Teaching Formal Languages with Visualizations and Auto-Graded Exercises

Mostafa Mohammed
profmdn@vt.edu
Virginia Tech
Blacksburg, VA
& Assiut University
Assiut, Egypt

Clifford A. Shaffer
shaffer@vt.edu
Virginia Tech
Blacksburg, VA

Susan H. Rodger
rodger@cs.duke.edu
Duke University
Durham, NC

ABSTRACT

The material taught in a Formal Languages and Automata (FLA) course is mathematical in nature and requires students to practice proofs and algorithms to understand the content. Traditional FLA textbooks are heavy on prose, and homework typically consists of solving many paper exercises. Instructors often make use of Finite State Machine simulators like the JFLAP package. JFLAP allows students to interactively build models and apply different algorithms to these models, providing both a more interactive and a more visual approach. However, course materials have still traditionally relied largely on prose and hand-graded exercises, limiting both the interaction and the amount of practice. In this paper, we propose an eTextbook with integrated tools (simulators and auto-graded exercises) that allow for greater interactivity and levels of engagement. To evaluate the pedagogical effectiveness of our approach, we conducted performance evaluations across different offerings of an FLA course. Results indicate that students using the integrated eTextbook performed better than did a control group using a traditional textbook approach. Students gave positive feedback regarding the usefulness of the auto-graded exercises for practicing different FLA concepts.

CCS CONCEPTS

• **Social and professional topics** → *Student assessment*; • **Applied computing** → **Interactive learning environments**; • **Software and its engineering** → *Simulator / interpreter*.

KEYWORDS

Formal Languages, Auto-graded Exercises, OpenFLAP, JFLAP, Visualizations, OpenDSA

ACM Reference Format:
Mostafa Mohammed, Clifford A. Shaffer, and Susan H. Rodger. 2021. Teaching Formal Languages with Visualizations and Auto-Graded Exercises. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education (SIGCSE '21), March 13–20, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3408877.3432398

SIGCSE '21, March 13–20, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8062-1/21/03…$15.00
https://doi.org/10.1145/3408877.3432398

1 INTRODUCTION

Formal Languages and Automata (FLA) is a core course in the Computer Science theory curriculum [14]. This course typically includes topics like finite state machines, languages, their representation by regular expressions and grammars, Turing machines, and possibly computability theory and complexity theory [29, 30]. FLA topics are applied in many daily activities in the life of computing professionals, like parsing and regular expressions. However, FLA courses face a few challenges. They are often presented as fairly abstract and highly mathematical. This has the benefit of making students practice useful skills like proof writing, but also may make it less appealing to students more used to the hands-on style of the typical CS programming course. A typical FLA class presents several models of computing (deterministic and non-deterministic finite state machines, regular expressions, push-down automata, context-free languages, Turing machines), with many proofs about their relationships and limitations. Understanding these limitations helps students to understand how computers work [26]. Often, students are asked to create machines or languages using the various models. To support this process, many instructors have their students use simulators, such as the popular JFLAP system [13, 24].

To help students better understand this hard topic, we implemented an eTextbook for FLA. Our eTextbook combines prose, visualizations to explain models and algorithms, simulators, and many interactive exercises. The book is built on the OpenDSA infrastructure [8] and uses the JavaScript Algorithm Visualization (JSAV) framework [15] to implement the visualizations and exercises. We have also re-implemented and expanded on key parts of JFLAP, and tightly integrated this into the eTextbook; this part of our system is referred to as OpenFLAP.

OpenDSA has a tradition of using a rich collection of interactive exercises to keep students engaged. The core data structures content relies on traditional simple question types (such as multiple choice), combined with small programming exercises [4] and proficiency exercises, as originally popularized by TRAKLA [16, 18]. Proficiency exercises require students to apply specific algorithm steps to a visual presentation of a data structure, such as a tree or array. Thus, a student might be asked to demonstrate their knowledge of a sorting algorithm or a tree construction algorithm.

Our eTextbook extends these ideas to the FLA curriculum by providing the equivalent of both traditional programming exercises and OpenDSA proficiency exercises. Finite State Machine models (including Turing Machines) are represented within OpenFLAP as graphs, or by regular expressions and grammars. We provide a wide range of exercises where students are required to construct

a machine or a grammar to define a language. This has the feel of a traditional programming exercise in that we can then test the student's proposed machine or grammar against a collection of test cases, similar to traditional unit testing. We include auto-graded exercises to test the student's ability to create specified Deterministic Finite Automata (DFA), Nondeterministic Finite Automata (NFA), Push-Down Automata (PDA), or Turing Machines (TM). We also provide a collection of proficiency exercises, where a student demonstrates their knowledge of applying an algorithm to a given machine by showing the appropriate steps that would be taken. Examples include Minimizing a DFA, or converting a NFA to a DFA.

