Realizing the Promise of Visualization in the Theory of Computing[1]

Joshua J. Cogliati[2], Frances W. Goosey[3], Michael T. Grinder[4], Bradley A. Pascoe[5], Rockford J. Ross[6], and Cheston J. Williams[7]

http://www.cs.montana.edu/webworks/projects/theoryportal/

Abstract

Progress on a hypertextbook on the theory of computing is presented. The hypertextbook is a novel teaching and learning resource built around Web technologies that incorporates text, sound, pictures, illustrations, slide shows, video clips, and—most importantly—active learning models of the key concepts of the theory of computing into an integrated resource. Active learning models currently exist for finite state automata, regular expressions, regular grammars, the pumping lemma for regular languages, context free grammars, LL(1) parsing, and program execution. The seamless interweaving of these components into a browser-ready whole will help realize the goal of integrating visualization aids into theory courses.

Categories and Subject Descriptors: K.3.2 [Computers and Education]: Computer and Information Science Education—*Computer science education*; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages—*Classes defined by grammars or automata*; F.1.1 [Computation by Abstract Devices]: Models of Computation—*Automata;* H.5.4 [Information Interfaces and Presentation]: Hypertext/Hypermedia

General Terms: Theory

Additional Keywords and Phrases: computer science education, theory of computing, regular languages, finite state automata, regular grammars, regular expressions, hypertextbook, active learning, learning applets

1. Introduction

Software systems for visualizing concepts in computer science have been around for many years (see, for example, [Stasko et al. 1997] for an overview). Most of this work has been focused on algorithm animation.

The theory of computing is a subject that lends itself naturally to visualization techniques: the many models of computation, algorithms that convert one model to

[1] This work has been supported in part by the National Science Foundation, grant numbers NSF 0088728 and NSF 0089397.
[2] Computer Science Department, Montana State University, Bozeman, MT 59717. cogliati@cs.montana.edu.
[3] Computer Science Department, Montana State University, Bozeman, MT 59717. goosey@cs.montana.edu.
[4] Computer Science Department, Montana Tech of the University of Montana, Butte, MT 59701, grinder@mtech.edu.
[5] Computer Science Department, Montana State University, Bozeman, MT 59717. pascoe@cs.montana.edu.
[6] Computer Science Department, Montana State University, Bozeman, MT 59717. ross@cs.montana.edu.
[7] Computer Science Department, Montana State University, Bozeman, MT 59717.

another, pumping lemmas, problem reductions, and virtually all other aspects of the theory beg to be visualized. Indeed, instructors generally spend a large proportion of their time in the classroom acting as visualizing agents, attempting to illustrate the dynamic nature of these topics. Unfortunately, whereas students may take notes of these demonstrations, once they leave the classroom they are stuck with only static reminders of the dynamic processes they watched in the classroom. The prospect of software that would allow instructors to present such topics in an animated, error-free, and repeatable fashion in the classroom, and that would empower students to study these topics on their own time as often and as extensively as desired using the same software, has long held a strong allure. Even more enticing was the hope that students would be able to learn the theory of computing better, or at least find themselves motivated and excited about learning the theory—a subject generally deemed the most difficult and least interesting in the computer science curriculum.

A number of software systems for visualizing various concepts from the theory of computing have been developed over time. Most have been "toy" versions produced locally, not widely distributed, and rarely maintained. Just a handful have become widely known (see, for example, [Chesnevar et al. 2003]). Among these, only a few represent concerted efforts to produce comprehensive resources for support of an entire course or course module on the theory of computing. Most notable are the works of Susan Rodger [Akingbade et al. 2003, Hung and Rodger 2000] and those of the authors of this paper [Boroni et al. 1999, Boroni et al. 2001, Grinder et al. 2002]. Readers are especially encouraged to visit http://www.cs.duke.edu/~rodger/tools/tools.html, home of Susan Rodger's theory tools web site.

In spite of the fact that some very good visualization software for the theory of computing has been developed, its use in the curriculum is not widespread. Indeed, this phenomenon is not restricted to the theory of computing. None of the excellent visualization systems for teaching and learning computer science has seen broad use. The reasons for this are probably not hard to understand in retrospect and have been discussed in the literature (see, for example, [Ross 2002, Naps et al. 2003]). These have largely to do with the amount of time an instructor must invest for a small return: visualization systems must be located on the Web, learned, possibly installed in the local computing environment, taught to students, and somehow integrated into the fabric of an existing course that may use different terminology and illustrations than those incorporated in the visualization software—all for perhaps a few lectures' worth of material (consider, for example, integrating the use of a software system that visualizes the actions of finite state automata into an existing course).

A second observation that had an initial chilling effect on the promise that visualization software would help students learn appeared in [Byrne et al. 1996] where it was reported that algorithm animation software did not seem to help students learn the algorithms involved. Later results, however, showed that students were indeed helped if the visualization software required active participation on the part of the students [Stasko 1997, Stasko 1999]. Indeed, in all of the reports on student learning an important fact was often overlooked: students certainly did not learn *worse* when using visualization

software and indeed were often much more motivated and excited about learning when using visualization software as opposed to traditional text-based resources alone [Ross 2002, Grinder 2003].

