Using Programmed Instruction to Help Students Engage with eTextbook Content*

Mostafa Mohammed[1][0000−0002−0652−2817], Susan Rodger[2][0000−0002−2524−7718], and Clifford A. Shaffer[3][0000−0003−0001−0295]

[1] Virginia Tech, Blacksburg, Virginia, USA
and Assiut University, Assiut, Egypt
profmdn@vt.edu
[2] Duke University, Durham, North Carolina, USA
rodger@cs.duke.edu
[3] Virginia Tech, Blacksburg, Virginia, USA
shaffer@vt.edu

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. Students need to read a significant amount of text and do practice problems by hand to achieve understanding. Electronic textbooks have many useful methods to display the content to students. However, unless carefully designed, students abuse these methods to earn grades without studying the content carefully. Inspired by the principles of the Programmed Instruction (PI) teaching method, we seek to develop a new Formal Languages eTextbook capable of conveying Formal Languages 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 are permitted to continue on to other frames of information, or must retry to solve the exercise. Our goal is to present all algorithms using algorithm visualizations and to produce auto-graded exercises to let students demonstrate understanding. To evaluate the pedagogical effectiveness of our new eTextbook, we plan to conduct time and performance evaluations across three offerings of the course CS4114 Formal Languages and Automata at Virginia Tech. The time spent by students looking at instructional content with text and visualizations will be compared with time spent using PI frames to determine levels of student engagement.

Keywords: Formal Languages · eTextbook · Programmed Instruction · Auto-graded exercises · Frames.

* We gratefully acknowledge the support of NSF through grants DUE-1139861, IIS-1258571, DUE-1432008, and DUE-1431667.

1 Introduction

In this paper, we discuss how to help students in a Formal Languages course to better engage, understand, and practice the course content. We do this through new implementations of an old pedagogy called Programmed Instruction. We argue for why this is superior for the type of material taught in Formal Languages than is used in traditional textbooks.

In 1953, BF Skinner went to his daughter's school to ask about her progress [10]. There he noticed these flaws in the traditional teaching methods. a) Instructors give little attention to individual students in class, b) textbooks provide no immediate evaluation of student solutions, and c) not all students have the same prior knowledge, thus some of them are not prepared to acquire knowledge from the textbook. These observations led Skinner to think about developing a new teaching technique, which resulted in the Programmed Instruction (PI) machine [15]. The PI machine works by presenting a small piece of information (a sentence or short paragraph, called a *frame*) to the student. The student then must answer a question about the given frame. If the student successfully answers the question, he/she is allowed access to the next frame. Failing to solve the question will prevent the student from moving forward to the next frame until they come up with the correct solution. The PI system acts as a personal tutor to the student. The PI approach gives students immediate evaluation for each response. Therefore, students will have motivation to solve each frame question correctly to move forward to further frames. Skinner [15] claimed that students would be able to learn twice as much with the same time and effort as compared to the traditional teaching method. Skinner's PI machine itself is not as important to us as the PI approach.

PI and similar teaching machines were once leading areas of research and development. Much of this research was conducted during the 1950s and died out in the early 1970s. During this period, PI research revolved around addressing instructional effectiveness, learner pacing, reinforcement strategies, and long-term effects [13]. Although there was considerable interest in PI due to Skinners *Teaching Machines* and *Technology of Teaching* [16], PI was superseded by Computer-Assisted Instruction (CAI), and the Keller Plan in the 1970s [10, 8].

The Personalized System of Instruction (PSI, or Keller Plan) became widely used during the 1970s and 1980s [8]. Under the Keller Plan, students work on the given materials at their own pace. 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. Instructors then can focus on students that struggle with the material. 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. Class time under the Keller Plan is just a place for students to study their materials and take tests. Students who feel that they have mastered a 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.