Our eTextbook helps students understand FLA content through simulations, visualizations, and auto-assessed exercises. Therefore, we hypothesize that we have increased students' interaction, and their opportunity to practice the course content beyond what was previously possible. We conducted studies to test this hypothesis.

This paper is organized as follows. We present a review of prior efforts to enhance FLA courses in Section 2. Section 3 presents OpenFLAP. Section 4 gives an overview to the implemented book with visualizations and different types of exercises. Our evaluation protocol and results on student satisfaction and student performance are presented in Sections 5 and 6. We discuss possible threats to validity in Section 7, and conclusions and future work plans in Section 8.

2 LITERATURE REVIEW

The Java Formal Languages and Automata Package (JFLAP) [24] simulates most of the models that are used in FLA courses. It allows students to view models, apply algorithms on these models, or test these models with different input strings for acceptance [9]. For example, a FSA simulator can help a student to determine which strings are accepted and which strings are rejected by the machine. JFLAP increases student engagement and interaction with the course content [9, 25]. JFLAP does not allow students to identify if their solutions are correct or not. Attempts have been made to provide such feedback [27, 28], but without assessing their impact on students' understanding.

Of course, JFLAP is not the only simulator for FLA as [2] summarizes 50 years of automata simulation. While JFLAP is broad, covering most FLA topics, some authors have created special simulators for specific models [3]. Examples include the FSA simulator [10], the Interactive Pushdown Automata Animation (IPAA) [20], a PDA simulator [5], and Turing Machine simulators [1, 11]. Most of this functionality can be found in JFLAP.

Many have attempted to enhance the FLA course. Vijayalaskhmi and Karibasappa [35] studied the use of activity-based learning. They had student groups discuss problems and then present their solution to other students. Groups were evaluated by how well they expressed the solution. [34] presents a web-based online Intelligent Learning System for Automata (ILSA). Each student is presented with a series of questions to answer. The tool provides features to help students solve those questions like a glossary, tips, simulators, extra challenges, peer references, a leader board, badges, and a scoreboard. Students are free to use any available feature to get help. To test ILSA, the authors asked 36 students to try the tool for 30 minutes. Then the students completed a survey.

Algorithm analysis content in OpenDSA was originally presented using text and static images, like any traditional textbook. By examining logging files, it was found [6] that students often skip reading this material completely, and as a result, their understanding of this content is poor. The authors developed Algorithm Analysis Visualizations (AAVs), that deliver the abstract mathematical concepts of algorithm analysis using more engaging interactive visualizations. AAVs helped the students to engage more with the material, and they scored higher on relevant test questions than did the control group.

3 OPENFLAP

JFLAP is used extensively in FLA courses to help students visualize and observe the behavior of models and associated algorithms [25]. However, JFLAP has three disadvantages from the point of view of integrating material into an eTextbook. First, it was written in Java and is a stand-alone application that runs on the student's machine. This does not allow it to easily tie to online tools like OpenDSA, or to an LMS [21, 22]. Second, JFLAP does not have any mechanism for auto-grading exercises. Students can use JFLAP to help solve many typical homework problems, such as creating a machine to recognize a given language. But they get little feedback from JFLAP about whether their answer is correct. Instead, they must wait until the homework is hand graded by instructional staff. In contrast, we have reached the state where many programming assignments can be done with immediate feedback from auto-graders, largely based on testing the program against unit tests. Third, JFLAP is a separate tool that is not associated with any automata book. There is a JFLAP book [23], but the book just shows how to use JFLAP features. It doesn't explain FLA concepts; students must use another textbook to learn these. We would prefer to integrate simulations directly with the course content.