The message from these two observations for developers of visualization software is clear. To be effective and used, visualization software must be designed for active learning, and it must become an integrated part of a larger, comprehensive educational resource. In the rest of this paper we discuss our approach to the design of visualization software and our efforts to interweave it into the fabric of a hypertextbook on the theory of computing. This work represents the combined efforts of many students and colleagues under the direction of Rocky Ross, who serves as the director of the Webworks Laboratory at Montana State University where this work is underway. We restrict this discussion to just those components to be used in a chapter on finite state automata, regular expressions, and regular grammars.

2. Active Learning Models

We have developed a number of active learning applets and identified many others for inclusion in the hypertextbook on the theory of computing. Some of the inspiration for this work comes from the ideas of constructivism [Ben-Ari 2001] and mental models [Craik 1943, Gentner and Stevens 1983, Byrne 2000]. An *active learning applet* is interactive software that can be integrated seamlessly into the fabric of a hypertextbook at arbitrary points to illustrate a concept. When students encounter an active learning applet they must interact with it through mouse clicks and other responses in exploring the concept in question. Currently our repertoire includes active learning applets for helping students learn about

- deterministic and nondeterministic finite state automata
- regular expressions
- regular grammars
- various versions of the pumping lemma for regular languages
- context free grammars
- LL(1) and LR(1) parsing

We sometimes refer to the above kinds of active learning applets as active learning *models*, because the concepts visualized by these applets are models (e.g. of finite state automata, regular grammars, and so forth). We have also designed different sorts of active learning applets to support other aspects of teaching and learning. We sometimes refer to these kinds of applets as active learning *tools*. Among these active learning tools are

- a slide show presentation system
- a video clip display module
- a program animator

2.1 Versions of Active Learning Models

For applets intended for use as active learning models we have identified a number of variations that are helpful for teaching and learning.

A passive learning example version. At first glance it might seem a paradox to have instances of active learning models that are passive. However, in a teaching and learning environment a passive form of an active learning model turns out to be quite helpful for illustrating the model and how it is used when students first encounter it. The initial exposure to the finite state automaton active learning model, for instance, might be in an embedded example that just requires students to click a "step" button continuously to observe the series of state transitions that the illustrated finite state automaton performs as it processes a predefined input string (i.e., the students are not allowed to change the input string or modify the finite state automaton in this passive version of the applet). Passive learning versions of an applet are thus intended for use in situations in which an instructor or hypertextbook author wants to convey a point without any of the distractions that might occur if students were allowed simultaneously to change aspects of the model.

An active learning example version. Most examples that appear in the hypertextbook will call for active learning versions of the model applet. In these cases students are required to interact with the model in active learning mode. In the case of the finite state automaton model, for instance, the student may be required to provide different input strings to the finite state automaton.

An active learning exercise version. Exercises require more of active learning model applets. For example, in exercises involving finite state automata, students should be able to modify a finite state automaton in the applet or indeed to construct an entirely new one from scratch. Furthermore, to ensure the most benefit, the applet should provide feedback to the student about the correctness of the student's solution.

An authoring version. A version of the applet should be constructed that provides instructors and hypertextbook authors a way to construct passive learning examples, active learning examples, and active learning exercises easily.

A standalone application version. Although all of the active learning models are designed to be embedded as applets directly in the hypertextbook, it is also important that each of these models be available as standalone applications in an appendix of the hypertextbook. This allows for their use in independent projects and individual student exploration.

2.2 Design Considerations

Historically, visualization software has been developed to elucidate individual concepts in isolation. Scant consideration was given to the construction of a framework in which all designed applets would have common interface characteristics or would be able to

interact with each other. The initial efforts of the Webworks team were no different. However, as the concept of the hypertextbook evolved it became apparent that it would be beneficial to address these issues. Among the most important are

- a common design philosophy and structure based on XML
- similar graphical user interfaces
- the capability for step at a time execution
- an option for continuous execution
- smooth transitions between states
- sparing use of popup windows
- undo, or reversal, of steps
- audio for cues and voice narration of processes [Mayer and Anderson 1991, 1992]

3. The Finite State Automaton Active Learning Model

A snapshot of the exercise version of the finite state automaton active learning model is shown in figure 1. We can assume in this depiction that a student has been asked to construct a finite state automaton to recognize any binary string that ends in 101 or 110. In this case the student has apparently constructed a nondeterministic finite state automaton for this purpose and is testing it on the input string 101101.