Formal Languages is a core course in Computer Science Theory. It often includes topics on computability theory, complexity theory, state machines, and Turing machines. Formal Languages topics are applied in a number of real-world applications like compiler architectures and pattern matching. However, the material is mathematical and theoretical, requiring students to practice many difficult skills. The course uses a lot of models such as finite automata, push-down automata, and Turing machines. There are algorithms associated with each model that students must learn to apply. Currently, many books and instructors use simulators to help students understand the models and apply the associated algorithms. One state-of-the-art simulator is the Java Formal Languages and Automata Package (JFLAP) [6]. JFLAP simulates most of the models that are used in Formal Languages courses, so it helps students by allowing them to watch different models, apply algorithms on these models, or test the behavior of these models with different input strings. For example, a Finite State Machine simulator can help a student to understand which strings are accepted and which strings are rejected by the machine. This way, JFLAP increases student engagement and interaction with the course content [14].

We observe that often students read Formal Languages proofs and get nothing from them, yet continue on with their reading by skipping this material. We want to design an educational system that challenges students to engage at every step in the course. We seek to help students form a better understanding of the concepts taught in Formal Languages by forcing students to demonstrate their understanding at small increments. We do this by adopting PI principles. We seek to create an eTextbook will be made up of small frames. Each one includes a challenge question or problem that must be completed before they can continue to the next frame. We hypothesize that following this approach will increase student engagement, which will lead to a better understanding of the course content.

The OpenDSA project [4] provides infrastructure and content to build eTextbooks for different topics in computer science like Data Structures and Algorithms, Computational Thinking, or Formal Languages. OpenDSA eTextbooks are enhanced with embedded artifacts such as visualizations, exercises with automated assessment, and slideshows to improve understanding [9]. 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) library [7].

In the rest of this paper, we present our design and prototype implementation and evaluation plans for a new eTextbook to teach Formal Languages using the PI technique. The heart of the implementation is a modified form of JSAV slideshows built on the frames concept. A student will see a series of slides (the frames). Frames will add a constraint on the "next" button that normally appears in the slideshows. Students will not be able to click on the "next" button until 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.

By implementing the new eTextbook, we believe that students will gain a better intuition about Formal Languages content, which is hard to grasp when relying merely on textual discussion and static figures. Heavy use of interactive visualizations, as inspired by JFLAP, combined with the PI pedagogy of asking frequent questions, will increase student interaction, engagement, and practice with course content. Also, the eTextbook will keep track of students' click streams, scores, problem attempts, and time spent on each frame. These data will be used to improve the system, and to prove (or disprove) the effectiveness of the system in teaching Formal Languages.

Evaluating the pedagogical effectiveness of the new eTextbook is a challenge. It is a challenging task to measure student learning gains, especially if this gain is a result of technological interventions [3]. In this study, we will build our system over three phases: (1) traditional text prose, (2) prose with visualization and auto-assessed exercises, and (3) the PI-based eTextbook with frames. At each phase, we compare its effectiveness with the previous phases.

We also seek to create guidelines for other researchers who might seek to use PI principles to create other eTextbooks. The CS community knows how to design and implement eTextbooks, but has little experience with PI teaching techniques. Accordingly, a clear set of guidelines should be developed to motivate more work in developing this type of eTextbook.

2 The PI Approach

According to [10], there is no single approach to building PI materials. However, by studying PI systems, we can find that there are some commonalities. 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. This includes 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. This involves collecting data about students who will learn from the system: demography, pre-existing knowledge of the given topic, or other learner characteristics.
3. Behavior Analysis. The system designer designs materials 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). 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 appropriate because it enables students with high abilities to skip frames. This means that the PI system will skip some frames

for the student based on his earlier responses to the given questions. According to [11], there are no significant differences found in the effectiveness of learning between intrinsic and extrinsic CAI designs.

