AutomaTutor: An Educational Mobile App for Teaching Automata Theory

Steven Jordaan, Nils Timm(✉), and Linda Marshall

Department of Computer Science, University of Pretoria, Pretoria, South Africa
{sj.jordaan,nils.timm,linda.marshall}@up.ac.za

Abstract. Automata theory is one of the core theories in computer science because it allows scientists and practitioners to understand the complexity of computational problems, and thus, to develop efficient solutions to them. Several formal methods such as model checking are based on automata theory. Automata theory has traditionally been taught on a theoretical level. Students learned to define abstract machines via pen and paper without the possibility to actually run these machines. Over the years several automata simulators have been introduced and employed in teaching automata theory. These tools offer rich features for designing and manipulating automata, but do not provide pedagogical guidance to the user. In this paper we present the AutomaTutor, an educational tool on automata theory that particularly targets learners without prior knowledge of theoretical computer science. The tool is a mobile application that offers guided learning by solving interactive exercises. Exercises can be randomly generated or customised by an educator. The user-friendly touch interface allows learners to solve exercises by constructing finite automata or regular expressions that match with given languages. Learners receive immediate feedback. The application's focus on user experience and visualisation aims to make it accessible regardless of the technological background of the user. Our target is that the tool stimulates the students in their learning activities, and thus, leads to an improved understanding of automata theory and an increased interest in formal and theoretical aspects of computer science.

1 Introduction

Automata theory is the study of abstract machines and problems that can be solved by them. It is one of the core theories in computer science because it allows scientists and practitioners to understand the complexity of computational problems, and thus, to develop efficient hardware or software solutions to them. Several formal methods such as model checking are based on automata theory.

An integral part of teaching practical computer science is to make use of technology such as software development kits and tools for the visualisation of software components. Automata theory has traditionally been taught on a theoretical level. Students learned to define abstract machines via pen and paper without the possibility to actually run these machines. Over the years several

H. Barbosa and Y. Zohar (Eds.): SBMF 2023, LNCS 14414, pp. 131–140, 2024.
https://doi.org/10.1007/978-3-031-49342-3_8

automata simulators have been introduced and employed in teaching automata theory [15,18,19,21]. These tools offer rich features for designing and manipulating automata, but do not provide pedagogical guidance to the user. This may overwhelm novice learners and discourage them from using the tools in their learning activities.

In this paper we present the AutomaTutor, an educational tool on automata theory that particularly targets learners without prior knowledge of theoretical computer science. The tool is a mobile application that offers guided learning by solving interactive exercises. It brings abstract automata to life and allows students to get a practical experience of theoretical computer science. The application's focus on user experience and visualisation aims to make it accessible regardless of the technological background of the user. The tool is split into two major components: the tutorial and the sandbox.

The tutorial offers interactive exercises on finite automata and regular languages on different levels of difficulty. Exercises can be randomly generated or customised by an educator. The user-friendly touch interface allows learners to solve exercises by constructing automata or regular expressions that match with given languages. Learners receive immediate feedback on each exercise. For incorrect solutions feedback is provided in the sense of counterexample strings that are incorrectly accepted or rejected. The tool generates such counterexamples automatically. The tutorial also offers hints during exercises, providing users with additional guidance without revealing the full solution.

The sandbox provides similar functionalities as existing automata editors. While being less feature-rich than existing editors, the sandbox was designed with a focus on simplicity and user-friendliness. The purpose of the sandbox is to also offer a platform for experimental learning of automata theory. Learners can create their own finite automata in an easy touch-based manner and they can simulate runs of the automata for input strings. The sandbox visualises simulation runs by highlighting the taken transitions and indicating whether the run is accepting or rejecting.

From 2024 on the AutomaTutor will be officially used in teaching the undergraduate module "Theoretical Computer Science" at the University of Pretoria. The set of tutorial exercises will be further populated and aligned with the lecture content. By using the application, students will have the opportunity to enhance their understanding of automata theory without additional guidance by an instructor. Our target is that the AutomaTutor stimulates the students in their learning activities, and thus, leads to an improved understanding of automata theory and an increased interest in formal and theoretical aspects of computer science.

2 Related Work

The development of automata simulation tools started in the early 1960s [4]. A review of tools that have been developed since then can be found in [2]. Simulators can be classified into language-, table- and canvas-based tools. Language-based tools [1,9,11] present automata as programs of a programming language.

