Teaching Formal Languages through Visualizations, Machine Simulations, Auto-Graded Exercises, and Programmed Instruction

Mostafa Kamel Osman Mohammed

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science & Applications

Clifford A. Shaffer, Chair
Susan Rodger
Lenwood S. Heath
Dennis G. Kafura
Jeremy Ernst

June 17, 2021
Blacksburg, Virginia

Keywords: Formal Languages, Programmed Instruction, visualizations, pedagogical evaluation, automated assessment, Turing Machines, JFLAP, OpenFLAP, Engagement Taxonomy, Auto-graded exercises, Proficiency exercises

Teaching Formal Languages through Visualizations, Machine Simulations, Auto-Graded Exercises, and Programmed Instruction

Mostafa Kamel Osman Mohammed

(ABSTRACT)

The material taught in a Formal Languages course is mathematical in nature and requires students to practice proofs and algorithms to understand the content. Traditional Formal Languages textbooks are heavy on prose, and homework typically consists of solving many paper exercises. Some instructors make use of finite state machine simulators like the JFLAP package. JFLAP helps students by allowing them to build models and apply various algorithms on these models, which improves student interaction with the studied material. However, students still need to read a significant amount of text and practice problems by hand to achieve understanding. Inspired by the principles of the Programmed Instruction (PI) teaching method, we seek to develop a new Formal Languages eTextbook capable of conveying these concepts more intuitively. The PI approach has students read a little, ideally a sentence or a paragraph, and then answer a question or complete an exercise related to that information. Based on the question response, students can continue to other information frames or retry to solve the exercise. Our goal is to present all algorithms using algorithm visualizations and produce proficiency exercises to let students demonstrate understanding. To evaluate the pedagogical effectiveness of our new eTextbook, we conduct time and performance evaluations across two offerings of the course CS4114 Formal Languages and Automata. In time evaluation, the time spent by students looking at instructional content with text and visualizations versus with PI frames is compared to determine levels of student engagement. In performance evaluation, students grades are compared to assess learning gains with text and paper exercises only, with text, visualizations with exercises, and with PI frames.

Teaching Formal Languages through Visualizations, Machine Simulations, Auto-Graded Exercises, and Programmed Instruction

Mostafa Kamel Osman Mohammed

(GENERAL AUDIENCE ABSTRACT)

Theory textbooks in computer science are hard to read and understand. Traditionally, instructors use books that are heavy on mathematical prose and paper exercises. Sometimes, instructors use simulators to allow students to create, simulate, and test models. Previously, we found that students tend to skip reading the text presented in the books. This leads to less understanding of the topics taught in the course. To increase student engagement, we developed a new eTextbook for the Formal Languages course. We used pedagogy based on Programmed Instruction, presenting the content in the form of short bits of prose followed by the related question. If students can solve the question correctly, this means that they understood the content and are ready to move forward. To help both instructors and students, we developed a new Formal Languages simulator named OpenFLAP. OpenFLAP allows instructors to create many exercises, and OpenFLAP can grade these exercises automatically.

Dedication

This dissertation is dedicated to the loving memory of my father, Kamel Osman Mohammed. His support, encouragement, and constant love have sustained me throughout my life. I also dedicate this dissertation to my wife, Asmaa. I give my deepest expression of love and appreciation for the encouragement that she gave and the sacrifices she made during my graduate program. She was patient enough to take care of all of us in the USA during these five years.

Acknowledgments

First of all, all thanks due to ALLAH. May His peace and blessings be upon his prophet for granting me the chance to successfully complete my PhD. My heartfelt gratitude to my advisor and my father in work, Professor Clifford A. Shaffer for his inspiration, enthusiasm, invaluable guidance, and patience. He has taught me many things, and this work would not have been possible without his encouragement and support. I would like to especially thank Dr. Susan Rodger for her precious feedback, guidance and support during these years. I would also like to thank my other committee members Dr. Heath, Dr. Ernst and Dr. Kafura for their support, feedback and efforts reviewing my work and dissertation. It is my honor to have worked and learned from them. I would like to thank the OpenDSA research group Hossam Shahin, Jackson Wonderly, Ayaan Kazerouni, Hamza Manzoor, Rifat Mansur, and Arinjoy Basak who spent with me wonderful 5 years as brothers and friends. I wish to express my sincere gratitude to Mohammed Fargally for introducing me to Dr. Shaffer and helped me through out my work and helped my family once we arrived at the USA. I would like to extend my thanks to all the instructors who used OpenDSA in their classes, and to all the undergraduate students army who worked in developing different OpenDSA materials and libraries related to OpenFLAP and my book. Special thanks to my dad and mom who inspired me and supported me with his love and blessings throughout my life. They have been a constant source of inspiration, motivation, and strength. I will be ever grateful for their assistance, and am sorry that my dad did not live to see me graduate. I thank Asmaa, my love, for her unconditional support all through this journey. I thank Arwa, Roaa, and Belal, my children, for being so patient with a busy dad. I thank my mother-in-law and my father-in-law for their support, encouragement, and patience for living away from their

grandchildren and their lovely daughter. I also thank all my wonderful friends in Blacksburg, Mohamed Magdy, Abdelrahman Eldusuky, Mohamed Handosa, and Mohamed Mohamadin, who supported me during my hard times. Thanks to the people in the CSA department. Many thanks to Dr. Ribbens for his constant support even before coming to the USA. Whiout him and Dr. Shaffer I was not able to receive the GTA teaching excellence Award. Lastly, I thank Virginia Tech and the Blacksburg community for making my stay here a memorable one. Once again, thanks to ALLAH All mighty God for giving me the ability, mindset, and perseverance to be where I am now.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this study, our primary goal is to help Formal Languages course students engage, understand, and practice the course content.We seek to improve the Formal Languages course by providing more engagement through use of various interactive features including visualizations, machine simulators, auto-graded exercises, and use of the Programmed Instruction pedagogical approach.

1.1 Formal Languages

Formal Languages is a core course in Computer Science theory. It teaches students many important topics such as finite state machines, languages, their representation by regular expressions and grammars, Turing machines, and possibly computability theory and complexity theory [59, 60]. Formal Languages topics are applied in many daily activities in computing professionals' lives, like parsing and regular expressions. Formal Languages topics are also applied in many real applications like compiler architectures and pattern matching.

However, Formal Languages 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 might make it less appealing to students more used to the hands-on style of the typical CS programming course. A typical Formal Languages (FLA) class presents several models of computing (deterministic and non-deterministic finite state ma-

chines, regular expressions, push-down automata, context-free languages, Turing machines), with many proofs about their relationships and limitations. There are many algorithms associated with each model that students must learn to apply. Understanding the limitations of various models help students understand how computers work [56]. Often, students are asked to create machines or languages using various models. Many instructors have their students use simulators to support this process, such as the state-of-the-art simulator Java Formal Languages and Automata Package (JFLAP) [32, 53].

JFLAP simulates most of the models used in Formal Languages courses, so it helps students by allowing them to watch different models, apply different algorithms on these models, or test these models with different input strings. For example, a finite state machine simulator can help a student determine which strings are accepted and which strings are rejected by the machine. This way, JFLAP increases student engagement and interaction with the course content. JFLAP plays a significant role in helping students understand different Formal Languages models [25, 54].

1.2 OpenDSA

The OpenDSA project [18] at Virginia Tech is concerned with building complete eTextbooks for different topics in computer science like Data Structures and Algorithms, Computational Thinking, or Formal Languages. These eTextbooks are enhanced with various embedded artifacts such as visualizations, exercises with automated assessment, and slideshows to improve understanding. OpenDSA allows instructors to create instances of complete interactive eTextbooks that integrate interactive artifacts with the textual content. OpenDSA contains slideshows produced using the JSAV (JavaScript Algorithm Visualization) framework [33] to support various topics in undergraduate courses.

1.3 Programmed Instruction

In 1953, B.F Skinner went to his daughter's school to ask about her progress [40]. When he entered the class, he noticed that there are flaws in the traditional teaching methods. For example, a) instructors give less attention to the students in class, b) textbooks provide no immediate evaluation of students' solutions, and c) not all students have the same prior knowledge. Thus some students are not prepared to acquire the knowledge from the textbook. These observations led Skinner to think about developing a new teaching technique. Skinner then invented the Programmed Instruction (PI) machine [61]. The Programmed Instruction machine can teach students the required materials by presenting a small piece of information (frame) to the student. After that, the student is asked to answer a question about the given frame. If the student successfully finishes a frame by solving the related question, the PI system will allow him/her to access the next frame. On the other hand, failing to solve the question will prevent the student from moving forward to the next frame. This prevention means that the student has to reread the frame and try to solve the question again. The PI system acts as a personal tutor to the student. Another benefit of this system is that the PI machine gives students an immediate evaluation for each response. Therefore, students will have the motivation to solve each frame question correctly to move forward to further frames. In his papers[61], Skinner claimed that students would be able to learn twice as much with the same time and effort compared to the traditional teaching method. It is noteworthy to mention that Skinner's PI machine is not as important to us as the PI approach. We are interested in the PI system, where students read a sentence and answer a question related to it. Skinner invented the machine that applies PI principles.

Programmed instruction (PI) and teaching machines were once active areas of research and development. Much of this research was conducted during the 1950s and died out in the early 1970s. During this period, significant areas of Programmed Instruction research revolved

around addressing instruction effectiveness, learner pacing, reinforcement strategies, and long-term effects [49]. Although there was considerable interest in Programmed Instruction, mostly due to monumental works such as Skinner's *Teaching Machines* and the *Technology of Teaching* [63], Programmed Instruction was primarily superseded by Computer-Assisted Instruction (CAI), and Keller Plan in the 1970s [34, 40].

There are many definitions of the term Programmed Instruction. Programmed Instruction is a graded sequence of controlled steps used to present new subjects to students. Another definition of Programmed Instruction is that it is a teaching method that consists of a network of statements and tests. This network is used to teach students new knowledge based on their pattern of errors.

Programmed instruction is an instructional methodology centered on Skinner's principle of stimulus control and reinforcement to shape behavior [62]. Instruction follows a linearly logical sequence and decomposes content into well-defined small curriculum units. Programmed Instruction machines support learning through a systematic reinforcing approach in which students can advance incrementally, receive immediate feedback on their responses, and are rewarded in a self-paced manner.

A more advanced form of programmed instruction was originated by N. Crowder [10]. He designed a CAI program that trained US Air Force members to troubleshoot problems in electronic equipment. His system implemented a branching approach in which the program will select the next stage of instruction according to the learner's previous responses. This model differs from Skinner's view of focusing only on reinforcing correct behaviors. Crowder argued that a student should not only learn from correct answers but also mistakes. He believed that doing this would promote learning through cognitive reasoning. Therefore, in his branching approach, errors are anticipated by the system, and students' erroneous responses place them on a remedial learning path [40].

Computer-Assisted Instruction (CAI) uses computers in classrooms, and education in general, to improve instruction. With the widespread and availability of Computers, many researchers started to use computers in the education process to teach students different materials [68] by developing computer programs that use text, graphics, sound, and videos to enhance targeted materials. These programs allow students to study the materials on their own, either online or offline. CAI has variations like Computer-Aided Instruction (CAI), Computer Assisted Learning (CAL), Computer Based Education (CBE), Computer-Based Instruction (CBI), Computer Enriched Instruction (CEI), or Computer Managed Instruction (CMI). CAI includes six different approaches, which are:

- Drill-and-practice. This approach requires students to study given information (Drill), then the student should answer questions on that given information (Practice).
- Tutorials. This approach is used to teach students specific materials in an interactive way. Tutorials can use other approaches to present the information to students, like drill-and-practice, games, or simulation.
- Games. The software creates a contest to achieve a specific goal or score by beating others or the computer.
- Simulation. Software that can provide models of reality that do not require the expense of real-life or its risks.
- Discovery. This approach provides students with course-related challenges, and students should study, analyze, and infer from the available course information to tackle these challenges.
- Problem-Solving. This approach helps students to develop specific skills and strategies that are required to solve specific problems.

According to [40], CAI is considered the sophisticated extension to PI theory and concepts. However, many researchers conducted their research on CAI outside the PI context. Because computers can produce many instructional possibilities, CAI is considered to be a separate field from PI.

Another important model of instruction that was popular and widely used during the 1970s and 1980s is called Personalized System of Instruction (PSI or Keller Plan) [34]. The main idea behind the Keller Plan is that students can work on the given materials at their own pace, which means that students can study the modules at any time and as much as they want without the need to wait for the instructor to explain the topic, especially when instructors tend to focus on students that struggle with a module. Therefore, some students may finish the course before the end of the semester, and others may fail to finish the course at all.

In the Keller Plan, the instructor's role is minimized. Instructors should decide what content students have to master, guide students through their studying, and give tests and exams to students. Lecture in the Keller Plan is just a place for students to study their materials and take tests. Students who feel that they have mastered the given module are free to ask the proctor (a Teaching Assistant) for a test. If the student passes the test, he/she can start to study the next module. Otherwise, the student has to restudy the module and take another test.

1.4 Our Goal

The primary focus of this dissertation is the design, implementation, and evaluation of a new eTextbook to teach students the Formal Languages, using the Programmed Instruction technique, augmented with visualizations and interactive exercises. The eTextbook gives information to students in the form of frames. Each frame shows some information and then

asks the student a question related to the given information. Students may move forward to the next frame when they successfully solve the question. Similar to existing OpenDSA modules, our new eTextbook is implemented using modified JSAV slideshows. Students see a series of slides (the frames). Frames add a constraint on the forward button in the slide shows. Thus, students are not able to click on the next button unless they satisfy a pre-determined criterion for each frame, such as selecting the correct answer or applying a specific algorithm step on a given model. With this eTextbook, students gain a better intuition about Formal Languages content, which are hard to grasp when relying merely on textual discussion and static figures.

Our eTextbook helps students understand the Formal Languages content by reading the information and solving the related question given in each frame, watching simulations and visualizations for Formal Languages models and algorithms associated with them, and solving auto-graded exercises. Therefore, our book increased student's interaction, engagement, and practice with course content. Also, the book keeps track of students' click streams, grades, attempts, and time spent on each frame. These data are used to prove the impact of using our book in a course.

Evaluating the pedagogical effectiveness of the new eTextbook is a challenge. Based on [16], it is a challenging task to measure student learning gains, especially if this gain results from technological interventions. We built and evaluated our system over three phases: traditional text prose, prose with visualization and auto-graded exercises, and an eTextbook that has frames with information alongside the auto-graded exercises and visualizations. In each phase, we compared its effectiveness with the previous phases.

1.5 Research objectives

This study seeks to satisfy four main research aims:

- Design a new framework for Programmed Instruction Frames
- Build a Web-based version of JFLAP that can be integrated with an eTextbook.
- Design a new Formal Languages eTextbook that uses PI frames, visualizations, simulators, and auto-graded exercises.
- Evaluate the pedagogical effectiveness of the resulting eTextbook

1.6 Research Questions

The primary research questions for this study are:

1. What feedback do students give regarding their experience with the Programmed Instruction book and OpenFLAP? (based on a survey at the end of the semesters)
2. How does performance on exams for the intervention groups compare to the performance of the control group students on the same set of questions?
3. How does performance on exams for the PI group compare to the performance of the Visualization students on the same set of questions?

1.7 Major Contributions

The key contributions of this study are:

1. A set of interactive visualizations capable of conveying Formal Languages concepts visually that can be embedded in the curriculum as part of an online eTextbook.
2. A set of Auto-graded exercises and proficiency exercises for Formal Languages course
3. An new variation of JFLAP that is implemented using web-based technology. This tool allows instructors to create Auto-Graded exercises and proficiency exercises. OpenFLAP can be integrated with an eTextbook to report grades to a Learning Management System.
4. A new eTextbook for Formal Languages that follows the Programmed Instruction teaching method augmented with visualizations, auto-graded exercises, and proficiency exercises.
5. Add Programmed Instruction support to the OpenDSA project. This will allow instructors to create new Programmed Instruction ebooks for different CS topics.
6. A quasi-experiment and results analysis to gauge the effectiveness of different Formal Languages ebooks regarding student satisfaction and performance.

1.8 Dissertation Outline

In Chapter 2, we present some previous work-related to Programmed Instruction. We also present previous work related to enhancing the Formal Languages courses. Chapter 3 presents our motivation to this study. The OpenFLAP tool and how to create various exercises are presented in Chapter 4. Chapter 5 presents Programmed Instruction support in OpenDSA. The implementation process for our eTextbook is described in Chapter 6. Chapter 7 presents the evaluation of different Formal Languages eTextbook phases. Finally, Chapter 8.1 presents the conclusions and future research directions.

Chapter 2

Literature Review

2.1 Programmed Instruction

After Skinner and Crowder provided their Programmed Instruction approaches, many researchers started to focus their studies on PI by trying to address different approaches to PI, or different elements of the PI model. Other research focused on developing different PI materials or on comparing its effectiveness against conventional teaching methods.

2.1.1 The Programmed Instruction Approach

According to [40], there is no single approach to building Programmed Instruction materials. However, by studying the available PI systems, we can find that there are some commonalities among these systems. From these commonalities, researchers have extracted a process that can be followed to build a PI system.

1. **Specification of Content and Objectives**. The first step is to determine the content that will be taught using the PI system. The material determination consists of defining the terminal behavior and course objectives, in other words, the intended outcomes of the system, and planning the assessment items and evaluation strategies.

2. **Learner Analysis**. Collecting data about students who will learn from the system.

Examples of data to be collected are demographics, pre-existing knowledge of the given topic, or learner abilities.

3. **Behavior Analysis**. The system designer must select the intended materials concepts in a way that makes students enter a sequence of responses where each response creates the stimulus for the next response. The selection of concepts is based on the needs, abilities, strengths, and weaknesses of students.

4. **Selection of a programming paradigm**. There are two common types of sequencing in PI, Linear Sequencing (Extrinsic) and Branching Sequencing (Intrinsic), see Figure 2.1. The choice between these two paradigms is made based on earlier steps in the PI process. If there is a high variance in abilities of students, then branching will be suitable because it enables students with high abilities to skip frames. Skipping frames means that the PI system will skip some frames for the student based on his earlier responses to the given questions. According to [41], there are no significant differences found in the effectiveness of learning between intrinsic and extrinsic CAI materials.

5. **Sequence of Content**. After determining the programming paradigm, PI system designers should determine the sequence of frames. There is no standard sequencing for the frames, but there is research to identify some possible sequences.

    - A general PI sequence consists of an introduction, a diagnostic section, theory section, teaching, testing section, practice section, and finally review or summary to reinforce all of the concepts addressed in the program.
    - Pragmatic approach. The frame is based on the logical sequence of the material. Thus, this step ensures that the PI materials address all necessary information and components.
    - RULES and EXAMPLES (RULEG) system [14]. This approach is useful when

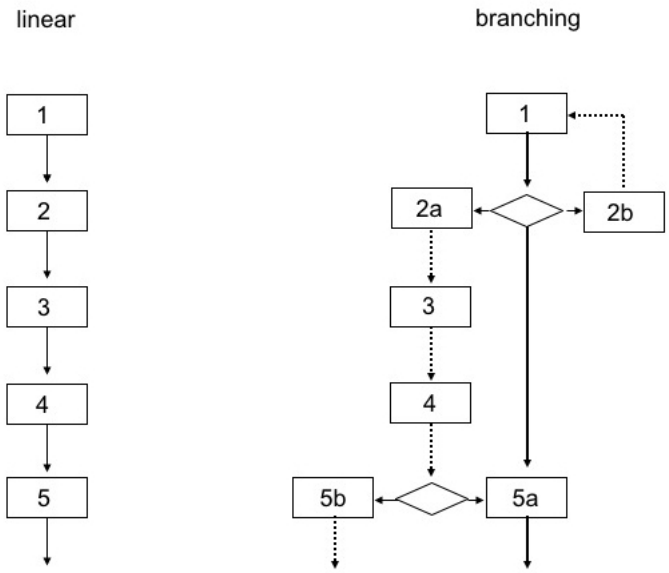


Figure 2.1: The difference between Extrinsic and Intrinsic branching. In Extrinsic branching, shown in the left, students cover the material sequentially. In Intrinsic branching, the PI machine may skip some frames based on learner performance on previous frames.

the material consists of rules and examples. Thus, the sequence will present a rule followed by an example to practice the rule.

- EXAMPLES and RULES (EGRUL) approach [45]. This is the reverse of the RULEG approach. In EGRUL, the frames will be a variety of examples, and these examples will guide the student to comprehend the rule.
- Conversational chaining [1]. In this approach, there is no programmed feedback. Therefore, after the learner responds to a frame, he/she checks the solution in the following frame.
- Mathetics (Backward Chaining) [21]. In this approach, the learner begins with a frame that has a piece of information. The following frames will let the student work backward through the process that led to that piece of information.

6. **Frame Composition**. Programmed instruction frames should contain some essential

components, which are:

- **Stimulus context** to which the occurrence of the desired response is to be learned. The information that the student should understand.
- **Incentive** that serves to obtain the targeted response. In other words, the frame should contain a question that leads the student to grasp the given knowledge.
- **Response** that leads the student to gain the intended knowledge. Which means that the student should understand the given information to respond to the given question correctly.
- **Other materials** that make the frame readable and understandable, like visualizations.

Another important frame component is the method that is used to present the information to the student (the prompt). The prompt plus some of the previously acquired learning will make the students able to answer the questions in the frames correctly. Prompts can be formal or thematic [65].

(a) Formal prompts provide the targeted response to students. Thus it is used to introduce new concepts, like defining a model and its components.

(b) Thematic prompts use pictures, grammatical rules, or any other supplementary data to guide students toward the production and application of the frame's targeted response. Like showing a model to the student to apply an algorithm step on it.

Another critical design consideration for the frames is the response (or question) type. Responses are necessary and help students to acquire the intended learning outcomes. These could be any question type that can be supported by the implementing system.

Some standard types include multiple-choice, true or false, fill-in-the-blank, and labeling. More advanced systems can provide students an interface where they demonstrate proficiency at some content-related skill or knowledge.

7. **Evaluation and Revision**. The final step in this process is to evaluate the programmed product and revise it. The primary source of evaluation is learner feedback about the product. Thus to determine the effectiveness of programmed instruction products, there are some factors to consider, like errors in content, accuracy, appropriateness, relevance, and writing style. The learner can not discover these factors. Thus, external reviewers (field experts) should review the initial product and a way for students who use the system to provide feedback to the content creators.

2.1.2 Applications

According to [49], researchers focused their studies on producing PI materials, especially from 1960 to 1967. After 1967, the number of produced materials declined gradually. Figure 2.2 shows products produced over time. After 1971, there is some research conducted in producing PI materials. However, the produced materials are small in numbers. Another factor in the decline in PI materials is using computers in education, which attracted many researchers to produce different education materials, including PI materials, in the form of computer programs (CAI) [40, 68]. However, we can find some research studies in the recent past that use PI materials in different topics, like genetics and gene interaction [22] in 2016. In this section, we present some programmed materials that researchers made using the PI teaching method.

Programmed Instruction inspired some writers to write their books in the form of frames of questions. Two examples of PI books are **The Little LISPER** [19] in 1987, and **The Little**

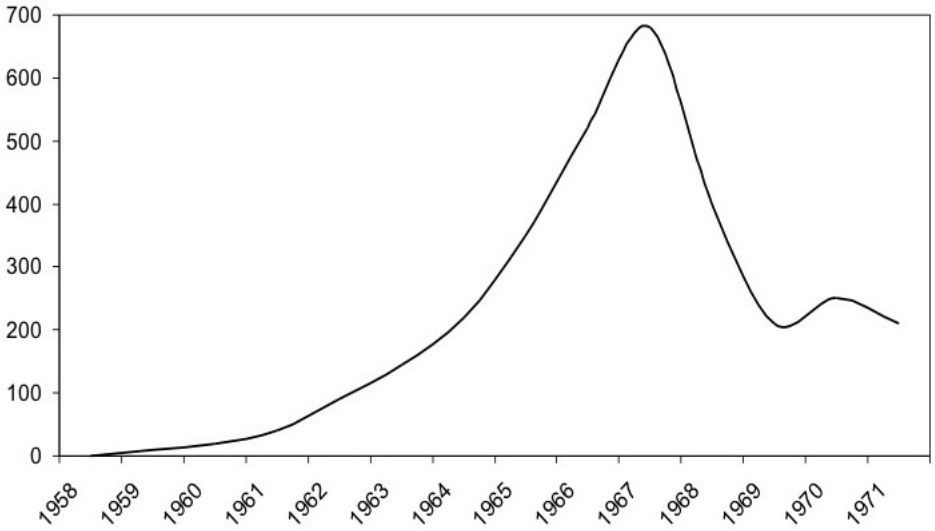


Figure 2.2: PI-based products 1958 - 1971 [49]

Schemer [20] in 1996. A page in these books consists of two columns, see Figure 2.3, one for the questions and the other for the answers. So the student will not read lengthy text to understand the topics. Instead, the student will answer a series of questions. These answers will give students a better understanding of the concepts by forming their definitions.

In the late 1960s and early 1970s, significant developments in educational technology began with the introduction of computer-assisted instruction (CAI) research. PLATO (Programmed Logic for Automated Teaching Operations) [5] was launched with the National Science Foundation (NSF) funding in 1961. The central area of research interest was to compare the effectiveness of CAI instruction to traditional forms of instruction. PLATO is still one of the most successful PI systems, and it was used widely in American schools.

In year 1989 [23], a self-managed module based on PI was presented to teach infection control of viruses like hepatitis B and human immunodeficiency virus to in-service nurses. The post-test results of nurses who studied this module were significantly higher compared to their pre-test results. This shows that the PI materials had a significant impact on nurses' performance in infection control of hepatitis B without the need for an instructor.

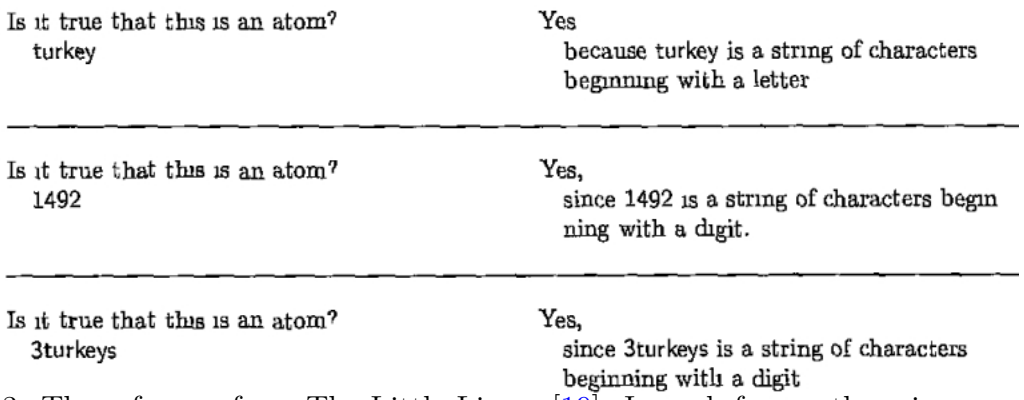


Figure 2.3: Three frames from The Little Lisper [19]. In each frame, there is a question on the left-hand side of the frame, and the answer with an explanation on the right-hand side of the frame.

During 2006 [38] a study was made to compare the success of PI with traditional teaching using two groups of stereo-chemistry teacher trainees. Both groups learned the same material in stereo-chemistry. The control group was taught the material using traditional instruction (through lectures), and the treatment group learned the same materials using a PI textbook. Both groups took a pre-test and post-test, and they found a significant difference between the groups in the post-test results. The study used a small set of frames. They used only 56 frames to teach the material. Each frame contained a long paragraph of text followed by a question. The major flaw of this study is that frames with long paragraphs are not following Skinner's definition of frames, where he suggested that the frame contain a small fundamental piece of knowledge followed by a question.

Recently, a study was conducted to teach students Genetics for graduate students [22] in 2016. Due to its complexity, students face many challenges to memorize and understand this topic using conventional instruction. The authors developed PI materials for the students, and they found an increase for student scores between the pre-test and post-test results. They also reported that students were able to memorize complex genetics terms, because of using frames that force students to answer the related question before going to the next

frame.

2.1.3 Comparisons between PI and Conventional Instruction

According to [40] there was much research that compared PI and conventional instruction (CI). The results of these comparisons varied from one study to another. Some research results found that PI has a better impact than the CI method. In 1959 [11], the authors presented many examples of the success of PI groups when compared with CI groups. Examples from the armed services show that PI students learned the subjects faster and more thoroughly. A second example is from the Department of Psychology at Georgia University where most PI students scored 100%. The third example is from an experimental course in the German language offered at Harvard. They found that PI students were able to learn the language faster than CI students. The last example we will address was from Ohio University. PI and CI groups used the same materials to study general chemistry. The PI group scored higher than the CI group by 20%. Also in 1962 [31], a comparison was made between PI and CI to teach two groups a psychology course. When students took an unannounced quiz, the PI group scored higher than the CI group. With announced quizzes, there was no significant difference between the groups. The PI group is always ready to take any exam at any time without explicitly preparing. On the other hand, the CI group is not on the same understanding level. That is why the PI group scored higher in the unannounced quiz. We can conclude that the PI group understands the content given by the machine, and they need less time to perceive the knowledge when compared to the CI group.

Some researchers targeted their research toward testing the effectiveness of merging PI and CI methods on learners. In 1963 [64], a study about the effect of PI in teaching disabled children reading and vocabulary found that the PI version was better than CI. The study

suggested that mixing PI with CI will be better than using PI alone. Also, in 1964 [8], the authors performed 12 studies to compare the different possible combinations of teaching methods. The result of 11 out of 12 studies that compare PI, CI, and PI with an instructor, suggests that PI with an instructor is the most effective choice.

Some studies found no significant difference between using PI or CI. For example, in 1962 [12, 24] the authors concluded their research by stating that there is no significant difference between PI and CI. In 1972 [30] reviewed 112 studies that compared different materials implemented using PI with the same materials taught to learners in CI. The authors of this review found that the PI materials were as good as the CI materials. However, in [4, 44, 51, 55, 68], the authors tested their materials with PI, and CI. They reported that CI is superior when compared with PI.

From the studies mentioned above, we can see that there is no consensus about which instruction method is better than the other. However, the majority of PI related papers agreed that PI requires learners to spend smaller time periods to make them understand equivalent content.

2.1.4 Keller Plan

The Keller Plan was adopted by many psychology instructors and by individuals outside of psychology. In [26], the authors converted their Physics course at MIT to follow Keller Plan principles. The authors presented how good the result was for instructors and students. They also claimed that most of the students got A's and B's on their hard physics course.

The Keller Plan and Programmed Instruction share some common characteristics.

- **Mastery-Based Learning**. With both instruction methods, students must master

the initial content in order to get access to further course materials. Students who successfully satisfy the mastery criterion will be able to move forward. Others who fail it have to restudy again. A mastery model is most distinguished from conventional instruction by the fact that students are allowed to repeat until they reach the necessary threshold. As a consequence, the parts that they complete give full credit, unlike a typical test situation where the class median might be 75%.

- **The instructor's role is minimized**. In Keller's plan, the instructor's role is to prepare the material for the course by picking the appropriate visualizations/exercises from a pool of existing material. Instructors also determine the sequence that students should follow to study the course content.

- **Students work at their own pace**. Students are provided with the material as instructors planned. After that, students can study them based on their own pace. Students do not need to wait for a lecture to study. They can go and study the material when they can and as they want.

There is a major difference between the two methods. In PI, students should master little information (found in a frame) before moving to the next frame. On the other hand, the Keller Plan requires students to master a module (maybe a chapter) before moving to the next module.

OpenDSA has many features found in the Keller Plan. OpenDSA has modules, and through out each module, there are exercises that students should solve to prove their mastery of the module. However, OpenDSA does not prevent students from moving further if they failed to solve an exercise. But it does follow the mastery approach in that students can repeat exercises as many times as they wish, eventually receiving credit whenever they demonstrate mastery. Our work is a combination between CI and PI (we give lectures and exams in

addition to the PI frames). Adding OpenDSA allows us to use a combination of Programmed Instruction, conventional instruction and Keller Plan.

2.2 Formal Languages

Formal Languages topics are abstract and challenging and at the same time these topics are important and students need to understand them. This led many researchers to look for different ways to enhance the Formal Languages course. Some researchers focused their research on enhancing tools that can be used to simulate the algorithms and models [47], and other researchers focused on attempts to enhance the teaching methods for Formal Languages courses.

2.2.1 Enhancing Formal Languages simulators

The Java Formal Languages and Automata Package (JFLAP) [53] simulates most of the models that are used in Formal Languages courses. It allows students to view models, apply algorithms on these models, or test these models with different input strings for acceptance [25]. 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 [25, 54]. JFLAP does not identify for students whether their solutions are correct or not. Attempts have been made to provide such feedback [57, 58], but without assessing their impact on students' understanding.