5. Sequence of Content: After determining the programming paradigm, the PI system designers should determine the sequence of frames. There is no standard sequencing for the frames, but there is research to suggest some possible approaches.
   - A typical PI sequence consists of an introduction, diagnostic section, theory section, teaching, testing section, practice section, and finally review or summary to reinforce the concepts addressed.
   - Pragmatic approach. The frame is based on the logical sequence of the material. The PI materials must simply address all necessary information and components.
   - RULES and EXAMPLES (RULEG) system [2]. This approach is useful when the material consists of rules and examples. Thus, the sequence will present a rule followed by an example to practice the rule.
   - EGRUL approach [12]. 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 synthesize the rule.
   - Conversational chaining [1]. In this approach, there is no programmed feedback. After the learner responds to a frame, he/she is given the solution in the following frame.
   - Mathetics (Backward Chaining) [5]. The learner begins with a frame that has a piece of information, for example, a minimized Deterministic Finite Automaton. The following frames will let the student work backward through the process that led to that piece of information, step by step (the algorithm steps that minimized the original Deterministic Finite Automaton).
6. Frame Composition. Programmed instruction frames should contain these essential components.
   - The information that the student should learn.
   - Incentive to obtain the targeted response. In other words, the frame should contain a question that leads the student to grasp the information.
   - Response that lets the student indicate that they have gained the desired knowledge.
   - 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 students able to answer the questions in the frames correctly. Prompts can be formal or thematic [17].
   (a) A formal prompt provides 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 frames is the question type. Questions are necessary and help students to acquire the intended learning outcomes. Many different types of questions can be used in frames like multiple choice, true or false, and labeling, name a few.

7. Evaluation and Revision. The final step is to evaluate the programmed product and revise it. The primary source of evaluation is learner feedback on the product. But learners are unable to judge some factors, like possible errors in content, accuracy, appropriateness, and relevance. Thus there should be external reviewers (field experts) to review the initial product.

3 Designing a Formal Languages eTextbook

3.1 Project Design

We are building our eTextbook within the OpenDSA eTextbook system in three phases. In the first phase, during Spring 2018, we used standard course materials to teach the Formal Languages course. While the "textbook" content was delivered through OpenDSA, it was largely text-based prose. We created thirteen (paper-based) homework assignments, two midterm exams, and a final exam. To construct the control data, we collected all student results from the homework assignments and exams.

In Spring 2019, (and planned for Fall 2019) students use the Phase 2 eTextbook. This builds on the existing OpenDSA eTextbook materials from Spring 2018 to include many visualizations for the various algorithms. We are adding to our collection of online exercises, and are gradually converting JFLAP to an integrated equivalent within OpenDSA (that we refer to as OpenFLAP). We also kept the same paper homework assignments and exam questions, collecting grades from them to use as a direct comparison with the Spring 2018 cohort from Phase 1.

The third phase eTextbook is planned to be available in Spring 2020. By then we hope to have converted the textbook content to be delivered as PI frames. The eTextbook will consist of a set of modules. Each module will be a set of frames. We will use the same homework and exams as in Phases 1 and 2.

By collecting meaningful data from these three phases, we can compare the impact of each phase on knowledge gain. For instance, we can compare Phase 1 and Phase 2 data to study the impact of evolving visualizations from prose. Comparing Phase 1 and Phase 3 outcomes lets us study the effectiveness of a PI eTextbook on student gains. Comparing Phase 2 and Phase 3 allows us to assess the relative impact of visualizations versus PI frames.

Another type of data we can collect is the student engagement with the course materials. In Phase 2 and Phase 3, we can record the time spent by students to finish each slideshow. In Phase 3, we can record more information like the number of attempts to solve each frame-related question, and the time

spent by students to solve each question. These data will constitute the time evaluation. Based on the results of time evaluation, we can conclude whether using PI materials lead to an increase in student engagement and interaction with the course content or not.

During our phases, different instructors, teaching assistants, and students will be involved, which may affect the quality of our results. For example, the instructor of the course in Phase 1 is different from the instructor for Phase 2. We expect the Phase 2 and Phase 3 instructors to be the same. The teaching assistants will be different for each instance of the course. These differences will affect some factors in the study, such as course presentation and explanation, homework, and exam grading.