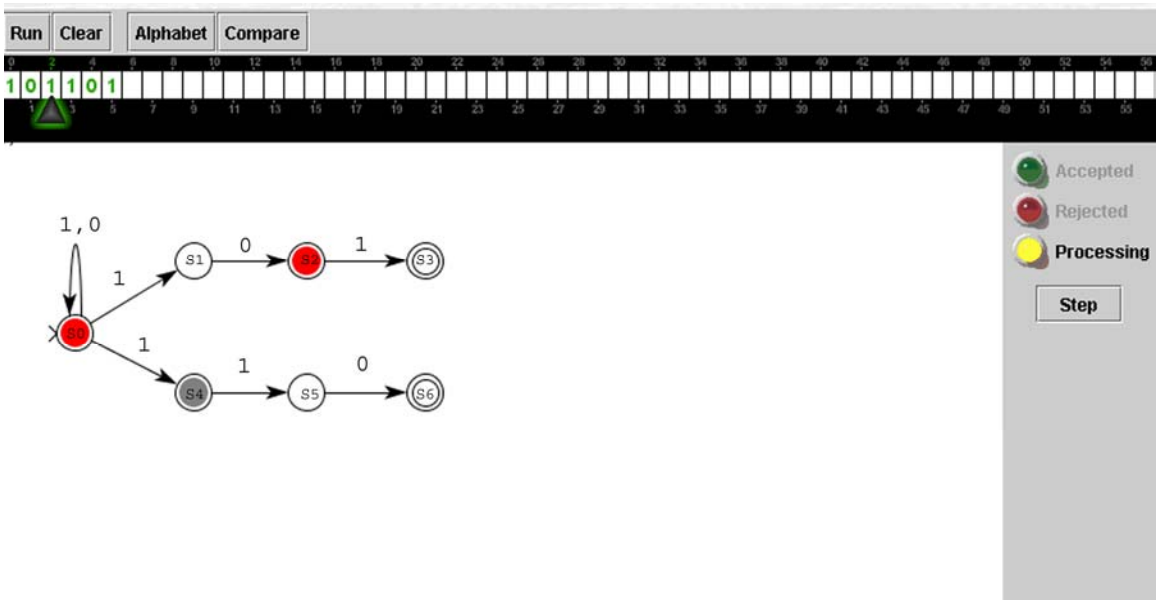


Figure 1. Finite State Automaton

Given the position of the input head on the string, it is clear that the student has clicked on the "Step" button (in the right panel of the applet window) twice, causing the automaton to process the leading 1 and 0 of the input string in succession and positioning the head to consume the next symbol (a 1). Notice that the automaton is concurrently in

states S0 and S2 (nondeterministically), as indicated by the red shading of these states. These are precisely the states at which the automaton would arrive upon processing the leading 1 and 0 of the input string. State 4, on the other hand, is colored gray to denote that processing of the previous input symbol (the 0) caused that nondeterministic branch of the automaton to "die."

Transitions are shown by smooth motion of the red disks from their current states across the correct arcs to the next states according to the input symbol being processed. It is known that presenting state transitions in any model in transitionally smooth fashion helps students see how a particular state of the model is arrived at from a previous state [Saariluoma 2000].

Note the three "indicator lights" on the right panel of the window. The yellow light is lit to indicate that the finite state automaton is in "Processing" mode. When the automaton has finished processing the entire input string the Processing light will turn off and either the "Accepted" or the "Rejected" light will illuminate to indicate whether the input string has been accepted or rejected, respectively.

Notice also the four buttons on the top panel of the window. The "Run" button positions the read head under the first symbol of the input string and prepares the automaton for processing after a string has been typed onto the tape. The "Clear" button clears the input string in preparation for entering a new string. The "Alphabet" button allows the user to select the alphabet symbols that can be used to form strings for a finite state automaton under construction. And the "Compare" button, when clicked, compares the language recognized by the student's constructed finite state automaton with the language of a correct (hidden) finite state automaton provided by the author of the exercise (see [Grinder 2003] for details). If the two languages are the same, the student receives a congratulatory message. If the two languages differ the student is given feedback about strings that the student's automaton accepts or rejects in error, allowing the student to repair the automaton.

Automata are constructed by simple mouse clicks that create states, and by click-and-drag operations to connect states with arrows. Similarly, provisions for labeling arrows with symbols from the chosen alphabet are made through mouse clicks. At any point during construction of a finite state automaton, states can be repositioned by clicking and dragging them to desired locations; the arrows and all labels follow automatically.

The work of Michael Grinder, the finite state automaton applet has undergone many revisions since its inception. Its definition in standard, display-independent XML notation and its feedback mechanism are all recent enhancements. Another recent extension, not shown in figure 1, is a state description tool that helps students learn that the states in a finite state automaton represent memory. For example, a description can be associated with a state that reads, "This state remembers that an even number of 1s has been seen in the in the input string so far." State descriptions can be read by hovering the mouse pointer over a state.

Finally, the finite state automaton active learning model includes sound effects that provide audio cues for the actions of the automaton as it processes a string.

4. The Regular Expression Active Learning Model

A snapshot of the exercise version of the regular expression active learning model in action is presented in figure 2. In this exercise version of the applet, the student is initially presented with a regular language; it is then up to the student to construct a correct regular expression for that language in the proper input pane (the one labeled "Enter Final Regular Expression:"). For this illustration the student has been asked to construct a regular expression that represents the language consisting of all floating point numbers in a particular programming language (the definition of this language is not shown in figure 2, as it was presented previously to the student under the tab "Regular Expression").

