







Automata Simulator: A mobile app to teach theory of computation

Tuhina Singh  | Simra Afreen  | Pinaki Chakraborty  | Rashmi Raj  |
Savita Yadav  | Dipika Jain 

Division of Computer Engineering, Netaji
Subhas University of Technology, New
Delhi, India

Correspondence

Pinaki Chakraborty, Division of
Computer Engineering, Netaji Subhas
University of Technology, New Delhi
110078, India.
Email: pinaki_chakraborty_163@yahoo.com

Abstract

We developed a mobile app named Automata Simulator to simulate different types of automata and used it to teach the course on theory of computation. The students used the app to design and simulate automata during the lectures. The students also used a more sophisticated desktop application, JFLAP, to study advanced concepts related to automata theory after the lectures. An analysis of the examination results revealed that this approach helped the students to learn the concepts better and score 1% more marks. Moreover, a strong correlation between the marks obtained by the students for questions related to designing of automata and their total score was identified. In a survey conducted at the end of the semester, the students said that it was difficult to use two tools for learning the same concepts. However, they also felt that the combination of the two tools made the course more interactive and contributed positively in the process of learning.

KEYWORDS

automata simulator, automata theory, educational software, mobile app., Theory of computation

1 | INTRODUCTION

A course on theory of computation is included in the curriculum of undergraduate computer science program in most universities. The course typically discusses automata theory and computational complexity theory. Automata theory studies how computational tasks can be solved by different types of abstract machines, while computational complexity theory studies how efficiently those computational tasks can be solved.

Professors teaching the course on theory of computation typically use automata simulator software tools to explain the working of different types of abstract machines to the students. The first automata simulator tool was developed in the early-1960s [9] and increasingly sophisticated tools have been developed since [4,5]. Empirical studies have revealed that automata simulator tools can help students in studying the course on theory

of computation [15,33,37,38]. A typical automata simulator tool helps students in the following ways:

1. It allows a student to design an automaton.
2. It can display a transition diagram for an automaton designed by a student.
3. It can simulate the behavior of an automaton designed by a student for an arbitrary input string. This helps students to verify if their design is correct.
4. It often supports features for converting one type of automata into another and minimizing the number of internal states.

Most automata simulator tools developed till recently were implemented as applications for desktop computers. However, widespread use of smartphones has now motivated educationists to implement educational software as mobile apps [23]. Recently, two automata

simulator tools have been implemented as mobile apps [8,29]. Implementing an automata simulator tool as a mobile app makes it more pervasive and allows it to be used by students during lectures just as they use calculators during mathematics lectures.

We have developed a mobile app, named Automata Simulator, to simulate finite accepters, finite transducers, pushdown accepters, and Turing machines. The app can simulate both deterministic and nondeterministic forms of automata. We have used the app to teach the course on theory of computation in the Autumn 2018 semester. In this paper, we present the detail of the app and our experience of using it for teaching.

2 | RELATED WORK

Professors have been using automata simulator tools to teach the course on theory of computation since the 1960s. Different types of automata simulator tools have been developed over these years. We review their design paradigms and implementation media.

2.1 | Design paradigms

An automata simulator tool allows the user to model an automaton in some way and then simulates its behavior for an input string provided by the user. On the basis of how the user can specify the details of an automaton, automata simulator tools can be classified into language-, table-, and canvas-based tools.

In a language-based automata simulator tool, the automaton to be simulated is presented as a program in a suitably designed programming language. The program is then processed and simulated using appropriate tools. Different types of languages have been used by such tools as follows:

1. *Notational languages.* In a notational language, an automaton is specified using formal symbols. There are restrictions on the structure of the programs and the symbols that can be used in them. Notational languages are easy to learn and have been used in automata simulator tools over the years primarily because of their simplicity [9,20,21,36].
2. *Assembly-like languages.* In an assembly-like language, the behavior of an automaton is specified using low-level instructions like those in an assembly language. Instructions for jump and subroutine call are typically included in such languages. Some tools developed in the 1960s and the 1970s used assembly-like languages [11,30,34].

3. *Procedural languages.* Knuth and Bigelow [25] designed a procedural language to model automata. Procedural languages support high-level constructs for data abstraction and flow control, which can be used to efficiently define different types of automata. However, it requires more effort to learn a procedural language and no other tool uses procedural language for modeling automata.