To decrease the effects of variability, we are taking the following steps.

- We created a unified rubric for all homework given to the students in Phase 1. Therefore, different Teaching Assistants will use the same rubric to grade students answers for homework throughout all phases. This way we will limit the variability due to different Teaching Assistants. In Phase 3, we will implement a number of these exercises in the form of auto-assessed exercises.
- We use the same midterm and final exams throughout all phases of the project. Using the same exams will maintain grading consistency using a unified rubric. In this way, we will limit the effect of different graders that may yield to different grades on similar answers.

3.2 The Frames framework

We have created a JavaScript framework to support PI materials delivery. Frames are visually similar to existing JSAV slideshows. OpenDSA uses reStructuredText (RST) as its content authoring system. The Sphinx compiler combines RST files with JavaScript visualizations to form HTML pages that make up the OpenDSA modules. Adding frames to book modules will be similar for content authors to adding slideshows.

We implemented the required functions to gain control over the "next slide" button to enforce students to study the frame and satisfy the criterion that will allow them to move forward. Design changes give students a different look-and-feel to separate them from traditional slideshows. We have completed the support infrastructure, and are starting to write the actual PI materials needed for Phase 3. Figures 1 and 2 show PI-Frames examples.

3.3 Building the Programmed Instruction eTextbook

The eTextbook constitutes the main artifact for this project. It consists of a set of modules, each module consisting of a series of framesets. Each frameset presents visualizations to help the students understand a Formal Languages model, and the underlying algorithms that are applied to these models. While we have completed development of the Frames framework and the bulk of the Phase 2 book, this does not mean that the Phase 3 book will be implemented easily. We will face some challenges to complete the Phase 3 materials.

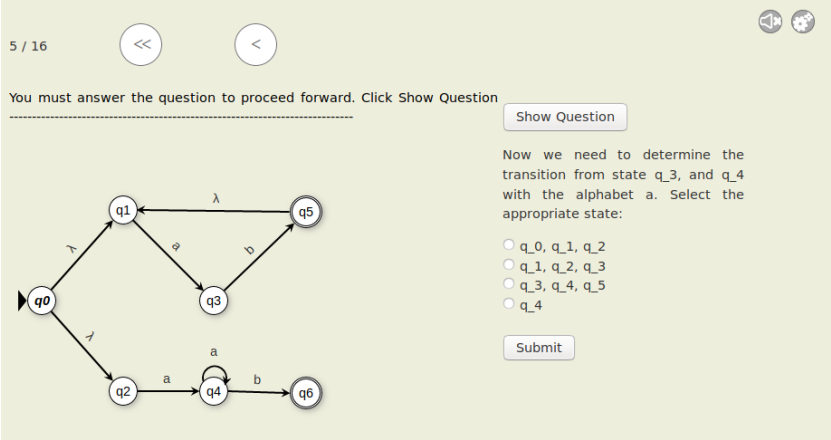

Fig. 1. An example PI-Frame with a question. Note that the "next" button is hidden until the student solves the question.

- Converting all textual descriptions into framesets. We need to think about converting each topic in the course to be in the form of frames. Each frame has a small piece of information and an appropriate question based on that given information. This conversion can be direct in some topics, for instance describing an algorithm, but it also can be hard for other topics.
- Framesets length. We need to develop the framesets to be of moderate length. We do not want to make students feel tired from completing a frameset that is too long.
- PI design decisions. There are some design decisions we have to take to produce the book. For instance, we must decide whether the framesets will be sequential or not. In each decision, there are pros and cons.
- Collecting data from student usage for the new book. We need to collect appropriate data related to student time and attempts on each frame. These data will help us to identify the impact of our system on student engagement.
- Define a mechanism to help struggling students. Based on the principals of the PI teaching technique, students cannot move forward from the current frame until they successfully satisfy the associated frame criterion. Of course, we do not want a student to be stuck on any frame for too long. We need students to try again by rereading the frame or previous frames. However, what if the student can not satisfy the criterion? Or what if there is a bug that makes it impossible to move forward by "correctly" answering the question? We need to find a mechanism that can help the students to go forward by giving the student the appropriate hints and guidance that will make the student try to satisfy the criterion again with a successful solution. These hints and guidelines will be frame specific and must be implemented in a way that ensures that the student understands the information in order to go forward. Alternatively, there may be value in providing a bypass mechanism, but this needs to be done in a way that will avoid abuse.