Of course, JFLAP is not the only simulator for Formal Languages courses as [7] summarizes 50 years of automata simulation. While JFLAP is broad, covering most Formal Languages courses topics, some authors have created special simulators for specific models [9]. Examples

include the FSA simulator [27], the Interactive Pushdown Automata Animation (IPAA) [43], a PDA simulator [13], and Turing Machine simulators [2, 28]. Most of this functionality can be found in JFLAP.

Algorithm analysis content in OpenDSA was originally presented using text and static images, like any traditional textbook. By examining log files, it was found [15] that students often completely skip reading this material, 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.

2.2.2 Enhancing the teaching methods for Formal Languages courses

Vijayalaskhmi and Karibasappa [67] studied the use of activity-based learning in the Formal Languages course. They had student groups discuss problems and then present their solution to other students. Groups were evaluated by how well they expressed the solution. The paper [66] 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.

Some researchers used Game Building to enhance student's understanding of Formal Languages content. In [37], the authors used a game to teach students Finite Automata (FA) and Turing Machines (TM). Students learn by completing a game-building assignment in which

there is a direct and transparent mapping between the game that the student is building and the model in the framework they are trying to master. In this approach, students learn by constructing a game about FA or TM, which should improve the student's understanding of them. For example, a student is asked to build a game where an FA keeps track of information about the game state, allowing the game to respond to players' decisions. Certain requirements were put in place to ensure that the created FA was sufficiently complex to assess the students' understanding of the subject material. The drawback of this approach is that different students have different motivations, so students whose main motivation is to do as little work as possible to obtain their desired grade will not increase their performance using this model.

Other researchers focused on the relation between the instructor and the student. In [35], the authors applied Cognitive Apprenticeship Approach to reducing student failure rates in an introductory theory course at the University of Potsdam. The practice of cognitive apprenticeship focuses on various opportunities for the student and the instructor to collaborate: modeling and articulation, where the teacher provides an exposition on the course material; coaching, where the teacher assists the students in putting the material to use; and scaffolding, where the teacher creates methods and structures to aid the students in applying the material. The authors believe that coaching was already accomplished by feedback on homework and exams. To facilitate modeling and articulation, the authors added weekly class sections to ask questions and observe the instructors working through the problem-solving process. To help with scaffolding, the authors created exercises for the students to work on together to prepare them for similar questions on the homework assignments.

Chapter 3

Motivation

Students usually do not do the level of work required to effectively learn hard topics. Student tend to skip the material and jump immediately to solve the exercises to collect credits. In general, instructors build their courses in a way that they expect some efforts from students to read and practice the course material. However, some research showed that students tend to skip the material and not do their work as expected. In [29], the authors collected surveys from students and instructors about the time and effort that is required/spent by students to study a hard CS topic like recursion. The authors found that instructors expected that students should spend about seven hours to study and practice recursion while students reported that they only spend half of that time. The result is students do not have a good practice and understanding for this crucial topic. In another OpenDSA study [15], it was found that students skipped the mathematical material on algorithm analysis in a data structures course. Students do not spend the required time studying the book materials with text and images, and some skip them entirely. In addition to these problems, Formal Languages courses typically require students to solve several problems by hand. Example problems include building a machine that recognizes a language, writing a grammar for a language, or appliing an algorithm on a specific model. Instructors struggle to create and grade paper-based problems because it is time consuming and even so, students do not get enough practice. Worse is that there is no feedback to the students until much later about whether they got the right answer. Students spend time to solve the exercises, and they

do not know if they are working correctly on these exercises. For example, when a student solves an NFA to DFA conversion, they may start with an incorrect step and keep going on the exercise, but the result will be a completely incorrect solution. Thus, the student did not get many benefits from spending the time solving the exercise.

These problems inspired us to find a better way. In particular, we have written an eTextbook that applies the Programmed Instruction approaches along with auto-graded exercises to achieve the following benefits:

- Address inadequate engagement and interaction by presenting most content in the form of Programmed Instruction frames-sets. Students are forced by the questions to read and reflect on the material.
- Evaluates on-the-spot if the student understands or not the concepts taught. In our book, if students understood the sentences, then they may solve the corresponding PI frame question correctly. If not, students will have to reread the sentences in the current/previous frames.
- Provides feedback on student responses such that they will correct themselves if they misunderstand the concepts. Instructors can assign messages to show to students after they solve the question correctly or incorrectly.
- Provides many auto-graded exercises that require students to practice content by doing activities like building different models for languages and writing grammars. These exercises provide immediate feedback on whether the student's solution is correct.
- Provides several proficiency exercises to apply algorithms on models.

A key goal of our study is to find methods that can increase student engagement. Based on the Engagement Taxonomy [50], there are six levels of engagement for students.

1. No viewing means that students do not interact with the book at all. This can be found in traditional books where the book is made of prose and paper assignments. According to [15], students tend to skip certain types of textbook material without reading it and leave the chapter without understanding it. In our experiment, the control group book is a traditional book with text and some static figures only.

2. Viewing, this is the lowest possible form of interaction. Here, students watch visualizations passively, like slideshows, without additional interaction with the visualizations. In our second control group, where we build the phase 2 book that contains slideshows, students can see visualizations about different models and algorithms on these models. Still, students can skip reading the text or the visualization.

3. Responding, means that students have an interaction with the material. In our study, this is the goal behind using the Programmed Instruction technique. Students are required to answer questions to be able to move on in the book. This response increases the interaction with the book. Based on the taxonomy, it increases student understanding of the content.

4. Changing, means that students can modify the visualization, like select the algorithm input or step. In our book, students are asked to solve proficiency exercises. To solve these exercises, students select the next step they want to perform. Thus, they have the ability to test the same problem (algorithm) with different inputs and different orders.

5. Constructing, means that students can construct their own visualization. In our book, students are asked to build models as auto-graded exercises. During these exercises, students freely build the model in many different ways. They can see a visualization about how their model processes a string for acceptance/rejection for every model.

6. Presenting, means to present their solutions to other students. This is the only level of the Engagement taxonomy that is not implemented in our books.

With this taxonomy in mind, we find that our approach provides multiple levels of engagement with the material. We conducted assessments to determine if this approach yields a better understanding of the content.

Chapter 4

OpenFLAP

The Java Formal Languages and Automata Package (JFLAP). JFLAP is used extensively in Formal Languages courses to help students visualize and observe the behavior of models and associated algorithms [54]. However, JFLAP has three disadvantages from the point of view of integrating the material into an eTextbook. First, it was written in Java and therefore 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 [46, 48]. 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, introductory courses have reached the state where many programming assignments can be graded with immediate feedback from auto-graders, largely based on testing the program against unit tests. We apply lessons learned from programming exercise auto-graders to create machines to answer Formal Languages exercises. Third, JFLAP is a separate tool that is not associated with any automata textbook. There is a JFLAP book [52], but that book shows how to use JFLAP features. It doesn't explain FLA concepts, so students must use another textbook to learn them that might not present the information in the same way as JFLAP does, and certainly is not closely integrated with JFLAP. We 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 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.

4.1 Building Process

One of the main challenges in this study is to re-implement JFLAP functionality using web technologies, HTML5 and JavaScript. Thus, it will be accessible on the Internet and without the need to install any other packages. Our eTextbook for Formal Languages can also use the same functionality found in JFLAP. The main drawbacks of the current JFLAP tool are that it is built using Java, and there is no support to give students questions and auto-assess them. Students can use JFLAP to solve any exercise, but it is their responsibility to determine if the answer is correct or not. For example, suppose that a student wants to build a finite state machine that accepts a string with an even number of a's. The student can use JFLAP to build the model and try different strings one by one to determine if his/her model is correct or not But they will not know for sure if they are correct until the exercise is hand graded at some point. We refer to our web-based version of JFLAP as OpenFLAP. OpenFLAP is implemented using the JSAV library. The implementation of OpenFLAP provides us with many visualizations to help students understand different aspects of Formal Languages. It also allows us to form proficiency exercises and auto-assess them through the OpenDSA system. However, the implementation of OpenFLAP was challenging and time-consuming. JFLAP has many models and functions that required a significant amount of

time to implement.

JFLAP supports a large number of model types and a wide variety of algorithms to manipulate them. To help manage this effort, we first classified JFLAP functionality into three groups. The first contains the minimum functionality required to create visualizations of the core material covered in the course. These include fundamental models like deterministic finite automata (DFA), non-deterministic finite automata (NFA), pushdown automata (PDA), and Turing machines (TM), along with their core algorithms like converting an NFA to a minimized DFA. The second group contains the functions and models that are not essential to build our visualizations, or they do not represent the core for our targeted visualizations and exercises. This group has model variations 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.

In the priority implementation, we included the basic engines for automatic generation of visualizations of a machine executing 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).

Over the course of this work, approximately 20 undergraduate students helped with OpenFLAP development as part of capstone and independent study projects.

After finishing all the first group functions, we implemented some functions from the second and third groups like Empty Stack Pushdown Automata and CYK parsing. The reason

for implementing the Empty stack is to show the students the difference between the two different acceptance models for Pushdown Automata. CYK parsing plays an important role in testing the correctness of context-free grammars. As we present in the auto-graded exercises, to test a grammar's correctness, we need to test a student's solution against several test cases. This is an application for the membership problem. The membership problem is defined as deciding if an input string is a member of a language represented by a given grammar. Grammars that have unit productions ($A \rightarrow B$) or lambda productions ($A \rightarrow \lambda$) may take forever in deciding if the string is in the language or not due to an infinite loop. In the beginning, we were using the exhaustive search algorithm to decide every test-case string. Since students can write any grammar to test in our exercises, some of them get stuck in an infinite loop when they try to grade their solutions. One possible way to overcome this problem is to implement the CYK parsing function in OpenFLAP and determine students' solution correctness.

4.2 Exercises

OpenFLAP also allows us to create exercises, auto-assess them, and report the result to an LMS through OpenDSA's standard framework. Creating exercises benefits both instructors and students. Instructors can create any number of exercises without the need to worry about grading these exercises. All an instructor needs is to create the exercise and provide the test cases and the solution for the exercise. OpenFLAP also has the ability to take the solution for the exercise and auto-generate test cases for the instructor. OpenFLAP allows instructors to determine which test cases will be shown to the students and which will be hidden. In the auto-generated test cases, instructors determine the number of test cases, and how many of them should be hidden. OpenFLAP auto-generates the test cases and

hides some of them. Figure 4.1 shows the web page for creating an auto-graded exercise. Instructors select the model type (FA, PDA, TM, or Grammars). Under the description, instructors write the question prompt to the students and add all test cases to test students on them. For each test case, instructors can specify if the test case string should be accepted or rejected by students' answers. Clicking on "get JSON" will generate a JSON object for the exercise as shown in Figure 4.2. The JSON object allows instructors to tell the system more details about the exercise. In 4.2 line 3, instructors determine the number of test cases that should be accepted by students' solutions (in the figure, "totalTruecases": 5 means 5 test cases that students' solutions should accept including the previously added test cases on the web page). The same is true about the false test cases. Students' solutions must reject these test-cases. In line 5, instructors indicate the alphabet set for the generated test cases. This line starts with an alphabet based on strings the instructor added in the web page in Figure 4.1. Line 6 shows the range of generated test-cases lengths. Lines 10 to 39 show the test cases that instructors added through the web page. In each of these test cases, instructors can determine if the test case is visible or not by setting the "ShowTestCase" to true or false, respectively. Finally, instructors can specify the path to the exercise solution. We plan to add all these capabilities to the web page to make things easier for instructors.

4.2.1 Auto-Graded Exercises

Auto-graded exercises ask students to build different models. These exercises are similar to programming exercises that ask students to write some code to solve a problem. To test students' solutions, instructors can assign static and auto-generated test cases. The exercises are shown to students in an editor and some buttons that they need to use to build the model or write the grammar.

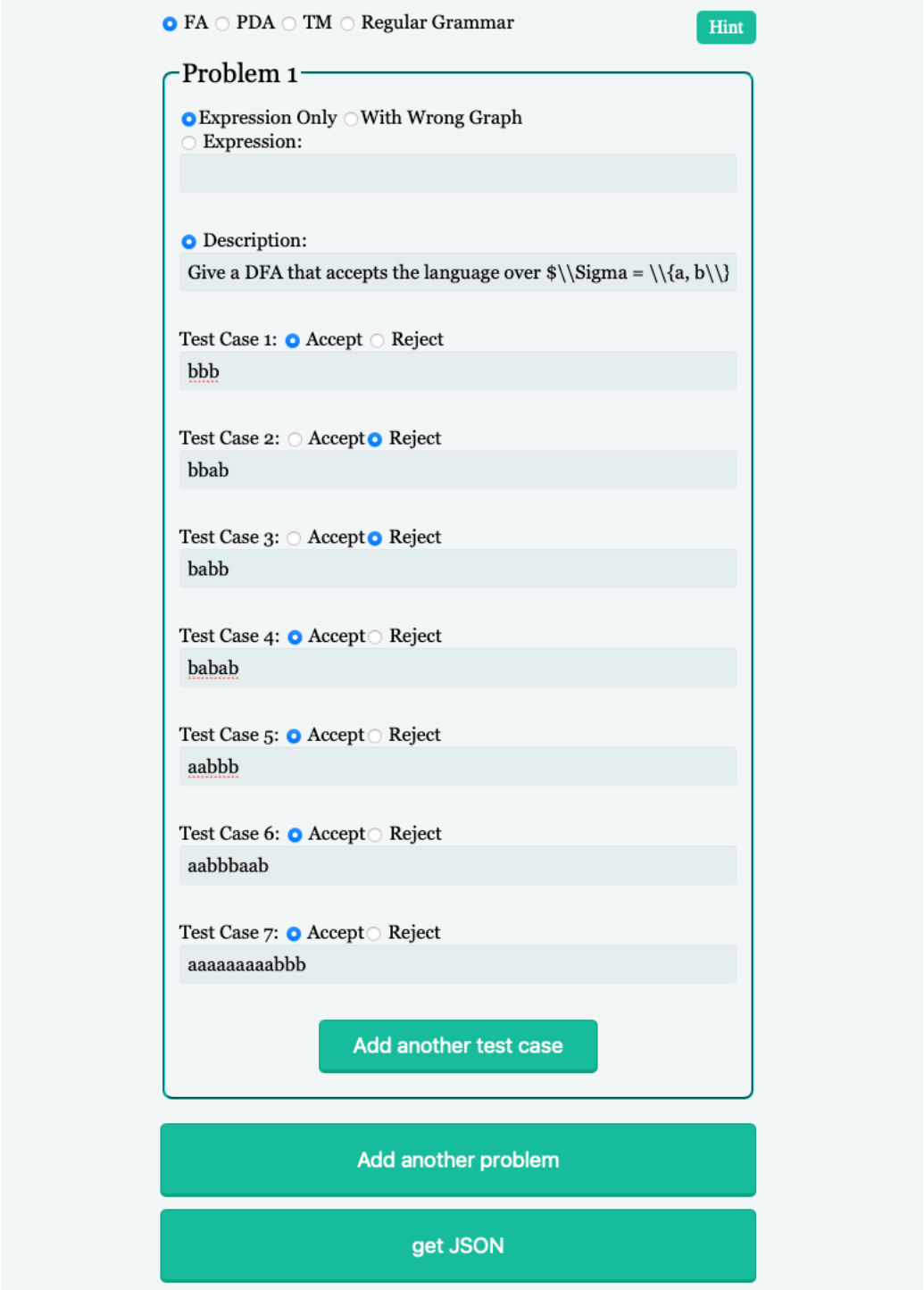


Figure 4.1: Example for creating an auto-graded exercise for the problem [Give a DFA that accepts the language over $\sigma = \{a, b\}$ of strings with any odd number of a's and any even number of b's] using OpenFLAP web page

```
[{
  "exerciseType": "DFA",
  "totalTrueCases": 5,
  "totalFalseCases": 5,
  "containLetters": ["a", "b"],
  "randomStringLength": [0, 15],
  "description":
  "Give a DFA that accepts the language over $\\Sigma = \\{a, b\\}$ of strings
   with any even number of a's and at least three b's.",
  "testCases": [
    {
      "bbb": true,
      "ShowTestCase": true
    },
    {
      "bbab": false,
      "ShowTestCase":true
    },
    {
      "babb": false,
      "ShowTestCase":true
    },
    {
      "babab": true,
      "ShowTestCase":true
    },
    {
      "aabbb": true,
      "ShowTestCase":true
    },
    {
      "aabbbaab": true,
      "ShowTestCase":true
    },
    {
      "aaaaaaaaabbb": false,
      "ShowTestCase":true
    }
  ],
  "graph": {
    "nodes": [],
    "edges": []
  },
  "solution": "../../../machines/FA/DFAevenamin3b.jff"
}]

```

Figure 4.2: Example for creating an auto-graded exercise using OpenFLAP JSON objects

Once an exercise is auto-graded, the score is sent to the LMS through the OpenDSA system. This means that instructors will not spend time to grade or store student's grades. In addition, OpenFLAP remembers student's solutions to all exercises. This means that students will always find their previous solutions for each exercise. Thus they can retry to solve the exercise with a different solution, study the existing solution, or continue their exercise. If students decided to try different exercises, the system would not update students' grades except if the new grade is higher than the current grade. This allows students to try as many different solutions as they want without any fear of losing grades.

In OpenFLAP, we have an editor for each model. Each exercise shows students the appropriate model editor with embedded three buttons(Figure 4.5) that appear in any auto-graded exercise.

- Reset button to clear both the shown and stored solution
- Show Test Cases button to show the visible test cases. For hidden test cases, OpenFLAP shows students that there are some hidden test cases. See Figure 4.3
- Grade button to grade the current solution and store it in the system. After grading, OpenFLAP shows students the result of every visible test case with the expected result of the test case and student's solution result, as shown in Figure 4.6.

To be compatible with JFLAP, OpenFLAP uses similar button icons as JFLAP. These buttons are used to build the solution model for DFA, NFA, PDA, and TM exercises. These six button functions are

- The mouse arrow is used to drag model nodes
- The circle button is used to create model nodes. Each click in the big square area will create a node labeled with q_0, q_1, etc.

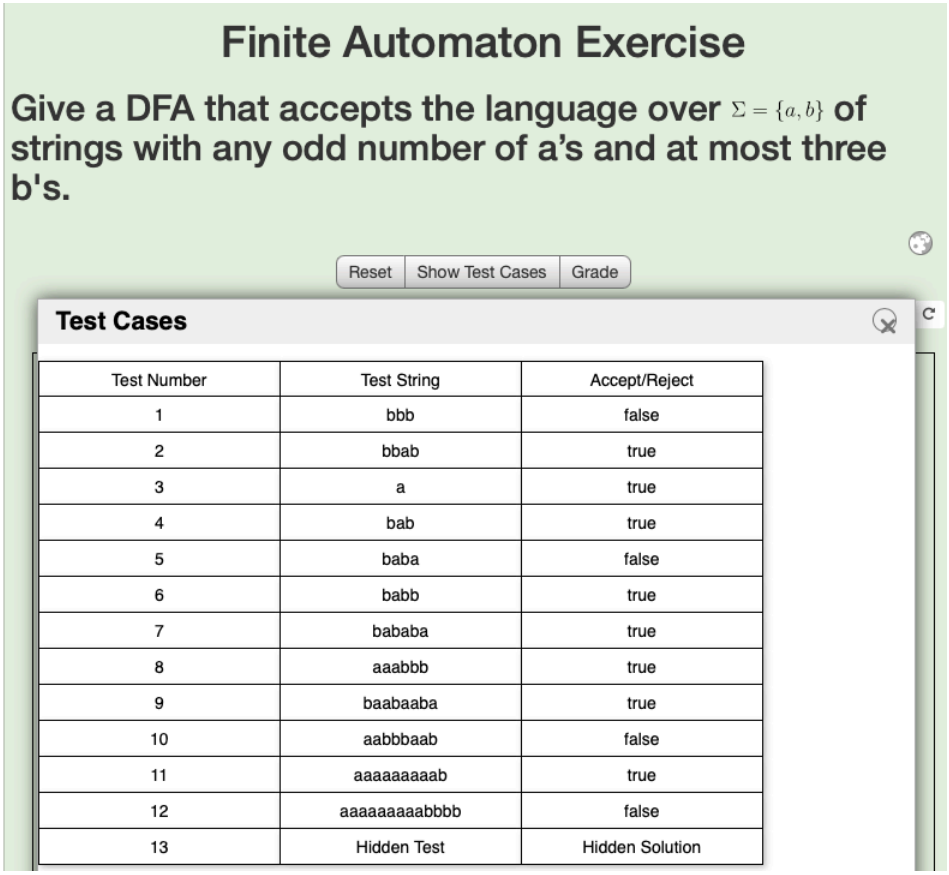


Figure 4.3: Example for building DFA auto-graded exercise after clicking on Show Test Cases button

- The arrow button is used to create a transition between two nodes, or a loop transition if a node is clicked twice. Different models have different transition formats used in the edge label.
- The X button is used to delete a node or a transition by clicking on it.
- The last two buttons undo a step or redo an undone step.

Figure 4.4 shows these buttons with their descriptions.

For writing grammar exercises OpenFLAP shows a grammar editor to students, Figure 4.7, with the same embedded buttons (Reset, Show Test Cases, and Grade). In this example the

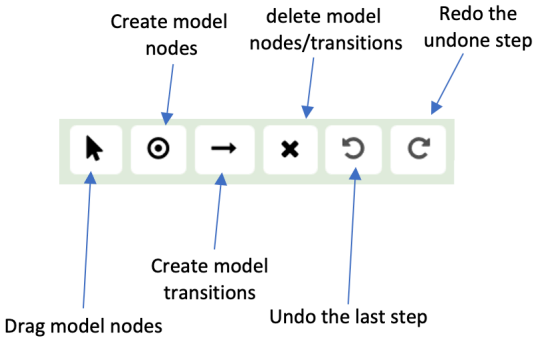


Figure 4.4: OpenFLAP button descriptions.

student solution satisfied only 5 correct test cases out of 6, so the student will earn 80% of the exercise grade only. Regular expressions exercises are similar.

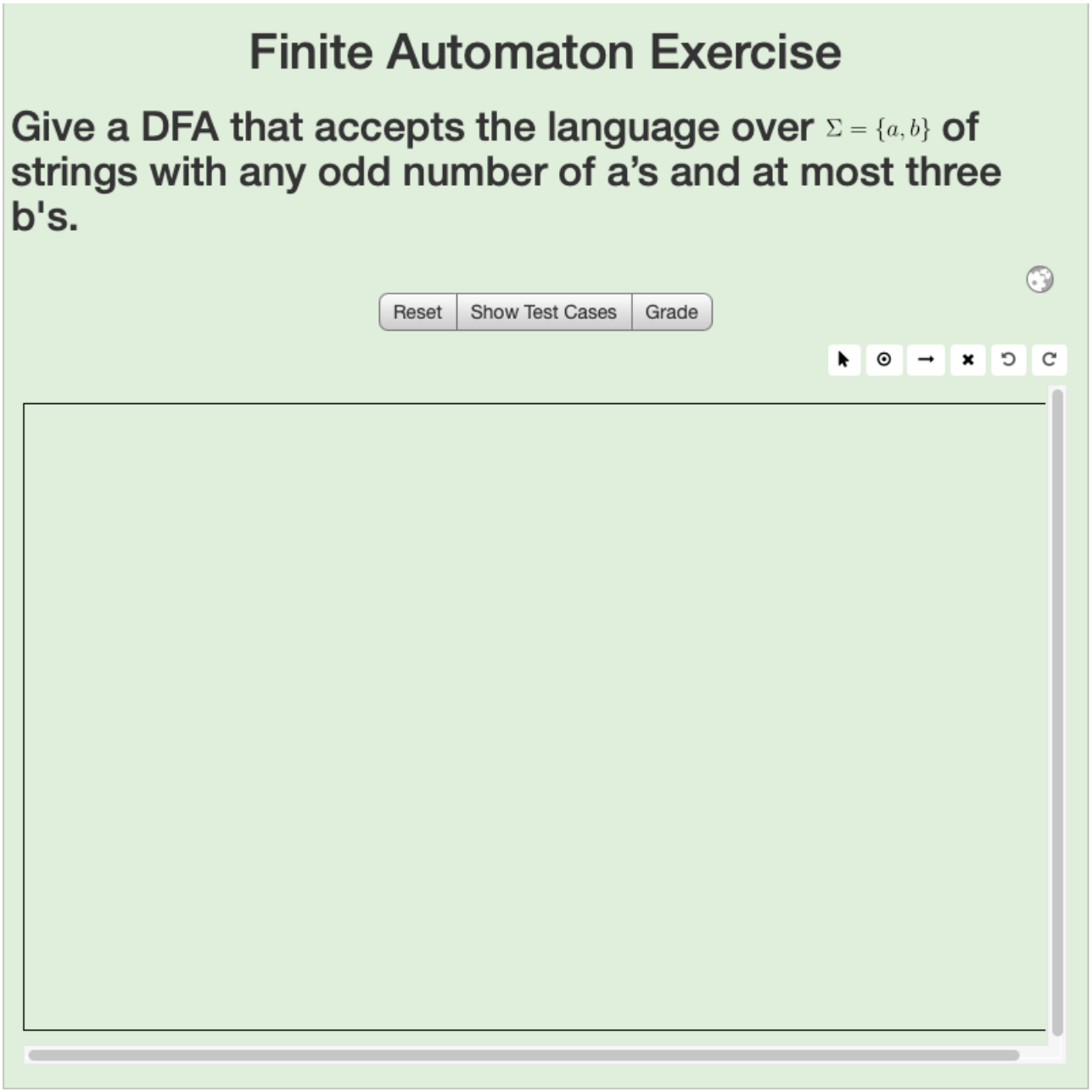
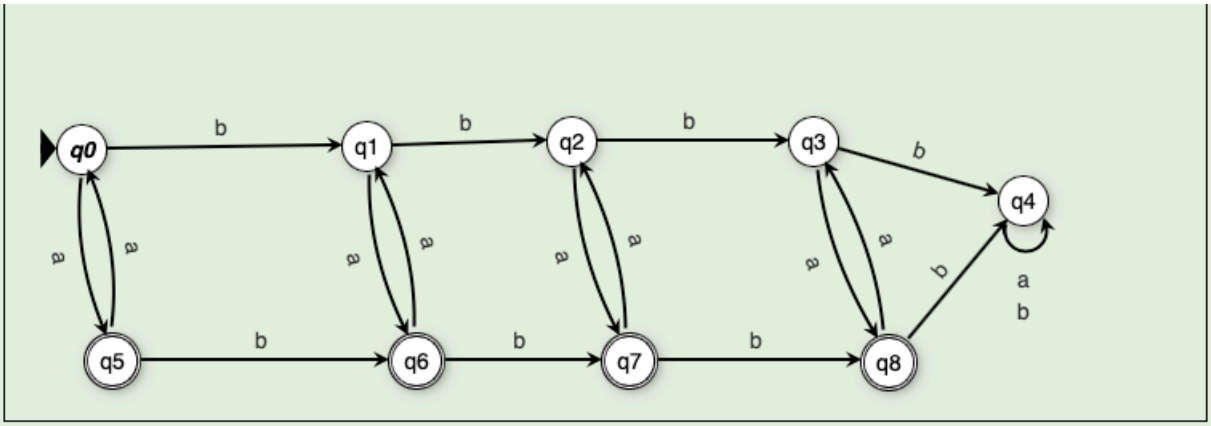


Figure 4.5: Example of editor window for building DFA auto graded exercise



Correct cases: 20 / 20

Test Case	Standard Result	Your Result
bbb	Reject	Reject
bbab	Accept	Accept
a	Accept	Accept
bab	Accept	Accept
baba	Reject	Reject
babb	Accept	Accept
bababa	Accept	Accept
aaabbb	Accept	Accept
baabaaba	Accept	Accept
aabbbaab	Reject	Reject
aaaaaaaaab	Accept	Accept
aaaaaaaaabbbb	Reject	Reject
Hidden Tests	Accept	Accept

Figure 4.6: 100% correct solution for an exercise. OpenFLAP shows all visible test case results to students. The hidden test case line tells students that there are some hidden test cases and whether they are all correct, or at least one is incorrect.

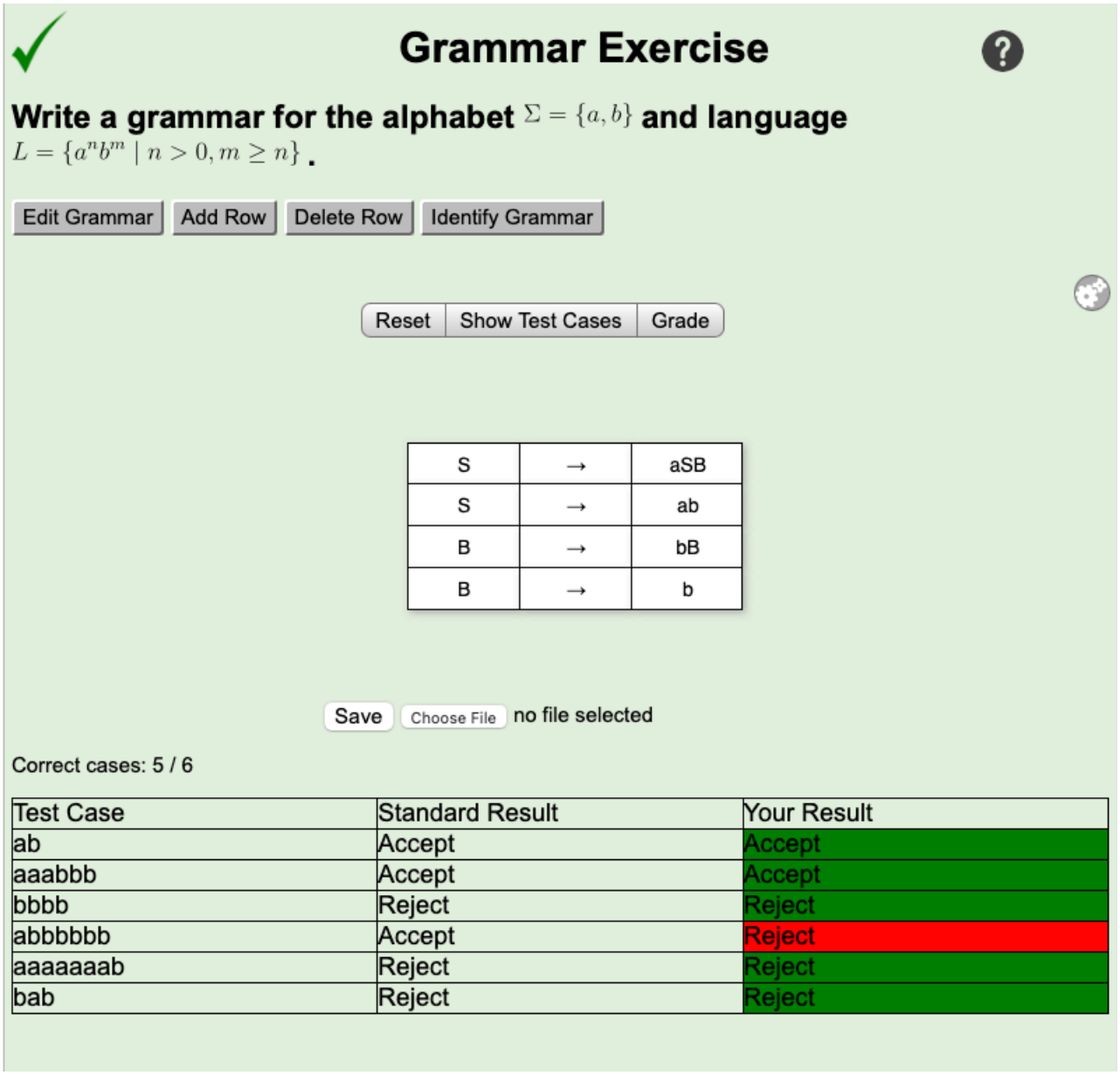


Figure 4.7: A grammar writing auto graded exercise.

4.2.2 Proficiency Exercises

Proficiency exercises ask students to apply an algorithm to a given model or grammar. Our Formal Languages course contains several algorithms like

- Convert n non-deterministic finite automata (NFA) to a deterministic finite automata (DFA)
- Convert a regular grammar to a NFA
- Convert an NFA to a regular grammar
- Convert an NFA to regular expressions
- Convert regular expression to an NFA
- Convert a context-free grammar (CFG) to a PDA
- Remove lambda, unit, and useless productions
- Transform a grammar to Chomsky normal form

To provide practice exercises about these algorithms, OpenFLAP allows instructors to create proficiency exercises where students need to apply algorithm steps on a given model. Currently, OpenFLAP supports NFA to DFA, minimize DFA, and transforms a grammar to Chomsky normal form (this one includes remove lambda, unit, and useless productions). To create a proficiency exercise, instructors need to identify the algorithm type and provide the model/grammar that the algorithm should work on. Since proficiency exercises ask students to apply a series of steps on the input model, students are prone to mistakes that may lead to totally incorrect solutions. OpenFLAP monitors student's steps and prompts them immediately if there is an incorrect step (and deducts credit for this step). In other words,

OpenFLAP checks the correctness of every student step and shows a message to the student to prompt him/her to retry the incorrect step before moving forward to the next steps.