In table-based tools [7,8] automata can be constructed by means of a transition table. While these two kinds of tools lack visual features, the technological advances in the 1990s allowed to introduce canvas-based simulators [5,12,14] where users can draw automata as state-transition diagrams. Today canvas-based tools are still the most popular ones with prominent examples such as JFLAP [15] and JFAST [21]. Most simulators are desktop applications where user input is performed via mouse and keyboard. In recent years, a number of mobile applications for automata simulation has been introduced. CMSimulator [3], FLApp [13] and Automata Simulator [18] are mobile applications that allow a touch-based construction and simulation of automata. Each of the above-mentioned tools comes with a particular range of supported automata types, such as finite automata, pushdown automata, Turing machines and transducers. Educators have reported on successfully using simulation tools in teaching automata theory at university level [16]. Using the tools required an instructor-guided approach where the instructor had to manually create exercises to be solved with the tool. In contrast, our AutomaTutor offers guidance by the tool itself. The exercises are already integrated into the tool. They can be automatically generated and graded, and the learner receives immediate feedback. Currently, our AutomaTutor is limited to finite automata and regular expressions, and thus, does not offer as rich features as alternative tools. During the development of the AutomaTutor particular emphasis was put on following usability guidelines [6,10] in order to make the application as user-friendly as possible, which is an aspect that is typically not addressed in related work. Our work is loosely related to game-based approaches to learn automata theory [17,20]. The approaches integrated automata aspects into classical games such as Mastermind and Tower Defence. In the proposed games the focus is more on the fun aspect than on comprehensive learning.

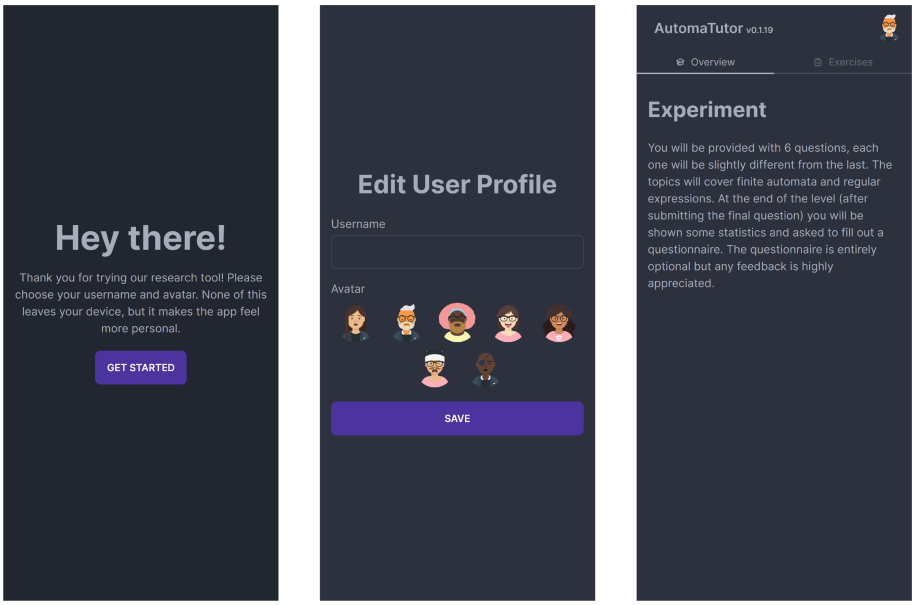
3 The AutomaTutor

In this section we present the AutomaTutor, a mobile application for teaching and learning automata theory. The application was designed with the purpose to provide computer science students a guided and engaging learning experience. In the design of the application particular emphasis was put on *usability* and *feedback* features. Moreover, *generation* features were integrated into the application which include the automatic generation of random exercises. After a brief tool overview we discuss the features of these types in separate subsections. The AutomaTutor can be accessed via the following link: https://sj-jordaan.github.io/masters-tool/. We recommend to use the application on a mobile phone. The source code of the AutomaTutor is available under https://github.com/SJ-Jordaan/masters-tool.

3.1 Overview

Upon first accessing the application, users are presented with a landing page (Fig. 1a) that emphasises the tool's experimental status and conveys appreciation

for their engagement. They are then guided to a profile customisation interface (Fig. 1b), which hints at forthcoming personalisation enhancements. Subsequently, users are ushered into the Tutor segment, where they receive a descriptive overview of the 'Experiment' category (Fig. 1c). From here, they have the option to transition to the 'Exercises' tab (Fig. 2a) to select their desired level. Upon selecting their avatar located at the top right corner, users are directed to their profile page (Fig. 2b). From this interface, they have the option to either revert to the Tutor segment or proceed to the Sandbox environment (Fig. 2c). As users delve deeper into the application, they encounter one of its most essential features:

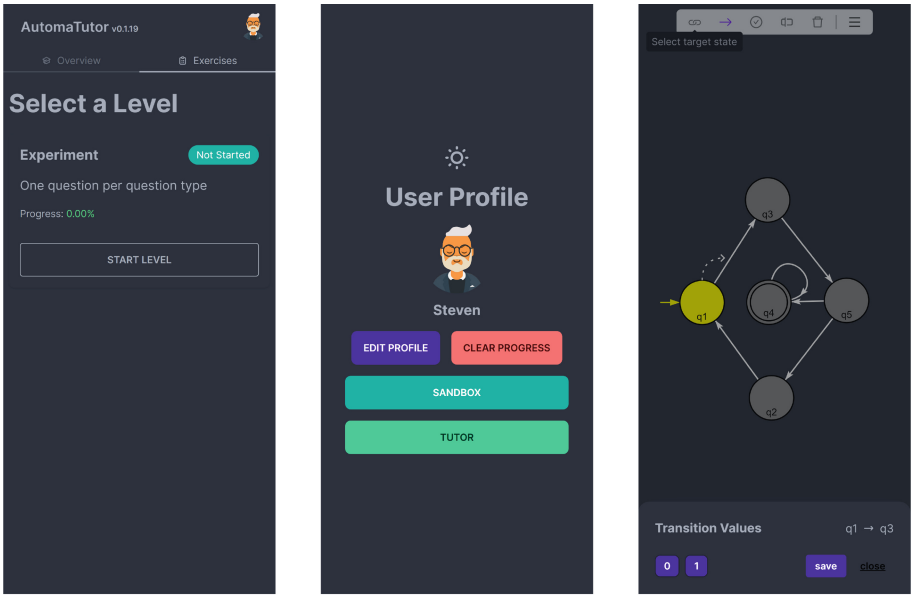


(a) Landing page (b) Add profile (c) Description

Fig. 1. Initial Application Journey.

Finite Automata Editor. The Finite Automata Editor is a pivotal feature of the mobile application, meticulously designed to provide an intuitive and efficient interface for users to interact with finite automata. The editor is implemented in two distinct contexts within the application: the exercise interface and the sandbox interface. Each interface encapsulates a range of functionalities, each contributing to the overall usability of the application.

Exercises. Exercises are a crucial component of the tool, providing users with opportunities to apply their knowledge and understanding of finite automata and



(a) Exercises (b) Profile page (c) Sandbox

Fig. 2. Navigating to Sandbox.

regular expressions. The exercises are designed to be diverse and challenging, offering a range of question types that cater to different learning styles and objectives. The tool supports exercises that involve providing a regular language, either textually, as a regular expression, or as an automaton, and asking the user to construct a corresponding regular expression, automaton, or perform a conversion between the two. Another variant of the exercise requires the user to provide a string that is contained within the given language. These exercises are designed to test the user's understanding of the core concepts and their ability to apply this knowledge in practical scenarios.

3.2 Usability

Prioritising user experience, the application combines intuitive design with efficiency. This section highlights features that enhance interaction with finite automata and foster a conducive learning environment.

Adding, Removing, and Modifying Components. Both the exercise and sandbox interfaces facilitate users to seamlessly add and modify transitions of an automaton. In the exercise interface, the states, including initial and accepting states, are predefined, and users can add or modify transitions between these states by labeling them with appropriate symbols. In contrast, the sandbox interface allows users to add, alter, and delete states and transitions using a context

menu, providing a more flexible and advanced environment for creating and modifying finite automata. Both interfaces provide adequately sized touch targets for the components of the automaton, ensuring sufficient spacing between touch targets to minimise the risk of accidental inputs.

Re-Arranging Components. Both interfaces employ an automatic arrangement algorithm for states and transitions within the editor to create a visually appealing and organised layout for the automaton. When a state is repositioned, the transitions connected to it automatically adjust their paths to maintain a clear and uncluttered representation of the automaton. The sandbox interface further enhances this feature by allowing users to lock the layout once they are satisfied with the arrangement, providing a balance between automatic layout optimisation and user control.

Zooming and Panning. Both interfaces automatically adjust the zoom level and position of the automaton diagram to ensure that it fits nicely and is legible on the screen. The sandbox interface supports gesture-based zooming and panning, which is a standard feature in modern mobile applications. It also automatically adjusts the zoom level to fit all elements on the screen when states are moved out of the interface's bounds, enhancing the usability of the interface, especially when working with larger automata.