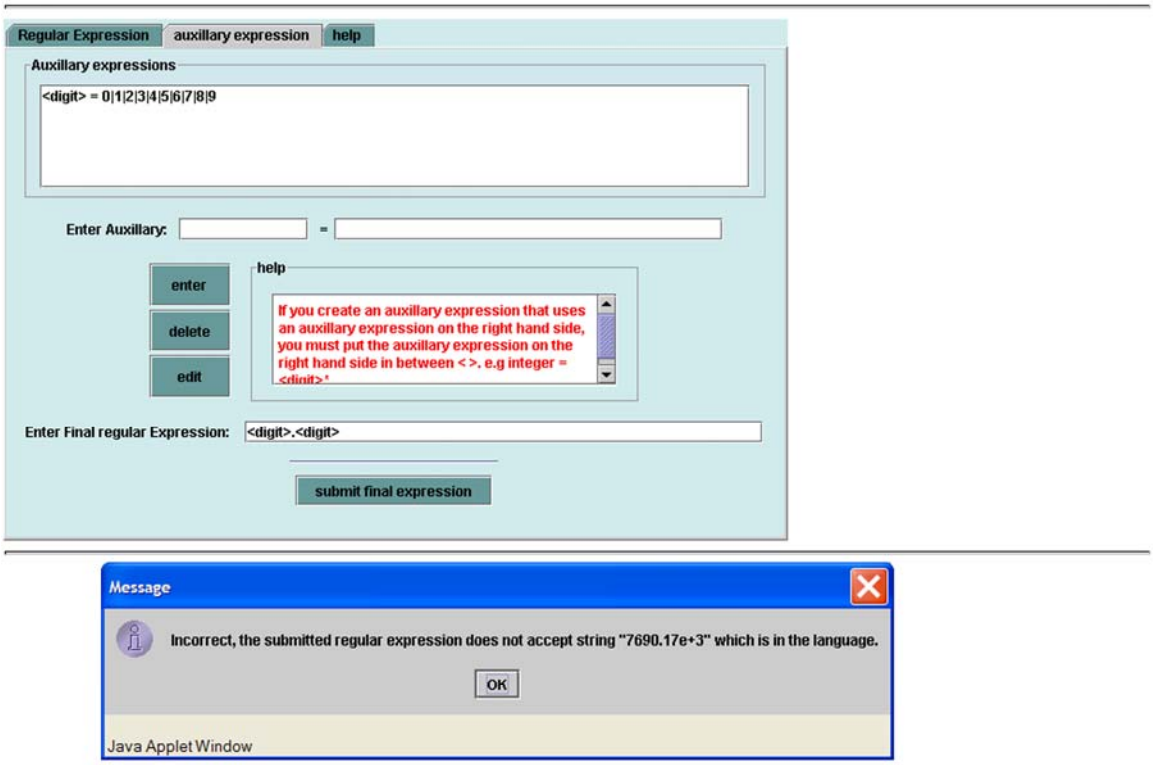


Figure 2. The Regular Expression Active Learning Model

Rather than requiring the student to construct a complex regular expression from individual characters, provision is made for defining auxiliary regular expressions that can be used in the final answer. For example, in figure 2, the student has chosen to construct an auxiliary regular expression named <digit> that denotes the decimal digits in the top input window. The student has then used this auxiliary regular expression to enter the following attempted solution to the exercise:

```
<digit>.<digit>
```

In this illustration, the student has apparently clicked on the "submit final expression" button at the bottom of the applet window to see whether his or her attempted solution is correct. The popup window indicates to the student that the solution is flawed and the message provides feedback to indicate why. (As a technical note, the feedback mechanism is based on the one described earlier for finite state automata. The student's submitted regular expression and a hidden, correct regular expression are both converted to finite state automata. Both of these automata are then fed to the finite state automata comparator that was previously constructed for the finite state automaton exercise active learning model to obtain feedback on whether the two automata—and hence, the two regular expressions—represent the same regular language.) The student can use this feedback to continue to work on the regular expression until it is correct.

The original version of this applet was the work of Katie Walsh and was reported in [Grinder et al. 2002]. Recent enhancements were made by Brad Pascoe, including the compare feature.

5. The Regular Grammar Active Learning Model

The regular grammar active learning model also has many features to help students learn about regular grammars. A set of grammars can be provided with each instance of this applet from which a student can select one for exploration. In the exercise version of this applet, a student can be required to create a regular grammar that generates a given regular language. This is the case shown in figure 3. Here a student has input a simple grammar to generate the language that is the set of all strings of length 0 or greater consisting only of *a*'s.

Once a grammar has either been selected from a list or constructed from scratch, the student can generate parse trees based on this grammar. An unexpanded nonterminal in the parse tree is selected by clicking on it, and then a rule from the list in the upper right pane of the applet is selected by clicking in turn on it. Finally, the chosen rule is applied to the selected nonterminal when the "EXPAND" button is clicked. Following the design rule that changes in a model image should be shown in transitionally smooth fashion, the lines from the nonterminal being expanded to its children are drawn one at a time. The speed of this drawing is controlled by the slider bar entitled "Animation Speed."

Notice that there are buttons that allow a student to undo rule applications (arbitrarily far), to select a node and collapse the entire sub tree beneath it for reconstruction, or to clear the entire tree and start over.

As with the finite state automaton and regular expressions active learning models, this applet also includes a compare feature that can be used in exercises to see whether the regular grammar constructed by a student generates the language specified in the exercise. Again, this comparison is carried out by an algorithm that converts the student's grammar and a hidden, correct grammar supplied by the author of the exercise to finite state automata, which are both fed to the finite state automata comparator for feedback.