AV > OpenFLAP > exercises > FLAssignments > FA > {} NFAtoDFAex1.json > ...

```
[
  {
    "description": "Convert the following NFA to a DFA.",
    "testCases": [
      {
        "The resulting FA is a DFA": true
      }
    ],
    "graph": {
      "nodes": [{"left": 21,"top": 54,"i": true,"f": false},
        {"left": 56,"top": 186,"i": false,"f": false},
        {"left": 179,"top": 157,"i": false,"f": false},
        {"left": 291,"top": 161,"i": false,"f": true},
        {"left": 341,"top": 57,"i": false,"f": true},
        {"left": 170,"top": 51,"i": false,"f": false}],
      "edges": [{"start": 0,"end": 0,"weight": "a<br>b"},
        {"start": 0,"end": 1,"weight": "a"},
        {"start": 0,"end": 5,"weight": "b<br>&lambda;"},
        {"start": 0,"end": 2,"weight": "&lambda;"},
        {"start": 1,"end": 1,"weight": "a"},
        {"start": 1,"end": 2,"weight": "b"},
        {"start": 2,"end": 2,"weight": "a"},
        {"start": 2,"end": 3,"weight": "b"},
        {"start": 4,"end": 4,"weight": "a<br>b"},
        {"start": 5,"end": 4,"weight": "b"},
        {"start": 5,"end": 5,"weight": "a"}],
      "shorthand": false}
  }
]
```

Figure 4.8: Defining an NFA to DFA proficiency exercise.

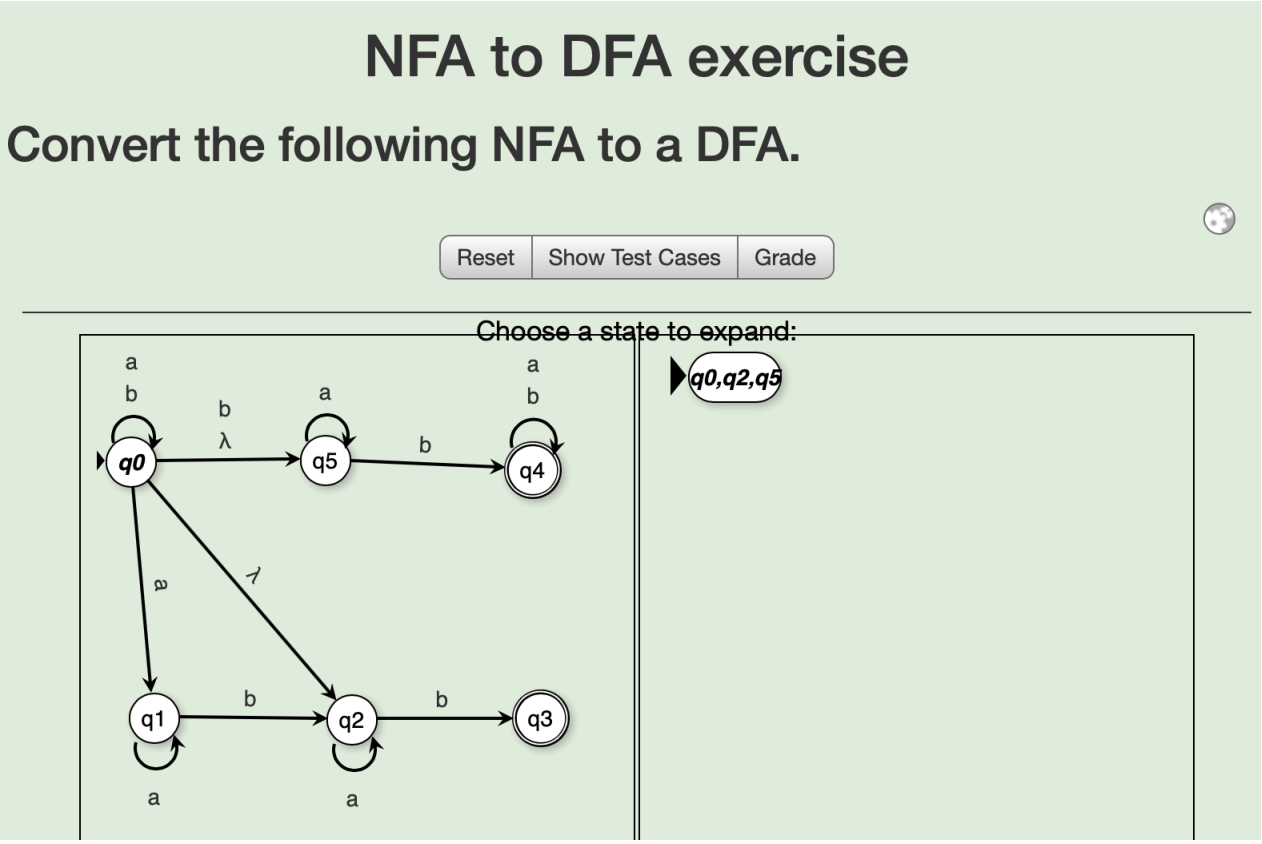


Figure 4.9: An NFA to DFA proficiency exercise. In the left hand side is the NFA and students should use the space on the right to create the equivalent DFA.

4.2.3 Converting an NFA to a DFA exercises

Converting an NFA to a DFA is one of the famous exercises found in any Formal Languages course. OpenFLAP allows instructors to create NFA to DFA exercises by creating a JSON object that contains the NFA. Figure 4.8 shows how to define an NFA to DFA exercise, and Figure 4.9 shows the resulting exercise. In defining the exercise, instructors need to define the initial model, defined as a **graph** object in Figure 4.9. The format of the graph object is the same as the JFLAP file format (JFF files). Instructors can create the exercise easily by using the create exercise web page, Figure 4.1. Instructors will draw the model and OpenFLAP will create the appropriate exercise file.

To solve the NFA to DFA exercise, a student needs to start with the given DFA start state (in the right-hand-side in Figure 4.9). In this exercise, the start state represents all the states from the NFA start state where you start without seeing any symbols yet. Based on the NFA to DFA algorithm, clicking this node will ask the student about the transition letter and writing the correct transition letter to answer another question about the new state that the student wants to add. If the state is a new state on the DFA side, the student needs to click on any empty space to create the state, and OpenFLAP will ask about the state label. If the state exists on the DFA side, the student needs to click on an existing state to add a transition to that state. To prevent students from doing incorrect steps, OpenFLAP monitors all student steps and reports immediately any incorrect steps to the student (and deducts credit for that step). After finishing the exercise, students can click on the grade button to get a full report about the conversion process. This report shows the student the total number of steps, the percentage of the correct steps, and a table that list all incorrect steps made by the student. The student grade is calculated as the percentage of the correct overall steps. If this percentage is above a specific threshold, specified by the instructor, the student will receive full credit. Otherwise, the student will receive a zero and should retry the exercise. Figure 4.13 shows snapshots for the exercise steps.

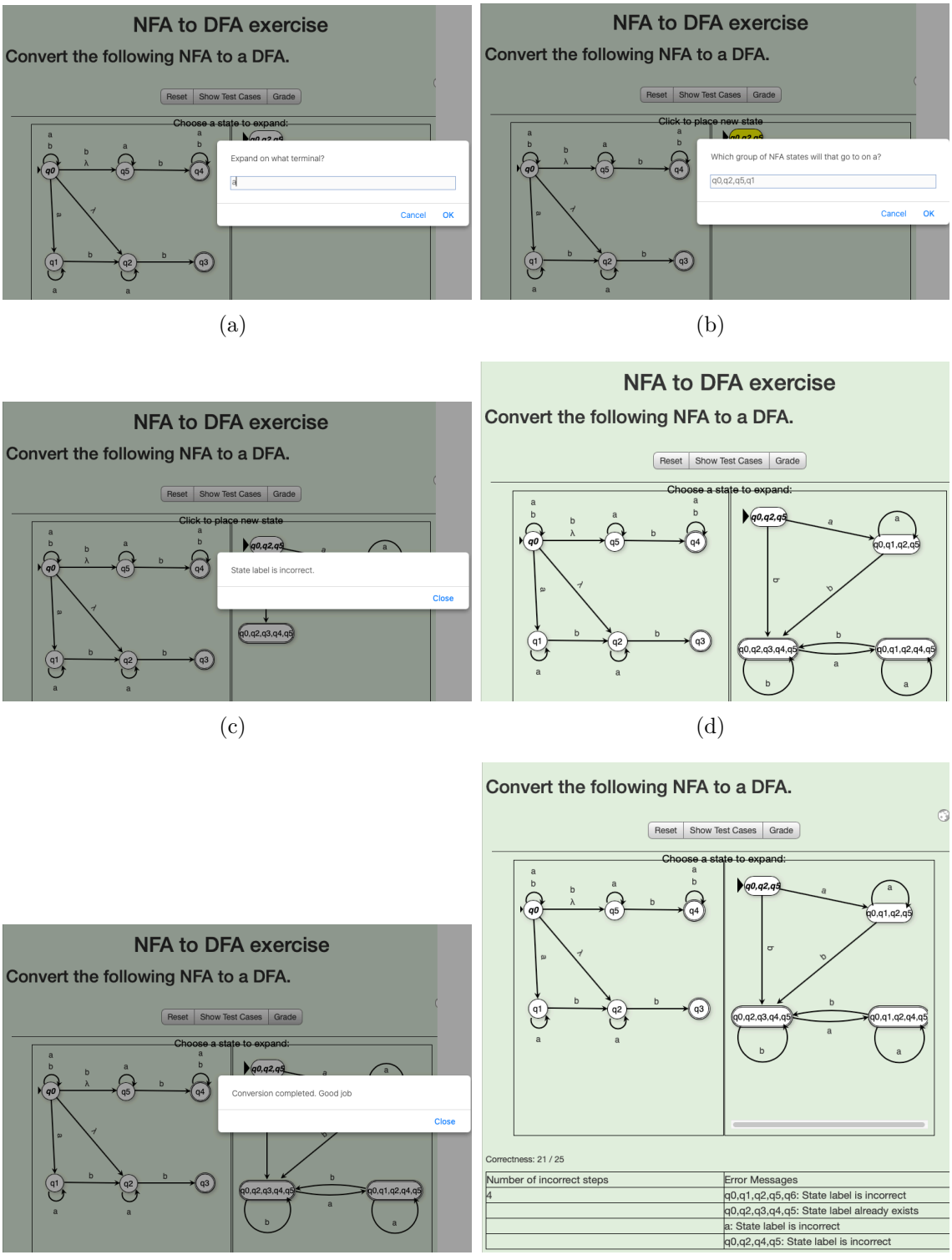


Figure 4.10: (a) click on a state and identify the terminal for the possible transition
(b) write the transition destination
(c) a message that tells the student about an incorrect step
(e) a student finished the conversion process and clicked on the grade button
(f) OpenFLAP presents all incorrect steps made by the student

4.2.4 DFA minimization exercises

Similar to NFA to DFA exercises, DFA minimization exercises require instructors to initialize the input DFA and students need to apply the algorithm steps on that model. Figure 4.11 shows the JSON object that instructors need to provide the DFA that students need to minimize. Similar to NFA to DFA exercises, instructors can create DFA minimization exercises by using the create exercise web page to draw the DFA and generate the exercise file that is shown in 4.11.

Figure 4.12 shows the DFA minimization exercise. On the left is the DFA to be minimized and on the right is the tree that will help students minimize the DFA. The root for the tree is the set of DFA states. This set is divided into two nodes, nonfinal states and final states. Students need to keep splitting the nodes in the tree in this exercise until there are no more splits. In DFA minimization, splitting a node means to find if all nodes have a similar behavior with reading the same input or not. If they are different, students should split the nodes into groups of nodes with similar behavior. To do that, students need to test each leaf in the tree with every alphabet letter in the exercise alphabet set. If a split is found, the student needs to click on the tree node and click on "Set terminal" button. OpenFLAP will check if this decision is correct. If it is, then the tree will add two children to the selected node. Students can add more children by clicking on "Add child" button. If they need to remove a child node, they can select the node and click on "Remove selected partition" button. Next, students need to add nodes to the created children nodes. Adding children nodes can be done by selecting one of the children nodes followed by clicking on the DFA states. If students selected an incorrect state, they could click on it again to remove it from the children node. After selecting the nodes for all children nodes, no matter the order, students need to click on "Check nodes" button to ask OpenFLAP to check if the split is correct or not. Since splitting the tree node is a tricky step, OpenFLAP provides

AV > OpenFLAP > examples > {} dfaminimization2.json > ...

```
[
  {
    "description": "Minimize the following DFA",
    "testCases": [
      {
        "": true
      }
    ],
    "graph": {
      "nodes": [{"left": 46,"top": 58,"i": true,"f": true},
        {"left": 171,"top": 58,"i": false,"f": false},
        {"left": 310,"top": 57,"i": false,"f": true},
        {"left": 40,"top": 187,"i": false,"f": false},
        {"left": 167,"top": 187,"i": false,"f": false},
        {"left": 314,"top": 186,"i": false,"f": false}
      ],
      "edges": [
        {"start": 0,"end": 1,"weight": "a"},
        {"start": 0,"end": 3,"weight": "b"},
        {"start": 1,"end": 2,"weight": "a"},
        {"start": 1,"end": 4,"weight": "b"},
        {"start": 2,"end": 1,"weight": "a"},
        {"start": 2,"end": 5,"weight": "b"},
        {"start": 3,"end": 0,"weight": "b"},
        {"start": 3,"end": 4,"weight": "a"},
        {"start": 4,"end": 5,"weight": "a"},
        {"start": 4,"end": 1,"weight": "b"},
        {"start": 5,"end": 4,"weight": "a"},
        {"start": 5,"end": 2,"weight": "b"}
      ],
      "shorthand": false
    }
  }
```

Figure 4.11: The JSON object that is used to create a DFA minimization exercise

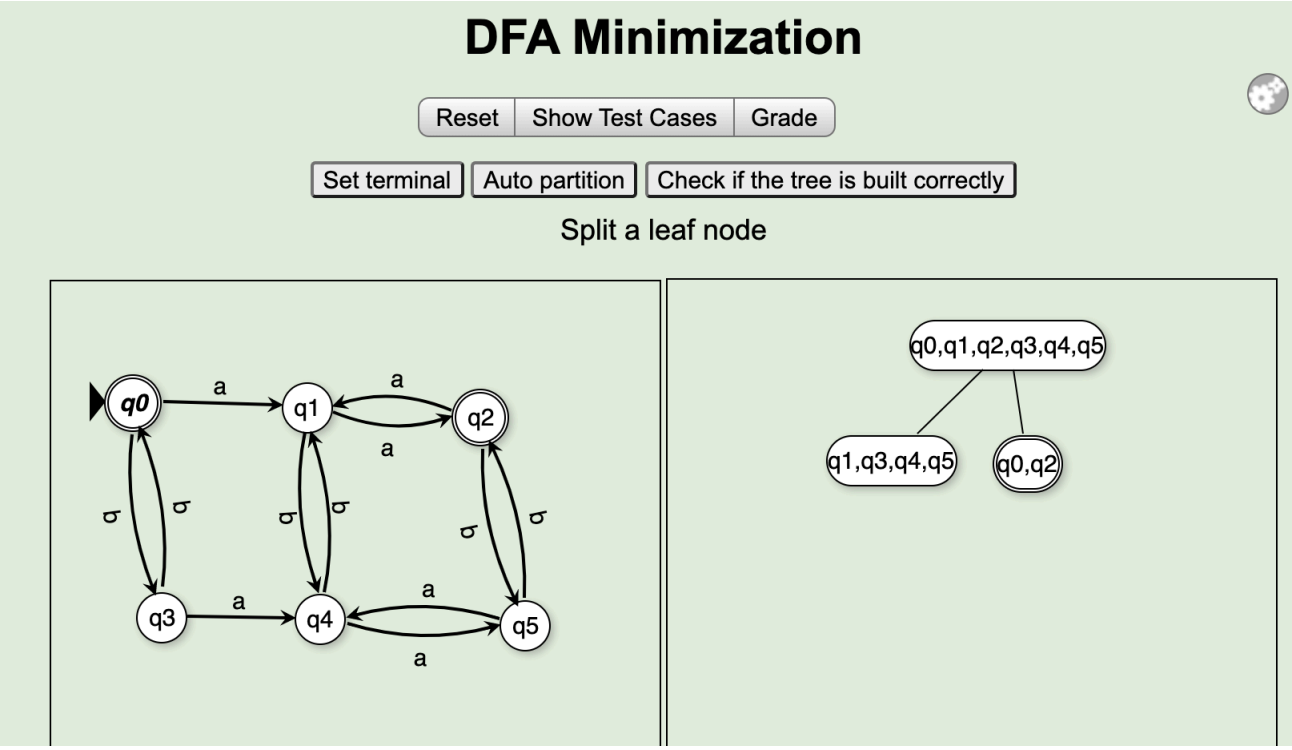


Figure 4.12: DFA minimization exercise

students with an "Auto partition" button that can be used twice. Each time students select a tree node and click on that button, OpenFLAP will auto split that node to the students. When students believe that they are done with splitting the nodes, they need to click on "Check if the tree is built correctly" button. If students correctly finish the splitting process, OpenFLAP will ask students to build the minimized DFA. The tree's leaf nodes will be the minimized DFA states. Students now need to build the transitions for the minimized DFA.

To build the minimized DFA, students need to click on the **Add Transition** button. A message will tell students to select the source state then select the target node. Next, OpenFLAP will ask students to enter the transition letter. If this transition is correct, OpenFLAP will add the transition to the minimized DFA. Students need to add all transitions to the minimized DFA, and if they are stuck, they can click on the **Hint** button to ask OpenFLAP to add a transition automatically. Students can use this button twice per exercise. Once

students believe that they are done, they can click on the **Grade** button. OpenFLAP will present all mistakes made by the student and show the correctness of the solution. Like the NFA to DFA exercise, the grade is calculated as the percentage of students' correct steps over the overall steps. If that percentage is larger than a threshold value (0.9 in our book), the student will get full credit. Figure 4.13 shows the exercise steps.

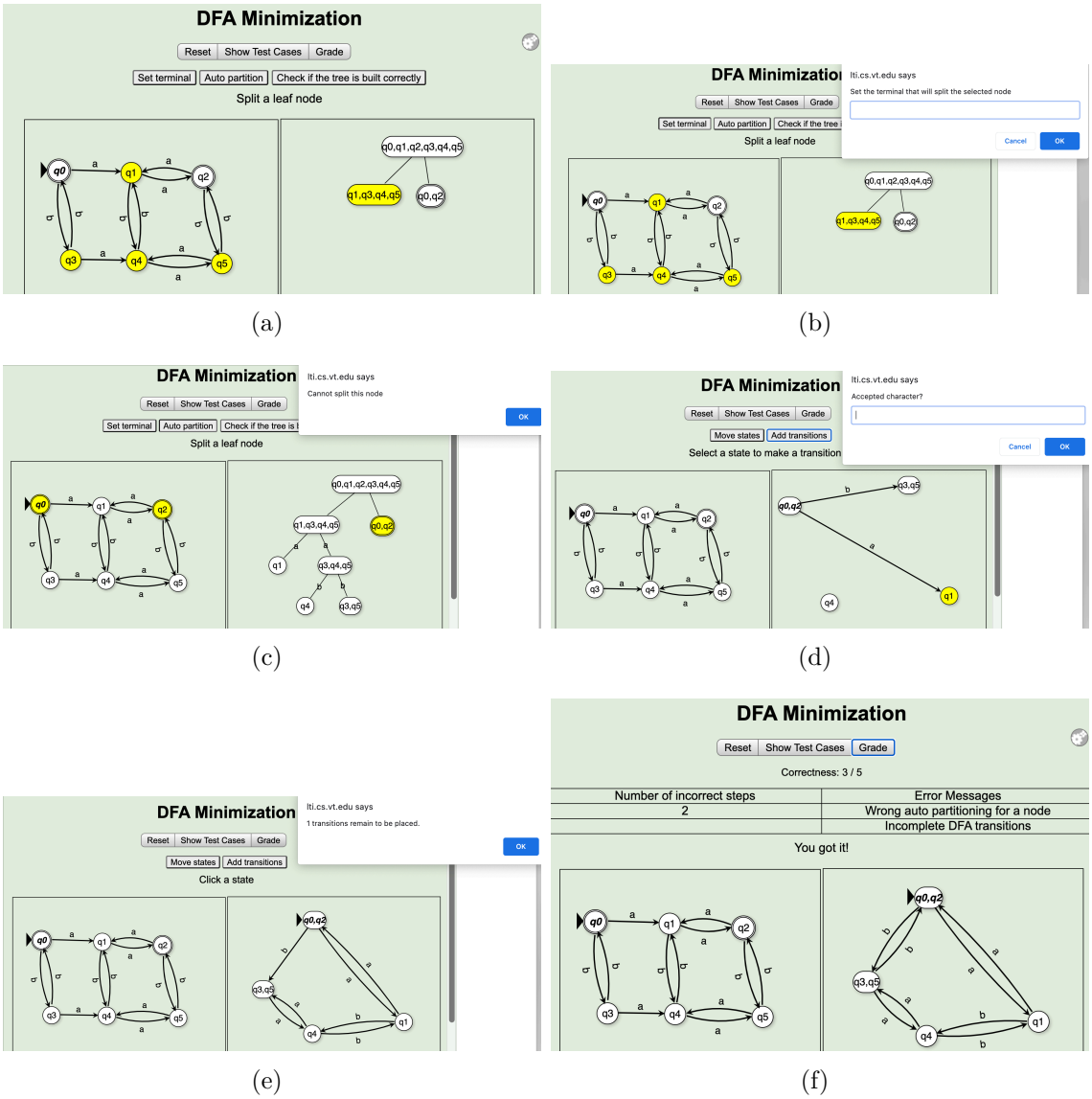


Figure 4.13: DFA minimization exercise steps
(a) Selecting a tree node will highlight all DFA states to help students find the split
(b) Writing the letter that will be used to split the selected tree node
(c) A message that tells the student about an incorrect split attempt
(d) Writing the transition letter
(e) A message that tells the student about an incorrect grade attempt
(f) OpenFLAP presents all incorrect steps made by the student

4.2.5 Grammar Transformation exercises

If the exercise is about grammar transformation, instructors need to identify the input grammar instead of the model. See Figure 4.14 for an example about defining a Grammar Transformation exercise and Figure 4.15 for the equivalent exercise. Similar to other proficiency exercises, instructors can use the create exercise web page to create this type of exercises.

In grammar transformation exercises, students are asked to remove lambda, unit, and useless transitions. Other grammar transformation exercises may ask students to convert the grammar to Chomsky normal form (CNF). CNF also requires the grammar to be lambda-free, unit-free, and useless-free. Thus, in both cases, students will need to remove those types of productions. In the beginning, students are asked to remove lambda productions. Lambda productions are defined as any production in the form

1. Direct lambda production: $A \rightarrow \lambda$, where A is a variable

2. Indirect lambda production: $A \rightarrow B$, and $B \rightarrow \lambda$ where A, and B are variables. In this case, A produces λ by going to B, and B produces the λ.

To remove lambda productions, students start by selecting the direct and indirect lambda productions; see 4.16 part a. After that, students need to remove the lambda productions. Then they are asked to write the new productions that include right-hand sides with variables that do not derive λ, see 4.16 part b. After correctly removing the lambda productions, students are asked to remove unit productions, i.e., $A \rightarrow B$ where A and B are variables. To remove unit productions, students start with building a directed dependency graph (DDG). The graph nodes represent each variable as a node, and the edges are the unit productions. For example, if the grammar has a unit production $A \rightarrow B$, the DDG will contain an

AV > OpenFLAP > examples > {} dfaminimization2.json > ...

```
[
  {
    "description": "Minimize the following DFA",
    "testCases": [
      {
        "": true
      }
    ],
    "graph": {
      "nodes": [{"left": 46,"top": 58,"i": true,"f": true},
        {"left": 171,"top": 58,"i": false,"f": false},
        {"left": 310,"top": 57,"i": false,"f": true},
        {"left": 40,"top": 187,"i": false,"f": false},
        {"left": 167,"top": 187,"i": false,"f": false},
        {"left": 314,"top": 186,"i": false,"f": false}
      ],
      "edges": [
        {"start": 0,"end": 1,"weight": "a"},
        {"start": 0,"end": 3,"weight": "b"},
        {"start": 1,"end": 2,"weight": "a"},
        {"start": 1,"end": 4,"weight": "b"},
        {"start": 2,"end": 1,"weight": "a"},
        {"start": 2,"end": 5,"weight": "b"},
        {"start": 3,"end": 0,"weight": "b"},
        {"start": 3,"end": 4,"weight": "a"},
        {"start": 4,"end": 5,"weight": "a"},
        {"start": 4,"end": 1,"weight": "b"},
        {"start": 5,"end": 4,"weight": "a"},
        {"start": 5,"end": 2,"weight": "b"}
      ],
      "shorthand": false
    }
  }
```

Figure 4.14: The JSON object that is used to create a grammar transformation exercise

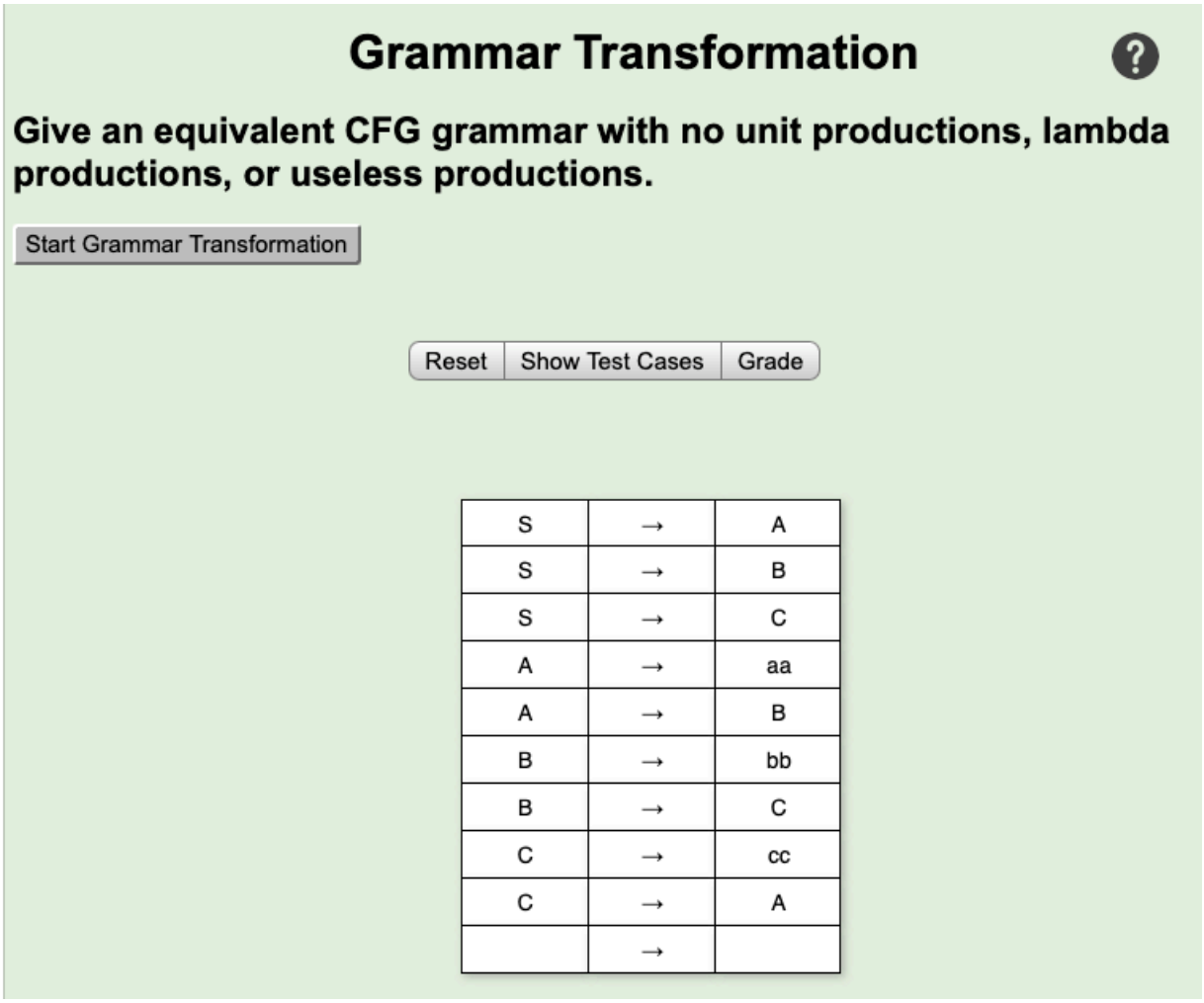
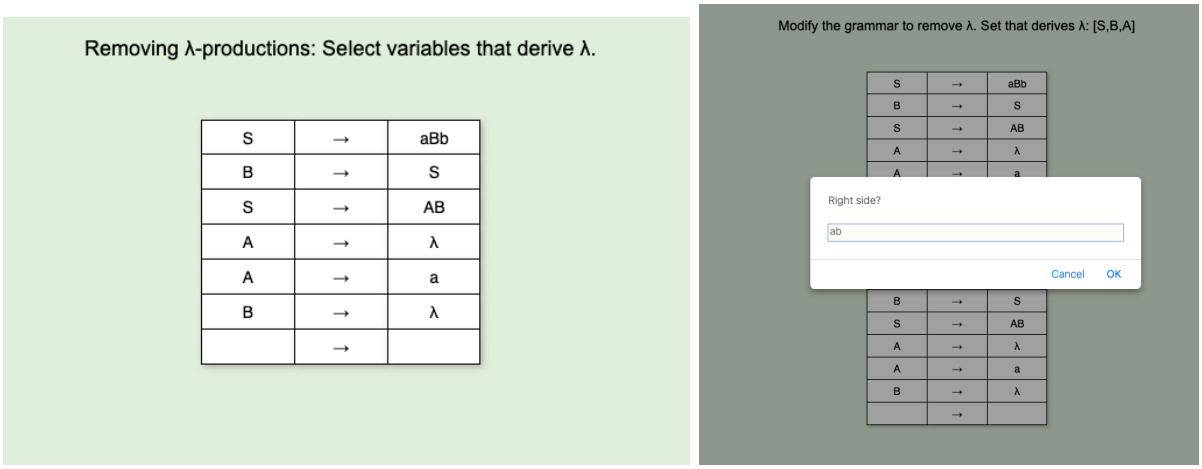


Figure 4.15: Grammar transformation exercise



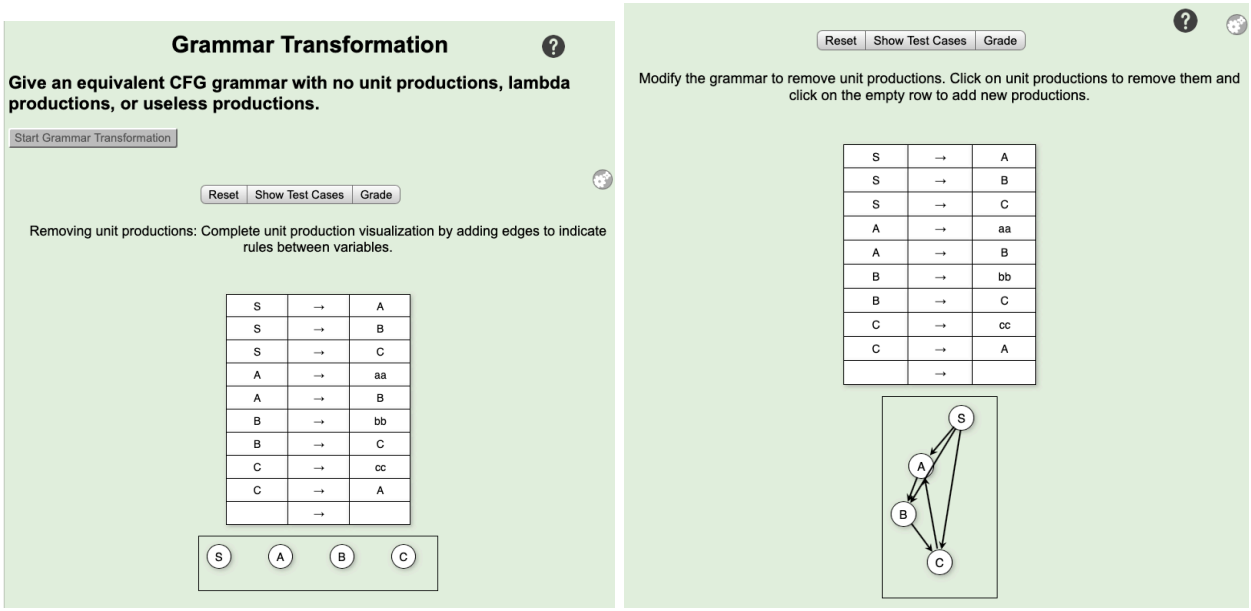
(a) (b)

Figure 4.16: Remove lambda productions
(a) Students are asked to select the variables that produces lambda either directly or indirectly
(b) Students rewrite the productions after removing the lambda productions

edge from the node A to the node B. OpenFLAP presents the DDG nodes without any edge. Students should add the graph edges based on the grammar. Students should select the source node then select the target node and the edge is then added automatically, see Figure 4.17. After building the DDG, students are asked to remove all unit productions by clicking on all unit productions. Then, students need to write the new productions that result from removing the unit productions.