4. *Descriptive languages.* In a descriptive language, an automaton is written in a format similar to one used in textbooks. This allows students to readily use tools that use descriptive languages. Descriptive languages have been used by some recent tools [3,4,6,13].

Notational languages and assembly-like languages can be classified as low-level languages. They lack sophisticated constructs for data abstraction and flow control. These languages are suitable for modeling small automata. Automata modeled in notational languages and assembly-like languages can be processed and simulated using simple tools. Alternatively, procedural languages and descriptive languages can be classified as high-level languages. Large and complex automata can be efficiently modeled in these languages. A compiled-simulation technique is used for automata defined in procedural languages and descriptive languages. In this technique, a program is first compiled and the object program is then simulated for an input string provided by the user. The technique ensures fast and error-free simulation.

A table-based automata simulator tool displays a form in which the user needs to fill in the details of the automaton to be simulated. Information to be provided includes the details of the internal states, input and output alphabets, and transition function. The tool can then display a transition diagram for the automaton and simulate the automaton for an input string provided by the user. Table-based automata simulator tools have been developed by Jagielski [22], Lee [26], Hannay [18,19], Vieira et al. [39], and Hamada [16]. Table-based tools allow users to define automata quickly. However, the fixed size of the form restricts simulation of large-sized automata.

Advances in software development technologies, especially the advent of Java, in the 1990s allowed professors to develop more interactive automata simulator tools. Many professors developed canvas-based tools and used them to teach their course on theory of computation [2,10,12,17,27,28,31,32,35,40]. Such a tool provides a canvas where the user has to draw the transition diagram of an automaton. The user has to drag and drop internal states on the canvas and connect them using transitions. The user also has to set the properties of the internal states and the transitions. The tool can

then simulate the behavior of the automaton for an input string provided by the user. Canvas-based tools have great didactic value. However, it requires more time to draw the transition diagram of an automaton than to define it in a programming language or specify its details in a table. The canvas also gets cluttered if the automaton has too many internal states or transitions.

Table 1 compares the advantages and disadvantages of the three design paradigms followed by automata simulator tools.

2.2 | Implementation media

Most automata simulator tools developed to date have been implemented as desktop applications. Professors distribute such tools among the students attending the course on theory of computation. Students can use such tools during the lectures and later for self study.

Some professors have also developed hardware to simulate automata. For example, Gilbert and Cohen [14] developed a hardware model of Turing machine. However, hardware models are rarely used to teach the course on theory of computation because they are not commercially available and difficult to construct in-house.

In recent years, two automata simulator tools have been implemented as mobile apps and used successfully in teaching the course on theory of computation. Chuda et al. [8] developed a mobile app named CMSimulator that can simulate finite accepters, pushdown accepters, and Turing machines. CMSimulator supports the table-based paradigm as well as the canvas-based paradigm of simulation. The app allows a user to draw the transition diagram of an automaton. Alternatively, the details of the automaton can be entered in an interactive form. Pereira and Terra [29] developed another mobile app, named FLApp, to simulate different types of automata. FLApp supports the canvas-based paradigm only and allows a user to define an automaton by drawing its transition diagram on the canvas. Both CMSimulator and FLApp are available for Android phones. Automata simulator tools implemented as mobile apps are more pervasive, that is, they can be used anywhere and anytime. Students can use these mobile apps more easily in the classroom during the lectures. However, smartphones have screens

smaller than that of desktop computers. This means that only a small portion of the canvas can be displayed on the screen. Moreover, a user operates a smartphone by touching its screen with a finger. The finger has lesser precision than an on-screen pointer and hides a part of the screen.

Chakraborty et al. [4] provides a detailed review of automata simulator tools.

3 | MATERIALS AND METHODS

We have developed a mobile app named Automata Simulator and used it to teach the course on theory of computation. Although software to simulate and teach about various forms of automata is being developed for decades, mobile apps are being used for the purpose only recently. We decided to implement our tool as a mobile app so that it can be used readily by students in the classroom. Since we implemented our tool as a mobile app, we had to select a simulation paradigm that can be appropriately realized using a smartphone. The user interface was also designed accordingly. Furthermore, our app uses terminology, notations, and illustrations similar to those used in textbooks to enhance pedagogy.

