Active learning animations for the theory of computation
by Michael Thomas Grinder

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science
Montana State University

Abstract:
This dissertation presents the author's design, implementation, and evaluation of active learning animation software for teaching the theory of computation. The software builds on techniques used in traditional textbooks, in which concepts are illustrated with static diagrams. Developers of animation software have worked to make these traditional static diagrams come to life using motion, color, and sound (a process commonly referred to as animation), allowing students to manipulate and explore concepts in a fully interactive graphical environment. However, the mere vivification of static diagrams exploits only a small amount of the potential that modern personal computing environments provide. It is possible for animation software to make further use of this potential by providing learning activities that would be impractical or even impossible to duplicate using traditional methods.

To support this claim, the author developed software for simulating finite state automata (FSAs), the FSA Simulator. The FSA Simulator is designed for a variety of uses from in-class demonstrations to integration into a comprehensive "hypertext book." Although many others have developed similar software, the FSA Simulator advances a step beyond conventional automaton simulations. Using algorithms that compute the closure properties of regular languages, the FSA Simulator can be used to create interactive exercises that provide instant feedback to students and guide them toward correct solutions.

The effect of the FSA Simulator on students' learning was evaluated in preliminary experiments in undergraduate computer science laboratories at Montana State University. While these initial investigations cannot be considered either comprehensive or conclusive, they do indicate that use of the FSA Simulator significantly improves students' performance on exercises and may have some positive impact on students' ability to construct FSAs without the assistance of the Simulator.

The development of the FSA Simulator represents significant progress in creating and evaluating active learning animation software to support the teaching and learning of the theory of computation. The author has demonstrated that such software can be created, that it can be effective, and that students find such software more motivating than traditional teaching and learning resources.

ACTIVE LEARNING ANIMATIONS FOR THE THEORY OF COMPUTATION

by

Michael Thomas Grinder

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 2002

APPROVAL

of a dissertation submitted by

Michael Thomas Grinder

This dissertation has been read by each member of the dissertation committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Rockford J. Ross Rockford J. Ross 11/26/02
(Signature) Date

Approved for the Department of Computer Science

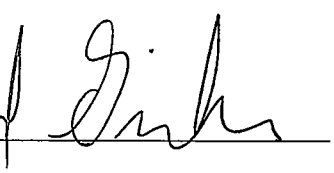
Rockford J. Ross Rockford J. Ross 11/26/02
(Signature) Date

Approved for the College of Graduate Studies

Bruce McLeod Bruce R. McLeod 11-26-02
(Signature) Date

STATEMENT OF PERMISSION TO USE

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this dissertation is allowable only for scholarly purposes, consistent with fair use as prescribed in the U.S. Copyright Law. Requests for extensive copying or reproduction of this dissertation should be referred to Bell & Howell Information and Learning, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted the exclusive right to reproduce and distribute my dissertation in and from microform along with the non-exclusive right to reproduce and distribute my abstract in any format in whole or in part.

Signature 

Date 11/26/2002

ACKNOWLEDGMENTS

Glory to God in all things.

Many people need to be thanked for their assistance in the process that brought me to this point:

- Rocky Ross, my dissertation advisor, for his advice, patience, and careful proofreading
- Jeff Adams, Fred Cady, Bob Cimikowski, Gary Harkin, and John Paxton, for serving on my dissertation committee
- Ann Defrance, Bob Cimikowski, and the TAs and students in CS 221 and 223 at Montana State University during Spring Semester 2002, for being my "guinea pigs"
- my parents, Edwin and Linda Grinder, and the rest of my family
- Lou Glassy, Chris Boroni, and Zuzana Gedeon, the other three "Horsemen"
- Marty Hamilton, for important advice about statistics
- Terri Dore and Jeannette Radcliffe

I do not have room to thank everyone else who helped me along the way. Please forgive my oversight.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - CONTINUED

ABSTRACT

This dissertation presents the author's design, implementation, and evaluation of active learning animation software for teaching the theory of computation. The software builds on techniques used in traditional textbooks, in which concepts are illustrated with static diagrams. Developers of animation software have worked to make these traditional static diagrams come to life using motion, color, and sound (a process commonly referred to as animation), allowing students to manipulate and explore concepts in a fully interactive graphical environment. However, the mere vivification of static diagrams exploits only a small amount of the potential that modern personal computing environments provide. It is possible for animation software to make further use of this potential by providing learning activities that would be impractical or even impossible to duplicate using traditional methods.