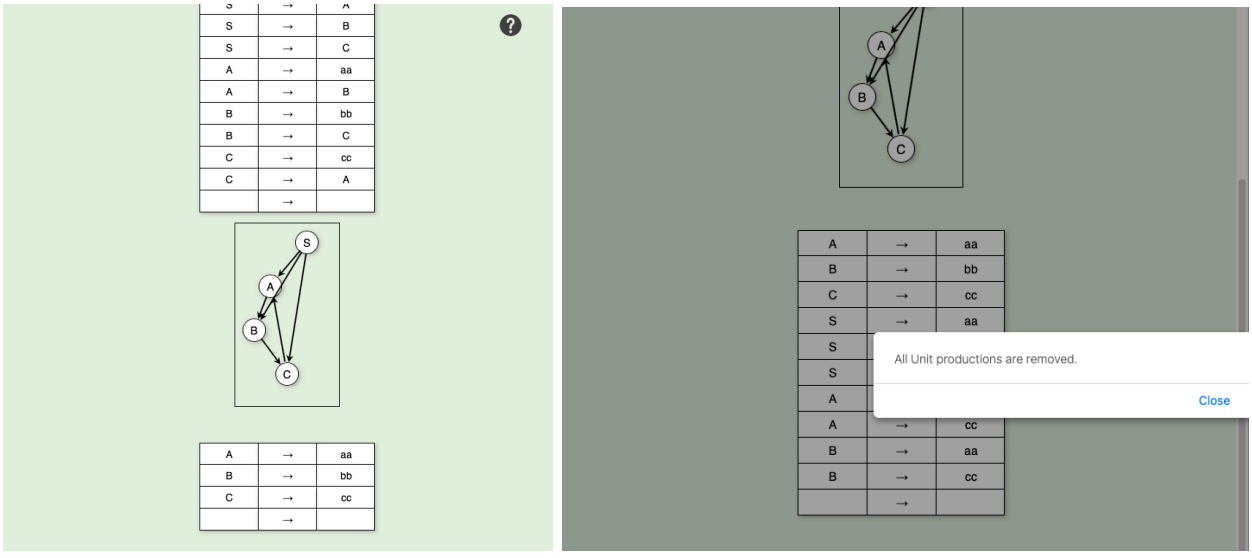
The third step is to remove the useless productions. There are two types of useless productions.

- Unreachable productions. Productions that can not be reached from the start variable. In this case, OpenFLAP will as students to build another DDG, similar to 4.16, to determine the unreachable states. Then students need to remove their corresponding productions.



(a) (b)

Figure 4.17: Remove unit productions
(a) Students are asked to build the DDG to determine the unit productions
(b) Students finished building the DDG for the given grammar

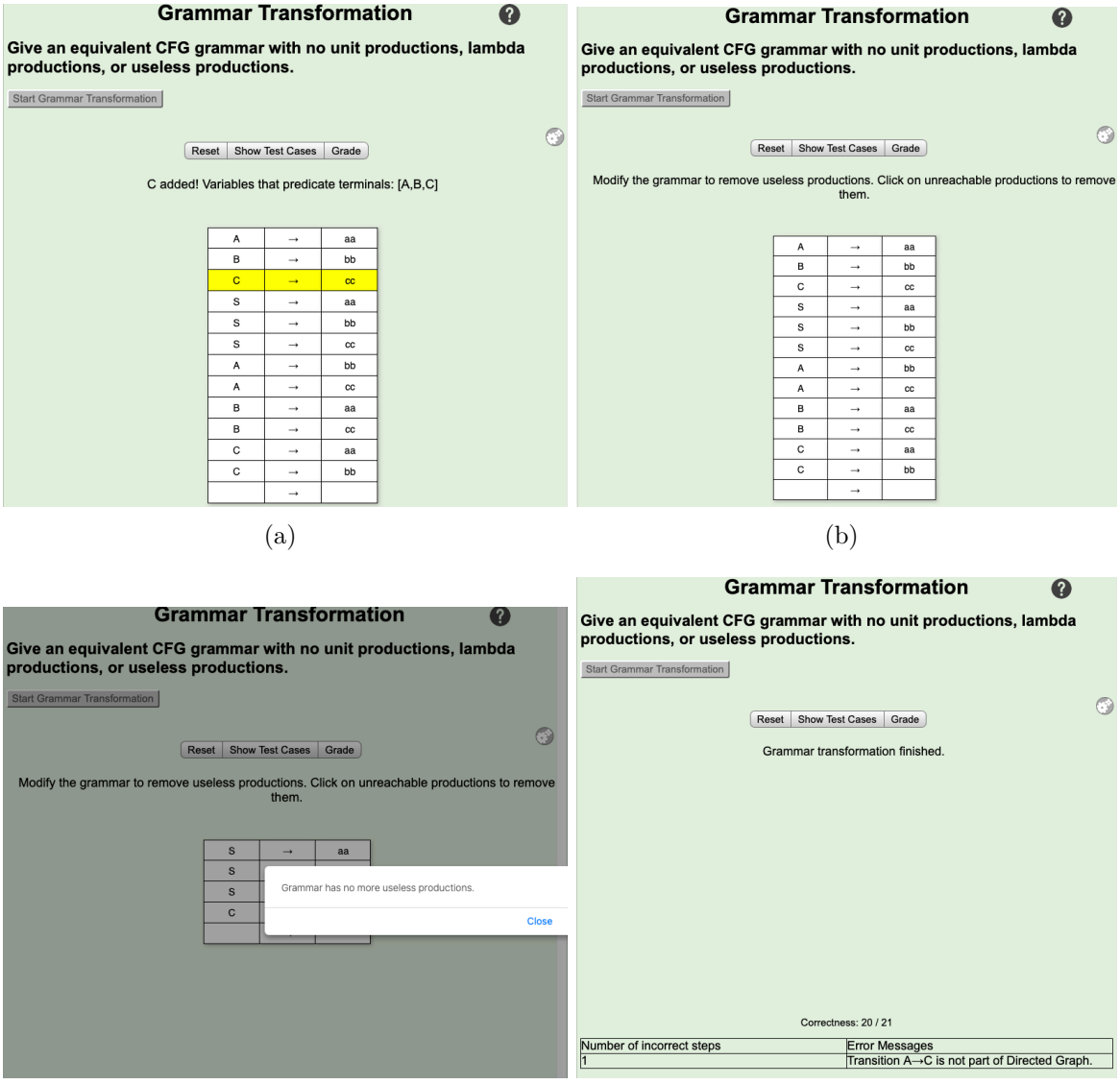


(a) (b)

Figure 4.18: Remove unit productions
(a) Students rewrite the productions after removing the unit productions
(b) Students successfully finished removing all unit productions

- Non-terminating variables. OpenFLAP will ask the students to select the variables that terminate, i.e., productions that have the form $A \rightarrow x$ where A is a variable and x is one or more terminals. Then students should remove useless productions by selecting all of them, see Figure 4.19.

Finally, once students finish removing useless productions, OpenFLAP will show the percentage of the correct steps over the total performed steps. Again, if the correctness percentage is higher than the predefined threshold value (0.9), students will earn the full credit. Otherwise, students will not get any points and need to redo the exercise.



(a) (b)

(c) (d)

Figure 4.19: Remove useless productions
(a) Students are asked to select the variables that terminate
(b) Students rewrite the productions after removing the lambda productions
(c) Students removed all useless productions
(d) Finally, OpenFLAP reports the exercise score and students mistakes

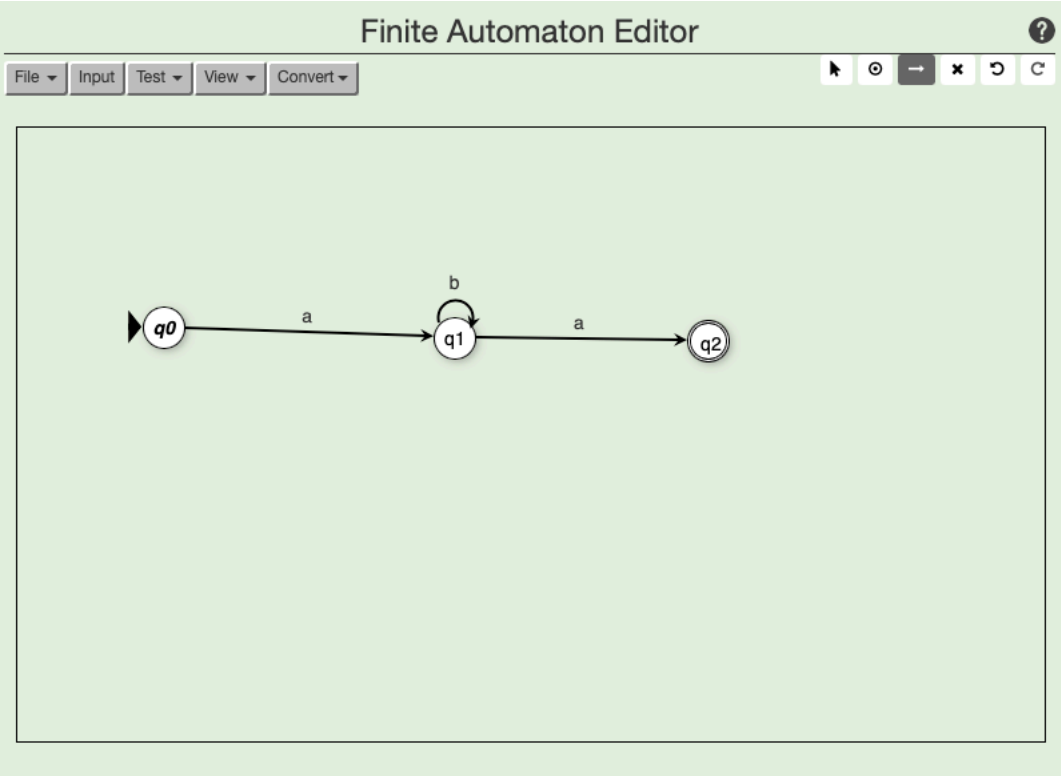


Figure 4.20: OpenFLAP editor for finite automata

4.3 Editors

Similar to JFLAP, OpenFLAP contains a set of editors to allow students to build various models and test them. OpenFLAP contains

- NFA/DFA editor, see Figure 4.20: This editor is used to create an NFA or DFA then
  - Test the machine with strings and trace the acceptance/rejection
  - Convert an NFA to a DFA
  - Minimize a DFA
  - Convert a machine to regular grammar
  - Convert a machine to a regular expression

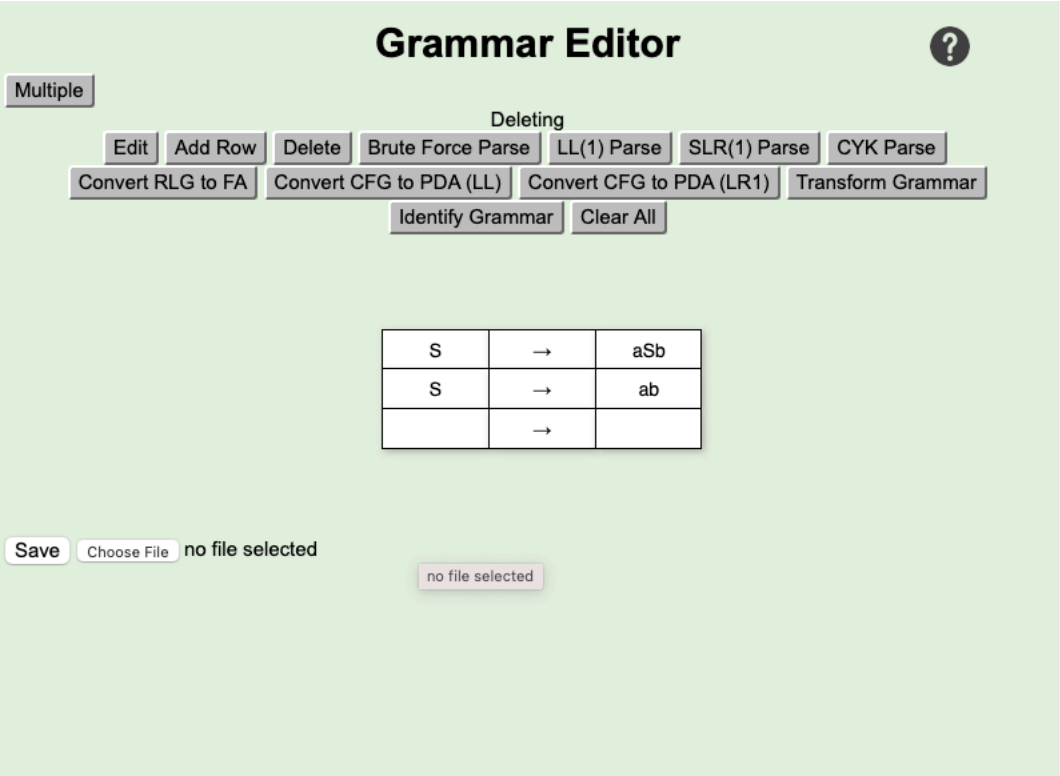


Figure 4.21: OpenFLAP editor for Grammars

- Grammar editor, see Figure 4.21. This editor allows students to write different grammars such as regular grammars and context-free grammars. The editor allow students to:
  - Brute force parse for a string to test if the string is a member in the language represented by the grammar or not.
  - LL(1) parsing
  - SLR(1) parsing
  - CYK parsing
  - Convert a regular grammar to an NFA
  - Convert a CFG to a PDA
  - Transform the grammar to Chomsky normal form

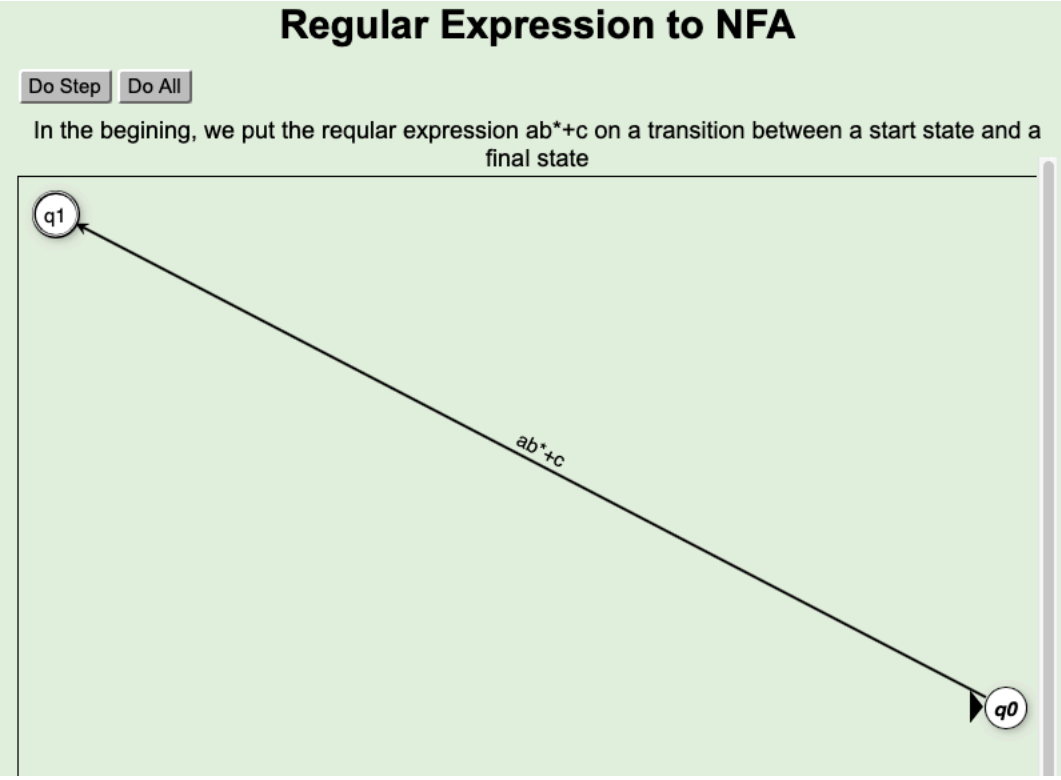


Figure 4.22: OpenFLAP Regular expression to NFA editor

  - Identify the grammar type

- Regular expression to NFA converter, see Figure 4.22
- PDA editor, see Figure 4.23. Allows students to create a PDA, test it on strings and convert it to a CFG.
- Turing machine (TM) editor, see Figure 4.24. Allows students to create a TM, and test it on strings

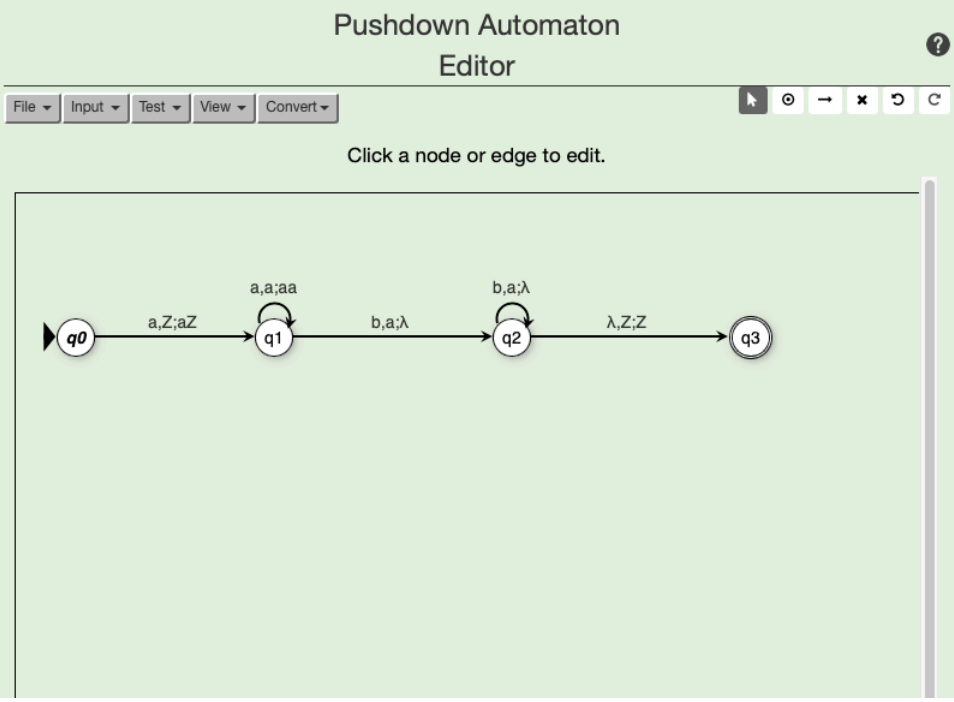


Figure 4.23: OpenFLAP editor for pushdown automata

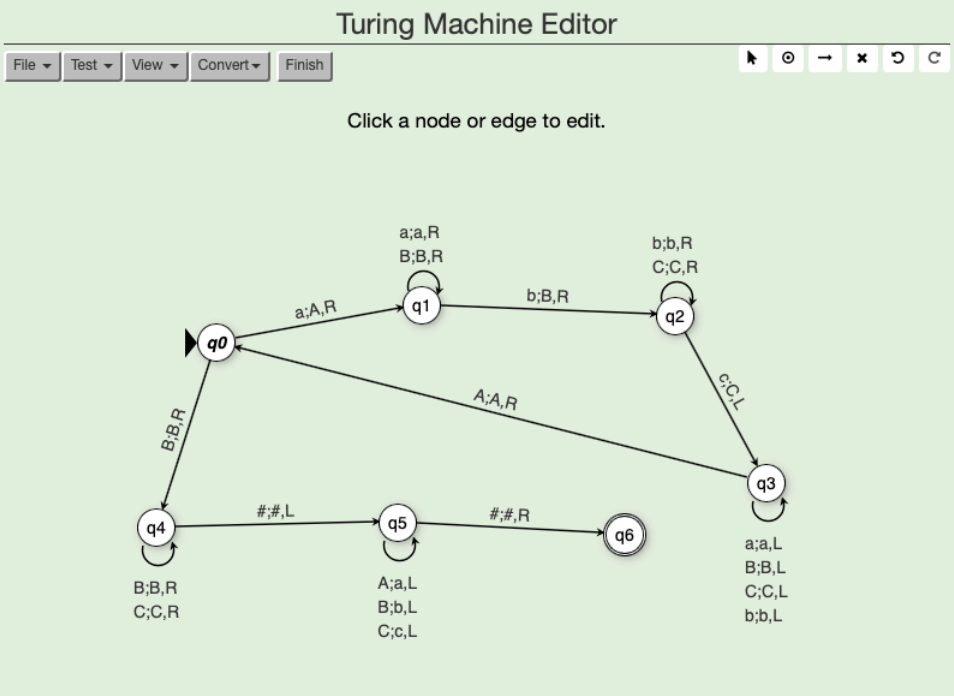


Figure 4.24: OpenFLAP editor for Turing Machines

Chapter 5

Programmed Instruction Support in OpenDSA

To present the required Programmed Instruction materials, we need a system to deliver and display the information and the question. The system should also have a mechanism to control when students can move forward, read and study the next piece of knowledge, and move backward, to review previously given information. Keeping this in mind, we found that OpenDSA has a suitable starting point called Slideshows. Standard OpenDSA Slideshows have a significant disadvantage that we need to address: students can freely skip any number of slides by using the "next slide" button or the "last slide" button. Since our primary goal is to follow the PI principles, we need to add a constraint that prohibits students from skipping the current slide. To achieve this goal, we extended the existing slideshow support library to support PI framesets. The frame is similar to the slide in that it has many of the same control buttons and space to give the information to students. However, there are differences between frames and slides.

- Satisfaction criterion. Each frame has its satisfaction criterion that allows a student to pass that frame. In other words, in each frame, the student has to do something to satisfy the satisfaction criterion. For instance, students may need to read a frame, answer a question, or modify a given graph.

- Forward button. Students can not use the forward button unless they satisfy the satisfaction criterion. However, students can use this button freely if they used the previous button to reread previous frames. In this case, students can move forward through frames until they reach one with an unsatisfied criterion.

- Last slide button. Students can not use the "last slide" button unless they fulfill the satisfaction criterion for all frames in the module. However, students can use this button freely if they go back to previous frames. In this case, clicking on the "last slide" button will redirect students to the first frame with an unsatisfied criterion.

- Questions space. JSAV slideshows do not have a satisfactory mechanism to present questions to students. The current version of slideshows can display a problem in a new iframe. HTML iframes allows embedding an HTML document. On the other hand, frames should have a mechanism to present a question on the same frame and take the answer to check its correctness.

- Storing student's progress. The frames system stores all student's progress. This allows students to skip all previously satisfied frames. This option will save students time as they are not forced to redo all the questions every time they reopen the book. So, students may work on a frameset over several days, and the frames system identifies for every frameset if the student can access specific frames inside the set or not.

The Frames system allows developers to create questions that generate random elements each time the student practices the same frameset. In some cases, the system can present different examples every time the page is reloaded. Suppose a student wants to study for the final exam. If he/she sees the same question or the same example that he/she already solved before, this will not help him/her practice effectively. The frames system randomly selects a different example from a pool of available questions and auto-generates frame questions.

We developed the frames framework to be flexible and easy for developers to create new framesets. We hope that others will build Programmed Instruction materials for their courses.

5.1 Frames questions types

Currently, the frames system supports a number of different types of questions

- Multiple choice questions (MCQ). The majority of frames questions are MCQ. For most frames, we do not intend the questions to be difficult. We simply want to keep the student engaged, and we feel that frequent easy to answer questions will do that. So, having an MCQ question allows students to find a way to escape from the question if they could not find the correct answer. This may cause some students to pass some questions without understanding the related information to the given question. To overcome this problem, we added support to frames questions to give students a message to tell them why their answer is correct / not correct. This message can give students hints if they do not answer the question correctly; see Figure 5.1. Some messages give students reinforcement or new information if they answer the question correctly/incorrectly (see Figure 5.1).

- Embed an exercise HTML iframe. In general, instructors can embed any HTML exercises in an iframe inside the frameset. For the Formal Languages course, instructors can ask their students to solve a proficiency exercise or an auto-graded exercise after finishing a frameset. For example, if students read a frameset about converting an NFA to DFA, the instructor can add at some point in the frameset an auto-graded exercise that students should answer to show their mastery of the topic. In [47] we

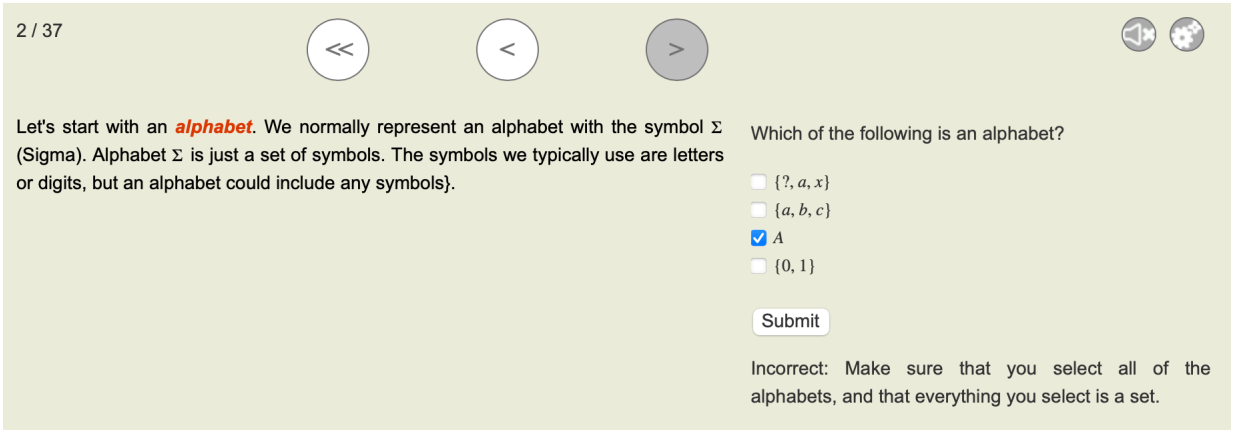


Figure 5.1: A hint to help students know why their answer is incorrect.

show the process for creating these types of exercises for the Formal Languages course.

- Free response questions. In this type of question, the students should answer a question that asks about a general idea or a summary of the topic at hand. If students understand the general idea, they can write the answer using some terminology. These questions only look for these common terminologies to allow instructors to know if students clearly understand the topic at hand or not.

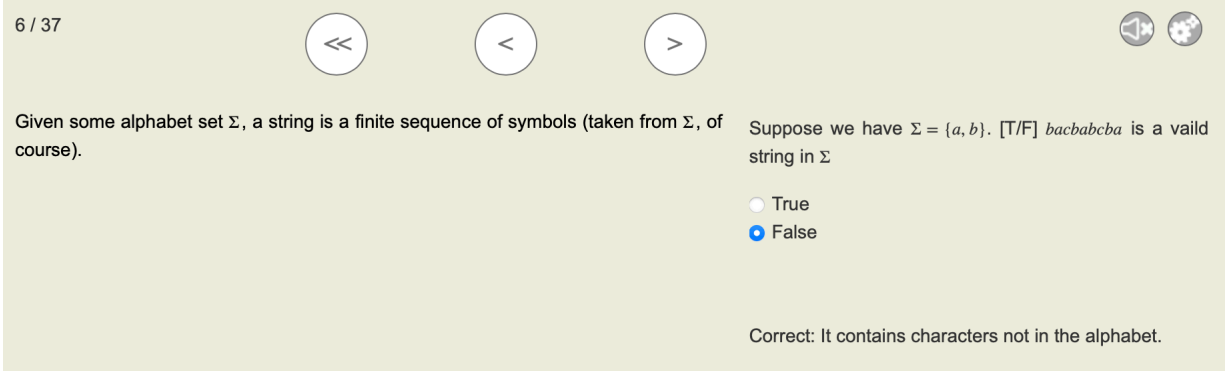


Figure 5.2: A hint to reinforce why the answer is correct.

5.2 Creating frame questions

Our ultimate goal is to develop the frames framework so it helps us and any researcher build Programmed Instruction materials. We believe that creating the frames framework will encourage the researchers to build Programmed Instruction materials for their courses. To do so, we created two different methods to create framesets.

5.2.1 Using the JSAV library

OpenDSA is a popular system that many instructors can use to create different online courses, especially in Data Structures and Algorithms. Thus, there is a good number of instructors who know how to create slideshows for OpenDSA. Creating OpenDSA slideshows requires the instructor to have two files for each slideshow. These files are a JS file for the visualization code and a CSS file that contains the style for these visualizations. Since the frame system is an extension to the slideshows, instructors will need to use the same two files and add a JSON file. The JSON file will contain the questions for the frameset. Figure 5.3 shows an example for a JSON object that represents a question for a frame that is shown in Figure 5.2. The object consists of

- the first line is the question identifier.
- "type" represents the question type. The value can be
  - "select": the question is an MCQ question with a single/multiple correct answers. The system determines if it is a single answer or multiple answers based on the value of the "answer" field.
  - "iframe": this option allows instructors to embed any HTML exercise inside an iframe.

```
"strings": {
  "type": "select",
  "description": "Given some alphabet set $\\Sigma$, a string is a finite sequence of symbols (taken from $\\Sigma$, of course).",
  "question": "Suppose we have $\\Sigma$ = $\\{a, b\\}$. Which of the following are valid strings?",
  "answer": ["$bababababaa$", "$a$"],
  "choices": ["$bababababaa$", "$a$", "$\\{aaa, bbb\\}$", "$bacbabcba$"],
  "incorrectFeedback": ["Strings are not sets, and a valid string can only have characters from the alphabet."],
  "correctFeedback":""
},
```

Figure 5.3: An example for JSON object that represent a question for a frame.

  - "text": text means that the question is a free response question type.

- "description": The value for this field is the frame information that the student should read and understand.

- "question": The value for this field is the frame question that students must answer to proceed.

- "answer": The value for this field is the correct answer that students must select. If there are multiple correct answers, the value for this field will be a list of correct answers.

- "choices": The value of this field is the possible answers that the students need to read and find the correct answer/answers.

- "incorrectFeedback": The value of this field is the message that displays when the student gives an incorrect answer.

- "correctFeedback": The value of this field is the message that displays when the student gives a correct answer.

To show the question inside the frame, instructors should call a function that takes the frame question identifier in the appropriate frame. Figure 5.4 show a JavaScript snapshot for the code that is executed to load the frame question from the JSON object.

5.2.2 Using Google Spreadsheets

To help instructors create framesets without dealing with the above details, we created an easy-to-use method to create all required files for the frameset from a customized Google Spreadsheet. We have added a button that converts the spreadsheet to the JS, CSS, and JSON files. Figure 5.5 shows an example of creating a frameset for deterministic finite automata (DFA). Each row creates a frame in the frameset. The frameset name will be the spreadsheet name. The sheet columns are

- "ref_name": is the identifier for the frame.
- "Hide": checking the button in any row means hiding the frame in the generated frameset. The benefit of this hiding is to allow instructors to debug their frames by hiding the complete frames and showing only the new frames that require more attention / debugging.
- "Type": The value for this column represents the type of information that the instructor wants students to see/read.
  - "code": This lets the instructor write JSAV code that will generate content in

```
$(document).ready(function () {
  "use strict";
  var av_name = "EquivFS";
  var av = new JSAV(av_name);
  var Frames = PIFRAMES.init(av_name);

  // Frame 1
  av.umsg("An :term:`equivalence relation` is an especially important type of relation. Relation $R$ on set $S$ is an equivalence
  av.displayInit();

  // Frame 2
  av.umsg(Frames.addQuestion("equivalent"));
  av.step();

  // Frame 3
  av.umsg(Frames.addQuestion("eqclass"));
  av.step();
```

Figure 5.4: An example for JS code that loads a question for a frame from the JSON file.

the frame.