3.1 | The app

The objective of Automata Simulator is to allow students to quickly model and simulate automata in the classroom so as to augment the lectures in real-time. Automata Simulator follows the table-based paradigm of automata simulation. The app can be used to model and simulate finite accepters, finite transducers, pushdown accepters, and Turing machines (Figure 1). Both deterministic and nondeterministic automata can be simulated using the app. Figure 2 presents a flowchart describing the simulation process followed by the app. The app provides the user a form where details regarding the internal states and the transitions of the automaton are to be filled in (Figure 3). The app can then display a formal definition (Figure 4) and a transition diagram (Figure 5) of the automaton. The app can simulate the behavior of the automaton for an input string provided by the user.

TABLE 1 Comparison of the design paradigms

Design paradigm	Advantage	Disadvantage
Language-based	Can be used to simulate large and complex automata	Requires learning a programming language
Table-based	An automaton can be defined quickly	Suitable only for small automata
Canvas-based	Gives the feel of drawing transition diagrams on paper	Suitable only for small automata

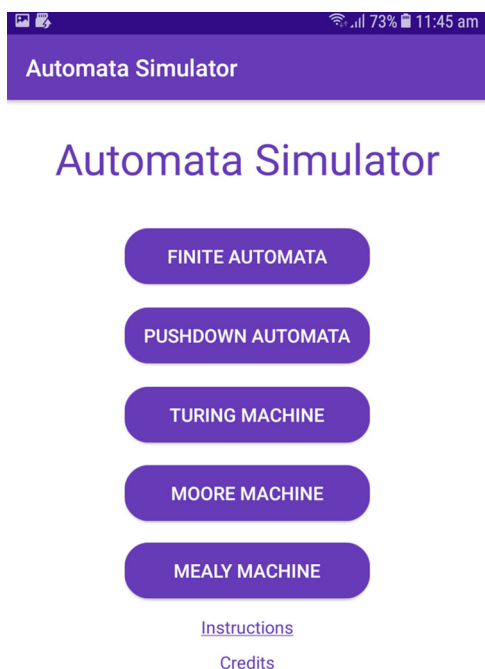


FIGURE 1 The main menu of Automata Simulator

The simulation process ends either with an acceptance (Figure 6a) or a rejection (Figure 6b) of the input string by the automaton. The steps in the simulation process are also displayed. Automata Simulator does not support features like conversion of one type of automata in another and minimizing the number of internal states, which are typically supported by tools that are implemented as desktop applications. Automata Simulator can be downloaded for free from Google Play [1].

3.2 | Teaching using the app

A course on theory of computation was offered to undergraduate computer science students at Netaji Subhas University of Technology in the Autumn 2018 semester. The course was attended by 185 students (41 female and 144 male). The students were divided into three sections having 58, 60, and 67 students, respectively.

Students in Sections I and II were encouraged to use Automata Simulator during the lectures. The instructor taught about different types of automata during the semester and the students were asked to design a few automata of each type in the classroom. The students designed and simulated those automata using Automata Simulator. The students were advised to use JFLAP [32] for self study after the lectures. The students were specifically recommended to use the advanced features of JFLAP. Students in Section III were taught without using any automata simulator tool during the lectures but were advised to use JFLAP for self study.

At the end of the semester, an examination was conducted and the students were awarded a score out of 100. Students of the three sections solved the same question paper. The questions were of average difficulty level and had no particular connection to either JFLAP or Automata Simulator. The question paper had questions worth 23 marks related to designing of automata. We compared the students who studied using Automata Simulator with those who studied without the app on the basis of the marks they received for the questions related to designing of automata and their total score. We also determined the correlation between the marks obtained by the students for the questions related to designing of automata and their overall score.

We also conducted a survey at the end of the semester. We asked the students how easy was it to use the automata simulator tools, how much did the tools help in making the course interactive and interesting, and how much did the tools help them in learning about the various types of automata. We asked the students to rate the tools in a 5-point Likert scale on these three criteria. We also asked the students for their general comments on the tools.

4 | RESULTS

We analyzed the scores obtained by the students in the examination and the feedback we received from them to assess the utility of Automata Simulator in teaching the course on theory of computation.