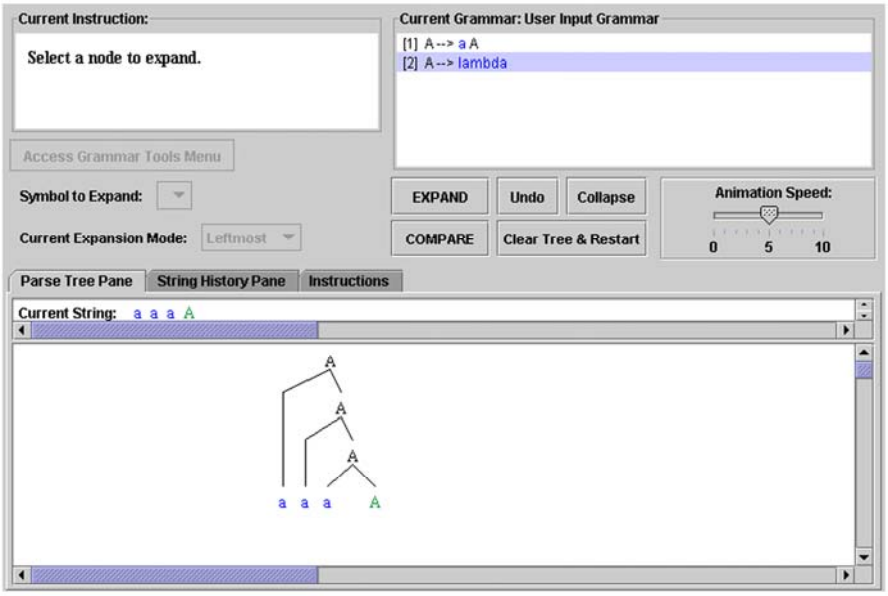


Figure 3. The Regular Grammar Active Learning Model

There are also three tabs in the applet. The first tab, labeled "Parse Tree Pane," shows the parse tree as it is being constructed, as depicted in figure 3. The second tab, labeled "String History Pane," shows an alternate view of the parse as a succession of intermediate strings in a standard derivation. The third tab reveals a help window that provides instructions on how to use the applet. The currently derived string in the parse (i.e., the leaves of the current parse tree) is maintained in the narrow pane just above the main lower pane. The slider bars on the lower pane ensure that arbitrarily large parse trees can be constructed and viewed.

This applet was the work of Teresa Lutey and was reported in [Grinder et al. 2002]. Extensions were made by Brad Pascoe, including the compare feature[8].

6. The Regular Language Pumping Lemma Active Learning Model

Developing active learning applets for the standard models of the theory of computing—finite state automata, regular expressions, and regular grammars—while time consuming and painstaking—is a fairly straightforward process. It is a greater challenge to design active learning applets that help students understand and apply various results of the theory. The pumping lemma for regular languages is one such result that students have tremendous difficulty grasping and applying.

[8] It should be noted that the standalone applications for creating finite state automata, regular expressions, and regular grammars can be used in exercises in which students create an instance of one of these according to written directions and save and submit their creation electronically to an instructor, who can then run the submissions against a correct instance for easy grading.

The purpose of the pumping lemma for regular languages is, of course, to provide a means of demonstrating that languages are *not* regular. However, correct application of the pumping lemma requires that students clearly understand the pumping lemma itself. That is, students must realize that the pumping lemma in its various forms reveals characteristics that all regular languages have, and, by implication, that any language that does not exhibit theses characteristics is therefore not regular. The first order of business is thus to help students learn what the pumping lemma is and why it is true. Among the facts to be learned are

- for each regular language there is a finite state automaton that recognizes it
- each finite state automaton has some finite, fixed number of states, k
- every string that has at least as many symbols in it as the number of states (k) in the finite state automaton processing the string is guaranteed to cause the automaton to loop while processing the first k symbols of the string (and, in fact, although just one loop can be guaranteed, there are likely many different loops encountered by the automaton while processing the first k symbols of an input string)
- furthermore, in any string being processed by a finite state automaton with k states, each substring of that string that has at least k symbols will cause the automaton to loop
- if the string being processed by a finite state automaton is accepted by that automaton and if it also causes the automaton to loop, new strings are readily constructed that will also be accepted by the automaton (and hence must be members of the regular language recognized by the automaton) by repeating the symbols of a substring that causes the automaton to loop, or by taking the symbols that cause the automaton to loop out of the string.

Our experience in teaching the theory course at both the undergraduate and graduate levels indicates that many students seem not to be able to grasp these facts readily and thus have trouble understanding their abstraction as the pumping lemma. The problem may well be one of time. Instructors cannot spend the time necessary to ensure that students really do understand all of the issues described above before moving on to the statement, proof, and application of the pumping lemma. Thus it seems apparent that active learning models that help students explore these issues in depth on their own time outside the classroom would help students learn the pumping lemma.

Josh Cogliati has designed and implemented an active learning model for this purpose. Figure 4 shows one version of this model.