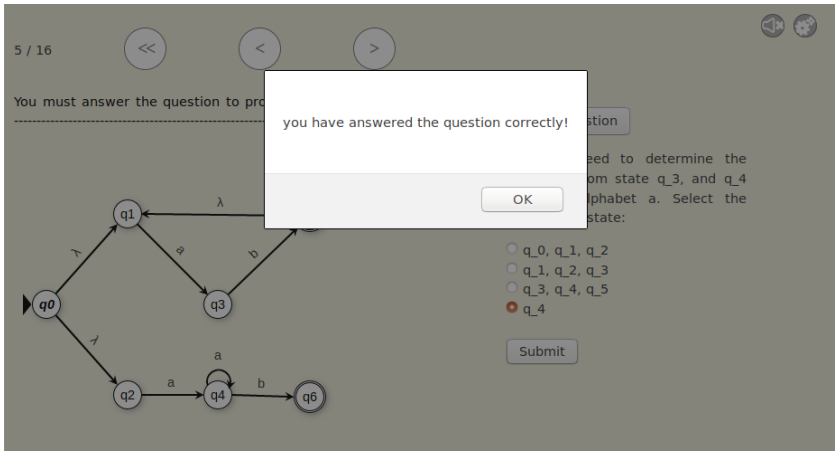

Fig. 2. Example PI-Frame after the student solves the question correctly.

4 Auto-graded Exercises

Our project includes creating auto-graded exercises. In the original Phase 1 course offering, we gave thirteen homework assignments to students. Many of these exercises can be converted from paper exercises to auto-assessed exercises. Doing that will help us save the time needed to grade these exercises, will standardize the grading policies for Phase 2 and Phase 3 homework assignments, and encourage other Formal Languages instructors to use our book. This is one of the value-added features that we can gain by converting JFLAP to its OpenFLAP version, and thereby take advantage of the exercise frameworks and expertise available from the OpenDSA project.

However, we will not be able to convert all questions to auto-graded proficiency exercises. The main reason is that many of the most valuable questions are mathematical in nature, requiring students to do things like write proofs. So while we know how to automate grading of exercises that involve constructing machines or showing the behavior of an algorithm, and we certainly can automate questions that can be cast as multiple choice or other common question formats, many important parts of the assessment process will have to be performed by human graders.

Fortunately many interesting questions can be auto-graded. We expect that many questions will be of the nature where students must create a finite state or other type of machine, similar to writing a program to solve a particular problem. Just as when writing a program, that can be auto-graded using test cases, we can auto-grade student machines by running test cases on them. To help instructors build such "proficiency" exercises, we created a web page, with an example in Figure 3. This guides the developer when designing the exercise. The exercise generator allows the developer to easily include "unit" test cases that will determine the correctness of the student's answer. The generator will produce an HTML page that can be embedded inside an OpenDSA module.

The main benefit from this generator is that instructors can easily generate any number of exercises to their students without thinking about the grading burden. When necessary, these exercises can easily be modified, so developers can improve their exercises. Students will be able to test their understanding of the content by trying these exercises.

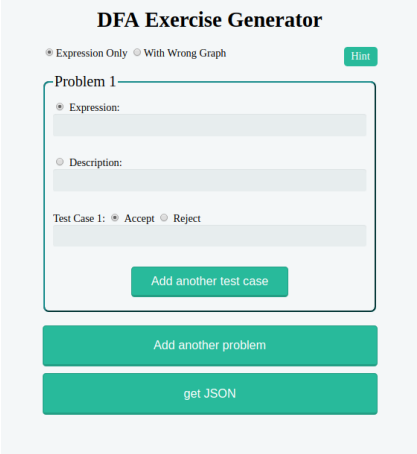

Fig. 3. The exercise generator web-page.