These drawbacks inspired us to develop an open-access, web-based version of JFLAP with enhanced support for auto-graded exercises. We have largely re-implemented JFLAP functionality within the OpenDSA framework. We refer to it as OpenFLAP. OpenFLAP is implemented using the JSAV library. OpenFLAP provides many visualizations to help students understand different aspects of Formal Languages, similar to JFLAP. OpenFLAP also allows us to create exercises, auto-assess them, and report the result to an LMS through OpenDSA's standard framework. However, any re-implementation of JFLAP is challenging and time-consuming. JFLAP supports a large number of model types, and a rich variety of algorithms to manipulate them. To help manage this effort, we first classified JFLAP functionality into three groups. The first contains minimum functionality required to create visualizations of the core material covered in the course. These include fundamental models like DFAs, NFAs, PDAs, and TMs, along with their core algorithms like converting a NFA to a minimized DFA. The second group contains the functions and models that are not essential to build our book. This group has models like "Accept by empty stack" PDAs and "single character input" PDAs. The final group has content outside our course scope. This includes models and algorithms that are used in compiler courses, like LL parsing and CYK parsing. For the FLA eTextbook we implemented the functions found in the first two groups, and left the last group as future work.

Table 1: Visualizations (V), exercises (PE, AE), and figures (F) in our eTextbook

Topic	# PE	# AE	# V	# F
CF Languages, Grammars	17	10	6	9
Regular languages	3	24	18	23
Identifying non-regular languages	0	0	15	0
Pushdown Automata	0	8	2	7
Identifying Non-CF Languages	0	0	13	0
Turning Machines	0	9	3	10
Total	20	51	57	49

In the first priority implementation we included the basic engines for automatic generation of visualizations of a machine executing on a specified input, and auto-grading by unit tests. These core functionalities are similar, since if a machine can be executed on an input (and accept or reject on the input decided by the machine), then it is a simple next step to determine if the machine (as provided by a student) matches the correct behavior on a collection of pre-selected inputs (i.e., unit tests).

To do the necessary implementation, we enlisted a small army of undergraduate volunteers and independent study students. All told, over the past three years, approximately 20 students have worked on this project.

4 AN ETEXTBOOK FOR FLA

Previous studies [6] have found that students skip reading prose in eTextbooks, especially mathematical material that is hard to understand. Past success in this area motivated us to create a pool of JSAV visualizations and exercises that reduce the amount of text needed to cover a given topic, and promote increased engagement.

Our eTextbook covers topics typically taught in a FLA course. These include DFAs, NFAs, equivalence and conversion between them, RegEx, regular grammars, their equivalence with NFAs, and closure properties. Additional topics include context-Free (CF) grammars and languages, conversion to special forms, PDAs, the Pumping Lemma for recognizing non-CF languages, and Turing machines. We also include material that might not be in all FLA courses about basic complexity and computability theory, including NP-completeness, unaccountably infinite sets, and unsolvable problems.

The eTextbook currently includes 57 visualizations (V) in the form of slideshows, 71 exercises (including proficiency exercises (PE) and auto-graded "create and test a machine" style exercises (AE)), and 49 additional static figures. Table 1 shows a breakdown for the topics covered by exercises, figures, and visualizations.

4.1 Exercises

In addition to standard multiple choice or T/F style questions, we present two types of exercises: a) auto-graded exercises (AE) and b) proficiency exercises (PE). AE are similar to programming exercises. Students are required to devise a machine (expressed as a graph) that recognizes a specified language (meaning it identifies if a given string is in the language or not), or a grammar for the language. The student's solution is tested by applying multiple test cases, in the form of strings that must be correctly identified as in the language or not. Passing all test cases, if they were selected properly, should give strong evidence that the solution is correct. The percentage of correct test-cases will determine the grade they earn for each exercise. There are exercises of this type to build many types of models: DFA, NFA, PDA, TM, RegEx, or grammar.

Figure 1 shows an example of an exercise to create a DFA. In this example, the student has created a DFA for the problem statement with eight states shown at the bottom of the figure, and graded their proposed answer for the problem by clicking on the Grade button. Since the solution is not correct, we see that some test cases are labeled red to indicate that they have failed.