To support this claim, the author developed software for simulating finite state automata (FSAs), the FSA Simulator. The FSA Simulator is designed for a variety of uses from in-class demonstrations to integration into a comprehensive "hypertext-book." Although many others have developed similar software, the FSA Simulator advances a step beyond conventional automaton simulations. Using algorithms that compute the closure properties of regular languages, the FSA Simulator can be used to create interactive exercises that provide instant feedback to students and guide them toward correct solutions.

The effect of the FSA Simulator on students' learning was evaluated in preliminary experiments in undergraduate computer science laboratories at Montana State University. While these initial investigations cannot be considered either comprehensive or conclusive, they do indicate that use of the FSA Simulator significantly improves students' performance on exercises and may have some positive impact on students' ability to construct FSAs without the assistance of the Simulator.

The development of the FSA Simulator represents significant progress in creating and evaluating active learning animation software to support the teaching and learning of the theory of computation. The author has demonstrated that such software can be created, that it can be effective, and that students find such software more motivating than traditional teaching and learning resources.

CHAPTER 1

INTRODUCTION

This dissertation presents the author's design, implementation, and evaluation of active learning animation software for teaching the theory of computation. The software builds on techniques used in traditional textbooks, in which concepts are illustrated with static diagrams. Developers of animation software have worked to make these traditional static diagrams come to life using motion, color, and sound (a process commonly referred to as *animation*), allowing students to manipulate and explore concepts in a fully interactive graphical environment. However, the mere vivification of static diagrams exploits only a small amount of the potential that modern personal computing environments provide. It is possible for animation software to make further use of this potential by providing learning activities that would be impractical or even impossible to duplicate using traditional methods.

To support this claim, the author developed software for simulating finite state automata (FSAs), the FSA Simulator. The FSA Simulator is designed for a variety of uses from in-class demonstrations to integration into a comprehensive hypertext-book [15, 35]. Although many others have developed similar software (for example, [74, 5, 67, 36, 70]), the FSA Simulator advances a step beyond conventional automaton simulations. Using algorithms that compute the closure properties of regular languages, the FSA Simulator can be used to create interactive exercises that provide instant feedback to students and guide them toward correct solutions.

The effect of the FSA Simulator on students' learning was evaluated in prelimi-

nary experiments in undergraduate computer science laboratories at Montana State University. While these initial investigations cannot be considered either comprehensive or conclusive, they do indicate that use of the FSA Simulator significantly improves students' performance on exercises and may have some positive impact on students' ability to construct FSAs without the assistance of the Simulator.

Enhancing Computer-Aided Instruction with Active Learning

As personal computers have improved and become ubiquitous over the years, their potential for enhancing education has increased dramatically. From their earliest days, personal computers have been integrated into educational settings in one form or another. In the beginning, they were most often used in elementary school classrooms to aid rote memorization tasks such as learning basic arithmetic and spelling. As their capabilities increased, personal computers equipped with CD-ROM drives began to be used as reference tools. Electronic books and educational multimedia presentations were provided for online use. Many science labs were also equipped with computerized measurement devices. In recent years, especially with the arrival of the Internet, computers have routinely been used by students for doing research and collaborating with others through e-mail and other forms of electronic communication.

So far, most applications of computer technology within education have largely been passive. For example, electronic encyclopedias include photographs, diagrams, movies, and sound clips, but few accompanying opportunities for students to interact with the subject they are studying in a way that promotes active learning. Passive learning programs tap into only a small portion of the promise for enhanced

learning afforded by personal computers. It is well known that students learn better when they are engaged in active, rather than passive, learning [11, 52]. Thus, it is imperative that interactive learning software be developed in order to exploit the full potential of computer-enhanced learning.

Active Learning Software

In contrast to the computer-based passive learning environments alluded to above, active learning software provides opportunities for a student to directly control the animation of a concept being learned. The most elementary means of providing a student with active learning opportunities is to allow the student to submit differing inputs for the system to use during an animation of a concept. For example, a finite state automaton animator might allow a student to input a string for the automaton and then watch as the automaton processes it. While the software is animating the concept (e.g., a finite state automaton) based on the student's input, other opportunities for interaction are also provided: The student can often control the speed and appearance of the animation or even choose among several different views of the concept being animated.