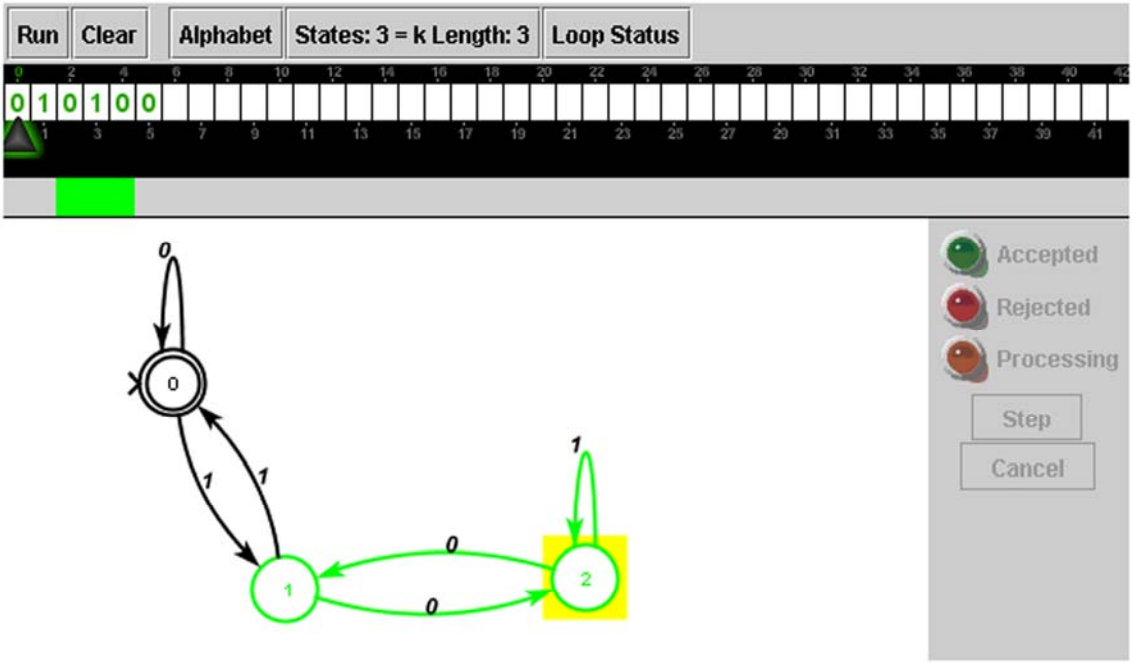


Figure 4. The Pumping Lemma Active Learning Model

This applet was constructed by extending the finite state automaton active learning model described earlier to incorporate features for elucidating the pumping lemma. There are versions of this applet for

- passive learning examples. Students can watch a preconfigured automaton locate the first loop encountered while processing the input string; the applet factors the input string into a prefix substring, *x*, a loop substring, *y*, and a suffix substring, *z*, in usual fashion as the input string is read by the automaton.
- active learning examples. Students can be required to supply strings for input and/or to select any substring of the input string of length greater than or equal to the number of states in the finite state automaton and watch the automaton identify the first encountered loop in the selected substring.
- active learning exercises. Students can be asked to identify a substring that causes the automaton to loop by highlighting a portion of the input string; figure 4 illustrates this mode of the applet. The green bar beneath the input string indicates that the student has identified the substring just above it as one that will cause the automaton to loop. The student then checks whether this selected substring causes the automaton to loop by running the automaton on the input string. The applet colors the transition arrows green that are encountered as the automaton reads the portion of the string that the student highlighted, illustrating for the student whether the selected substring actually causes the automaton to loop or not. Students can be required to locate all of the loops in a string.

We expect that the pumping lemma active learning model will help students understand the pumping lemma. The next step in this project, not yet completed, is to design and

implement an active learning model that leads students to a clear understanding of how to apply the pumping lemma to show that a language is not regular.

7. The Program Animator Active Learning Tool

A number of algorithms are integral parts of the theory of computing, including algorithms that convert one model to another (e.g., regular expressions to finite state automata and vice versa) and algorithms that implement the models themselves (e.g., programs that simulate finite state automata). These can be illustrated using the program animator applet, illustrated in figure 5 (note that the program animator is not intended to take the place of active learning visualizations of these processes—yet to be developed—but rather to augment them).