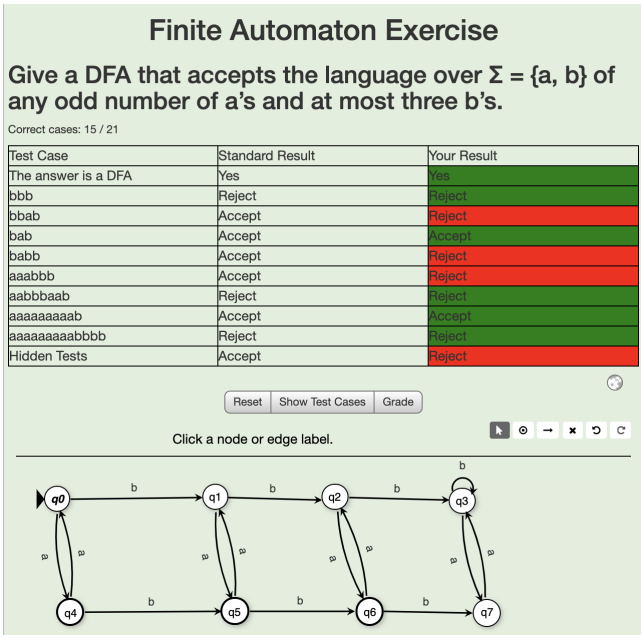

Figure 1: DFA auto-graded exercise

FLA courses require students to understand many algorithms, such as how to convert an NFA to an equivalent DFA. So we want to include exercises that ask students to imitate the algorithm's steps on a given model. This is similar in spirit to a major motivation for creating OpenDSA in the first place for data structures and algorithms courses. This exercise type is referred to as a proficiency exercise (PE). Proficiency exercises ask students to apply an algorithm and show its steps operating on a machine or grammar to produce the correct answer. If the student is able to do all of the steps in sequence, then the final result will be correct and the student will earn the full credit. Getting some or all steps incorrect will result in reduced credit.

Using OpenFLAP functionality for executing algorithms on formal languages models, we were able to use the OpenDSA PE framework to create the necessary exercises. The student is given the model and an input, and must indicate how the model is changed at each step. Since OpenFLAP can determine the correct next change to the model for that algorithm, it is relatively easy to match the student's modification to the correct one. Students can then be notified if their step is correct or not, and if necessary their view of the model can be corrected to show the correct step. The student

is scored based on how many steps they get correct. Details on the use of proficiency exercises in data structures and algorithms courses can be found in [7, 16]. As an example, a student could try to convert an NFA to DFA. Their solution might score 85% if the student did 17 correct steps out of 20 total steps. The incorrect steps would be displayed to the student as they progress through the exercise, to help them identify their mistakes. They might get messages such as "b: There are no paths on that terminal", "q5: State label is incorrect", and "You're not done yet".

4.2 Visualizations

We have implemented a series of visualizations to help students see how different models and their respective algorithms work. Figure 2 shows a typical example. Each visualization is composed of a series of slides. A slide typically has a brief text statement, supported by visual components. The user can choose to hear text-to-speech narration for the text.

Figure 2 shows a slide from a demonstration for how a TM accepts or rejects strings in the language $L = \{a^n b^n c^n \mid n > 0\}$. Students are able to see how the tape changes at each step, until the machine accepts or rejects the string. Using OpenFLAP's ability to execute an algorithm on a machine model, the visualizations are auto-generated from the machine specification and the input string. This makes it easy for a content author to generate examples, and for the student to see behavior on their own selected input. In a similar way, new proficiency exercises are easy to generate.