  - "text": This means that the instructor can write the information text that students should read in the frame.
  - "select / IFrame / Free response": Determines the type of the frame question.

- "Description": Text displayed in the frame.
- "Question / Code": If the cell type is "code", then the content of this cell will be the JSAV code used to show the intended visualization. Otherwise, the cell content will be the question that will be displayed to students.
- "Answer": Represent the correct answer(s) for the frame question.
- "Choices": Represent the possible choices that students will read to select the correct answer(s).
- Export to OpenDSA: This button allows instructors to create the JS and JSON file for the frameset.

5.3 Auto-generating frames questions

Creating different examples with frame questions can be a tedious task for instructors. In the Formal Languages course, students study different algorithms like converting non-deterministic finite acceptors (NFA) to deterministic finite acceptors (DFA), minimizing DFA, or convert NFA to regular expression. Students need to see how different models work when they accept/reject a string. Formal Languages courses include models like NFA, DFA, PDA, and Turing machines. In addition to the time and effort required to create a single example for every algorithm and model, students will not enjoy studying from the book if they

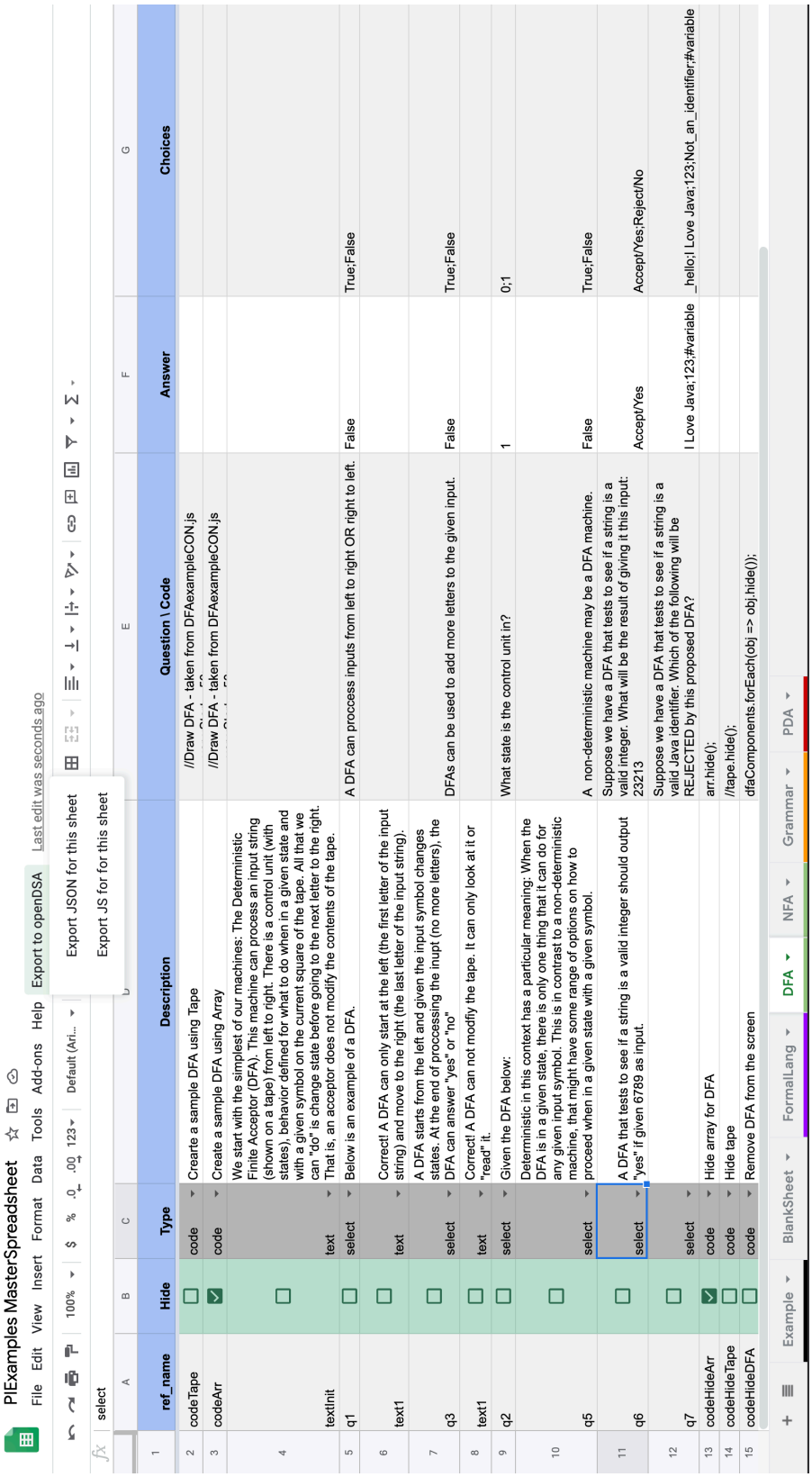


Figure 5.5: Example for generating a DFA frameset by filling a customized Google Spreadsheet

are required to see the same examples every time. To overcome these problems, we created a mechanism to auto-generate frame questions for all algorithms and models in the Formal Languages course. The result of this mechanism is that the Formal Languages eTextbook will include as many examples as the instructor wants without writing a single question for the generated frames. Integrating this mechanism with auto-generating Formal Languages models will yield a book with a different example every time students reload the eTextbook.

5.4 Frames checkpoints and mastery points

Students are required to solve frame questions to proceed in the frameset successfully. Every time students solve a question, the frames system stores their progress. This way, students can retrieve their progress if they closed their book or refreshed the module page and save their time and effort to redo all the questions they already solved.

The Programmed Instruction teaching method is based on the mastery philosophy. This means that students should earn points for mastering book sections and students can repeat a question until they get it correct. Our Programmed Instruction book stores students progress in every frameset. Once students reach the end of the frameset, the Frame system awards points for completing the frameset. Since students read all the frames and answered all the frameset questions correctly, this indicates that they mastered this part of the book and they earn points for their efforts.

Chapter 6

Methodology

We built the eTextbook by using the OpenDSA eTextbook system in a three-Phase process. In the first Phase, we created a fairly standard version of an Formal Languages textbook in OpenDSA, based on course notes from prior course offerings, paper assignments, and JFLAP. We created thirteen homework assignments, and three exams similar to those given in prior offerings of the course. We supplemented our materials with a popular textbook [39]. To construct the control data, we collected all student results in these homework assignments and exams during Spring 2018.

In Phase 2, we added to the existing Phase 1 OpenDSA eTextbook with many visualizations to all methods and algorithms in the Formal Languages course. We kept the same/similar assignments and exams questions and collected student grades for them. In addition, we have created a number of auto-graded exercises and proficiency exercises. We removed use of the Linz book from the course. Students used the Phase 2 book in Spring 2019, Fall 19, and Spring 2020. For homework assignments, we converted all questions that can be implemented as OpenFLAP exercises, and we kept the remaining questions, such as the pumping lemma questions, and added more questions to these assignments.

Our final Phase is the Programmed Instruction eTextbook. This book inherited all visualisations, auto-graded exercises, homework assignments, and exams from Phase 2 ebook. The main difference between this book and the Phase 2 book is that the Phase 3 book replaced most prose content with Programmed Instruction framesets.

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

Table 6.1: Visualizations (V), exercises (PE, AE), and figures (F) in our eTextbook

The rest of this chapter provides more details on the materials used in the various phases.

6.1 Phase 2 eTextbook - the Visualizations eTextbook

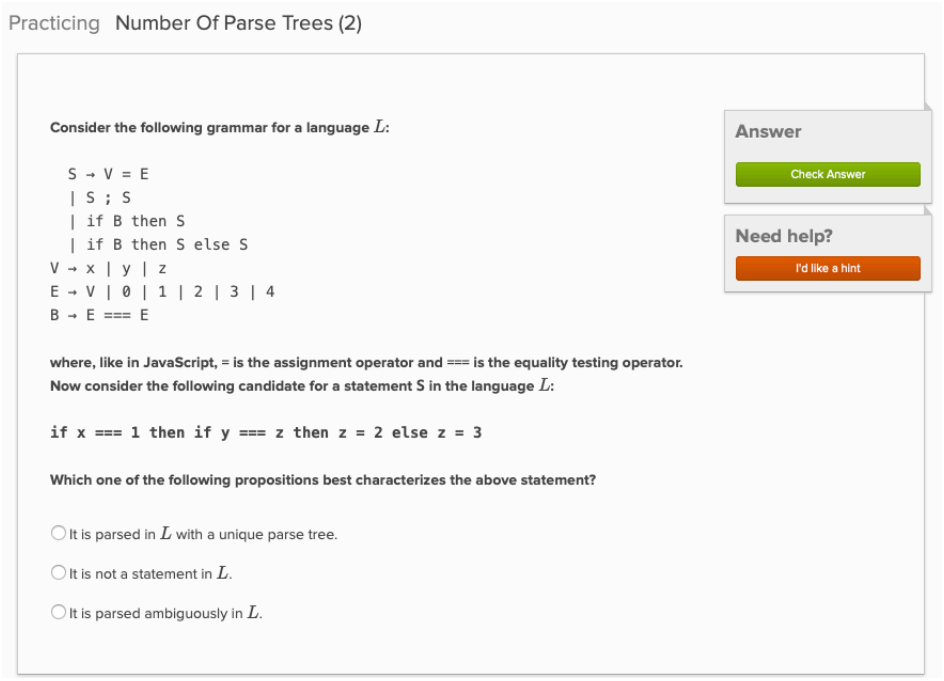
Previous studies [15] have found that students skip reading prose in eTextbooks, especially mathematical material that is hard to understand. Replacing prose with more visual slideshows has had success in motivating students to read materials on algorithm analysis. This result motivated us to create a pool of JSAV visualizations and exercises that reduce the amount of text needed to cover a given topic, and promote increased engagement.

The Phase 2 eTextbook includes 57 visualizations (V) in the form of slideshows, 71 exercises (including proficiency exercises (PE), auto-graded "create and test a machine" style exercises (AE)), and 49 additional static figures (F). Table 6.1 shows a breakdown for the topics covered by exercises, figures, and visualizations.

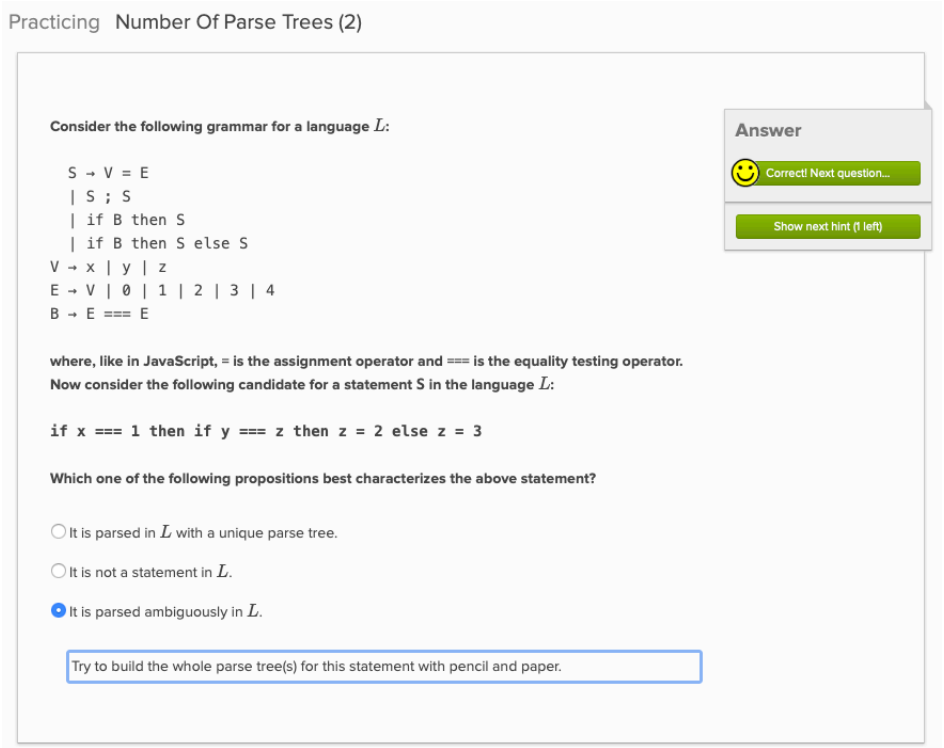
6.1.1 Exercises

In addition to OpenFLAP exercises (auto-graded exercises (AE) and proficiency exercises (PE)), the Phase 2 book contains a number of Khan Academy exercises. OpenDSA's core

framework provides tools to build another form of proficiency exercises, called Khan Academy exercises. Figure 6.1 shows an example. In our Phase 2 book, all Khan Academy exercises ask MCQ questions. Khan Academy exercises allows students to ask for hints to lead them for the correct solution.



((a)) OpenDSA Khan Academy exercise



((b)) Student answer after reading a hint

Figure 6.1: Example for Khan Academy exercises.

FLA courses require students to understand many algorithms, such as how to convert an NFA to an equivalent DFA. So we included 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 [17, 36].

6.1.2 Visualizations

We have implemented a series of visualizations to help students see how different models and their respective algorithms work. Figure 6.2 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 6.2 shows a slide from a demonstration for how a TM accepts or rejects strings in

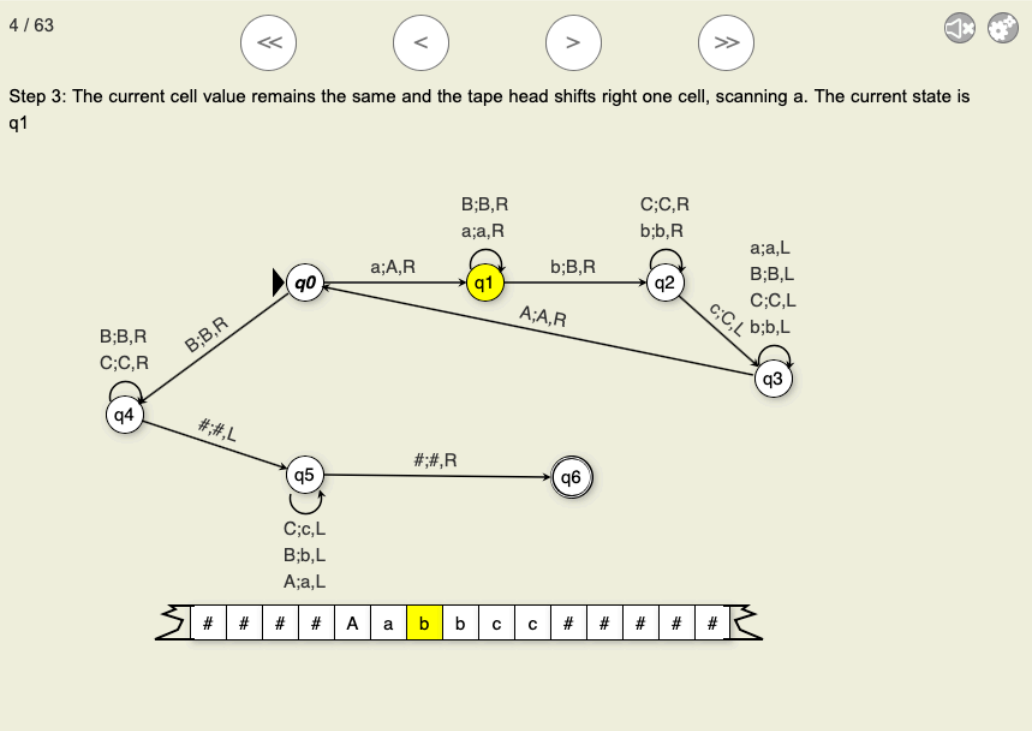


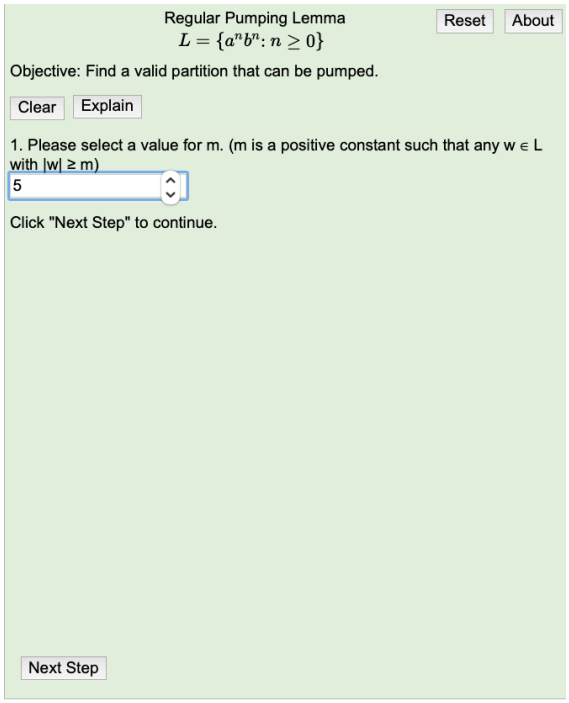
Figure 6.2: A Turing machine accepting a string

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.

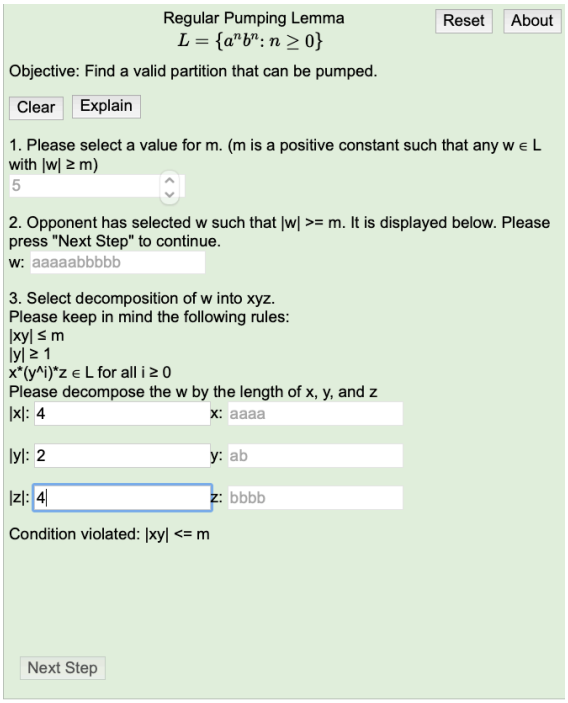
6.1.3 Pumping Lemma Game

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

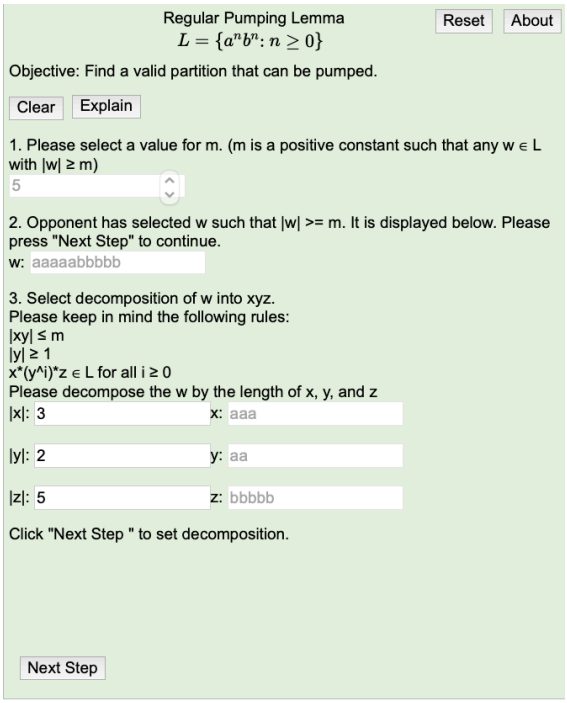
correctly distinguish an infinite number of strings as being in a language if looping through a subset of the states an arbitrary number of times always generates a string in the language. The pumping lemma can be viewed as an adversary game [39]. The adversary game presents the pumping lemma in the 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 6.3 shows partial sample game steps to prove that a language is not regular. First the student selects one of the thirteen languages and role. In this example, the student selected the language $\{a^n b^n | n \geq 0\}$ and selects to be the first player in the game. Next, 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.



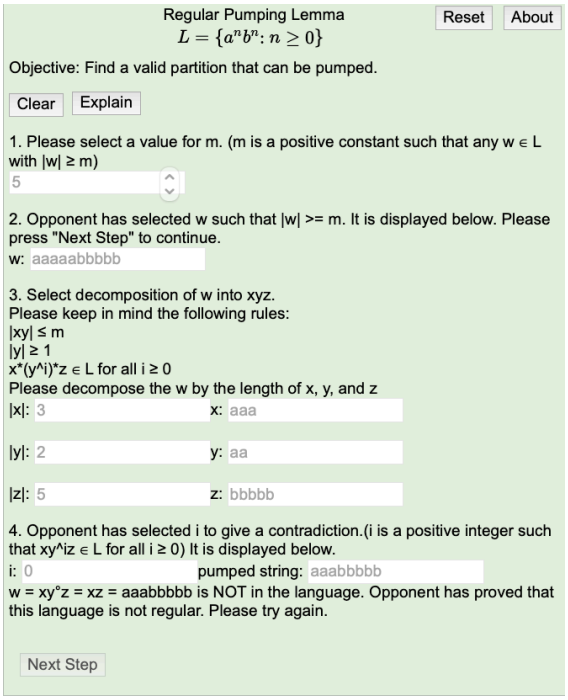
((a)) The student selected to be the first player, and selects the value of m.



((b)) Opponent picks a string such that $|w| \geq m$, then the student selects a bad decomposition of w to xyz that satisfies the constraints.



((c)) The student retries and selects a better decomposition of w to xyz that satisfies the constraints.



((d)) The opponent selects a value for i that makes $xy^iz \notin L$, so the language is not regular.

Figure 6.3: An adversary game to prove that a language is not regular.

6.2 Phase 3 - Programmed Instruction eTextbook

The benefit from developing the visualizations book is that we now have all required visualizations and exercises that we need to use in the Programmed Instruction book. To build the PI book, we refactored most of Phase 2 text into small sentences and we created questions related to these sentences. We have created over 2500 individual frames where most of the frames have questions. Each chapter in the Visualizations book is divided into some framesets. Each frameset describes a topic or provides an example about a previously described topic. Each frame presents some sentences with visualizations. Between these frames, we have added some questions about the current frame sentences. We tested the PI book in Fall 2020 and Spring 2021 semesters course offerings. We used the same exams as in Phase 1 and Phase 2, and the same homework assignments as in Phase 2.

Creating this number of frames and questions was a challenge in our study. There is no defined bank of questions available that contained questions we can use. We created questions about definitions, models specifications, and algorithms steps. Students have the ability to start any module in any order. The same is true about the framesets inside each module. However, students have to study any frameset in-order. The means that students must go through the whole frameset from the first frame to the last frame and answer the frames questions correctly to be able to move forward to the next frame.

6.2.1 Comparative Example

To illustrate the difference between the various versions of the book, we will use NFA to DFA conversion as an example.

Figure 6.4 shows the proof from the the Phase 1 book used by the control group to study

Theorem 3.2.1 and Proof

Theorem: Given an NFA $M_N = (Q_N, \Sigma, \delta_N, q_0, F_N)$, there exists a DFA $M_D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ such that $L(M_N) = L(M_D)$.

Proof: We can use an algorithm to convert M_N to M_D.

- $Q_D = 2^{Q_N}$
- $F_D = \{Q \in Q_D \mid \exists q_i \in Q \text{with} q_i \in F_N\}$

  Interpretation: A state q_D in M_D is final if **any** of the states from M_N in the subset that q_D corresponds to is final.
- $\delta_D : Q_D \times \Sigma \to Q_D$

Algorithm to construct M_D

1. Start state is $\{q_0\} \cup \text{closure}(q_0)$ (Note that "closure" of q_0 is a set of states defined as q_0 plus all states reachable from q_0 by λ transitions.
2. While can add an edge (that is, while missing a transition from δ_D)
   a. Choose a state $A = \{q_i, q_j, \ldots, q_k\}$ with missing edge for $a \in \Sigma$
   b. Compute $B = \delta^*(q_i, a) \cup \delta^*(q_j, a) \cup \ldots \cup \delta^*(q_k, a)$
   c. Add state B if it doesn't exist
   d. Add edge from A to B with label a
3. Identify final states.

   For a state in Q_D, if any of its base Q_N states are final, then it is final.
4. If $\lambda \in L(M_N)$, then make the start state final.

Figure 6.4: An example for what students see to study NFA to DFA conversions.

the conversion of NFA to DFA. This was followed by an example to convert an NFA to a DFA, shown in Figure 6.5. Control group students could also read a similar presentation in the Linz book used as a supplement that semester. As we see, students need to read that mathematical proof and try to read the example and do it by hand to test their understanding for the algorithm.

Example:

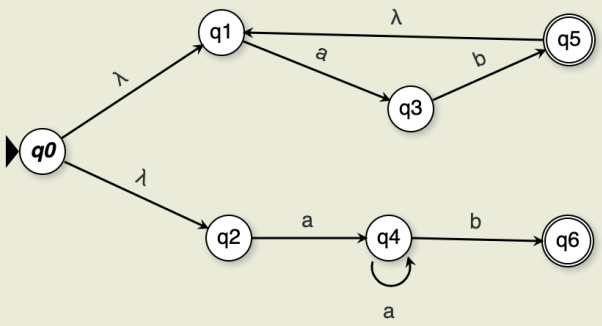


Figure 3.2.5: Another NFA to convert

Let's begin with the start state. Closure(q_0) in M_N is $\{q_0, q_1, q_2\}$. So this is the start state.

Now, keep repeating the steps of the algorithm:
While δ_D is not total, pick a missing transition and deal with it.

For example: From M_D state q_0, q_1, q_2, determine the subset of states that can be reached from any of those states on letter a. This would be the subset q_3, q_4.

Note

Do this conversion using JFLAP. You should get the following result.

Answer:

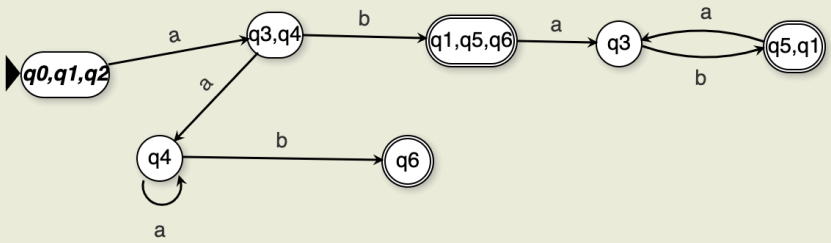


Figure 6.5: Phase 1 eTextbook example for NFA to DFA conversion.

In Phase 2, we added a slideshow that shows the example for students step by step, see Figure 6.6. The slide show allows students to go back and forward to see any step in the algorithm. It also contains useful descriptions about each algorithm step. Students see also the change that occurs after applying every algorithm step.

Thus, students still see the proof as presented in Figure 6.4 with a visualization about the steps of the algorithm to help them understand the algorithm. This means that students can skip reading the proof and jump to the slideshow as there is no control to prevent them from skipping the reading. They also have the freedom to skip the slideshows as well. After that, students should do an OpenFLAP proficiency exercises to apply these steps to convert a given NFA to a DFA as shown in Figure 4.13.

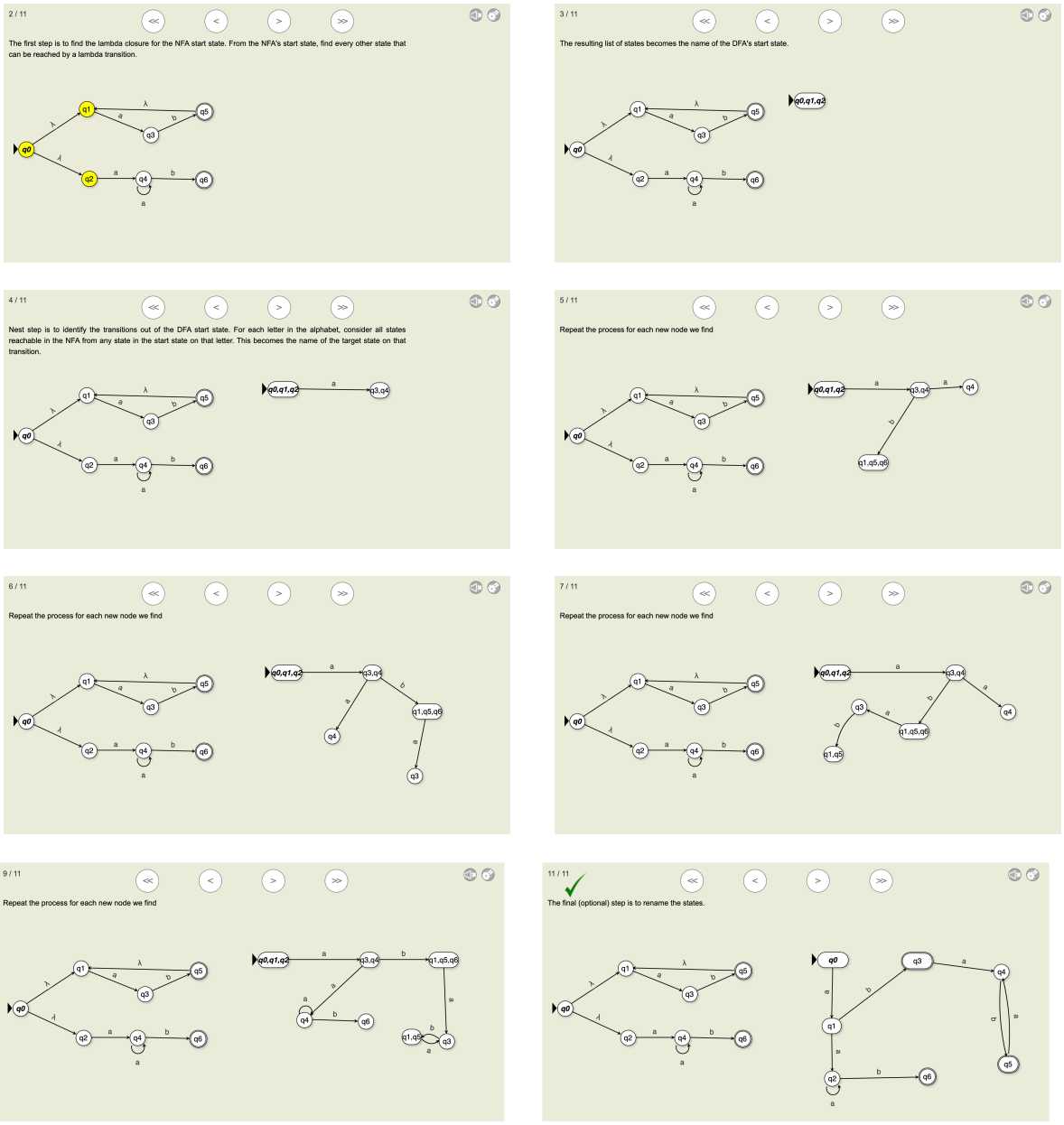


Figure 6.6: Slides from a visualization that presents an example to convert an NFA to a DFA

Finally, in Phase 3, students still see the proof as in the previous books, but it appears within a PI frameset. Thus, they must read it step by step, and answer questions as they go along. This is required to receive homework credit for this material. The frames present the step, an example, and ask students to apply the step on the example machine. The answers for the questions are simple MCQ questions with meaningful hints in case of correct/incorrect answers. This allows students to understand how the algorithm works, and test their understanding by solving these questions. Figures 6.7, 6.8, 6.9, and 6.10 show detailed steps for the NFA to DFA conversion PI frameset.