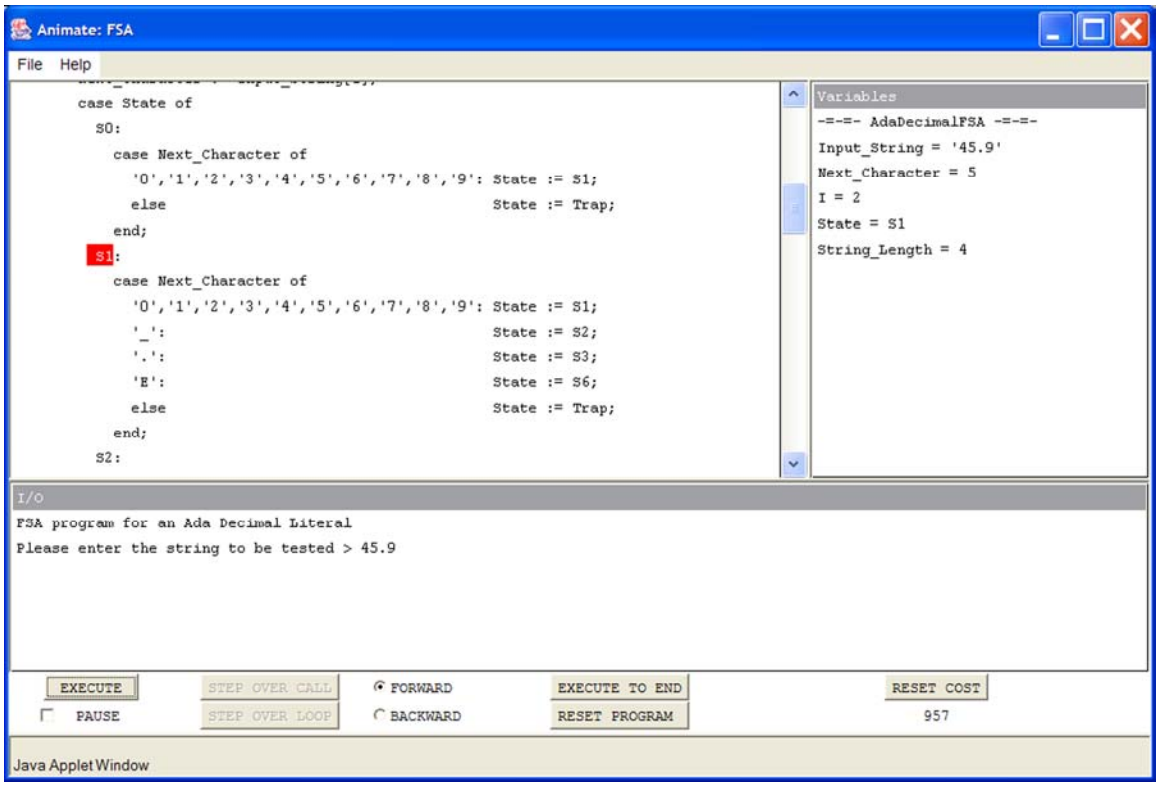


Figure 5. The Program Animator Active Learning Tool

This applet executes Pascal programs in a highly visual fashion (note that Pascal serves these days as a convenient pseudo language for expressing algorithms). The program animation applet allows a student to select a program to run from a library pulldown menu. Once the program is loaded it can be executed a step at a time or it can be set to execute without interruption. In step mode, the current line being executed is highlighted. Changing variable values are shown in the upper right pane of the applet window. Provisions are made for both forward and backward execution so that puzzling parts of the program can be reviewed as many times as desired. The "Cost" pane in the lower right corner of the applet window keeps track of the number of underlying virtual

machine instructions executed to provide for time complexity analysis of a running program.

The program in figure 5 is an implementation of a finite state automaton that recognizes the set of all valid Ada decimal literals. Students studying this program can learn one standard way of implementing finite state automata as programs.

The program animator was one of the first active learning applets to be developed in the Webworks Laboratory, and has been discussed in the literature numerous times (see, for example [Boroni et al. 1999]). It was converted to applet form by Frances Goosey.

8. The Slide Show Tool

The slide show applet, shown in figure 6, is the work of Brad Pascoe and was originally developed for a separate project underway in the Webworks Laboratory, the construction of a hypertextbook on the subject of biofilms [Ross 2003]. It soon became apparent that this tool would be useful in a hypertextbook on any subject.

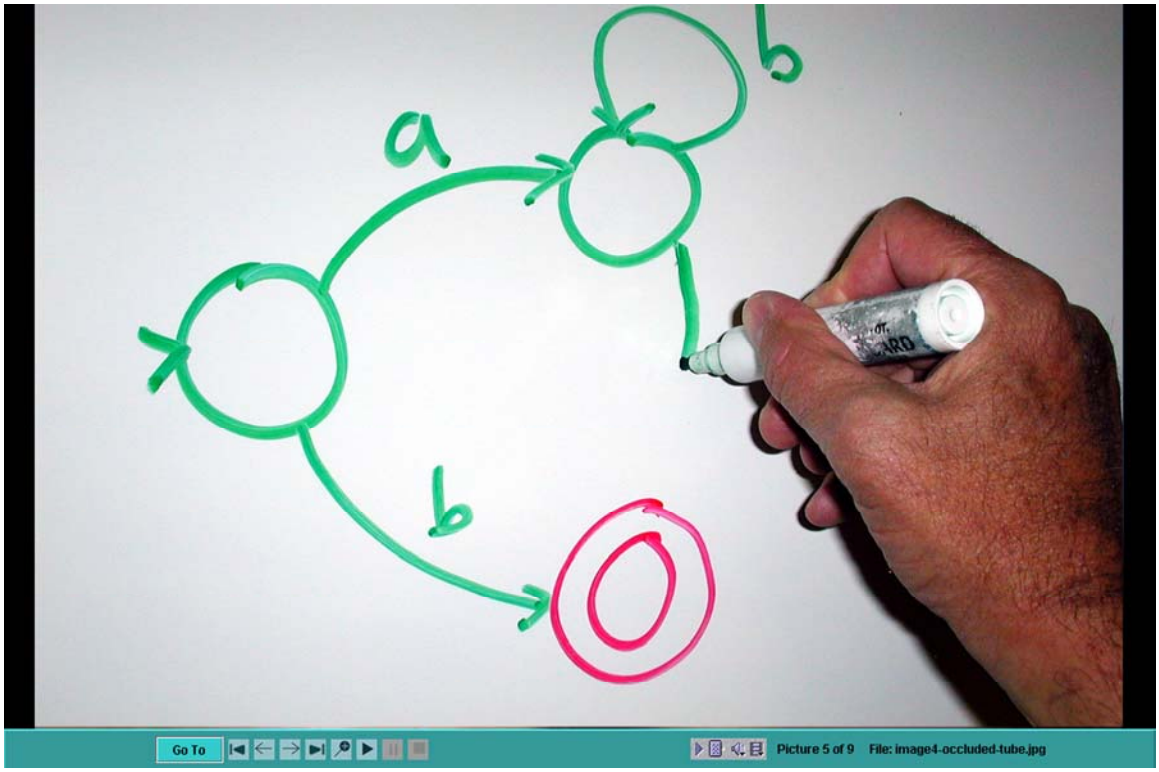


Figure 6. The Slide Show Applet

In the illustration of figure 6, the image shown is just one in a planned slide show that helps students learn how to construct a finite state automaton by hand in traditional "pencil and paper mode." A series of slides accompanied by voice narration would lead students from the beginning to the end of the construction a step at a time.

The various buttons on the lower bar of the slide show applet provide a student with many options for viewing a slide show. The "Go To" button brings up a list of all of the images in the show, and students can select one from the list to access immediately. The next buttons in succession from left to right allow a student to (1) go to the first slide in the show, (2) go to the previous slide in the show, (3) go to the next slide in the show, (4) go to the last slide in the show, (5) magnify the current image where desired an arbitrary amount, (6) start the slide show in automatic mode, (7) pause the slide show, and (8) stop the slide show. The next buttons off to the right in gray allow a student to play the accompanying audio narration or to turn the sound on or off.