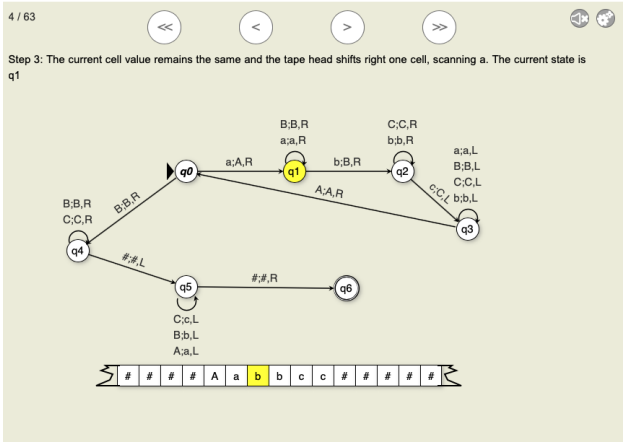

Figure 2: A Turing Machine accepting a string

4.3 Pumping Lemma Game

We created an ad hoc special activity for the purpose of explaining the Pumping Lemma, which is typically viewed as a central result in any FLA course, but is also difficult to understand. The central concept is that a machine with a fixed number of states can only correctly distinguish an infinite number of strings as being in a language if looping through a subset of the states an arbitrary number of times always generates a string in the language. The Pumping Lemma can be viewed as an adversary game [17]. The adversary game presents the Pumping Lemma in the form of a rule-based game with two players, the student and the machine. One player tries to prove that a language is not regular and the other player tries to stop this. The student can play either role. We provide thirteen languages that students can play, to prove that these languages are not regular. Figure 3 shows partial sample game steps to prove that a language is not regular. First the student selects one of the thirteen languages and role. The student selected the language $\{a^n b^n | n \geq 0\}$ and selects to be the first player in the game. Next is that he needs to select the value of m. The opponent then picks a string such that $|w| \geq m$. Then the student selects a decomposition of w to xyz that satisfies the constraints. If the student selected a bad decomposition for w, the game will provide a message that tells the student to retry another decomposition. Then the student retries and selects a better decomposition of w to xyz that satisfies the constraints. Next the opponent selects a value for i that makes $xy^i z \notin L$, so the language is not regular. In the case that the student selects to be the second player, then all steps will be the same but the roles will be swapped between the student and the opponent.

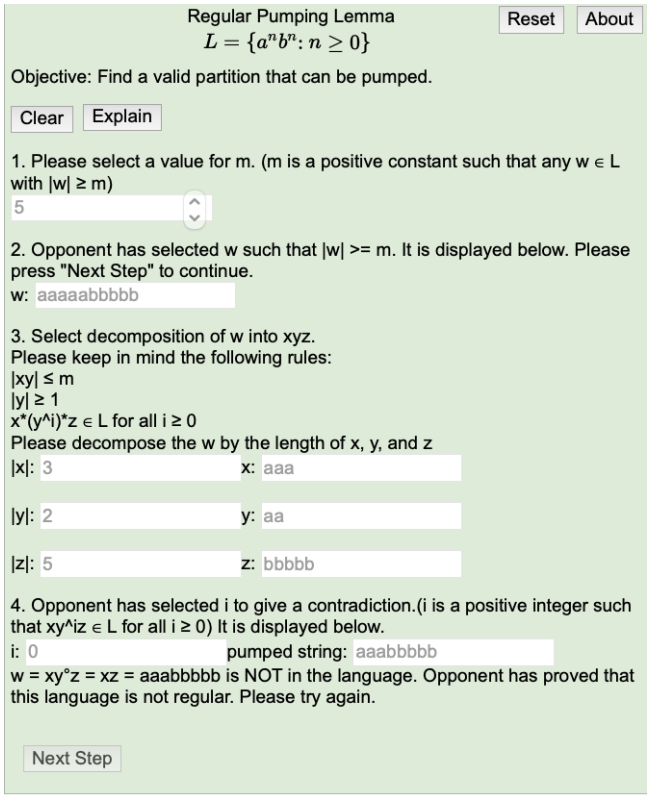

Figure 3: An adversary game to prove that a language is not regular.

5 EVALUATION PROTOCOL

In Spring 2018, one of the authors, Shaffer, taught an FLA course to 44 students at Virginia Tech, using a version of the eTextbook that had prose written jointly by the authors Shaffer and Rodger from

prior course notes. Some of the material on Turing Machines and computability theory included a few visualizations from prior work. We also supplemented with a popular FLA textbook by Linz [17], and we made use of the original, off-line, version of JFLAP. We used thirteen weekly paper-based homework sets, and three exams, all of which were fairly typical from prior instances of the course. Thus, this version of the course essentially used a treatment that is a typical presentation for the past decade. We consider this our control group for comparison.

Our intervention groups consist of 271 students at Virginia Tech taking the same FLA course over three different semesters, Spring 2019, Fall 2019, and Spring 2020. All intervention groups used the OpenDSA eTextbook for FLA as the main source of instruction and were taught by the same instructor, Mohammed. These groups did not use the Linz book as a supplement, and JFLAP was replaced by equivalent features available in OpenFLAP.

We sought to answer two research questions about the use of our eTextbook.

RQ1 What feedback do students give regarding their experience with the various exercises?

RQ2 How does performance on exams for the intervention group compare to performance of the control group students on the same set of questions?

In order to evaluate the FLA eTextbook in terms of student learning, both groups were given a set of questions as part of their three exams - two midterm exams and a final exam. Each exam tested students on different parts of the course. Some questions are related to the areas that are affected by the FLA eTextbook (that is, new visualizations and new interactive exercises), and other questions are on parts that are unchanged from what the control group saw. These items form the basis of our performance comparison between both groups. Exams were graded by the same person for both groups, to minimize inter-rater reliability problems.