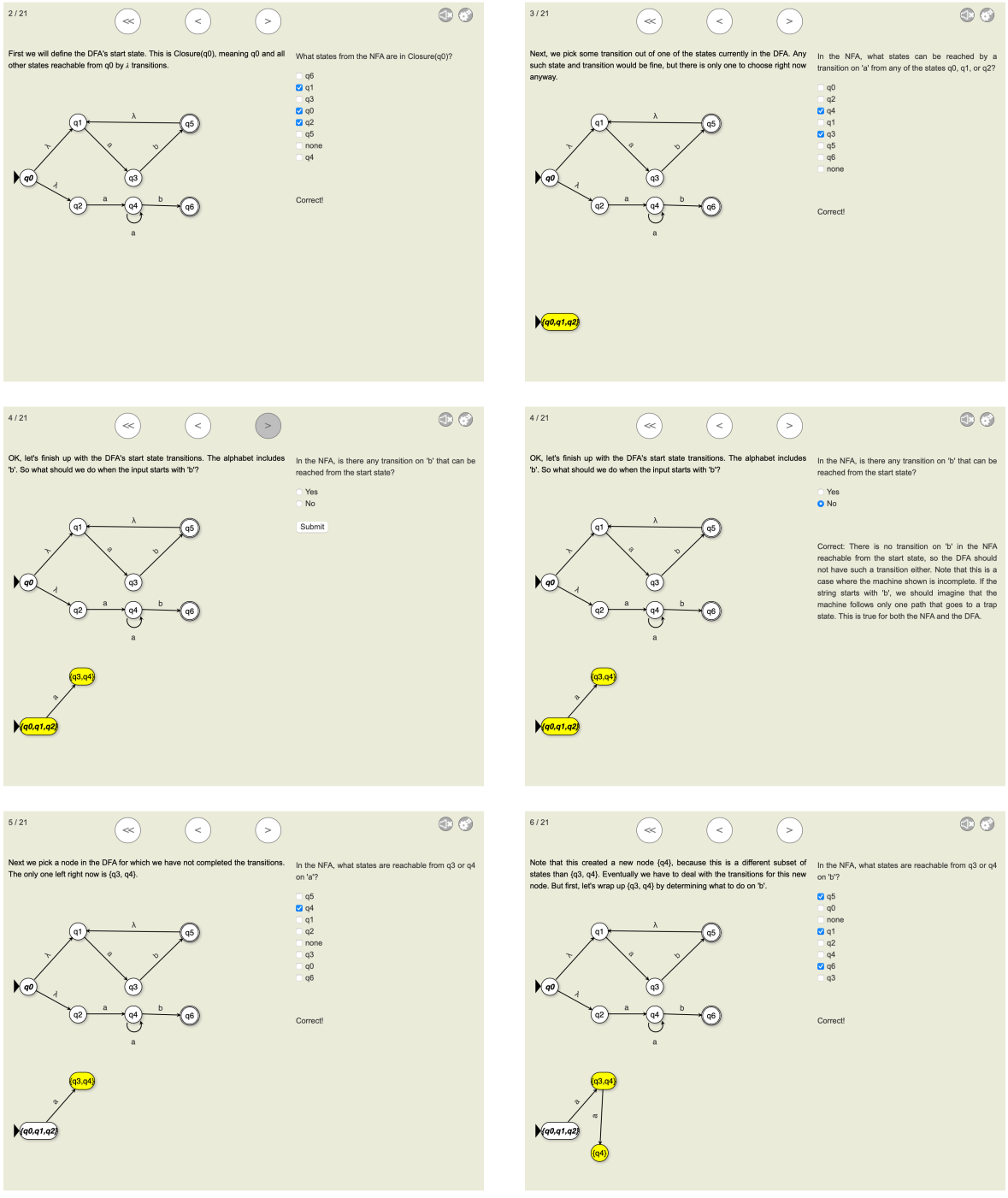


Figure 6.7: Frames that present an example to convert an NFA to a DFA - Part 1

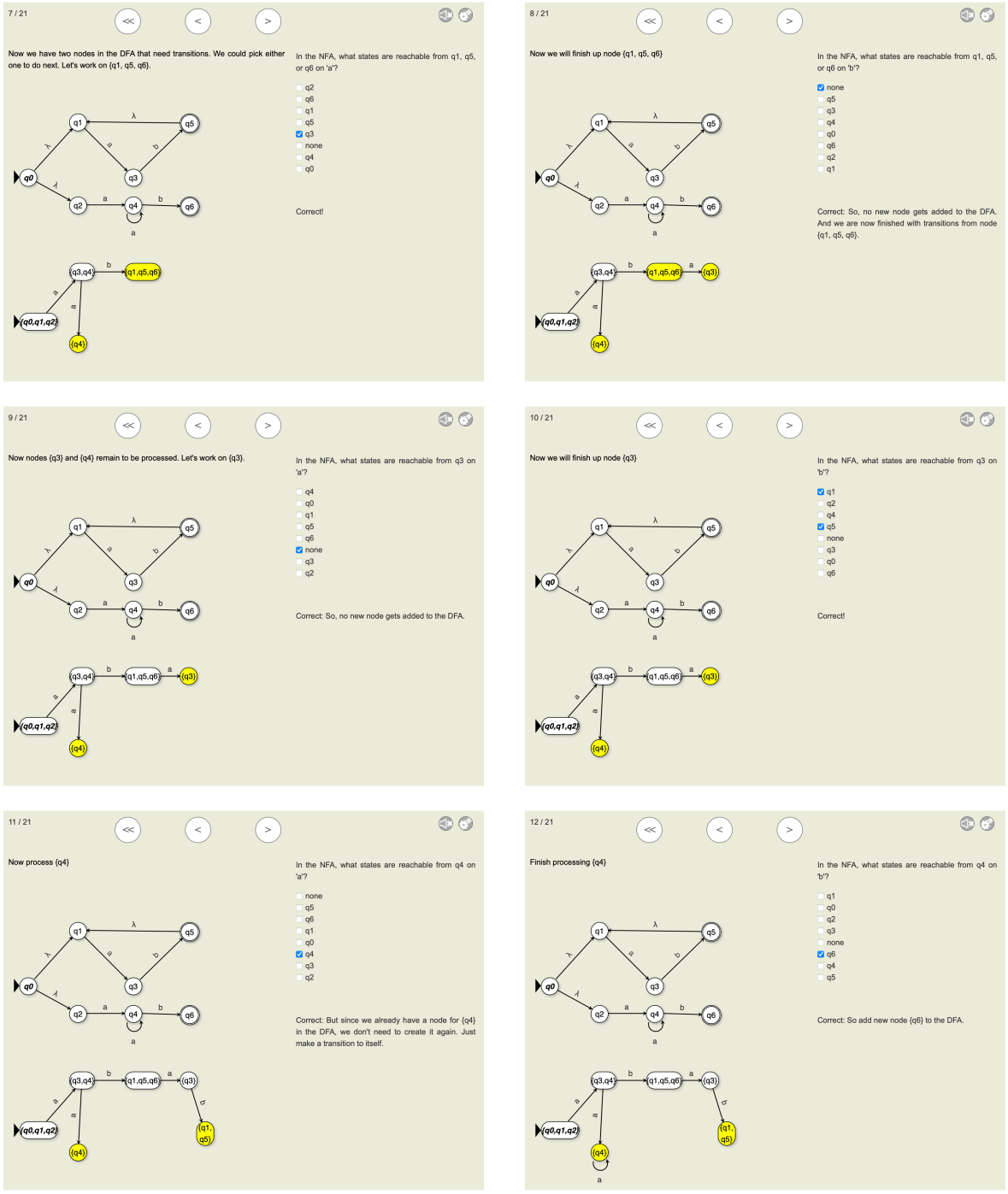


Figure 6.8: Frames that present an example to convert an NFA to a DFA - Part 2

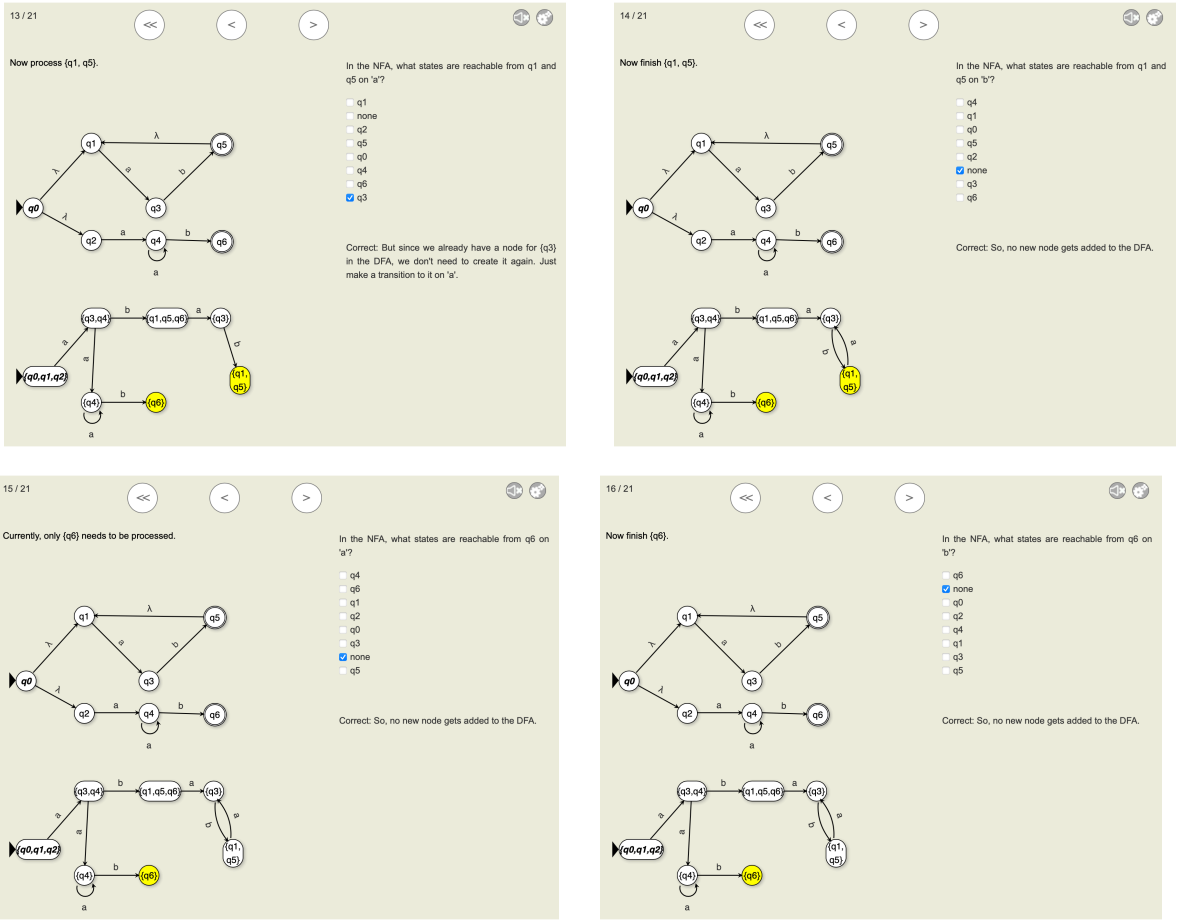


Figure 6.9: Frames that present an example to convert an NFA to a DFA - Part 3

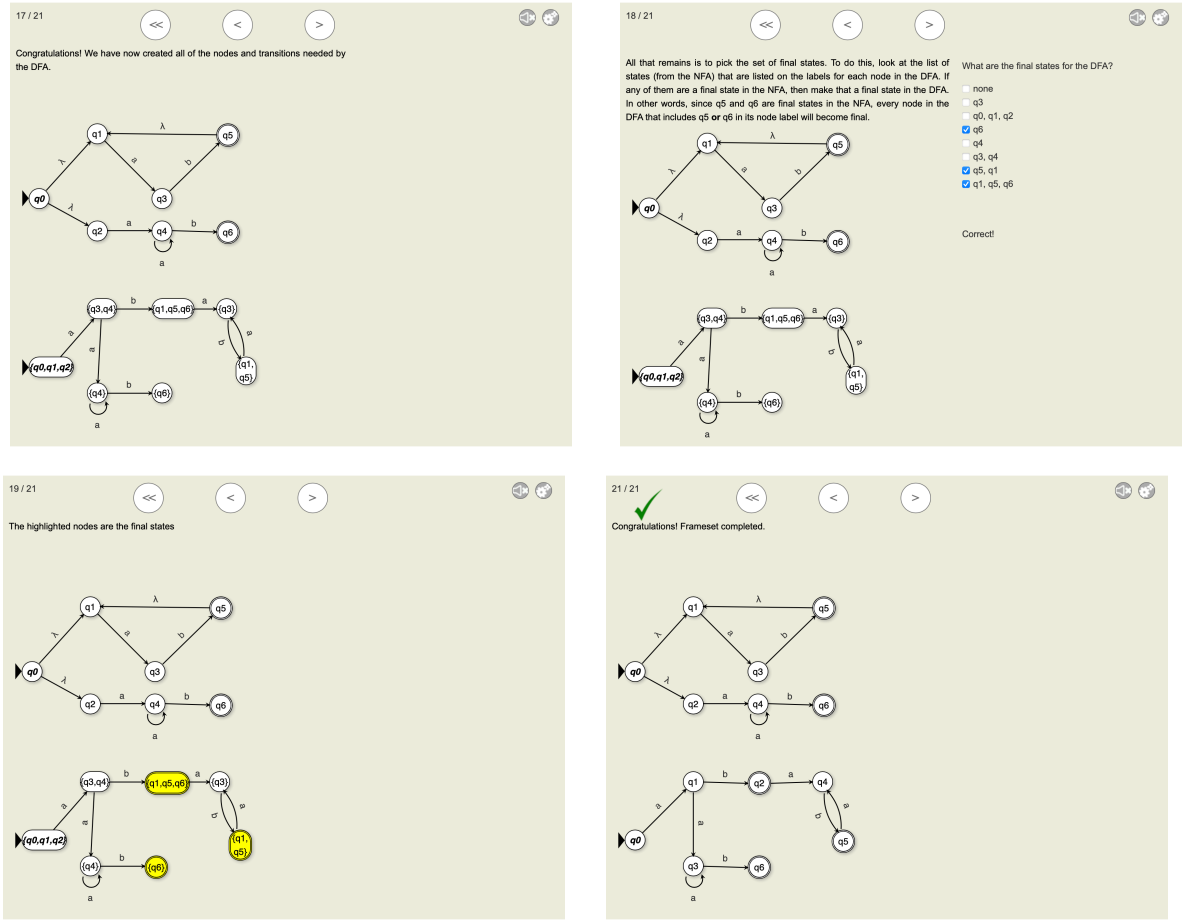


Figure 6.10: Frames that present an example to convert an NFA to a DFA - Part 4

6.3 Auto-Generating slideshows and Frame-sets

OpenFLAP has the ability to auto-generate slide shows and framesets. There are several topics that OpenFLAP can auto-generate, including:

- Show a model (DFA, NFA, PDA, or TM) and how that model accepts or rejects a string. In this case, the instructor can provide a JFF file for the machine along with some strings to test. OpenFLAP will generate a slideshow that shows students how the given model processes the input string, character by character. The visualizations will show how the machine will move from one state to another by applying the model transitions. In the end of the visualization, the machine will show if the string will be accepted or rejected.
- Apply an algorithm on a model or a grammar. In this case, the instructor can provide the model / grammar and OpenFLAP will apply the algorithm on that input. OpenFLAP will then generate the corresponding slideshow automatically and add it to the book. The slideshow will present the algorithm steps where each slide shows one step and presents some text to help students understand the algorithm.
- OpenFLAP can also auto-generate the model for the example automatically. In this case
  - Instructors provide a number of models and OpenFLAP will randomly select one of them to auto-generate the slideshows.
  - Instructors do not provide any model and OpenFLAP will randomly generate the model and auto-generate the slideshow.

  In framesets, the frames library will add to the auto generated slideshow some frames questions.

– In string acceptance/rejection, the frame system will ask students about the transition that will be applied given the current string input, the next state, or if the string will be accepted or not.

– In algorithms slideshows, the frame system can ask students about the result of applying an algorithm step, or select the next algorithm step that should be applied.

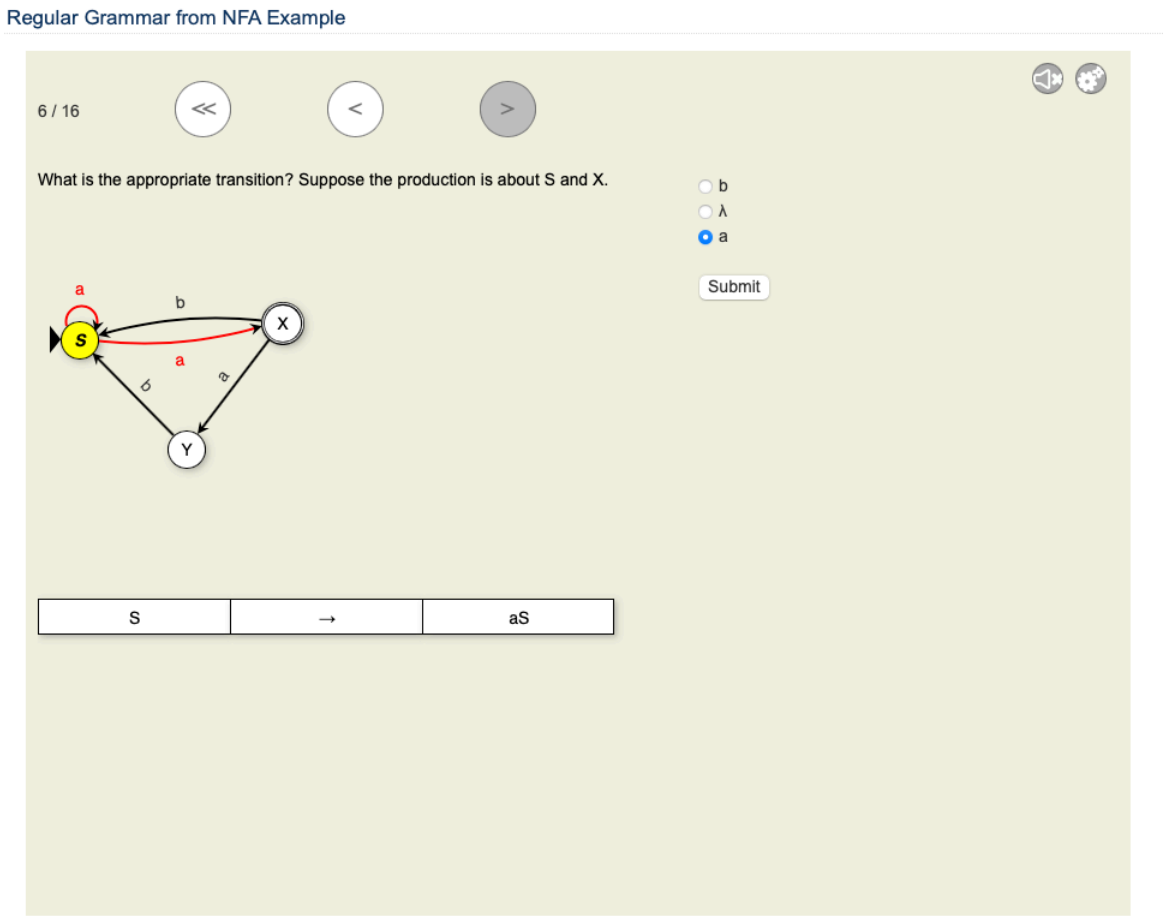


Figure 6.11: An example for an auto-generated slideshow with frames question. In this example, the instructor provided an NFA model. OpenFLAP applied the algorithm to convert the NFA to a regular grammar. Finally, the frames system auto-generated questions about the conversion algorithm.

With OpenFLAP exercises, OpenFLAP auto-generated slideshows, and frames auto-

generated questions, we provide a complete system that can help Formal Languages instructors to create course content. Figure 6.11 shows an example for an auto-generated slide show with auto-generated questions. The slideshow presents meaningful auto-generated sentences and the student is asked to answer a question about the current algorithm step. To see how easy it is to create this auto-graded generated slideshow with auto-generated question, Figure 6.12 shows the code that instructors need to run to generate the auto-graded slideshow and questions shown in Figure 6.11. As the figure shows, instructors only need to change the NFA to auto-generate the slideshows and the questions to a different example.

AV > PIExample > RegularGrammars > JS NFAToReExampleFF.js > ...

```
1  /*global Minimizer*/
2  $(document).ready(function() {
3      "use strict";
4      var av_name = "NFAToReExampleFF";
5      var arrow = String.fromCharCode(8594);
6      var av = new JSAV(av_name);
7      var url = "../../../AV/OpenFLAP/machines/FA/NFAtoRE.jff";
8      var arr = new Array(7);    // arbitrary array size
9      for (var i = 0; i < arr.length; i++) {
10         arr[i] = ["", arrow, ""];
11     }
12     var lastRow = 0;
13     var grammarMatrix = av.ds.matrix(arr, {style: "table", left: 0, top: 250});
14     // hide all of the empty rows
15     for (var i = lastRow + 1; i < arr.length; i++) {
16         grammarMatrix._arrays[i].hide();
17     }
18     var FA = new av.ds.FA({width: 300, height: 150, left: 10, url: url});
19     av.umsg("Suppose we need to convert this NFA to a Regular Grammar");
20     var FAtoGrammar = new FAtoGrammarConverter(av, FA);
21     av.displayInit();
22
23     //FAtoGrammar.convertToGrammar(grammarMatrix);
24     convertToGrammarWithQuestions(av_name, av, FAtoGrammar, grammarMatrix, {top: -10, left: -20});
25     //av.step();
26     av.umsg("The resulting Grammar is: ");
27
28     av.recorded();
29 });
```

Figure 6.12: Code that generates an NFA to regular grammars slideshow with auto-generated questions. Lines 7 defines the NFA that the instructor wants to show to the students. Lines 9-17 define the grammar area that will contain the generated grammar from the conversion algorithm. Line 20 calls an OpenFLAP function that converts the NFA to a Grammar and generate the slideshows that show the algorithm steps. Finally, line 24 calls the Frames function that takes the auto-generated steps to add the appropriate frames questions

Chapter 7

Evaluation

In this chapter, we describe our efforts to evaluate the effectiveness of our work with Formal Languages online materials. As mentioned earlier, we implemented two ebooks to teach students the Formal Languages course. One book uses visualizations, OpenFLAP auto-graded exercises, OpenFLAP proficiency exercises, and Khan Academy exercises. The second book inherited all components from the visualizations book but it replaces most text with a series of frames with questions. Our evaluation relies on:

- Collecting student feedback about the OpenFLAP tool: we surveyed the students to collect their feedback about OpenFLAP exercises (auto-graded exercises and proficiency exercises) in terms of whether they were engaging and helpful to them in understanding the Formal Languages concepts.

- Collecting student feedback about the Programmed Instruction ebook: we surveyed the students to collect their feedback about the idea of having frames of sentences followed by must-to-answer questions.

- Evaluating student performance: We used three exams to test students in different Formal Languages topics. Similar exams were used by each group pf students, so that the results were comparable.

Our eTextbooks cover topics typically taught in a FLA course. Since there are three exams,

we can think about the course as three parts, and each part is ended by giving the students an exam. The first part is about

- Reviewing some basic mathematical topics like set theory and mathematical proofs.
- An introduction to grammars
- Models for regular languages, this includes
  - Deterministic finite acceptor (DFA)
  - Non-deterministic finite acceptor (NFA)
  - Converting NFA to DFA
  - Minimizing the number of states for DFA
  - Regular expressions (RegEx)
  - Regular grammars
  - Equivalence between RegEx, regular grammars and NFAs
  - Closure properties of regular languages
- Identifying non-regular languages by using the pumping lemma
- An introduction to context-free languages, this includes
  - Parse trees
  - Context-Free grammars
  - String derivations
  - Grammar ambiguity

In the second part of the course, students cover

- Membership problem
- Transforming context-free grammars, this includes
  - Removing lambda productions
  - Removing unit Productions
  - Removing useless Productions
  - Transforming context-free grammars to Chomsky normal form (CNF)
  - Transforming context-free grammars to Grebach normal form (GNF)
- Push-down automata (PDA)
- Properties of context-free languages
- Identifying non-context-free languages using pumping lemma
- Turing machines
- Combining Turning machines

We also include material that might not be in all FLA courses about basic complexity and computability theory. Thus, the third part of the course is about Limits to Computing, this includes

- Reductions
- NP-Completeness
- NP-Completeness proofs
- Coping with NP-Completeness

- Countability
- Unsolvable problems

The evaluation experiment was performed during Spring 2018, Spring 2019, Fall 2019, Spring 2020, Fall 2020, and Spring 2021 for CS4114 Formal Languages and Automata at Virginia Tech. Spring 2018 students were our control group (Traditional book). Spring 2019, Fall 2019, and Spring 2020 students used our Visualization ebook. Finally, Fall 2020 and Spring 2021 students used our Programmed Instruction ebook. For all groups, we performed two evaluation activities. First, we surveyed the students to collect their feedback on the Open-FLAP exercises and visualizations (Spring 2020) and the Programmed Instruction ebook (Fall 2020 and Spring 2021). Secondly, we compared the student performance on the course exams.

7.1 Evaluation Protocol

In this section, we outline our protocol for Formal Languages ebook evaluation in terms of student satisfaction, and performance.

1. Research questions. We have three main research questions that we have answered through the evaluation.
   (a) What feedback do students give regarding their experience with the Programmed Instruction book?
   (b) How does performance on exams compare for the three treatments (control, visualizations, and Programmed Instruction)?

(c) How is performance on exams affected by attempting to solve Programmed Instruction questions, or spending more time studying from the books?

2. Participants. Our subjects are (a) 44 students who took CS3114 at Virginia Tech during Spring 2018, (b) 271 students who took the same course during Spring 2019, Fall 2019, and Spring 2020, and (c) 135 students took the same course during Fall 2020 and Spring 2021.

3. Materials.

   (a) In Spring 2018, Shaffer taught the Formal Languages course, using a version of the eTextbook that had prose written largely from either prior materials written by Shaffer (for Turing Machines, NP-Completeness, and computability), or course notes written by Rodger and Heath for other Formal Languages topics. Mohammed was the TA fro the course in that semester. Some of the material on Turing Machines and computability theory included a few visualizations from prior work. We also supplemented with a popular Formal Languages textbook by Linz [39], and we made use 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 at least a decade. We consider this our first control group for comparison.

   The second group consists of 271 students who took the same Formal Languages course over three different semesters, Spring 2019, Fall 2019, and Spring 2020. All of them used the Phase 2 OpenDSA eTextbook for Formal Languages as the main source of instruction, and were all taught by Mohammed. OpenFLAP replaced JFLAP for Phase 2.

The third group consists of 135 students who took the same Formal Languages course over two different semesters, Fall 2020 and Spring 2021, taught by Mohammed. All of them used the Phase 3 OpenDSA eTextbook for Formal Languages as the main source of instruction.

(b) To evaluate the Formal Languages eTextbook in terms of student learning, all groups were given a set of questions as part of their three exams. Each exam tested students on different parts of the course. Some questions are related to the areas affected by the Visualization book (that is, new visualizations and new interactive exercises). Other questions are on parts unchanged in the Visualization book and changed in the Programmed Instruction book only. These items form the basis of our performance comparison between all groups. The same person (Mohammed) graded exams for all groups to minimize inter-rater reliability problems.

(c) We gathered students' opinions about the OpenFLAP exercises and the Programmed Instruction materials through a survey at the end of the Spring 2020, Fall 2020, and Spring 2021 semesters. We asked students to evaluate the Frames in terms of how useful they were in helping them understand the Formal Languages concepts presented in the course.

4. Procedure. To evaluate student performance, we compared the groups based on the same set of exam questions. The same person graded exams for both groups to minimize inter-rater reliability problems.

5. Analysis. To make inferences about our hypotheses, we compared all groups according to the exam grades. We decided to use non-parametric tests (Mann-Whitney) with a 5% significance level. We chose this test to avoid the normality assumption that should be met if we would apply a parametric test (ANOVA) and mitigate the effect of unbalanced sample sizes of our three groups. It was stated in [42] that the Mann-

Whitney test is robust to unbalanced sample sizes.

7.2 Collecting Student Feedback

To gauge student satisfaction, we offered student surveys at the end of the intervention semesters. The survey solicited students' satisfaction with OpenFLAP exercises and Programmed Instruction ebook materials regarding whether they helped with understanding the Formal Languages course.

7.2.1 OpenFLAP exercises

To collect students' satisfaction with OpenFLAP exercises (auto-graded exercises and proficiency exercises), we asked them clearly to express their opinions about each type of OpenFLAP exercise. Since OpenFLAP exercises exist in both ebooks, we asked the students about the exercises at the end of interventions semesters. 70% responded to the surveys in total. The majority of the surveyed students report that all exercises are helpful for them to understand and practice the Formal Languages topics. Table 7.1 showed the students' responses in detail when they were asked to answer the question: "Much of the homework involved completing auto-graded exercises, where you had to show the steps for an algorithm build a machine or grammar. For each of the following exercise types, how well do you believe that these exercises helped you understand the course contents?". Students select whether the exercises are very unhelpful, somewhat helpful, neutral, somewhat helpful, or very helpful for each exercise type. The table shows that in all OpenFLAP exercises, most students select that the exercises are beneficial for them.

When we asked the students if the Formal Languages books need more exercises, 41% of the

Exercises	Very Unhelpful	Somewhat Unhelpful	neutral	Somewhat Helpful	Very Helpful
Writing Grammar Exercises	1%	1.9%	4.8%	33.3%	58.6%
Grammar Transformation	1%	4.8%	9.5%	38.6%	46.2%
Building DFA or NFA	1%	1.9%	5.7%	21.4%	70%
DFA Minimization	2.4%	4.3%	11.4%	30%	51.9%
NFA to DFA	1%	2.4%	10.5%	31.4%	54.8%
Building PDA	1%	1.9%	10%	24.8%	62.4%
Building Turing Machines	1%	5.2%	11.4%	27.6%	53.8%
Khan Academy	1.4%	5.2%	24.3%	39.5%	38.6%

Table 7.1: Percentage of the surveyed students satisfaction about the impact of different exercises on their learning for the Formal Languages courses

students reported that they need more exercises to practice course algorithms. 50% of the students believe that the available number of exercises is enough. 9% of students believe that there are more exercises than needed.

7.2.2 Programmed Instruction

At the end of the Programmed Instruction semesters, we asked students to answer the question: **Much of the content was presented as programmed instruction "frameset" slideshows, where many of the slides had a small question to answer. Which do you think helps your learning more? Presenting this material using the framesets with questions, as a series of slides with no questions, or as regular prose?**. 70% of students preferred to have a Programmed Instruction eTextbook, 23% preferred to have slideshows only, and 7% preferred the traditional text. Figure 7.1 shows the percentage of students selection between Programmed Instruction frames, slideshows, or traditional text.

Here is a sample of student answers about "The reason to select the Programmed Instructions Frames"

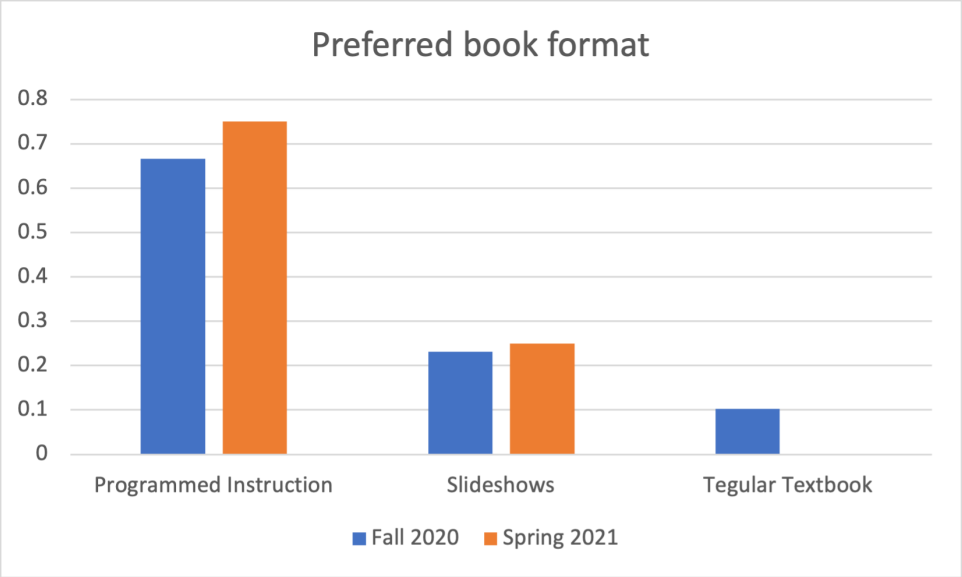


Figure 7.1: Student choices for the type of instruction that affected their learning more in the Formal Languages course.