4.1 | Analysis of examination results

The students who studied the course using both Automata Simulator and JFLAP performed slightly better in the examination than those who studied using JFLAP only. The students who used both the tools scored 0.55% more marks in the questions related to designing of automata and 1.05% more marks overall (Figure 7). We also found that the marks scored by students for the questions related to designing of automata were strongly correlated with their total score (Pearson correlation coefficient, $\rho = 0.68$).

4.2 | Analysis of student feedback

The students felt that it is easier to use JFLAP only than a combination of Automata Simulator and JFLAP (Figure 8a). However, the students felt that the combination of Automata Simulator and JFLAP made the course more interactive and interesting (Figure 8b). They also felt that the two tools together

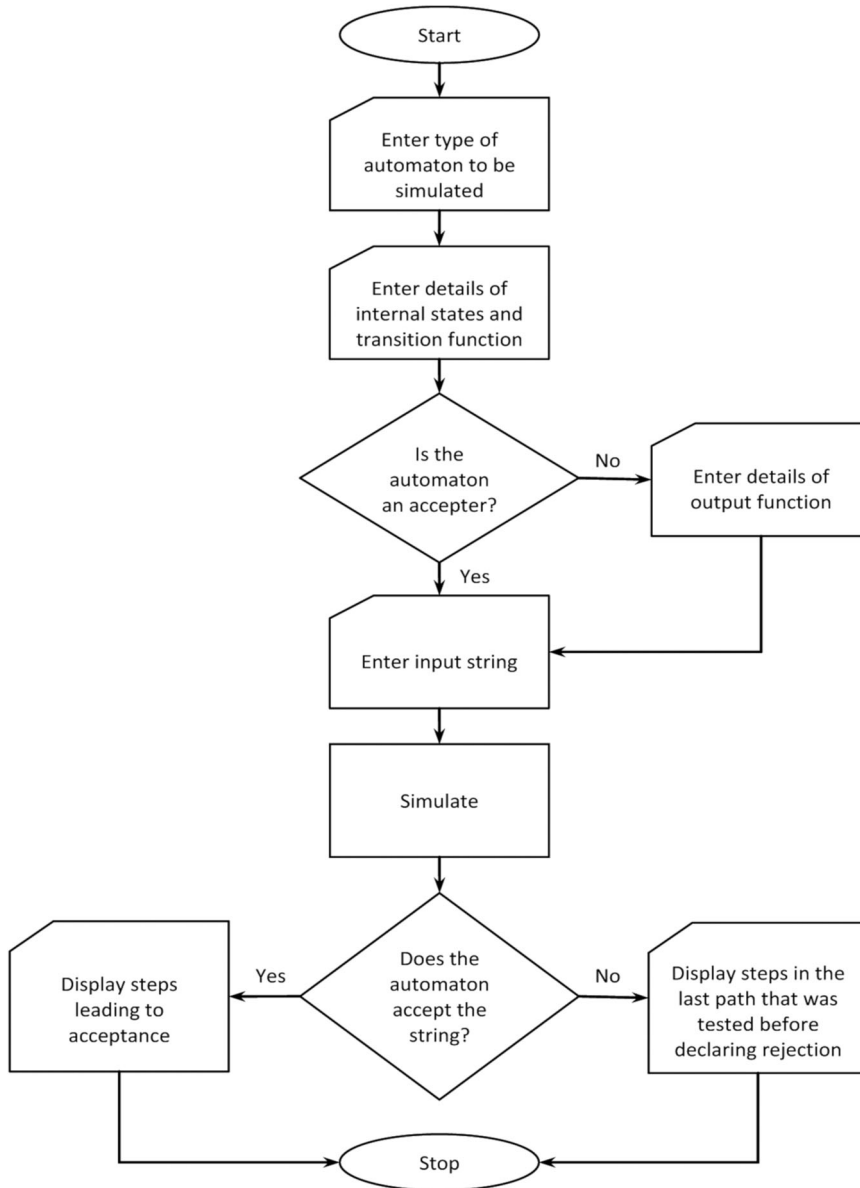


FIGURE 2 Flowchart describing the simulation process followed by Automata Simulator

provided a better overall learning experience (Figure 8c). In their comments, the students said that they felt Automata Simulator and JFLAP helped them in learning about the different types of automata. However, some students complained about the quality of the transition diagrams displayed by Automata Simulator.