We gathered students' opinions about the exercises through a survey at the end of the intervention semesters. We asked students to evaluate the exercises in terms of how useful they were in helping them understanding the FLA concepts presented in the course. We use non-parametric tests (Mann-Whitney) with 5% significance levels to analyze data comparing the control and intervention groups for exam grades. We chose this test to avoid the normality assumption required for a parametric test, and to mitigate the effect of unbalanced sample sizes [19], as there were collectively more students in the intervention groups.

6 EVALUATION RESULTS

6.1 Student Satisfaction

To gauge student satisfaction, we offered student surveys at the end of the interventions semesters. 89% of students responded to the survey. The survey solicited students satisfaction about various exercises in terms of whether they helped with understanding the Formal Languages course. Students were also asked to provide suggestions to add more exercises.

83% of the surveyed students reported that the exercises were useful to them in understanding the course concepts. When we asked the students if they wanted more exercises in the book, 58% indicated yes. Of these, 30% suggested adding exercises for the topics that are tested in the final exam, 22% suggested adding more exercises on Context-Free Grammar transformations, 14% suggested to add exercises related to the Pumping Lemma, and the remaining students suggested adding more exercises for PDAs, TMs, and Regular Languages.

6.2 Student Performance

We evaluated student performance by comparing the control versus intervention group scores on three exams.

6.2.1 Midterm Exams. Two in-class midterm exams were given. In both exams, we compared the control group versus the intervention group scores on topics where we created new visualizations and exercises. Table 2 shows the results using a Mann-Whitney test to compare performance at the topic level. The topics are (1) Context-Free Languages, (2) Pumping Lemma to prove that a language is not regular, (3) Regular expressions, (4) Turing Machines, (5) Pumping Lemma to prove that a language is not CFL, and (6) PDA. The last column shows the effect sizes of the differences between the two groups using Cohen's d for different sample sizes [12]. Table 2 shows that the new materials had an impact on students grades. All concepts where we added visualizations and auto-graded exercises had significantly higher grades for the intervention group than for the control group students.

To test the distinct effects of visualizations versus auto-graded exercises on student performance, we consider the change in performance of the different intervention groups on the topic of Turing Machines. The TM chapter was implemented in two stages. During the Spring 19 and Fall 19 semesters, the TM chapter included visualizations about TMs and how they work to manipulate different languages. In Spring 20, auto-graded exercises were added. These allowed students to test their understanding of TMs. Table 3 shows the results from comparing the control group versus Spring19 and Fall19 (intervention group 1), and Spring 20 (intervention group 2). The p-value is the result of applying the Mann-Whitney test comparing the control versus intervention group 1, and comparing the control group versus intervention group 2. These results show that students benefit more from seeing visualizations about TMs than reading traditional text about them. Students gain additional benefit when they have auto-graded exercises to practice. They score significantly higher (p-value < 0.0001) than students who used the visualizations only.

Table 2: Results and the effect size from comparing the grades in the Formal Languages part of the midterm exams from the control group (Cont., N = 44) to the grades in the same part from the intervention group (interv., N = 271).

Concept	Mean		STDV		p-value	effect size
	Cont.	Interv.	Cont.	Interv.		
1	24.8	27.3	7.03	5.33	0.0401	0.41
2	6.2	7.69	3.18	2.5	0.0053	0.52
3	3	3.4	1.05	1.15	0.0013	0.37
4	7.2	9.05	3.5	1.85	0.0002	0.69
5	9.16	12.28	3.56	5.04	<0.0001	0.73
6	6.43	7.52	3.31	2.81	0.0393	0.36

Table 3: The result of comparing the Turing Machine grades from the control group (N=44) to the grades in the questions from intervention group 1 (N = 142), and intervention group 2 (N = 129)

Groups	Mean	STDV	p-value
Control	7.2	3.5	
Intervention 1 (Visualization)	8.5	2.26	0.0552
Intervention 2 (Visualization & AE)	9.7	0.91	<0.0001

6.2.2 Final Exam. The final exam questions focus on the last part of the course. This part includes NP-Completeness, Countability and Unsolvable problems. For this part, both the control and intervention groups studied from the same OpenDSA materials that included some visualizations and no exercises. Students only read the text, saw the visualizations, and did hand-written exercises for homework. Thus, we can compare the groups on material that used the same treatment. Table 4 shows that there is no significance between the control and the intervention group in the topics that are not changed across iterations of the eTextbook. This supports the claim that changes in student grades on the midterm exams were due to the impact of the features being added to the eTextbook. (Note that we do not use Spring 2020 data for this comparison, since the Final Exam conditions were changed from prior semesters due to COVID-19.)