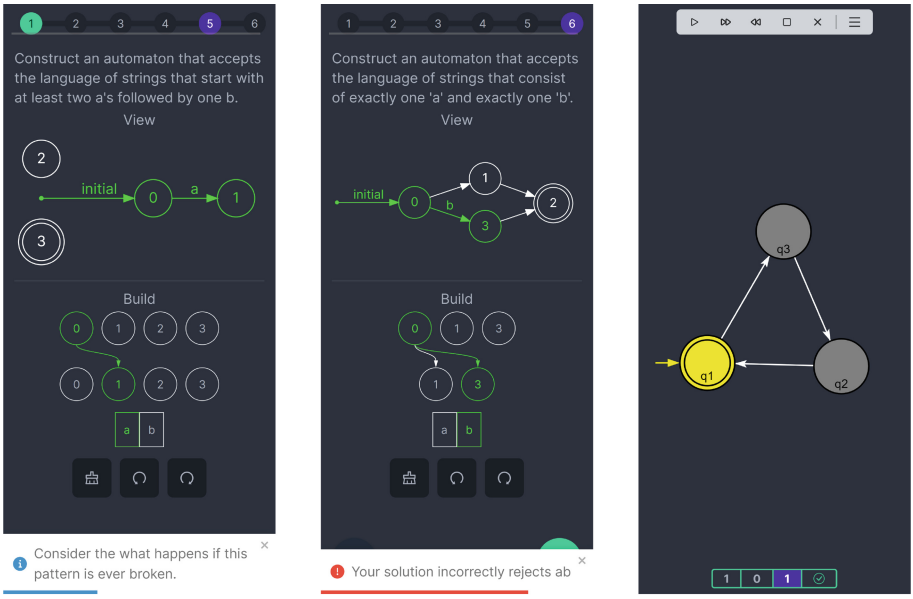
Progress Tracking: The tool implements a progress tracking feature which includes a timeline at the top of the exercise interface that shows users how far they have progressed and how many questions they have answered correctly. This feature allows users to gauge their progress, manage their time effectively, and stay motivated.

3.3 Feedback

Feedback in this application serves as more than just a response, it is a proactive tool that guides and informs the user's learning journey. This section details the feedback modalities, each designed to offer timely and constructive insights.

Hints. Hints, designed to scaffold problem-solving skills and alleviate user frustration, play a crucial role within the application. The application presents hints as textual prompts via a pop-up interface as seen in Figs. 3a. These hints, drawn from a manually curated pool, offer users a variety of suggestions to guide their problem-solving process.

Simulation. The Simulation (Fig. 3c) feature in the application is primarily integrated within the sandbox. This allows users to construct automata and simulate string inputs against them, fostering an active learning experience. The



(a) Requested Hint (b) Counter Example (c) Simulation Step

Fig. 3. Examples of Feedback.

automata simulation in the application utilises colour highlighting to indicate the active state and transitions during the simulation. The application also incorporates an animation that signifies the reading of the next input symbol. An accepted or rejected input is highlighted in green or red, respectively, providing clear feedback to the user.

Performance Feedback: Upon completing an exercise, users are presented with a summary of their performance. This feedback includes basic metrics such as the number of submission attempts, time taken, and percentage correct (in case some questions were left incorrect or unanswered). This information helps users identify areas where they excelled and those where improvement is needed, guiding their future learning efforts.

3.4 Generation

The generation features of the application encompass the creation of exercises and the generation of counterexamples.

Random Exercise Generation. The tool employs algorithms to generate random yet solvable exercises, ensuring a diverse range of tasks and providing users with new challenges each time they engage with the exercises. Users have

the option to select the difficulty scale and the number of questions, as well as the types of questions to generate, allowing for a personalised and targeted learning experience.

Counterexample Generation. The counterexample generation feature is a critical component of the application, providing users with immediate, constructive feedback upon the submission of their solutions. By comparing the user's solution to the memorandum solution, the application can generate counterexamples that highlight discrepancies in the user's understanding of the problem. This feedback is presented to the user in a clear, concise manner, accompanied by audio-visual cues to indicate an incorrect solution. The application generates two types of counterexamples: strings incorrectly accepted by the user's solution and strings incorrectly rejected by the user's solution. A generated counterexample is shown as a pop-up at the bottom of Fig. 3b.

4 Conclusion and Future Work

We presented a mobile application that can be used to construct and simulate finite automata as well as to solve interactive exercises on automata theory and regular expressions. In the design of the AutomaTutor emphasis was put on usability and feedback features. The application guides users in their learning activities without the need for additional intervention by an instructor. The question pool of the AutomaTutor is currently small but the implemented generation features allow to automatically generate random questions of several types. In preliminary user experiments we asked computer science students at the University of Pretoria to solve automata theory exercises via pen and paper, via the classical simulators as well as by using the AutomaTutor and to report on their experiences and preferences. The majority of students favoured the guided learning approach offered by the AutomaTutor. A more extensive experimental evaluation of the tool is in progress. From 2024 on the AutomaTutor will be officially used in teaching the undergraduate module "Theoretical Computer Science". Our conjecture is that the use of the tool will allow students gain a better understanding of the abstract topics of theoretical computer science. Although automata theory is not a formal method on its own, it it one of the core theories that is employed in several formal methods. Thus, with introducing our app we also intend to motivate and prepare students to study formal methods at postgraduate level.