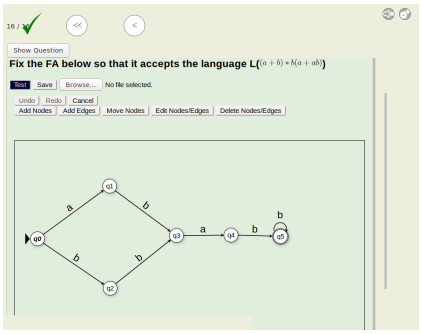

Fig. 4. Example of an auto-graded exercise inside a PI-Frame.

Auto-graded exercises can be integrated inside PI-Frames. Students use the framesets to understand the frame content, and prove their understanding by solving the frame question. Instructors can add auto-graded exercises in a frame. Figure 4 shows an auto-graded exercise that is displayed inside a PI-Frame.

5 Conclusions

OpenDSA provides infrastructure and materials to support courses in a wide variety of Computer Science-related topics. Traditionally, OpenDSA uses slideshows to explain and visualize Data Structures and Algorithms content. OpenDSA allows students to demonstrate knowledge of an algorithm by completing interactive exercises. However, static presentation of abstract, math-heavy material can be easily abused or skipped by students who quickly click through or skip the slides, or otherwise ignore the content. In general, static presentation is not an effective way for students to learn demanding material such as is presented in a Formal Languages course. In this paper we proposed to use Programmed Instruction (PI) pedagogy. PI is based on frames, small units of text along with a question or exercise that the student must answer before continuing to the next frame. We have implemented a frame-based system, and are collecting data to analyze the effectiveness of this approach.

References

1. John A Barlow. Conversational Chaining in Teaching Machine Programs. *Psychological Reports*, 7(2):187–193, 1960.
2. 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.
3. 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.
4. Eric Fouh, Ville Karavirta, Daniel A Breakiron, Sally Hamouda, Simin Hall, Thomas L Naps, and Clifford A Shaffer. Design and Architecture of an Interactive EText book–The OpenDSA System. *Science of Computer Programming*, 88:22–40, 2014.
5. Thomas F Gilbert. Mathetics: The Technology of Education. *Journal of Mathetics*, 1(1):7–74, 1962.
6. Eric Gramond and Susan H Rodger. Using JFLAP to Interact With Theorems in Automata Theory. In *ACM SIGCSE Bulletin*, pages 336–340, 1999.
7. 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, 2013.
8. Fred S Keller. GOOD-BYE, TEACHER 1. *Journal of applied behavior analysis*, 1(1):79–89, 1968.
9. Ari Korhonen, Thomas Naps, Charles Boisvert, Pilu Crescenzi, Ville Karavirta, Linda Mannila, Bradley Miller, Briana Morrison, Susan H Rodger, Rocky Ross, et al. Requirements and design strategies for open source interactive computer science ebooks. In *Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports*, pages 53–72. ACM, 2013.
10. Barbara Lockee, David Moore, and John Burton. Foundations of Programmed Instruction. *Handbook of research on educational communications and technology*, pages 545–569, 2004.
11. BB Lockee, MB Larson, JK Burton, and DM Moore. Programmed Technologies. *Handbook of Research on Educational Communications and Technology*, 3:187–197, 2008.
12. 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.
13. Michael Molenda. When Effectiveness Mattered. *TechTrends*, 52(2):53, 2008.
14. 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*, pages 403–407, 2009.
15. BF Skinner. Programmed Instruction Revisited. *Phi Delta Kappan*, 68(2):103–10, 1986.
16. Burrhus Frederic Skinner. Teaching Machines. *Science*, 128(3330):969–977, 1958.
17. Julian I Taber et al. Learning and Programmed Instruction. *ERIC*, 1965.