7 THREATS TO VALIDITY

One threat to validity for this study could be that we did not account for differences in students' pre-knowledge between the control and intervention groups. We did not adopt a traditional pre/post-test approach for both groups. However, the populations were primarily Junior and Senior Computer Science students in all versions of the course, having come to the course with the same prior courses.

Another issue of concern is whether other factors caused the differences between the groups in the performance on the exams. One factor could be that a different instructor taught the course in Spring 2018 (the control group) from the intervention groups (all intervention semesters were taught by the same instructor). However, the instructor for the intervention groups was the GTA for the control group, and both are authors of this paper and closely involved in the course. The contents of the course were the same, just the presentation method differed (adding visualizations and replacing paper homework exercises with interactive exercises for the intervention groups).

Table 4: Comparison of final exam grades on unmodified topics: control group (n = 44) vs. intervention group (Spring/Fall 2019, n = 142).

Topic	Mean		Median		p-value
	Cont.	Interv.	Cont.	Interv.	
NP-Comp	26.65	24.31	30	27	0.1871
Unsolvable	13.26	11.1	20	15	0.5636
Countability	13.35	12.73	15	15	0.2406

In Spring 2020, students moved to work online due to the pandemic. While the materials were covered pre-pandemic in the usual way, students took the second midterm online. This means there was a different exam environment between Spring 2020's second midterm and other semesters. However, we believe that since students covered the same content with the same instructor and took the same exam, then the exam environment was not a major factor in student grades. We deliberately excluded the final exam from Spring 2020 from our results.

8 CONCLUSIONS AND FUTURE WORK

Visualizations and interactive exercises were found to have a positive impact on student understanding of various Formal Languages topics. The difference was significant for all OpenDSA modules containing visualizations and exercises. The majority of the surveyed students (83%) mentioned positive feedback about the ability of visualizations and exercises to help them understand the topics. A significant proportion of students requested adding more exercises to the existing ones, or creating exercises for other topics that do not have any. The scores for the intervention group students were significantly higher than the scores for the control group students, with medium to large effect sizes. Equally important, this improvement is only found on those questions addressing topics that were presented using visualizations and exercises. There was no statistically significant difference in performance on the unchanged topics, as would be expected. This is good evidence that visualizations and exercises were a major contributor to this improvement.

The adversary game for the Pumping Lemma had a positive impact on students grades on Pumping Lemma questions. A few students (14%) suggest that we create auto-graded exercises for the Pumping Lemma. A majority of students suggested adding more auto-graded exercises to other sections of the book to help them understand different topics more clearly.

In future, we will focus on sections of the course that cover topics that do not present easy opportunities for a visual presentation, or interactive exercises. We will develop more PE for PDAs, and TMs.

We are investigating an approach based on the Programmed Instruction technique [31–33]. We believe that many aspects of a FLA course might benefit from a presentation based on Programmed Instruction since this can be made to be more interactive than the current plain prose approach for those topics.

The materials described in this paper are open source and freely available for use. For more information, including access to the course materials, visit opendsa.org.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grants DUE-1139861, DUE-1431667 and IIS-1258471. The Egyptian Ministry of Higher Education funded Mostafa Mohammed during his PhD. We are grateful to the many, many students who have worked on OpenDSA, OpenFLAP, and the FLA eTextbook over the years.

REFERENCES

[1] J. Barwise and J. Etchemendy. *Turing's World 3.0 for the Macintosh: An Introduction to Computability Theory/Book and Disk (Csli Lecture Notes).* Stanford University Press, 1993.

[2] P. Chakraborty, P. C. Saxena, and C. P. Katti. Fifty years of automata simulation: a review. *ACM Inroads*, 2(4):59–70, 2011.

[3] C. I. Chesñevar, M. L. Cobo, and W. Yurcik. Using theoretical computer simulators for formal languages and automata theory. *ACM SIGCSE Bulletin*, 35(2):33–37, 2003.

[4] S. H. Edwards and K. P. Murali. Codeworkout: Short programming exercises with built-in data collection. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '17, pages 188–193, New York, NY, USA, 2017. ACM.

[5] F. Erlacher et al. Pushdown automata simulator. *PhD, Institut für Informatik, Universität Innsbruck*, 2009.