5 | DISCUSSION

We observed that students find educational software implemented as desktop applications easier to use when compared to mobile apps. Superior computational power

and larger screen of a desktop computer allow educational tools running on it to perform computationally intensive tasks and display more information at a time, respectively. However, mobile apps are more pervasive. Students can use educational software implemented as mobile apps in classrooms and laboratories [23,24], and also in their homes. We used a combination of a mobile app and a desktop application to teach the course on theory of computation. Students used the mobile app in the classroom and the more sophisticated desktop application for self study. This helped to make the course more interactive and provided a better learning experience to the students.

Educational software implemented as mobile apps can be used successfully as teaching aid. However, the mobile

Finite Automata

Number of states: 3

The states are: q0, q1, q2

Initial state: q0

Final states: q2

Number of transitions: 5

ENTER TRANSITIONS

Current State	Current Symbol	New State
q0	a	q0
q0	b	q1
q1	b	q1
q1	c	q2
q2	c	q2

DEFINITION TRANSITION DIAGRAM SIMULATION

FIGURE 3 Filling in the details of an automaton. Here, a deterministic finite acceptor to accept strings of the form $a^l b^m c^n$ where $l, m, n > 0$ is being modeled

Finite Automata

$(\{q_0, q_1, q_2\}, \{a, b, c\}, \delta, q_0, \{q_2\})$

$\delta(q_2, c) = \{q_2\}$

$\delta(q_1, b) = \{q_1\}$

$\delta(q_0, b) = \{q_1\}$

$\delta(q_1, c) = \{q_2\}$

$\delta(q_0, a) = \{q_0\}$

FIGURE 4 A formal definition of the automaton as displayed by Automata Simulator

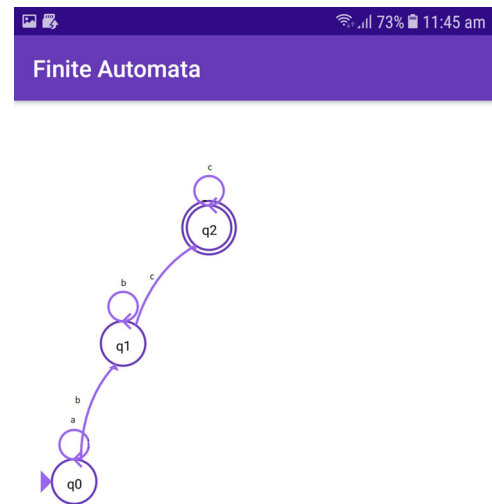


FIGURE 5 A transition diagram of the automaton as displayed by Automata Simulator

(a)

Finite Automata

Input String : aabbcc

PROCEED

q0aabbcc | q0abbcc | q0bbcc | q1bcc | q1cc | q2c | q2 (ACCEPT)

(b)

Finite Automata

Input String : aacc

PROCEED

q0aacc | q0acc | q0cc (REJECT)

FIGURE 6 Simulation of the automaton (a) for the input string aabbcc resulting in an acceptance and (b) for the input string aacc resulting in a rejection

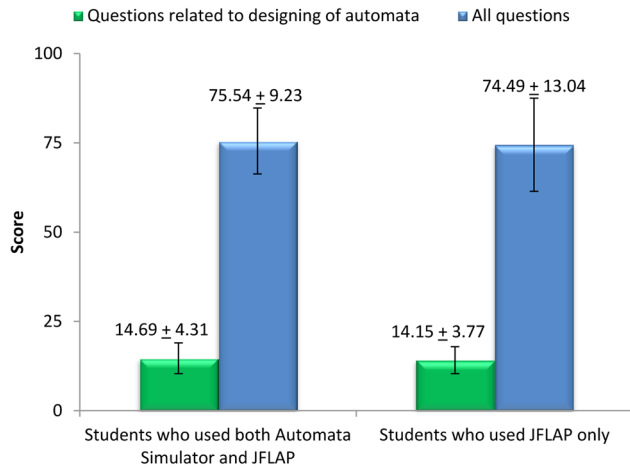


FIGURE 7 Marks (mean + SD) scored by the students in the examination. The examination was of 100 marks and had questions worth 23 marks on designing of automata. One hundred eighteen students studied the course using both Automata Simulator and JFLAP, while 67 students studied the using JFLAP only

apps should be integrated properly with the curriculum and used in conjunction to other software, if necessary. Care should be taken while developing the content and designing the user interface of the mobile apps.