- *It forces me to pay attention, something that is occasionally hard to do with online learning.*
- *It forces me to conceptualize the content. With math in general, being asked a question and hearing/answering helps me learn more than reading text.*
- *It allows the student to engage more with the questions step by step.*
- *I think the questions helped me reinforced my knowledge, validating what I learned was correct.*
- *Having to answer questions makes me read it more thoroughly*
- *Doing the course as programmed instruction framesets helped reassure my understanding of the material as I completed it. The questions force you to interact and actively read versus just skim. They act as quick self-checks to verify that I really understand something.*

- *It is easy to read something and move on, but the questions reinforced the concepts in my mind by making me reiterate what I just read.*

- *More interaction and eliminates the possibility of skimming through an entire chapter, however I think if you get a frame question wrong a few times it should let you continue as I was discouraged from completing a chapter in one sitting if I was stuck on a question.*

- *I don't mind either way, but I feel like when there are small questions in the slides, it gets me thinking about how to solve it. Since we are on the current topic, it helps me understand what it is about when thinking of a way to answer the question.*

- *The more engagements you have with students the better. Death by PowerPoint with no student interaction is not a great way of teach but too many teachers teach it this way. Honestly, it's lazy and a waste of time. But even if a teacher where to use slides, at least have stopping points to interact with students and see how they are absorbing the material.*

- *Going through the frameset's questions require me to be engaged in thinking about what's going on in the information. However, I do feel like at sometimes some of the questions were redundant or unnecessary. So there must also be a balance of questions, and it'd be more efficient to only insert questions in strategic parts of the slides.*

- *To be honest, having the questions forces me to actually understand the material to progress.*

- *Slides with questions force me to think about the answer and either develop or correct my thinking about the problem for myself. It becomes less about memorizing baseless facts and more about developing an intuition.*

On the other hand, when we asked students about "The reason to select slideshows / normal text against Programmed Instruction Frames"? we got a number of useful comments that can help us improve our Programmed Instruction book. We got 21 students responses to this question. These responses can be categorized into four main problems that they found while studying from our Programmed Instruction book.

- Breaking the information into frames instead of having it as a whole. Ten out of the twenty one responses were about disliking the idea of breaking the information. Some students prefer to have the information as whole first to get the big picture of the topic. After that, they accept to have the frames that asks questions about the topic. The responses were

  - *I liked the interactive textbook more when it was walking me through a process and examples. I didn't like it as much for general information because I kind of just want all the information laid out in front of me. It makes note taking a bit cumbersome.*
  - *The questions in the programmed instruction framesets didn't help me learn and made it harder to skip through the slide to reference information on a topic further down in the set or go back to a topic in the set.*
  - *The notes were not very detailed. Everything was not at one place. It will be much more easier if there was just one source to refer to*
  - *I recall being blocked from progressing with the slideshow until entering the correct answer. This often led to frustration. I'd prefer to just learn the material, then answer the questions after.*
  - *I liked to switch back and forth between slides, reading through examples shown there. As I am trying to understand something, I don't like being confused by*

being asked clarification questions with wrong options. I like those to come after I have gained understanding.

- *I want to have all available information to review before and after lecture in ONE area when possible. I don't want to go to ODSA, go back to a slide, review a question I don't remember during lecture, then search through lecture to find that question.*
- *I prefer without the questions so that I can easily skim through the slide to find the info I need without answering 10+ questions.*
- *I like text because it's easier to scan and find the information that I'm looking for. I find that I learn most quickly by reading text and generally find "improvements" on regular textbooks unhelpful.*
- *It's easier to find what I am looking for when reviewing in the traditional format than having to click through dozens of slides*
- *I can understand more if my reading isn't Interrupted.*

- Asking for better user interface and better quality questions. Six students mentioned that they need a better system interface or better questions for the frames. Their responses are

  - *The UX thing it takes way to many clicks to answer the questions. The slides should auto advance after you click submit instead of forcing the user to click the next button.*
  - *While it isn't selectable as an option, I believe a combination of slideshows without questions and ones with questions is optimal. I believe in the book there were perhaps too many questions by way of the Programmed Instruction framesets and felt like it encouraged me more to just push through it rather than stop and think*

about the questions and answer them carefully, and it may have been better to ask less questions but questions with more quality.

- *I like the slide demonstrations, but too many of the slides with questions had questions that made things impossible to progress. Either because I did not know the answer or the answer was incorrect and too much brute force would be required to guess.*
- *There were times where I would get stuck and literally just couldn't study anymore because I couldn't get past one question. The embedded question/frameset structure also makes it hard to search through the book and find exactly what I'm looking for.*
- *Regular prose can be long and hard to read, while many of the questions in the PI felt more like time wasters than meaningful questions that forced me to recall something about the information.*
- *Some of the questions doesn't make much sense and the reading didn't help.*

- Preferring the traditional book. Three responses were mentioning that some students do not like the idea of interrupted learning style. They need to not answer questions during their study process. Their responses are

  - *I would prefer slideshows without questions because there were times where I felt frustrated that I couldn't skip to a part of the textbook without having to redo the questions.*
  - *I found that I didnt really read the questions, I just kept guessing each answer until I got it right to move on*
  - *In the programmed book, I found myself needing to spend 15+ minutes looking for one specific fact I needed for the homework.*

- Asking for getting the correct answer after some incorrect attempts. Two students mentioned that their problem with the Programmed Instruction book is that they need to see the correct answer after some attempts or seeing the correct answer when they attempt to reread the same material again. Their comments are

  - *The programmed book offers no help if you can't solve the questions, so if you get stuck you can't finish the reading. The whole point of the reading is to learn and if you can't finish reading the book because you don't ALREADY know the material you're supposed to be learning about, the book is useless; which it is.*

  - *Because I stuck with questions sometimes, and I could not step to the next part until I figured it out which is time-consuming sometimes.*

7.3 Evaluating Student Performance

Measuring student learning gains made with technology is a difficult endeavor [16]. However, we believe that introducing visualizations, OpenFLAP exercises, and Programmed instruction frames to OpenDSA will help greatly in enhancing the effectiveness of the modules. The reason behind this belief is stated in Carroll's time-on-task hypothesis, that the longer a student spends engaging with the learning material, the more opportunities the student has to learn [6]. In addition, recall the engagement taxonomy, our OpenFLAP exercises, and Programmed Instruction frames satisfy five out of six levels of engagement. The only missing level of engagement that our books and tools do not satisfy is the presenting level, requiring students to present their solutions for the exercises.

To evaluate students performance, we created 3 exams. Each exam tests students in a subset of the course topics. In general, the exams test students on

- Regular languages which includes
  - Regular expressions
  - Deterministic finite automata
  - Non-deterministic finite automata
  - Closure properties for regular languages
- Context-Free languages, which includes
  - Parse trees
  - Writing context-free grammars
  - Grammar ambiguity
  - Closure properties for context-free languages
  - Pushdown automata
- Turing machines and universal Turing machines
- Pumping lemma
  - Pumping lemma for regular languages
  - Pumping lemma for context-free languages
- NP-Completeness and Reductions
- Countability
- Halting Problem
- Languages Hierarchy to test students knowledge about how to identify the accurate class for different languages.

Groups	Exam 1		Exam 2		Exam 3	
	Mean	Median	Mean	Median	Mean	Median
Control	74.4	75	64.1	66.5	98	100
Visualization	74.7	77.5	70.7	71.3	95.4	97.5
Programmed Instruction	**84.8**	87	**80**	81.5	**119.4**	119.7

Table 7.2: The Mean and Median for all groups and all exams. Exam 1 and 2 are scored out of 100 points, Exam 3 is scored out of 150 points.

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
Exam 1	44	271	135	74.4	74.7	84.8	2	57.96	<0.0001
Exam 2	44	271	135	64.1	70.7	80	2	63.122	<0.0001
Exam 3	44	271	135	98	95.4	119.4	2	83.187	<0.0001

Table 7.3: Kruskal–Wallis test for each exam. The result shows that there is a difference between exam means

- Set theory questions.

To reduce the variability in our study, the same instructor graded all the exams over all semesters in the assessment period.

7.3.1 Comparing the overall exam grades

Table 7.2 shows the mean and median for each of the three exams against each of the three treatment groups. Since we have three groups, we applied the Kruskal–Wallis test to see if there is a difference between groups' scores. If so, we applied multi-comparisons with adjusted p-value using the Benjamini-Hochberg (BH) [3] method to decrease the false discovery rate.

For each Exam, we found that there is a difference between the grade means across the three groups with p-value < 0.0001, see Table 7.3. Table 7.4 shows the result of pairwise comparisons between the groups in each exam.

Exam 1	Cont.	Viz.
Viz.	0.745	-
PI	0.0001	0.0001

Exam 2	Cont.	Viz.
Viz.	0.007	-
PI	0.0001	0.0001

Exam 3	Cont.	Viz.
Viz.	0.61	-
PI	0.0001	0.0001

Table 7.4: Pairwise comparison between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values for Multiple Comparisons [3] across all exams. The differences are significant for the control group against the visualization group for exam 2, between the control group against the PI group for all exams, and between the visualization group against the PI group for all exams.

7.3.2 Comparing grades per concept

It is clear from Table 7.4 that the Programmed Instruction group students have significantly better scores on the whole set of questions than the control and visualizations students. However, having no significance between the Visualization group in exams 1 and 3 made us think about a deeper comparison between the groups. Exams 1 and 3 have questions about topics that are not presented differently in the control and visualizations books. This supports that the Visualization ebook significantly impacted the students in the topics with visualizations and OpenFLAP exercises only because topics that remained the same showed no improvement in score, while those that had visualizations or exercises had improvements.

Another reason to look at a deeper comparison between the groups is that maybe the significance comes from some questions only in these exams. Their significance affected the overall grade differences. Thus, we grouped all questions related to a topic together and compared students' grades across all groups.

- **Regular languages**. We compared students grades in all regular languages related topics. These questions include
  - Writing regular expressions
  - Definitions. This includes MCQ and essay questions about regular languages, i.e.

Groups	Regular Languages Topics							
	RegEx		Definitions		Hierarchy		Overall	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Control	7.2	7	17.7	19	7.5	8	32.4	34
Viz	8.4	9	15.5	15.5	7.3	7	31.3	32
PI	**8.7**	10	**17.8**	19	**9.7**	10	**36.3**	37

Table 7.5: Mean and Median for regular languages concepts across all groups

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
RegEx	44	271	135	7.2	8.4	8.7	2	18.249	0.00011
Definitions	44	271	135	17.7	15.5	17.8	2	43.609	<0.0001
Hierarchy	44	271	135	7.5	7.3	9.7	2	39.485	<0.0001
Overall	44	271	135	32.4	31.3	36.3	2	51.829	<0.0001

Table 7.6: Kruskal–Wallis test for regular languages topics. The result shows that there is a difference between topics means

define a DFA, or list the difference between DFA and NFA.

– Determine if a language is regular or not without using the pumping lemma

Table 7.5 shows the mean and median between all groups grades in regular languages questions. As we see, the Programmed Instruction group has higher mean in almost all questions except for the definitions. The visualizations group has higher mean in questions that are affected by the exercises and visualizations such as writing regular expressions.

We started by testing the overall grade for this concept and compared it across all groups. Using the Kruskal–Wallis test, we found a difference between the means (with $pvalue < 0.0001$), see Table 7.6. Using the multi-comparisons with adjusted p-value using the BH method, we found that, in general, there is no significant difference between the Visualization group and the control group. On the other hand, the Programmed Instruction group has a significant difference from the other groups. When we looked at the topics grades, we found that the Programmed Instruction group has a

Total	Cont.	Viz.
Viz	0.26	-
PI	0.0007	<0.0001

Regular expression	Cont.	Viz.
Viz	0.0015	-
PI	0.0004	<0.018

Definitions	Cont.	Viz.
Viz	0.0003	-
PI	0.7	<0.0001

Hierarchy	Cont.	Viz.
Viz	0.24	-
PI	0.0006	<0.0001

Table 7.7: Pairwise comparison between between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values across Regular Languages concept Total grade, Regular expression grade, Definitions questions grade, and Languages Hierarchy grade.

significant difference in all topics. The visualization group has significance versus the control group in the regular expressions related topics as they are affected by the visualizations and the exercises. The visualization is either close to the control group or significantly lower than the control group for the other topics. The main reason is that the visualizations and exercises may help students practice writing different regular expressions or building different NFA/DFA. However, these exercises and visualizations may not affect writing correct definitions about regular languages.

- Context-Free languages (CFL). In this concept, we tested students on
  - Parse trees
  - Writing context-free grammars
  - Grammar ambiguity
  - Drawing pushdown automata (PDA)
  - Definitions, like define a PDA, and some MCQ questions

  - CFL closure properties, such as prove that the union between two CFL languages is also a CFL.
  - Language hierarchy to determine, without using the pumping lemma, if a language is deterministic context-free language, non-deterministic context-free language, or not a context-free language.

  Table 7.8 shows the mean and median for all questions across the three groups. As we see, the Visualizations group has a higher mean grade for topics that are affected by the visualizations and exercises only. The Programmed Instruction group has higher grade means for most topics except for the language hierarchy questions. We started by testing the overall grade for this concept and compared it across all groups. Using the Kruskal–Wallis test, we found a difference between the means (with $pvalue < 0.05$), see Table 7.9.

  Table 7.10 shows the pairwise comparisons between all groups in every Context-Free Language topic question type. As the tables show, the visualizations group has significance in topics with exercises such as drawing a PDA or writing context-free grammar. In topics with no exercises, the visualizations group has a significant improvement in the grammar ambiguity question and no significance in the parse trees question. The Programmed Instruction group has significant improvement in all topics related to context-free languages. The only exception is the hierarchy questions. We ask students to determine the language class if is a context-free language or not without applying the pumping lemma. For these questions, we need to enhance our Programmed Instruction book by adding more questions about language hierarchy for context-free languages.

- Pumping lemma questions include

Groups	Context-Free Languages Topics															
	Parse Trees		Definitions		CFG		Ambiguity		Closure Properties		PDA		Hierarchy		Overall	
	Mean	M	Mean	M	Mean	M	Mean	M	Mean	M	Mean	M	Mean	M	Mean	M
Cont.	8.4	10	25.6	25.8	16.5	15.5	6.3	8	6.5	6.5	6.4	7	9.3	10	78.9	79.5
Viz.	8.7	9.5	25	26	18.6	19	8.3	10	7.3	10	7.5	8.5	8.7	10	84.2	85.3
PI	**9.5**	10	**29.3**	29	**21.4**	23	**9.4**	10	**8.7**	10	**8.5**	9.5	**9.4**	10	**96.2**	98

Table 7.8: The mean and median (M) for Context-Free Languages questions across all groups

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
Parse Trees	44	271	135	8.4	8.7	9.5	2	20.85	<0.0001
Definitions	44	271	135	25.6	25	29.3	2	74.346	<0.0001
CFG	44	271	135	16.5	18.6	21.4	2	46.236	<0.0001
Ambiguity	44	271	135	6.3	8.3	9.4	2	51.977	<0.0001
Closure Properties	44	271	135	6.5	7.3	8.7	2	20.948	<0.0001
PDA	44	271	135	6.4	7.5	8.5	2	19.226	<0.0001
Hierarchy	44	271	135	9.3	8.7	9.4	2	4.7152	0.0947
Overall	44	271	135	78.9	84.2	96.2	2	111.35	<0.0001

Table 7.9: Kruskal–Wallis test for context-free languages topics. The result shows that there is a difference between topics means

Total	Cont.	Viz.
Viz.	0.009	-
PI	<0.0001	<0.0001

Parse Trees	Cont.	Viz.
Viz.	0.7	-
PI	0.04	<0.0001

CFG	Cont.	Viz.
Viz.	0.033	-
PI	<0.0001	<0.0001

Definitions	Cont.	Viz.
Viz.	0.74	-
PI	<0.0001	<0.0001

Ambiguity	Cont.	Viz.
Viz.	<0.0001	-
PI	<0.0001	<0.0001

Closure Properties	Cont.	Viz.
Viz.	0.167	-
PI	0.0001	0.0001

PDA	Cont.	Viz.
Viz.	0.044	-
PI	0.0005	0.0008

Hierarchy	Cont.	Viz.
Viz.	0.38	-
PI	0.86	0.12

Table 7.10: Pairwise comparison between between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values across Context Free Languages topic and its questions.

Groups	Pumping Lemma					
	PL for Regular Languages		PL for CFL		Overall	
	Mean	Median	Mean	Median	Mean	Median
Control	6.2	6	11.7	13	17.9	18
Visualizations	7.7	9	14.8	14	22.5	23
Programmed Instruction	**8**	9.5	**16.7**	14	**24.7**	24

Table 7.11: The mean and median of pumping lemma questions across all groups

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
PL for regular languages	44	271	135	6.2	7.7	8	2	13.727	0.001
PL for context-free languages	44	271	135	11.7	14.8	16.7	2	36.096	<0.0001
Overall	44	271	135	17.9	22.5	24.7	2	42.672	<0.0001

Table 7.12: Kruskal–Wallis test for Pumping Lemma topics. The result shows that there is a difference between topics means

- Use the pumping lemma to prove that a language is not regular
- Use the pumping lemma to prove that a language is not context-free

Table 7.11 shows the mean and median for students' grades in the pumping lemma questions across all groups, starting with testing if there is a difference in groups means for the total grade. With Kruskal-Wallis test $p - value \leq 0.05$, we find a difference between the groups, see Table 7.12. The same is found for the group means in the regular languages pumping lemma and context-free pumping lemma question grades.

Table 7.13 presents pairwise comparisons between the groups in the total grade for the pumping lemma questions, the regular languages pumping lemma question, and the context-free languages pumping lemma question. As the table shows, both experiments have significantly higher grades than the control group. For the visualizations group, the ebook provided some useful visualizations that show students the principles of the pumping lemma and how they can use it to prove that a language is not Regular/Context-Free. The ebook also provides 14 pumping lemma adversary games

Total	Cont.	Viz.
Viz.	<0.0001	-
PI	<0.0001	0.002

Reg. PL	Cont.	Viz.
Viz.	0.014	-
PI	0.004	0.015

CFL PL	Cont.	Viz.
Viz.	<0.0001	-
PI	<0.0001	0.003

Table 7.13: Pairwise comparison between between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values across pumping lemma total grade, regular pumping lemma grade, and context-free languages pumping lemma grade.

Groups	Turing machines questions					
	Definitions		Hierarchy		Over all	
	Mean	Median	Mean	Median	Mean	Median
Control	15.82	15	**10**	10	25.9	25
Visualization	22.3	22	9.1	10	31.4	31
Programmed Instruction	**23.8**	24	9.8	10	**33.7**	34

Table 7.14: Mean and median values for the Turing machines questions across all groups

that students can use to practice. On the other hand, the Programmed Instruction ebook added several detailed pumping lemma examples for each topic. In these examples, the frames go with students step by step through the pumping lemma. The questions ask students to do each step they read about. In the pumping lemma for context-free languages, the proof is even trickier. Students should prove multiple cases to prove that a language is not context-free. The frames questions ask students about multiple strings and cases to help them understand the lemma and its conditions. That is why we see that the Programmed Instruction ebook shows significant improvements in both pumping lemma questions.

- Turing machines. In this topic, we ask students about defining universal Turing machines and different types of Turing machines (decidable, acceptable, and transducers), determine if a Turing machine can decide a language, or comparing between Turing machines and other models such as the linear bounded automata or pushdown automata. Table 7.14 shows the mean and median values for the Turing machines questions. To

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
Definitions	44	271	135	15.82	22.3	23.8	2	59.869	0.001
Hierarchy	44	271	135	10	9.1	9.8	2	5.1714	0.075
Overall	44	271	135	25.9	31.4	33.7	2	40.115	<0.0001

Table 7.15: Kruskal–Wallis test for Turing machines topics. The result shows that there is a difference between topics means except in language hierarchy questions

Total	Cont.	Viz.
Viz.	<0.0001	-
PI	<0.0001	0.0002

Hierarchy	Cont.	Viz.
Viz.	0.18	-
PI	0.89	0.15

Definitions	Cont.	Viz.
Viz.	<0.0001	-
PI	<0.0001	<0.0001

Table 7.16: Pairwise comparison between between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values across Turing machines total grades, definitions grades, and hierarchy grades.

see if there is a difference between groups means, we applied the Kruskal-Wallis test. With a $p - value < 0.05$, we found that there are differences between groups means except in the language hierarchy questions, see Table 7.15.

Table 7.16 shows the pairwise comparisons between all groups in every question type. As the table shows, the Visualizations and Programmed Instructions groups did not have a significantly better score than the control group. This concludes that our ebooks lack the required visualizations and frame questions that better explain to students how to determine if a language can be decided/accepted by Turing machines. We also can infer that having OpenFLAP exercises to build various Turing machines for different languages is not enough to determine if a language is Turing decidable or not. It is worth noting that the hierarchy questions provide different languages and ask students to determine if the language is regular, context-free, decidable using Turing machines, or not solvable.

- Theory part for the Formal Languages course. This part includes the questions about

Groups	Sets		Complexity		Countability		Computability	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Control	16.5	16	46.1	46	13.5	15	**13.4**	20
Viz.	15.2	15	44.1	45	12.5	15	9.9	10
PI	**21.9**	22	**58.3**	60	**14**	15	13.1	10

Table 7.17: Mean and median values for the theory part in the Formal Languages course across all groups.

Experiment	n			Mean			df	x^2	p-value
	Cont.	Viz.	PI	Cont.	Viz.	PI			
Sets	44	271	135	16.5	15.2	21.9	2	150.84	<0.0001
Complexity	44	271	135	46.1	44.1	58.3	2	64.075	<0.0001
Countability	44	271	135	13.5	12.5	14	2	12.719	0.00173
Computability	44	271	135	13.4	9.9	13.1	2	15.582	0.0004

Table 7.18: Kruskal–Wallis test for theory part topics. The result shows that there is a difference between topics means

topics that do not have any OpenFLAP exercises. These topics are

- Set theory
- Complexity theory including P, NP, $NP-Hard$, $NP-Completeness$ and Reductions
- Countable and uncountable sets
- Computability theory

Table 7.17 shows the mean and median values for the study groups in all theory questions. We applied the Kruskal-Wallis test on every question type to test if there is a difference in group grades. We found that in all questions, there is a difference between our three groups, see Table 7.18.

Based on the tables that are presented in 7.3.2, we see that in all theory questions, the visualizations group students do not score better than the control group. This is a good result as the visualizations ebook does not provide any enhancements to the material

Sets	Cont.	Viz.
Viz.	0.16	-
PI	0.0001	0.0001

Countability	Cont.	Viz.
Viz.	0.136	-
PI	0.522	0.007

Computability	Cont.	Viz.
Viz.	0.0216	-
PI	0.7407	0.0043

Complexity	Cont.	Viz.
Viz.	0.35189	-
PI	0.00026	0.0001

Table 7.19: Pairwise comparison between between control group (Spring 2018, $n = 44$), the Visualization group (Spring2019/Fall2019/Spring2020, $n = 271$), and the Programmed Instruction group (Fall 2020/Spring 2021, $n = 135$) using BH adjusted p-values across Theory part questions.

used to teach these topics to students. The same is true for the exercises. We do not have any exercises on these topics. Thus, both the control and visualizations groups studied from the same OpenDSA materials, including some old visualizations and no exercises. Students only read the text, saw the visualizations, and did hand-written exercises for homework. This supports the claim that changes in student grades on all other questions were due to the features being added to the visualization eTextbook. For the Programmed Instruction, we find a significant impact on some topics, which are set theory and complexity theory. In the other topics, the Programmed Instruction students perform better than the visualization group but not significantly better than the control group. This motivates us to write better Programmed Instruction frames and questions to help students better understand these topics in the future semesters.

7.3.3 Discussion

Based on the presented results, we find that the Visualization group students have better performance than the control group on those topics where visualizations and exercises were

added, and not on those where no changes were made. The visualization ebook presents several visualizations to some Formal Languages topics. In addition, the ebook contains several exercises (OpenFLAP auto-graded exercise, OpenFLAP proficiency exercises, and Khan Academy exercises) on those topics. Collectively these visualizations and exercises helped students to score better than the control group. However, in topics that are not affected by the visualizations and the exercises, students did not score better than the control group. This strengthens our findings that the visualizations and exercises helped the students to understand the Formal Languages topics.

The Programmed Instruction ebook inherited all visualizations and exercises from the visualizations ebook. Then the text and visualizations are refactored by adding questions about the visualizations and the text through PI frames. So, students can interact with the book and examine their understanding of the material. These frames and questions enhanced the understanding of the content and helped students score better in most of the Formal Languages topics. However, we need to pay more attention to topics that the Programmed Instruction ebook could not help students score better, such as determining the language classes (Language Hierarchy), Computability, and Countability topics. We need to provide better frames and questions to allow students to understand them better.

As a result, to answer the question about "How does performance on exams for the intervention groups compared to the performance of the control group students on the same set of questions?", we find that both intervention groups score better than the control group in most of the course topics that are covered by the intervention ebooks.

When we compare the two intervention groups against each other, we find that in some topics having visualizations and exercises is enough for students, such as the pumping lemma topic. In other topics affected by the visualizations ebook, the Programmed Instruction ebook help students to understand the contents better as it provides more engagement with book

materials. In addition, the Programmed Instruction ebook modified nearly all the course materials. The visualizations ebook only modified a subset of the course topics. Thus to answer the question about "How does performance on exams for the PI group compare to the performance of the Visualization students on the same set of questions?", we can say that the Programmed Instruction ebook has better performance in most of the topics that are affected by the visualizations ebook. In addition, the Programmed Instruction ebook has a broader affect on students' grades as it covers all course topics. Finally, the Programmed Instruction ebook needs some updates to affect the other topics that are not affected by the current ebook.

To answer the question about students' satisfaction, we find that students found the exercises beneficial, and some of them asked for more exercises to have better practice. For the Programmed Instructions ebook, students found the ebook more engaging, and solving the questions helped them test their understanding and made them think more deeply about the sentences presented in the frames. Students who are against the ebook find that some changes make the book more valuable, like

- Even with only MCQ questions, students find that they sometimes struggle to answer the question if they could not determine the correct answer. They asked for providing the correct answer after a few wrong attempts. We can also use incorrect solutions to guide the students to know what makes them select an incorrect answer.

- Some students found that dividing the information into pieces is not helping them to get a good idea about the topic as a whole. They prefer to have the information presented as one piece then add frames with questions for the same information. We already started this mechanism in the Fall ebook for the first three chapters only in response to them. We plan to adopt the general approach of giving a brief overview of

the topic in prose, followed by a Frameset with details about the same topics.

- Some students believe that the answered frames questions should be shown with the correct answer when they reread the frames. Currently, we left the questions as optional to solve so students can answer them again. Based on OpenDSA logs, students do not solve these answered questions again. We believe that the ebook needs to support displaying new questions for the same answered questions. This will give students a chance to study the ebook with new questions that may help them better understand the content.

7.4 Students Interactions with the eTextbooks

In this section we will present our findings from measuring students time spent on our eTextbooks and their attempts to solve the OpenFLAP exercises.

7.4.1 Solving OpenFLAP exercises in the intervention groups

One major component in our ebooks is OpenFLAP exercises. These exercises are found in both the Visualization and PI Frames books. One way to compare the Visualization ebook and the Programmed Instructions ebook is to see if the PI frames caused students' grades or attempts on the OpenFLAP exercises to change. Also, since there is an unlimited number of auto-graded attempts, we will not compare students' overall grades in the auto-graded exercises. Students in both groups can solve the exercises at any time for any number of attempts. In general, there is no significant difference in students' total auto-graded exercise grades, see Table 7.20. As the table shows, students attempted the exercises less in the Programmed Instruction book to achieve an equivalent average score in all auto-graded

Books	OpenFLAP exercises attempts			OpenFLAP exercises scores		
	Mean	Median	p-value	Mean	Median	p-value
Phase 2 book	7.43	7.59		45.43	45.29	
Programmed Instruction book	**3.04**	2.48	<0.0001	45.74	47.38	0.888

Table 7.20: Average number of attempts for each auto-graded exercise and average score for exercises for the Phase 2 book and Programmed Instruction book

exercises. It seems that the Programmed Instruction ebook better prepares students to solve these exercises. As Section 6.2.1 shows how different books explain an NFA to DFA conversion algorithm, the Programmed Instruction ebook walks through the steps one at a time, asking the students to do the step by hand, via selecting the correct answer. This technique prepares students to do the exercises in fewer attempts.

7.4.2 Spending time on the eTextbooks

Students spend time using the ebooks to read the material, watch the visualizations, answer the exercises, and answer the frames questions. There are two questions that can be answered by students' spent time

- First, is there a difference between the time spent in the visualizations ebook and the Programmed Instruction ebook? Perhaps the time is longer for the Programmed Instruction book as students may be slowed down by the frames questions. Or perhaps their time is reduced because they need fewer attempts to answer the exercises after completing the PI frames. On the other hand, students may ignore answering the questions and lose the mastery points, since the mastery points are only 5% of the total grade.

  We started by identifying the percentage of PI frames that students did during Fall 2020 and Spring 2021. We found that the majority of students did the PI frames about

the key concepts of the course. Frames about DFA, PDA, grammars, Turing machines or theory parts of the course are done by mostly all students. On the other hand, students tend to skip reading the PI frames about repeated examples. For instance, in our book we have about seven examples about each type of the Pumping Lemma. Students tend to skip some of these examples as they are done from other similar examples. The same is found for NP-Completeness proofs. We provide students with a number of different NP-Complete problems and how they can use reductions to prove that these problems are NP-Complete. This gives us a sign that we need to provide a reasonable number of examples with different questions to give students a reason to read the provided examples. In general, we tested the impact of completing the PI frames on students grades. Figure 7.2 shows that there is a significant positive relation between student PI frames complete ratio and their exam scores. This means that as students study more from the PI frames and answer the questions they gain better understanding and therefore, better scores.

We also compared the spent time between the Spring 2020 semester (using the visualizations book) and the Programmed Instruction group. The reason to compare with Spring 2020 students is that Spring 2020 is the closest to the Programmed Instruction group. Spring 2020 book is the best version of the visualizations books as it includes all types of OpenFLAP exercises and visualizations that are used in the Programmed Instruction book. For instance, as shown in [47], the Turing machines exercises were used in the Spring 2020 book only. The comparison shows that the Programmed Instruction group spent significantly more time than the Spring 2020 students ($p - value < 0.05$). Table 7.21 show the comparison results.

- Does spending more time on the eTextbook help students get better scores? Students spent many hours studying from our books. Since Programmed Instruction is a mastery

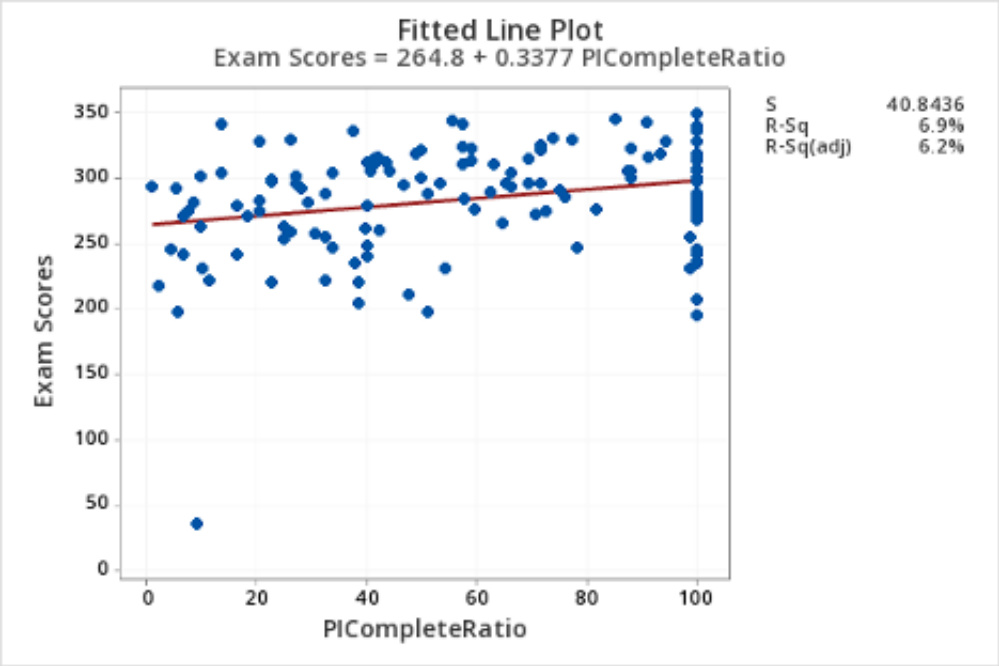


Figure 7.2: Scatter plot for the PI frames completion ratio versus students' exams grades along with the regression relation line.

Group	Mean	Median	p-value
Visualizations (Spring 2020)	88	71	
Programmed Instruction (Fall 2020 & Spring 2021)	113.1	143.3	0.044

Table 7.21: Comparison between the time spent by the visualizations group against the Programmed Instruction group in hours.