[6] M. F. Farghally, K. H. Koh, H. Shahin, and C. A. Shaffer. Evaluating the effectiveness of algorithm analysis visualizations. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, pages 201–206, 2017.

[7] E. Fouh, D. Breakiron, S. Hamouda, M. Farghally, and C. Shaffer. Exploring students learning behavior with an interactive etextbook in computer science courses. *Computers in Human Behavior*, pages 478–485, December 2014.

[8] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. L. Naps, and C. A. Shaffer. Design and Architecture of an Interactive ETextbook–The OpenDSA System. *Science of Computer Programming*, 88:22–40, 2014.

[9] E. Gramond and S. H. Rodger. Using JFLAP to Interact With Theorems in Automata Theory. In *Proceedings of the Thirtieth ACM SIGCSE Technical Symposium on Computer Science Education*, pages 336–340, 1999.

[10] M. T. Grinder. Animating automata: A cross-platform program for teaching finite automata. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, page 63–67, New York, NY, USA, 2002. Association for Computing Machinery.

[11] M. Hamada. Turing machine and automata simulators. *Procedia Computer Science*, 18:1466–1474, 2013.

[12] L. V. Hedges and I. Olkin. *Statistical methods for meta-analysis.* Academic press, 2014.

[13] JFLAP website. http://jflap.org, 2020.

[14] A. f. C. M. A. Joint Task Force on Computing Curricula and I. C. Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* Association for Computing Machinery, New York, NY, USA, 2013.

[15] V. Karavirta and C. A. Shaffer. JSAV: the JavaScript Algorithm Visualization Library. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, pages 159–164, 2013.

[16] A. Korhonen, L. Malmi, P. Silvasti, J. Nikander, P. Tenhunen, P. Mård, H. Salonen, and V. Karavirta. TRAKLA2. http://www.cs.hut.fi/Research/TRAKLA2/, 2003.

[17] P. Linz. *An Introduction to Formal Languages and Automata, 6th edition.* Jones & Bartlett Learning, 2017.

[18] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, September 2004.

[19] H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.

[20] J. McDonald. Interactive pushdown automata animation. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 376–380, 2002.

[21] M. Mohammed, S. Rodger, and C. A. Shaffer. Using programmed instruction to help students engage with etextbook content. *The First Workshop on Intelligent Textbooks*, 2019.

[22] M. K. O. Mohammed. Teaching formal languages through visualizations, simulators, auto-graded exercises, and programmed instruction. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 1429, New York, NY, USA, 2020. Association for Computing Machinery.

[23] S. H. Rodger and T. W. Finley. *JFLAP: an interactive formal languages and automata package.* Jones & Bartlett Learning, 2006.

[24] S. H. Rodger and E. Gramond. JFLAP: An aid to studying theorems in automata theory. *Integrating Technology into Computer Science Education*, 30(3):302, 1998.

[25] S. H. Rodger, E. Wiebe, K. M. Lee, C. Morgan, K. Omar, and J. Su. Increasing engagement in automata theory with JFLAP. In *Proceedings of the Fortieth ACM SIGCSE Technical Symposium on Computer Science Education*, pages 403–407, 2009.

[26] J. Sakarovitch. *Elements of automata theory.* Cambridge University Press, 2009.

[27] V. Shekhar, A. Prabhu, K. Puranik, L. Antin, and V. Kumar. JFLAP extensions for instructors and students. In *2014 IEEE Sixth International Conference on Technology for Education*, pages 140–143. IEEE, 2014.

[28] V. S. Shekhar, A. Agarwalla, A. Agarwal, B. Nitish, and V. Kumar. Enhancing JFLAP with automata construction problems and automated feedback. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 19–23. IEEE, 2014.

[29] M. Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[30] M. Sipser. *Introduction to the Theory of Computation.* Cengage learning, 2012.

[31] B. Skinner. Programmed Instruction Revisited. *Phi Delta Kappan*, 68(2):103–10, 1986.

[32] B. F. Skinner. Teaching Machines. *Science*, 128(3330):969–977, 1958.

[33] B. F. Skinner. The Technology of Teaching New York: Appleton-Century-Crofts. *The Behavior of the Establishment*, 1968.

[34] C. A. Tecson and M. M. T. Rodrigo. Tutoring environment for automata and the users' achievement goal orientations. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 526–533. IEEE, 2018.

[35] M. Vijayalaskhmi and K. Karibasappa. Activity based teaching learning in formal languages and automata theory-an experience. In *2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)*, pages 1–5. IEEE, 2012.