6 | CONCLUSION

We developed a mobile app named Automata Simulator that can be used to design and simulate different types of automata. We used the app along with JFLAP to teach the course on theory of computation to undergraduate computer science students. Analyses of student feedback and examination results reveal that the approach helped in making the course interactive (Figure 7) and facilitated the students to learn better (Figure 6). We recommend that educational software for teaching engineering courses be implemented as a combination of desktop applications and mobile apps. Students may use the

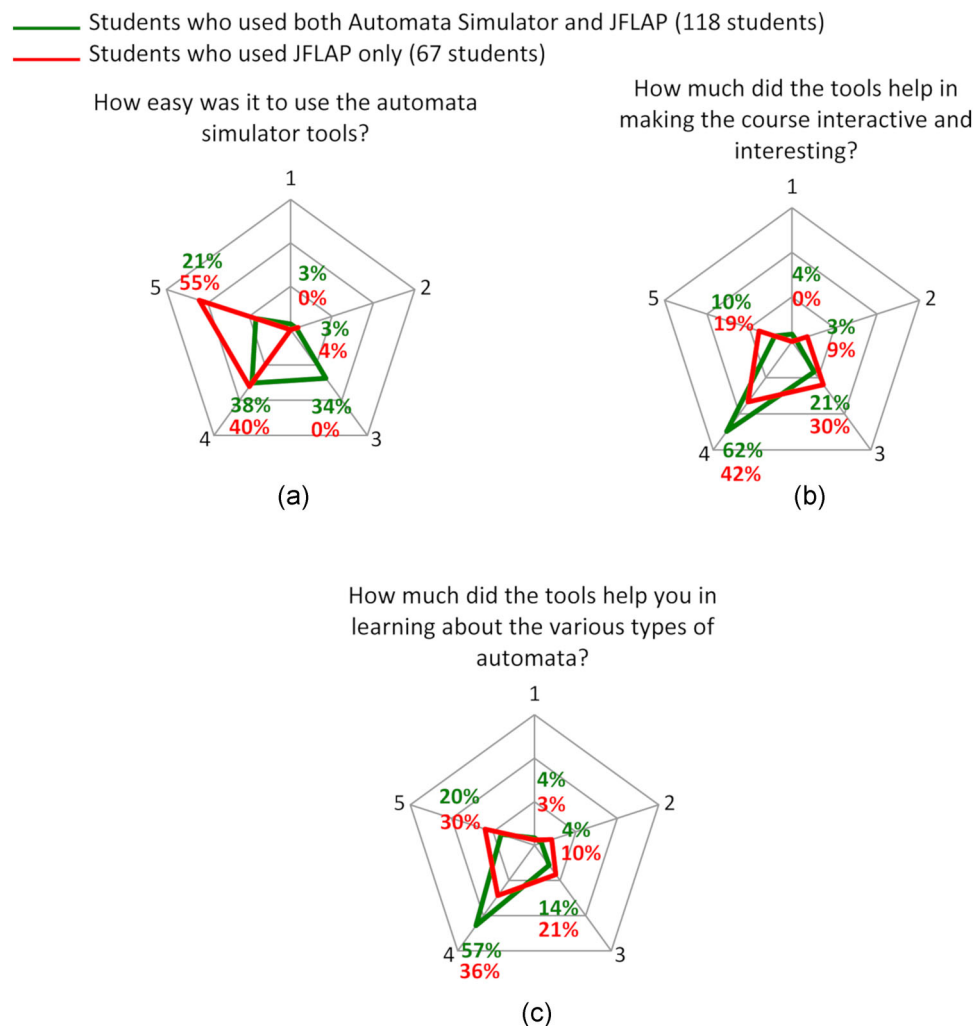


FIGURE 8 Feedback received from the students. A vertex in a polygon represents the percentage of students who gave a certain rating to the tools according to a specific criterion

mobile apps during lectures and laboratory sessions, and use the desktop applications for advanced and computation-intensive features.