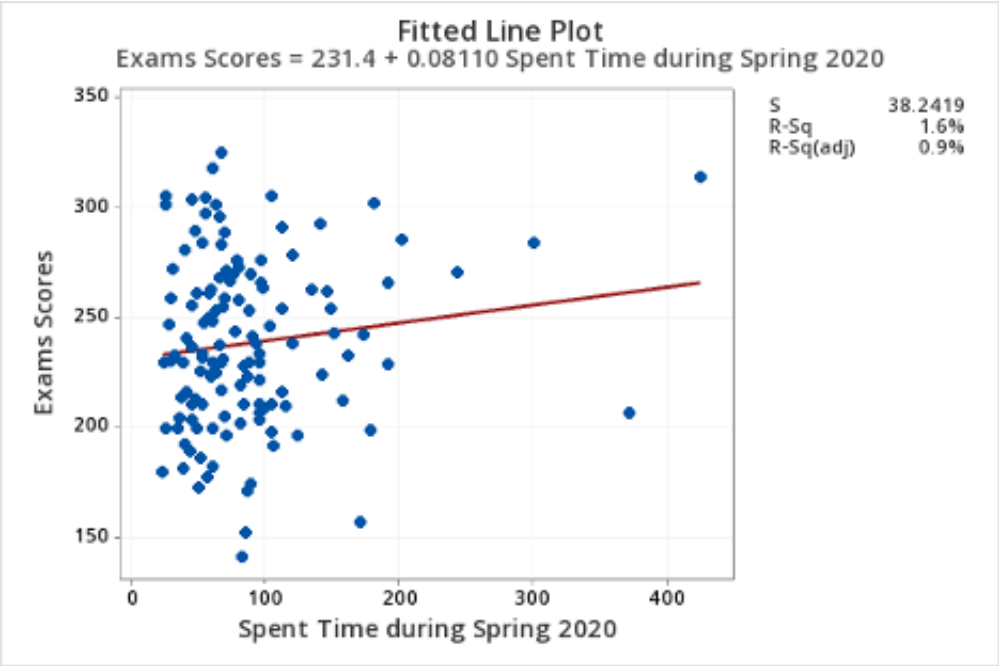


Figure 7.3: Scatter plot for the number of hours spent by the visualization group versus their exams grades along with the regression relation line.

based technique, we should expect that spending more time means reading frames, answer questions, and solving OpenFLAP exercises. These things should have an effect on students success and total grade.

To answer this question, we performed the following tests

- The correlation between the Spring 2020 group exams scores and the spent time. We found that there is no significant relationship between students' exam scores and their spent time, $p-value = 0.147$. This means that, if students spend more time on the visualization ebook, there is no evidence that this leads to an increase in their exam scores. Figure 7.3 shows the scatter plot for the number of hours versus the exams scores along with the regression relation line.
- The correlation between the Programmed Instruction group exam scores and their time spent on the book. We found that there is a significant positive relation,

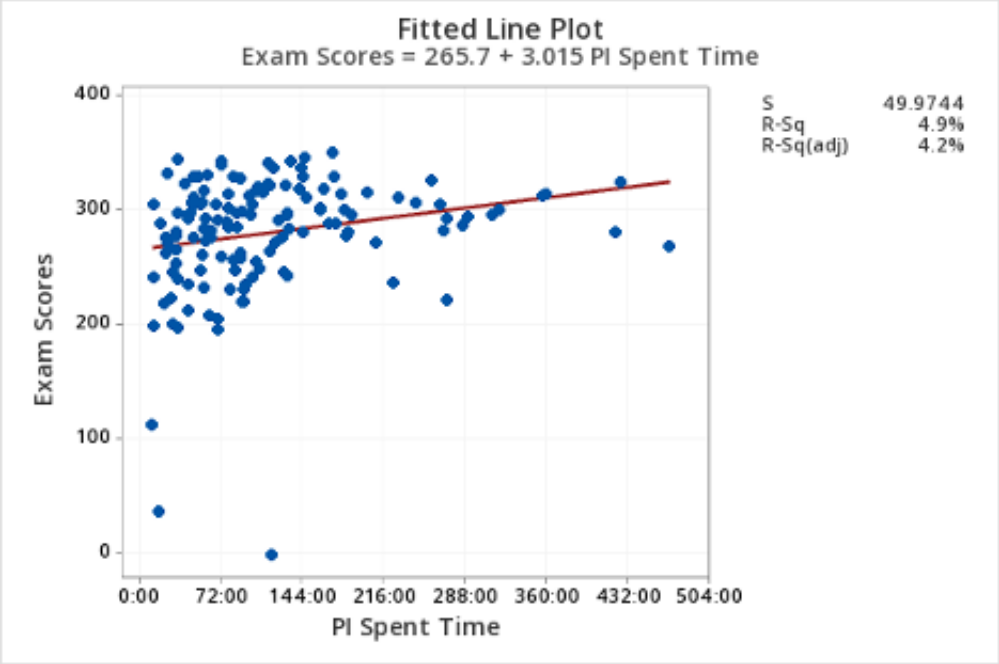


Figure 7.4: Scatter plot for the number of hours spent by the Programmed Instruction group versus their grades along with the regression relation line.

$p - value = 0.009$. This means that students who spend more time on the book are getting better scores on the exams. As Figure 7.4 shows, each hour spent by students in the Programmed Instruction ebook helps them to earn 0.08 points. This result means that students learn from the book as they spend more time reading and solving the frames questions.

By looking at the previous figures, we find that staying more than 200 hours in the book lead to slight increase in the grades. So, we studied the same correlations after removing all data related to students who spent more than 200 hours. Figures 7.5 and 7.6 show the correlations between spending less than 200 hours in the visualizations ebook and the Programmed Instruction ebook respectively. The same results are found. There is no correlation between spending more hours in the visualization ebook and the grades, $p - value = 0.985$. On the other hand, spending more time in the Programmed

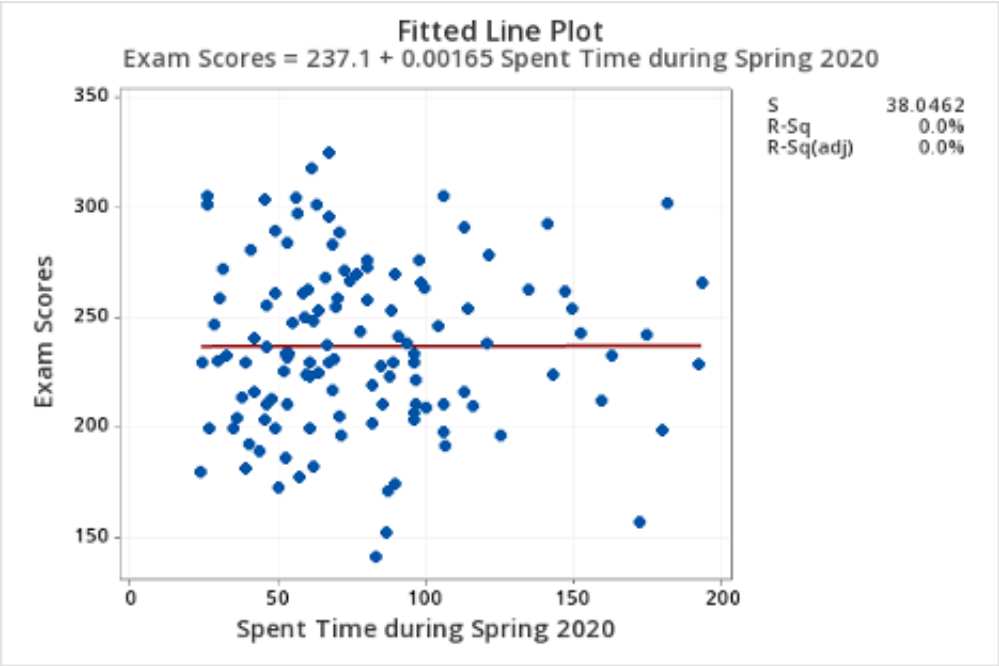


Figure 7.5: Scatter plot for studying less than 200 hours in the visualization group versus their grades along with the regression relation line.

Instruction ebook leads to better exam grades with $p - value = 0.001$ (in this case, spending one hour more helps students to earn around 8.4 points for the first 200 hours).

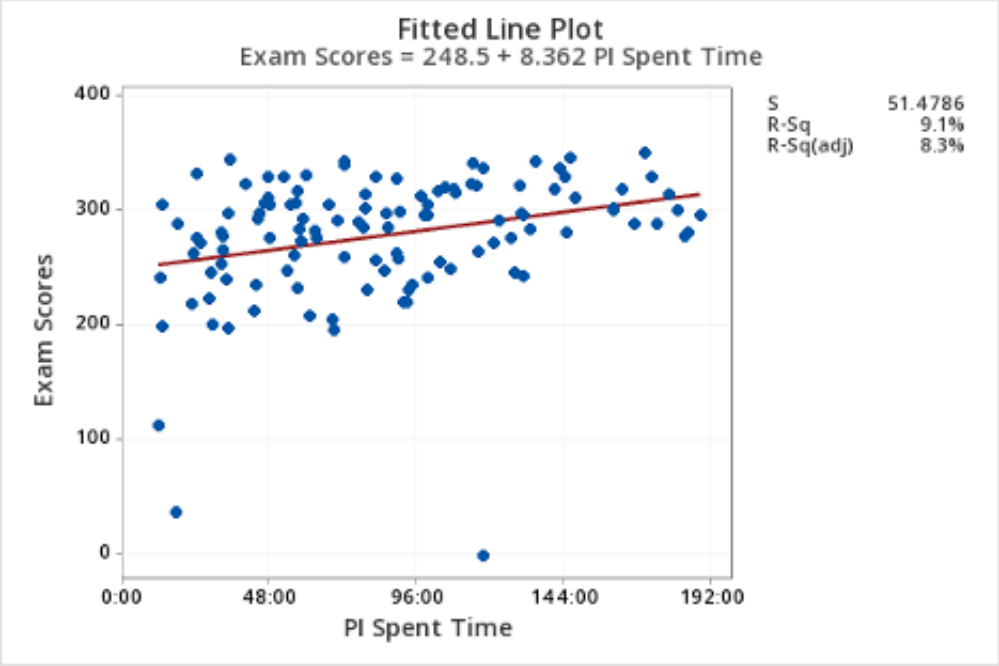


Figure 7.6: Scatter plot for studying less than 200 hours in the Programmed Instruction group versus their grades along with the regression relation line.

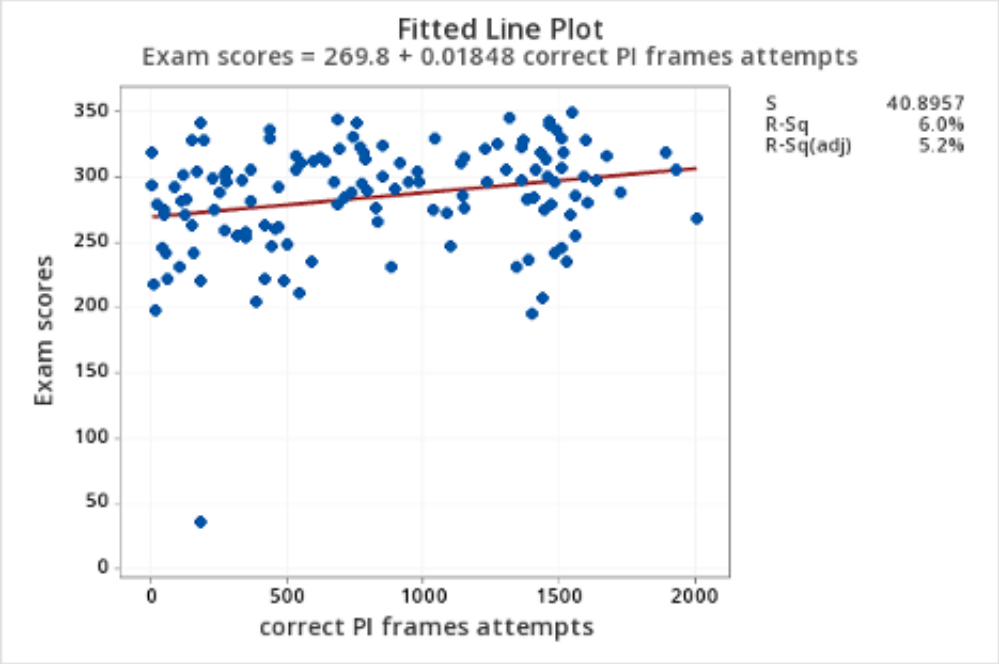


Figure 7.7: Scatter plot for studying the effect of answering the PI frames questions correctly on students exam scores.

7.4.3 Frames Questions Attempts

Students need to answer the frames questions correctly to advance to the next frame. To earn the mastery points, students can read and study the frames information, then test their understanding by answering the frames question. The other option is to game the frames system by trying every possible solution and earn the credit without learning at all. To test the impact of correctly answering the frames questions on students exams scores, we tested the correlation between the number of correct attempts against students scores in the three exams. We found that there is a significant relation between them with $p - value = 0.005$. This means that when students understand the content and answer the questions correctly, even after some incorrect attempts, students score better grades. Figure 7.7 shows the relation between students correct attempts and their scores.

What about gaming? students may tend to solve the questions randomly to earn the mastery

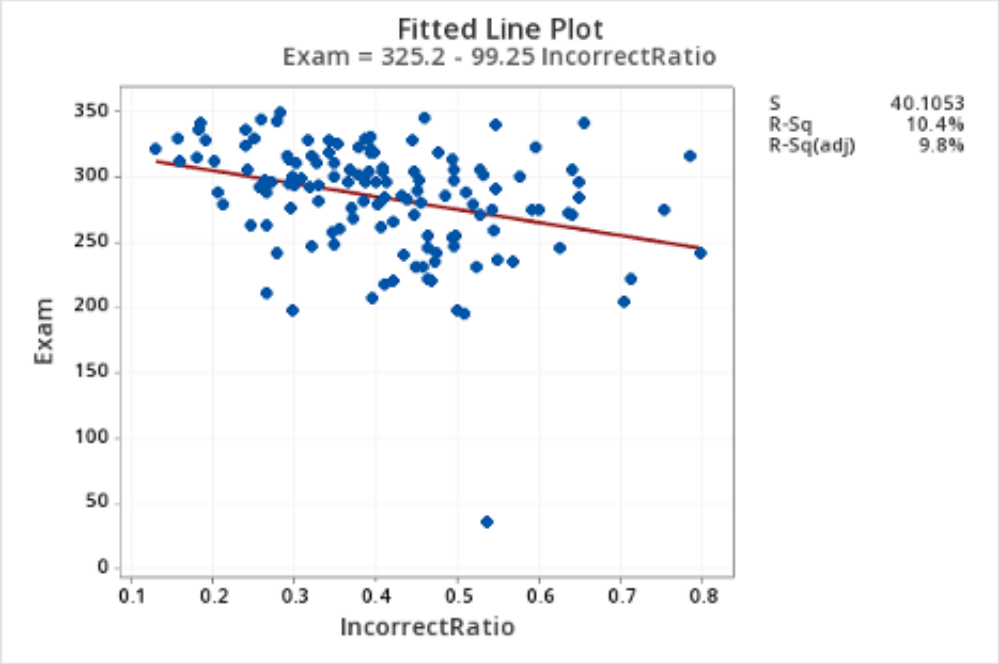


Figure 7.8: Scatter plot for studying the effect of gaming the frames questions and students' exam scores.

points. One factor that may help us to identify gaming is the ratio of incorrect PI attempts. High ratio means that a student is attempting to solve the question without understanding the material. Perhaps with more data, there are other factors that allow us to identify if the student mistakes are due to misunderstanding the material or due to gaming. For now, we will check the relation between the incorrect attempts ratio and student score in the exams. Figure 7.8 shows that there is a significant negative relation $p - value < 0.0001$ between students scores and their incorrect attempts for the PI frames questions. As the figure shows, when students mistakes increase, their grades decrease. This means as students do not focus to answer the question correctly from few attempts, then they are not gaining from their behavior. This result indicates that the Programmed Instruction purpose is to read and understand the content then prove that by solving the question correctly. However, just skipping the material and answering the questions will not help students to earn more exam scores.

Semester	Modality			Book
	Exam 1	Exam 2	Exam 3	
Spring 2018	Face to Face			Traditional
Spring 2019 and Fall 2019	Face to Face			Visualizations
Spring 2020	Face to Face	Virtual		
Fall 2020 and Spring 2021	Virtual			Programmed Instruction

Table 7.22: Different semesters exams' modality

7.5 COVID19 effect

Before Spring 2020, all students attended in-class lecture and took the normal face-to-face modality exam. Due to COVID19, students started to attend virtual classes and exams. This affected our results. Our results as reported in this chapter are on students who worked online in part during Spring 2020, and entirely online during Fall 2020 and Spring 2021. This means some of the visualization group and all of the Programmed Instruction group. Once we go back to the normal face-to-face modality, we plan to reapply the last experiment (the Programmed Instruction ebook) and compare the results with our control group. Table 7.22 shows all semester exams modality in our four years of study. Based on the table, we can compare the control group and the visualizations group as both groups took the same instruction and exam modality. The same is true between the visualizations group (exams 2 and 3) against the Programmed Instruction group. However, we do not have that between the control group and the Programmed Instruction.

By looking at Table 7.3.2, we compare all groups in the theory part of the course. As we mentioned earlier, these topics were taught in both the visualizations and control groups without any changes to the content. The table showed that both groups have no significant difference between the groups' means. This helps us to have confidence that the transition to online classes did not significantly impact learning outcomes, and so does not invalidate our results that show improvement in performance on the Programmed Instruction treat-

ment. We need to study these topics deeply because, as Table 7.22 showed, the visualization group is the only intervention group that students have been tested in these topics in both modalities. So, we can study the third exam grades between semesters (Spring 2018, Spring 2019, Fall2019, Spring 2020, Fall 2020, and Spring 2021) as four groups

- Control group. This is the group for students who took the material in the traditional book (Spring 2018).
- Face to face group. This group had third exams in traditional face-to-face modality. This group contains students' data from Spring 19 and Fall 19.
- Partly virtual group. This group of students took only the third exam virtually in Spring 2020 due to the pandemic.
- Programmed Instruction group. This group had the third exam contents in virtual modality as well, but it is different from the partly virtual group in that this group studied the contents of the exam using the Programmed Instruction ebook.

Table 7.23 shows all groups mean and median values for the topics covered by the third exam. These topics are Countability, Computability theory and Complexity theory. As the table shows, the control, face to face, and virtual groups have close total grades. This means that the difference between taking the exam virtually or face to face did not have a significant impact on student grades for these topics. By looking at the Programmed Instruction group, we see that it has a higher grade mean and median than the virtual group in all topics.

To test if there is a pair-wise significant difference between the groups, we found that the control, face to face, and the virtual groups do not have any significant difference when compared to each other in the Complexity theory questions, see Table 7.24. However, the Programmed Instruction group has significant difference when compared to all other groups.

Groups	Complexity		Countability		Computability	
	Mean	Median	Mean	Median	Mean	Median
Control	46.1	46	13.5	15	**13.4**	20
Face to Face	45.3	46.5	12.7	15	10.1	12.5
Virtual	42.9	42	12.9	15	8.7	5
PI	**58.3**	60	**14**	15	13.1	10

Table 7.23: Comparison of the third exam grades between control group (Spring 2018, $n = 44$), the Face to Face third exam group (Spring/Fall 2019, $n = 142$), the virtual group (Spring 2020, $n = 128$), and the PI group

Complexity	C	F2F	V
F2F	0.83	-	-
V	0.28	0.27	-
PI	0.0003	<0.0001	<0.0001

Computability	C	F2F	V
F2F	0.15	-	-
V	0.003	0.08	-
PI	0.56	0.15	0.0005

Countability	C	F2F	V
F2F	0.343	-	-
V	0.12	0.34	-
PI	0.47	0.07	0.011

Table 7.24: Pairwise comparison between between control group "C" (Spring 2018, $n = 44$), face to face "F2F" group (Spring/Fall 2019, $n = 142$), virtual group "V" (Spring 2020, $n = 128$), and the Programmed Instruction group "PI" for exam 3 topics using BH adjusted p-values for Multiple Comparisons

For the Computability theory questions we find that the control group and the face to face groups have higher grades than the virtual group. The control group is significantly higher than the virtual group. This means that having exams in virtual modality may have a negative effect on students grades. Finally, for the Countability questions, we find that the Programmed Instruction scores are significantly higher than the virtual group and no significance difference as compared to the control and face to face groups. This is similar to what we found in Section 7.3.

7.6 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 (the same instructor taught all intervention semesters). However, the instructor for the intervention groups was the GTA for the control group, and both were authors of the content, and both were heavily involved in this project throughout the study. The course contents were the same, just the presentation method differed (adding visualizations and replacing paper homework exercises with interactive exercises for the intervention groups).

Finally, since the pandemic still exists, students for the intervention group took all exams online, which is different than the control group who took all the exams in the face to face manner. We find in general that when the instructional material are the same, the exam results are the same. This gives us confidence that when the exam scores are different where the content presentation is different, that this is really being caused by the content presentation, not whether the exam is face to face or online.

Chapter 8

Conclusions and Future Work

In this study we implemented two eTextbooks and a simulation tool to replace a traditional textbook in a senior Formal Languages course. We started by re-implementing the JFLAP tool to be openly available to work with any eTextbook with LTI support to create and auto-grade exercises. The new version is called OpenFLAP. Then we created an OpenDSA Formal Languages eTextbook that uses OpenFLAP visualizations and exercises. Finally, we generalized the support for Programmed Instruction to support any OpenDSA eTextbook. This support allows instructors to add questions inside the slides to increase students engagement. Using the Programmed Instruction support, we developed a second eTextbook to teach students the Formal Languages course in addition to the OpenFLAP exercises.

Our eTextbooks and tools were found to be more engaging than the control group. In the visualizations ebook, students spent on average 87 hours studying the book materials and doing the exercises. In the Programmed Instruction ebook, students spent more time to read the frames and answer the questions. On average, students spent around 113 hours to work on the ebook and do the exercises. We found that the Programmed Instruction ebook helps students to score higher grades.

To collect students satisfaction, we surveyed the students about the OpenFLAP exercises and the Programmed Instruction materials. The majority of the surveyed students found that all OpenFLAP exercises are helpful for them to learn the Formal Languages topics. The majority also mentioned positive feedback about the idea of having questions inside the

frames that students need to answer before moving forward to read and study more. However, students who did not like the idea provided us with ideas to improve the Programmed Instruction eTextbook for future semesters.

By comparing different groups grades, we found that the Visualization eTextbook significantly improved students grades in all topics covered by the visualizations and the exercises. The Programmed Instruction ebook improved the students' grades in almost all topics covered by the course. There are few exceptions like Countability theory. For these topics we need to recreate frames questions that allow students to get better understanding and grades in these topics.

Based on student responses, we believe that the Formal Languages ebook should not have only Frames and questions. Students may find it better to have some introduction to the topic followed by a framesets. Students need to get a good overview about the topic then dig into the frameset and solve its questions. At the same time, not all course content should include frames questions. Some topics will be easier for students to have slideshows only to teach students an example for an algorithm. The example can be followed by a frameset to ask students about a similar example.

8.1 Future Work

The Programmed Instruction eTextbook still needs work. We have a list of tasks to improve the book. These include:

- Test Programmed Instruction ebook with flipped classrooms. Programmed Instruction can help students to prepare for hands-on activities in lecture. As Programmed Instruction is a mastery-based technique, students can study the material and learn

about the course at their own peace.

- Enhance the book by randomly changing the question provided with each frame. This requires the content developers to generate multiple questions for each frame and the frames system will select one of them. This will lead students to read and try the frames multiple times.

- Add free response questions for the frames. This requires us to implement Natural Language Processing (NLP) algorithms that can auto-generate questions and auto-grade the questions.

- Finally, we used a simple form of Programmed Instruction. In our model, we allowed students to freely select the order of the framesets. Students can leave a frameset to be answered later. We need to test other forms of Programmed Instruction like enforce the frameset order. For example, we need to force students to do a frameset about the NFA to DFA before doing the auto-graded exercise about the same algorithm. We believe that this type of Programmed Instruction will help students score better grades in fewer attempts at the exercises.

For OpenFLAP, we only reimplemeted the important functions in JFLAP. These are the functions that helped us to generate important visualizations and exercises. We hope to complete reimplementing all JFLAP functions to have a complete OpenFLAP system. We also need to continue supporting the creation of new exercises. This will encourage instructors to use our tool in their ebooks. JFLAP can be used to teach topics other than the Formal Languages. Course like Programming Languages, Compilers, and Theory of Computations can use OpenFLAP visualizations and exercises.

Bibliography

[1] John A Barlow. Conversational Chaining in Teaching Machine Programs. *Psychological Reports*, 7(2):187–193, 1960.

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

[3] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188, 2001.

[4] Howard C Berthold and Robert H Sachs. Education of the Minimally Brain Damaged Child by Computer and by Teacher. *Programmed Learning*, 11(3):121–124, 1974.

[5] Donald Bitzer, Peter Braunfeld, and W Lichtenberger. PLATO: An Automatic Teaching Device. *IRE Transactions on Education*, 4(4):157–161, 1961.

[6] John B Carroll. A model of school learning. *Teachers college record*, 1963.

[7] Pinaki Chakraborty, Prem Chandra Saxena, and Chittaranjan Padmanabha Katti. Fifty years of automata simulation: a review. *ACM Inroads*, 2(4):59–70, 2011.

[8] Barbara Chambers and John M Schulte. An Evaluation of Programmed-Instruction. *Education*, 85(4):245–249, 1964.

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

[10] Norman A Crowder. Automatic Tutoring by Means of Intrinsic Programming. *Automatic Teaching: The State of The Art*, 116, 1959.

[11] Jesse H Day. Teaching Machines. *Journal of Chemical Education*, 36(12):591, 1959.

[12] Donald J Dessart. A Study in Programed Learning. *School Science and Mathematics*, 62(7):513–520, 1962.

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

[14] James L Evans, Lloyd E Homme, and Robert Glaser. The RULEG System for the Construction of Programmed Verbal Learning Sequences. *The Journal of Educational Research*, 55(9):513–518, 1962.

[15] Mohammed F Farghally, Kyu Han Koh, Hossameldin Shahin, and Clifford 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. ACM, 2017.

[16] Richard E Ferdig. Assessing Technologies for Teaching and Learning: Understanding the Importance of Technological Pedagogical Content Knowledge. *British Journal of Educational Technology*, 37(5):749–760, 2006.

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

[18] Eric Fouh, Ville Karavirta, Daniel A Breakiron, Sally Hamouda, Simin Hall, Thomas L Naps, and Clifford A Shaffer. Design and Architecture of an Interactive ETextbook–The OpenDSA System. *Science of Computer Programming*, 88:22–40, 2014.

[19] Daniel P Friedman and Matthias Felleisen. *The Little Lisper*. MIT Press Cambridge, MA, 1987.

[20] Daniel P Friedman and Matthias Felleisen. *The Little Schemer*. MIT Press, 1996.

[21] Thomas F Gilbert. Mathetics: The Technology of Education. *Journal of Mathetics*, 1 (1):7–74, 1962.

[22] Mahesh V Gokhale, Prakash V Gaikwad, and Sunetra C Patil. Programmed Learning Technique: An Effective Tool in Teaching and Learning Genetics-Mendelism and Gene Interactions at Under Graduate Level. *International Research Journal of Engineering and Technology*, 3(8):157–162, 2016.

[23] Barbara A Goldrick. Programmed Instruction Revisited: A Solution to Infection Control Inservice Education. *The Journal of Continuing Education in Nursing*, 20(5):222–227, 1989.

[24] Leo S Goldstein and Lassar G Gotkin. *A Review of Research: Teaching Machines vs. Programmed Textbooks as Presentation Modes*. Center for Programed Instruction, 1962.

[25] Eric Gramond and Susan H Rodger. Using JFLAP to Interact With Theorems in Automata Theory. In *ACM SIGCSE Bulletin*, volume 31, pages 336–340. ACM, 1999.

[26] Ben A Green Jr. Physics teaching by the Keller Plan at MIT. *American journal of physics*, 39(7):764–775, 1971.

[27] Michael 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. ISBN 1581134738. doi: 10.1145/563340.563364. URL https://doi.org/10.1145/563340.563364.

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

[29] Sally Hamouda, Stephen H. Edwards, Hicham G. Elmongui, Jeremy V. Ernst, and Clifford A. Shaffer. Recurtutor: An interactive tutorial for learning recursion. *ACM Trans. Comput. Educ.*, 19(1), November 2018. doi: 10.1145/3218328. URL https://doi.org/10.1145/3218328.

[30] James Hartley. Strategies for Programmed Instruction: An Educational Technology. 1972.

[31] John B Hough. An Analysis of the Efficiency and Effectiveness of Selected Aspects of Machine Instruction. *The Journal of Educational Research*, 55(9):467–471, 1962.

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

[33] Ville Karavirta and Clifford 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. ACM, 2013.

[34] Fred S Keller. "Good-Bye, Teacher…" 1. *Journal of applied behavior analysis*, 1(1): 79–89, 1968.

[35] Maria Knobelsdorf, Christoph Kreitz, and Sebastian Böhne. Teaching theoretical computer science using a cognitive apprenticeship approach. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 67–72, 2014.

[36] 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.

[37] Laura Korte, Stuart Anderson, Helen Pain, and Judith Good. Learning by game-building: a novel approach to theoretical computer science education. In *Proceedings of*

the 12th annual SIGCSE conference on Innovation and technology in computer science education, pages 53–57, 2007.

[38] N Izzet Kurbanoglu, Yavuz Taskesenligil, and Mustafa Sozbilir. Programmed Instruction Revisited: a Study on Teaching Stereochemistry. *Chemistry Education Research and Practice*, 7(1):13–21, 2006.

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

[40] Barbara Lockee, David Moore, and John Burton. Foundations of Programmed Instruction. *Handbook of research on educational communications and technology*, pages 545–569, 2004.

[41] BB Lockee, MB Larson, JK Burton, and DM Moore. Programmed Technologies. *Handbook of Research on Educational Communications and Technology*, 3:187–197, 2008.

[42] Henry B Mann and Donald 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.

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

[44] Edward Nugent McKeown. A Comparison of the Teaching of Artithmetic in Grade Four by Teaching Machine, Programmed Booklet, and Traditional Methods. *Ontario Journal of Educational Research*, 1965.

[45] Francis Mechner. Behavioral Analysis and Instructional Sequencing. *Programmed Instruction: The sixty-sixth Yearbook of the National Society for the Study of Education*, pages 81–103, 1967.

[46] Mostafa Mohammed, Susan Rodger, and Clifford A Shaffer. Using programmed instruction to help students engage with etextbook content. *The First Workshop on Intelligent Textbooks*, 2019.

[47] Mostafa Mohammed, Clifford A Shaffer, and Susan H Rodger. Teaching formal languages with visualizations and auto-graded exercises. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 569–575, 2021.

[48] Mostafa Kamel Osman 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. ISBN 9781450367936. doi: 10.1145/3328778.3372711. URL https://doi.org/10.1145/3328778.3372711.

[49] Michael Molenda. When Effectiveness Mattered. *TechTrends*, 52(2):53, 2008.

[50] Thomas L Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, et al. Exploring the role of visualization and engagement in computer science education. In *Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 131–152. 2002.

[51] Glenn Richmond. Comparison of Automated and Human Instruction for Developmentally Retarded Preschool Children. *Journal of the Association for the Severely Handicapped*, 8(3):78–84, 1983.

[52] Susan H Rodger and Thomas W Finley. *JFLAP: an Interactive Formal Languages and Automata Package*. Jones & Bartlett Learning, 2006.

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

[54] Susan H Rodger, Eric Wiebe, Kyung Min Lee, Chris Morgan, Kareem Omar, and Jonathan Su. Increasing Engagement in Automata Theory with JFLAP. In *ACM SIGCSE Bulletin*, volume 41, pages 403–407. ACM, 2009.

[55] Dennis C Russo, Robert L Koegei, and O Ivar Lovaas. A Comparison of Human and Automated Instruction of Autistic Children. *Journal of Abnormal Child Psychology*, 6 (2):189–201, 1978.

[56] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[57] Vinay Shekhar, Akshata Prabhu, Kavitha Puranik, Lakshmi Antin, and Viraj Kumar. JFLAP extensions for instructors and students. In *2014 IEEE Sixth International Conference on Technology for Education*, pages 140–143. IEEE, 2014.

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

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

[60] Michael Sipser. *Introduction to the Theory of Computation*. Cengage learning, 2012.

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

[62] Burrhus Frederic Skinner. The Science of Learning and The Art of Teaching. *Cambridge, Mass, USA*, pages 99–113, 1954.

[63] Burrhus Frederic Skinner. Teaching Machines. *Science*, 128(3330):969–977, 1958.

[64] Lawrence M Stolurow. Chapter IX: Programed Instruction for the Mentally Retarded. *Review of Educational Research*, 33(1):126–136, 1963.

[65] Julian I Taber et al. Learning and Programmed Instruction. 1965.

[66] Cesar A Tecson and Ma Mercedes 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.

[67] M Vijayalaskhmi and KG 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.

[68] Henry R Weinstock, Francis W Shelton, and Jerry L Pulley. Critique of Criticisms of CAI. In *The Educational Forum*, volume 37, pages 427–433. Taylor & Francis, 1973.