In its current version the AutomaTutor only includes exercises on finite automata and regular expressions. As future work we are planning to extend the application such that further types of automata such as pushdown automata and Turing machines are supported. It is also planned to include Kripke structures and Büchi automata such that model checking subjects can be taught via the tool. Moreover, based on student feedback the usability, feedback and generation features of the AutomaTutor will be further improved.

References

1. Chakraborty, P.: A language for easy and efficient modeling of Turing machines. Prog. Nat. Sci. **17**(7), 867–871 (2007)
2. Chakraborty, P., Saxena, P.C., Katti, C.P.: Fifty years of automata simulation: a review. ACM Inroads **2**(4), 59–70 (2011)
3. Chuda, D., Trizna, J., Kratky, P.: Android automata simulator. In: Proceedings of the International Conference on e-Learning, pp. 80–4 (2015)
4. Coffin, R.W., Goheen, H.E., Stahl, W.R.: Simulation of a Turing machine on a digital computer. In: Proceedings of the November 12–14, 1963, Fall Joint Computer Conference, pp. 35–43 (1963)
5. Cogliati, J.J., Goosey, F.W., Grinder, M.T., Pascoe, B.A., Ross, R.J., Williams, C.J.: Realizing the promise of visualization in the theory of computing. J. Educ. Resour. Comput. (JERIC) **5**(2), 5–es (2005)
6. Google LLC.: Material design guidelines (2023). https://m3.material.io/
7. Hamada, M.: Supporting materials for active e-learning in computational models. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008. LNCS, vol. 5102, pp. 678–686. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69387-1_79
8. Hannay, D.G.: Interactive tools for computation theory. ACM SIGCSE Bull. **34**(4), 68–70 (2002)
9. Harris, J.: Programming non-deterministically using automata simulators. J. Comput. Sci. Coll. **18**(2), 237–245 (2002)
10. Inc., A.: Human interface guidelines (2023). https://developer.apple.com/design/human-interface-guidelines/
11. Knuth, D.E., Bigelow, R.H.: Programming language for automata. J. ACM (JACM) **14**(4), 615–635 (1967)
12. LoSacco, M., Rodger, S.: FLAP: a tool for drawing and simulating automata. Media **93**, 310–317 (1993)
13. Pereira, C.H., Terra, R.: A mobile app for teaching formal languages and automata. Comput. Appl. Eng. Educ. **26**(5), 1742–1752 (2018)
14. Robinson, M.B., Hamshar, J.A., Novillo, J.E., Duchowski, A.T.: A java-based tool for reasoning about models of computation through simulating finite automata and turing machines. In: The Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, pp. 105–109 (1999)
15. Rodger, S.H., Finley, T.W.: JFLAP: An Interactive Formal Languages and Automata Package. Jones & Bartlett Learning, Burlington (2006)
16. Rodger, S.H., Wiebe, E., Lee, K.M., Morgan, C., Omar, K., Su, J.: Increasing engagement in automata theory with JFLAP. In: Proceedings of the 40th ACM Technical Symposium on Computer Science Education, pp. 403–407 (2009)
17. Silva, R.C., Binsfeld, R.L., Carelli, I.M., Watanabe, R.: Automata defense 2.0: reediçao de um jogo educacional para apoio em linguagens formais e autômatos. In: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educaçao-SBIE), vol. 1 (2010)
18. Singh, T., Afreen, S., Chakraborty, P., Raj, R., Yadav, S., Jain, D.: Automata simulator: a mobile app to teach theory of computation. Comput. Appl. Eng. Educ. **27**(5), 1064–1072 (2019)
19. Traoré, M.K.: SimStudio: a next generation modeling and simulation framework. In: 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems (2010)

20. Vieira, M., Sarinho, V.: Automatamind: a serious game proposal for the automata theory learning. In: van der Spek, E., Göbel, S., Do, E.Y.-L., Clua, E., Baalsrud Hauge, J. (eds.) ICEC-JCSG 2019. LNCS, vol. 11863, pp. 452–455. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34644-7_45
21. White, T.M., Way, T.P.: JFAST: a java finite automata simulator. In: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education, pp. 384–388 (2006)