ORCID

Tuhina Singh  <http://orcid.org/0000-0002-3292-3366>
 Simra Afreen  <http://orcid.org/0000-0003-1407-6009>
 Pinaki Chakraborty  <http://orcid.org/0000-0002-2010-8022>
 Rashmi Raj  <http://orcid.org/0000-0002-1690-5782>
 Savita Yadav  <http://orcid.org/0000-0002-4334-1767>
 Dipika Jain  <http://orcid.org/0000-0002-7573-6851>

REFERENCES

1. Automata Simulator, <https://play.google.com/store/apps/details?id=com.singh.tuhina.automatasimulationcopy&hl=en>.
2. J. Barwise and J. Etchemendy, *Computers, visualization, and the nature of reasoning*, The Digital Phoenix: How Computers are Changing Philosophy (T. W. Bynum and J. H. Moor eds.), Blackwell, 1993, pp. 93–116.
3. P. Chakraborty, *A language for easy and efficient modeling of Turing machines*, Prog Nat Sci **17** (2007), 867–71.
4. P. Chakraborty, P. C. Saxena, and C. P. Katti, *Fifty years of automata simulation: A review*, ACM Inroads **2** (2011), 59–70.
5. P. Chakraborty, P. C. Saxena, and C. P. Katti, *Automata simulators: Classic tools for computer science education*, Brit J Educ Technol **43** (2012), E11–E13.
6. P. Chakraborty, P. C. Saxena, and C. P. Katti, *A compiler-based toolkit to teach and learn finite automata*, Comput Appl Eng Educ **21** (2013), 467–74.
7. P. Chakraborty, S. Taneja, N. Anand, A. Jha, D. Malik, and A. Nayar, *An optimizing compiler for Turing machine description language*, IUP J Comput Sci **5** (2011), 33–9.
8. D. Chudá, J. Trizna, and P. Kratky, *Android automata simulator*, Proceedings of the International Conference on e-Learning (2015), 80–4.
9. R. W. Coffin, H. E. Goheen, and W. R. Stahl, *Simulation of a Turing machine on a digital computer*, Proceedings of the Fall Joint Computer Conference (1963), 35–43.
10. J. J. Cogliati, F. W. Goosey, M. T. Grinder, B. A. Pascoe, R. J. Ross, and C. J. Williams, *Realizing the promise of visualization in the theory of computing*, ACM J Educ Res Comput **5** (2005), 1–17.
11. M. W. Curtis, *A Turing machine simulator*, JACM **12** (1965), 1–13.
12. A. Esmoris, C. I. Chesñevar, and M. P. González, *TAGS: A software tool for simulating transducer automata*, Int J Elec Eng Educ **42** (2005), 338–49.
13. M. Čerňanský, M. Nehéz, D. Chudá, and I. Polický, *On using of Turing machine simulators in teaching of theoretical computer science*, J Appl Math **1** (2008), 301–12.
14. I. Gilbert and J. Cohen, *A simple hardware model of a Turing machine: Its educational use*, Proceedings of the ACM Annual Conference (1972), 324–329.
15. M. T. Grinder, *A preliminary empirical evaluation of the effectiveness of a finite state automaton animator*, inroads – ACM SIGCSE Bull **35** (2003), 157–61.
16. M. Hamada, *Supporting materials for active e-learning in computational models*, Lecture Notes Comput Sci **5102** (2008), 678–86.
17. M. Hamada, and K. Shiina, *A classroom experiment for teaching automata*, inroads – ACM SIGCSE Bull **36** (2004), 261.
18. D. G. Hannay, *Hypercard automata simulation: Finite-state, pushdown and Turing machines*, ACM SIGCSE Bull **24** (1992), 55–8.
19. D. G. Hannay, *Interactive tools for computation theory*, inroads – ACM SIGCSE Bull **34** (2002), 68–70.
20. J. Harris, *Programming nondeterministically using automata simulators*, J Comput Sci Coll **18** (2002), 237–45.
21. J. Harris, *YATS—Yet another Turing machine simulator*, J Comput Sci Coll **13** (1998), 31–5.
22. R. Jagielski, *Visual simulation of finite state machines*, ACM SIGCSE Bull **20** (1988), 38–40.
23. D. Jain, P. Chakraborty, and S. Chakraverty, *Smartphone apps for teaching engineering courses: Experience and scope*, J Educ Technol Syst **47** (2018), 4–16.
24. D. Jain and P. Kumar, *Teaching algorithms using an Android application*, Towards Extensible and Adaptable Methods in Computing (S. Chakraverty, A. Goel, and S. Misra eds.), Springer, 2018, pp. 351–61.
25. D. E. Knuth and R. H. Bigelow, *Programming languages for automata*, JACM **14** (1967), 615–35.
26. M. C. Lee, *An abstract machine simulator*, Lecture Notes Comput Sci **438** (1990), 129–41.
27. M. LoSacco and S. H. Rodger, *FLAP: A tool for drawing and simulating automata*, Proceeding of the World Conference on Educational Multimedia and Hypermedia (1993), 310–317.
28. J. McDonald, *Interactive pushdown automata animation*, inroads – ACM SIGCSE Bull **34** (2002), 376–80.
29. C. H. Pereira and R. Terra, *A mobile app for teaching formal languages and automata*, Comput Appl Eng Educ **26** (2018), 1742–52.
30. J. C. Pierce, W. E. Singletary, and J. E. Vander Mey, *Tutor – A Turing machine simulator*, Inf Sci **5** (1973), 265–78.
31. M. B. Robinson, J. A. Hamshar, J. E. Novillo, and A. T. Duchowski, *A Java-based tool for reasoning about models of computation through simulating finite automata and Turing machines*, inroads – ACM SIGCSE Bull **31** (1999), 105–109.
32. S. H. Rodger and T. Finley, *JFLAP – An interactive formal languages and automata package*, Jones and Bartlett (2006).
33. S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar, and J. Su, *Increasing engagement in automata theory with JFLAP*, Proceedings of the Fortieth SIGCSE Technical Symposium on Computer Science Education (2009), 403–407.
34. L. L. Rose, N. D. Jones, and B. H. Barnes, *Automata: A teaching aid for mathematical machines*, ACM SIGCSE Bull **3** (1971), 12–20.
35. L. A. Sanchis, *Computer laboratories for the theory of computing course*, J Comput Sci Coll **16** (2001), 262–269.
36. T. A. Scott, *Turing machine simulation used in a breadth first computer science course*, J Comput Sci Coll **22** (2006), 240–245.
37. V. Shekhar, A. Prabhu, K. Puranik, L. Antin, and V. Kumar, *JFLAP extensions for instructors and students*, Proceedings of the IEEE Sixth International Conference on Technology for Education (2014), 140–143.