More sophisticated active learning features include such things as allowing the learner to change the model being animated on the fly, providing facilities to a learner to allow construction of entirely new models for animation, and providing feedback to students to guide them toward successful completion of exercises. Examples of these features are provided as part of this dissertation.

Incorporating active learning into teaching and learning resources has numerous benefits. There is some evidence that active learning animation software, when

used properly, can improve student learning (see page 35). Even if animation software does not directly improve learning, its use often appears to markedly increase students' enthusiasm for a topic, indirectly improving their performance in a course.

There is also evidence that active learning animation software can also benefit those who already understand a concept. The process of creating or watching a visualization of a subject may trigger new insights. For example, an animation of sorting algorithms inspired a worst-case analysis of Shell-sort [24] and visualization software for teaching logic has caused researchers to reconsider the nature of reasoning [6].

Despite these benefits and the availability of many quality educational software packages, active learning animation software is not widely used. One can speculate about the reasons for this. One is that animation software for education was historically written for a single, specific computing environment, which precluded its use on other systems. Currently, with the popularity of the cross-platform Java programming language and the dominance of the Windows operating system, this is not as much of a problem as it was in the past.

A second possible reason for the lack of use of animation software is that it often requires complex installation and configuration. Since most animation packages usually only deal with one particular topic, an instructor wanting to use animation software in a course, must find, install, and integrate each animation system of interest into the course. Most instructors do not have the time to do this. Thus, helpful software often remains unused.

To alleviate these problems, an integrated, cross-platform learning environment that incorporates animation software is needed. One effort to develop such an environment is underway in the Webworks Laboratory of the Computer Science

Department at Montana State University. The eventual objective of the Webworks Laboratory is the construction of a framework that supports the development of hypertextbooks for the World Wide Web. A *hypertextbook* combines hyper-linked text, images, audio, and video, with active learning Java applets (the animations) to provide a dynamic, web-based teaching and learning resource that greatly extends the capabilities of traditional textbooks. Since hypertextbooks are based on cross-platform web technology, they can be used on any platform that has a Java-enabled web browser. Hypertextbooks can be distributed either over the Internet through a web site or on some form of electronic media such as a CD-ROM, requiring little or no installation or configuration. Since the animation software is already integrated into the text, no extra effort by the instructor is required to use animations in a course. Thus, hypertextbooks address most of the issues discussed above that have limited the adoption of active learning educational software in computer science courses. The work presented in this dissertation represents an important step towards making hypertextbooks a reality.

Conventional Computer Science Education

Conventional instructional methods have many drawbacks when used to teach the numerous dynamic processes found in computer science. For example, topics such as algorithms, data structures, and models of computation require descriptions of constantly changing information. Presentation of these topics can be accomplished, in part, by an instructor at a whiteboard using diagrams and illustrations, but it is still difficult using such means to clearly convey these ideas to students.

Dedicated instructors often hone their lecturing skills in order to improve their

presentation of complex, dynamic topics. A teaching style that actively engages students through dialog and that incorporates whiteboard diagrams of the subject is known to be effective [11], but it requires much effort and many years of experience to develop. Even then, the best lecturers do not get through to many students. Although students appear to comprehend the topic of a dynamic lecture as it is being presented, they must struggle to recapture this dynamic information later from their notes. As memories fade, the notes (which are a static representation of a dynamic event) often become a source of frustration and misunderstandings rather than a helpful study aid. Exacerbating the problems inherent in traditional teaching methods are large class sizes and heavy class loads that often prevent instructors from giving sufficient and timely feedback to their students on assignments and exams, further impeding the learning process.

Textbooks, being entirely static, have even more limitations in the presentation of dynamic concepts. Unlike teachers, books cannot be queried for additional information or alternative explanations. The clearest, most engaging texts often fail to successfully inform many students, even after repeated readings.

Clearly, conventional teaching methods are often not an efficient way to teach the inherently dynamic topics that are ubiquitous in computer science. Web-based, active learning resources offer powerful new ways of presenting information to augment traditional teaching and learning methods.