38. V. S. Shekhar, A. Agarwalla, A. Agarwal, B. Nitish, and V. Kumar, *Enhancing JFLAP with automata construction problems and automated feedback*, Proceedings of the Seventh International Conference on Contemporary Computing (2014), 19–23.
39. L. F. M. Vieira, M. A. M. Vieira, and N. J. Vieira, *Language emulator, a helpful toolkit in the learning process of computer theory*, inroads – ACM SIGCSE Bull **36** (2004), 135–139.
40. T. M. White, T. P. and Way, *jFAST: A Java finite automata simulator*, inroads – ACM SIGCSE Bull **38** (2006), 384–388.

AUTHOR BIOGRAPHIES



Tuhina Singh is pursuing her degree in Computer Engineering at Netaji Subhas University of Technology. Her area of interest includes theory of computation, artificial intelligence and educational software.



Simra Afreen has completed her degree in Computer Engineering from Netaji Subhas University of Technology. Her area of interest includes computational intelligence and theory of computation.



Pinaki Chakraborty received his BTech from Indraprastha University and his MTech and PhD from Jawaharlal Nehru University. He is an assistant professor at Netaji Subhas University of Technology. His area of research includes systems

software and educational software.



Rashmi Raj received her BTech from Uttar Pradesh Technical University and her MTech from Indraprastha University. She is a guest lecturer at Netaji Subhas University of Technology. Her area of research includes theory of computation and compiler design.



Savita Yadav received her BSc and MSc in Physics from the University of Delhi and her MTech in Computer Engineering from Guru Jambheshwar University of Science and Technology. She is an assistant professor at Netaji Subhas University of Technology where she research and teaches courses on human-computer interaction.



Dipika Jain received her BTech and MTech from Maharshi Dayanand University. She is a guest lecturer at Netaji Subhas University of Technology. Her area of research includes theory of computation, computer networking and educational technology.

How to cite this article: 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*. 2019;27: 1064–1072. <https://doi.org/10.1002/cae.22135>