Unconventional Computer Science Education

Interactive animation software provides one possible solution to the limitations of traditional teaching and learning resources. Such software can present dynamic

information in ways that are virtually impossible to do with traditional methods. Animation software also has the advantage of being "repeatable." Students are able to review lecture examples exactly as they were presented in the classroom. When using animation software, students are not required to rely solely on their memories and cryptic handwritten notes to review material taught in class.

Additionally, rather than being restricted to the limited set of examples provided by an instructor in a classroom, students using active learning animation software have the opportunity to explore a topic in greater detail and to further deepen their knowledge of a subject. Unfortunately, the learning opportunities provided by animation software are not usually a sufficient incentive to provoke most students to investigate beyond what they need to know for an assignment or the next exam. True active learning software needs to entice students into becoming actively involved with the topic being animated. Carefully designed animation software will not only demonstrate a concept, it will also capture students' attention and guide them toward a proper understanding by providing feedback as they progress through a session using the software.

The research discussed in this dissertation is an initial step toward providing such an "unconventional" active learning environment for the theory of computation. Much more research and development needs to be done to provide a complete set of resources for teaching this topic, but we are well on the way towards our goal.

CHAPTER 2

LITERATURE REVIEW

Introduction

Animation software for computer science education can be divided into three general categories: *program animators*, *algorithm animators*, and *concept animators* [16]. *Program animators* allow the user to step line by line through the source code of a computer program as it executes. The user is provided with a view of variables and the program stack and can watch how each line of the program modifies the variable and stack values. *Algorithm animators* provide dynamic, graphical representations of the execution of an algorithm operating on a particular data structure of interest. *Concept animators* illustrate higher-level concepts in computer science, such as the execution of a theoretical model of computation. The dividing lines between these categories are somewhat fuzzy, since some educational software packages fit within the definitions of more than one category, but the distinctions are useful for gaining a general overview of the field. Prominent examples of each type of animator will be discussed below.

Although much of the animation software discussed in this chapter is not directly related to teaching the theory of computation, it is still important to review it. The author's animation software for simulating finite state automata was influenced by many of the animation projects that preceded it. Also, in an integrated hypertext-book environment, the FSA Simulator will need to be integrated with other types of animation software to provide a comprehensive view of the theory of computation

to its users.

Another important aspect of the study of animation software is evaluation. While many educators intuitively feel that active learning animation software helps their students learn, little empirical evaluation of animations has been done. Past efforts at empirical evaluation of animation software for computer science education will be discussed as well.

Evolution of Animators

All three types of animation software discussed in the previous section have passed through the same general evolutionary process. This process was fueled by improvements in computer hardware and by technology trends within the computing community. Many of these software systems were initially created in the late 1980s for specific computing platforms with very simple graphical or textual interfaces. As the speed and graphics capabilities of personal computers and low-end workstations increased in the early 1990s, animation software began utilizing more sophisticated graphical interfaces. However, most of these programs remained targeted at specific platforms, such as the Apple Macintosh, IBM PC, or a specific brand of Unix workstation, thus restricting their use to a small subset of the educational community that used various of these platforms in their curricula.

The release of the first version of the Java programming language in 1996 was a watershed event for the developers of educational animation software. The widespread adoption of Java by much of the computer industry made Java (and the Java virtual machine) an ideal platform for visualization software. The various animation software packages could be targeted to the Java virtual machine and then be

run without recompilation on any computer with a Java virtual machine installed, no matter what operating system the computer was running. During this time, many of the visualization software packages for computer science education were ported to Java and many new projects were written from scratch in the language.

Program Animators

As stated previously, program animators allow the user to step through the source code of a computer program as it runs. They provide a view of the program stack and display the changing values of variables as the program executes. Although program animators are similar to debuggers used for software development, program animators are specially designed for educational purposes. Some of the unique features of program animators include the ability to reverse execution at any point, automatic display of variable values, and special stepping modes that encourage students to predict the behavior of a program.

Dynalab

A prominent example of a program animator is Dynalab, which was developed at Montana State University under the direction of Rockford J. Ross. The Dynalab project started with the design of a virtual machine, known as the E-Machine [64], that allows reverse execution. An emulator for the E-Machine was developed shortly thereafter [9]. Once the E-Machine emulator was available, development of C, Ada, and Pascal compilers began [32, 33, 65]. As the compilers were created, the software for animating source-level programs in the corresponding high-level languages (i.e. the program animators) were developed concurrently. One program animator was