One appeal of the slide show applet is that it takes up little space in a hypertextbook. Students can watch an entire presentation without being required to leave their position on the current page. Indeed, multiple slide shows could be included in one applet and students who need additional help could select as many different shows as desired from a pulldown menu, each, for example, showing the construction of a different finite state automaton by hand. Slide shows can thus help alleviate the problem of limited time that instructors face in a course when presenting certain topics. For instance, students could observe the instructor present the hand-construction of a finite state automaton in class a few times and then study as many additional presentations as needed on their own time by way of slide shows included in a hypertextbook that accompanies the class.

9. The Video Tool

The video tool has a purpose similar to the slide show tool. It provides a way for including video clips of processes important to the subject under consideration. Small "lecturelets" on a topic are prime candidates for a video clip.

The snapshot of the video applet presented in figure 7 is of a professor explaining the workings of a pushdown automaton at a whiteboard. The bar at the bottom of the applet includes a play button, an advance-to-beginning button, and advance-to-end button, a slider bar for arbitrary positioning in the video clip, and a volume button. Audio recording can be done at the time of filming, or a separate audio track can be inserted as desired. The video applet is the work of the Frances Goosey.

We have found that both the slide show and video applets have given us a tremendous benefit in the ongoing construction of the hypertextbook on the theory of computing. They serve a useful and effective purpose in their own right and will see liberal use in the hypertextbook. It turns out, however, that they can also be used to fill the gaps still remaining which we hope to fill with new active learning models. That is, the ultimate goal of our project is to have effective active learning models of the many important concepts in the theory of computing. These, however, take a great deal of time to develop and hone. In the interim we are able to resort to slide shows and videos in their stead to help students learn these concepts. For example, in completing the chapter on finite state automata, regular expressions, and regular grammars, it would be nice to have active learning model applets of the main conversion algorithms in place (e.g., from finite

state automata to regular expressions and vice versa). While these applets are being developed, the conversions can be explained in slide shows and/or videos, allowing construction of the hypertextbook to proceed.

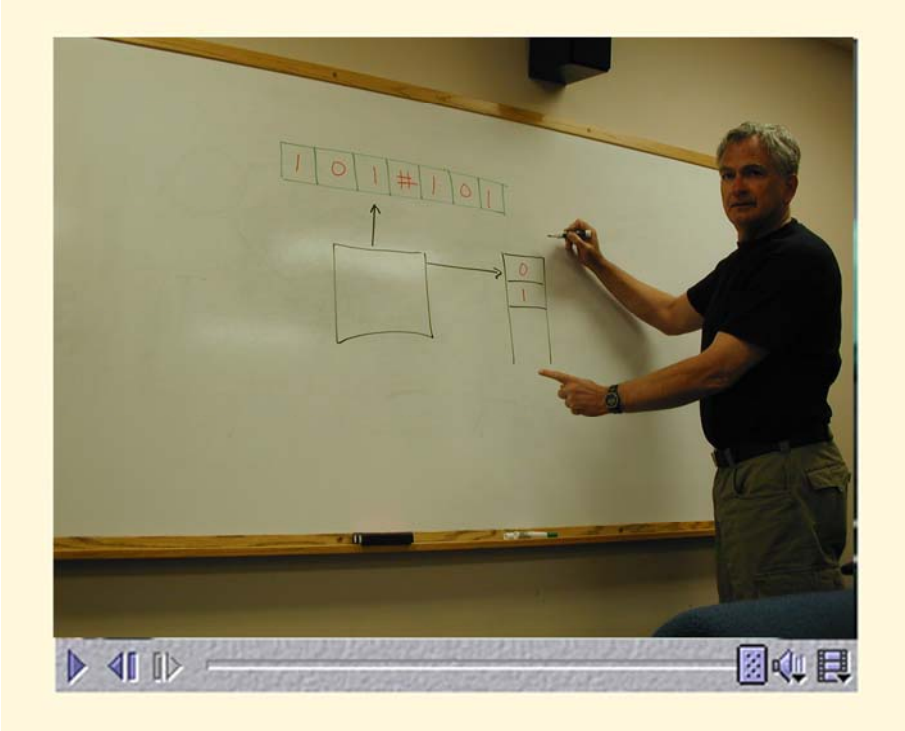


Figure 7. The Video Applet Tool

10. Tying it All Together—A Hypertextbook on the Theory of Computing

It should be clear that each of these active learning model and tool applets could be quite useful on their own. However, as noted at the beginning of this paper, standalone applets seem not to be widely used in the classroom for many reasons. It is therefore important that a comprehensive teaching and learning resource be developed and disseminated that seamlessly integrates standard text presentations of the material with the active learning models (see Greening 2000, Sutinen 2001, Ross 2002 for a comprehensive description of the concept of a hypertextbook).

There are a number of design objectives for the hypertextbook.

- It should be completely accessible in standard Web browsers, such as Netscape Navigator and Internet Explorer.
- It should work on any computer and operating system platform.
- It should be distributable on DVD or CD media.
- It should incorporate different levels of presentation of the material for different levels of learners.
- It should be easily modifiable and extensible.

Accessing the hypertextbook should be as simple as inserting a DVD into the player on a computer, which, in autostart mode, should immediately bring up the user's preferred browser with the cover page of the hypertextbook displayed, as illustrated in figure 8.

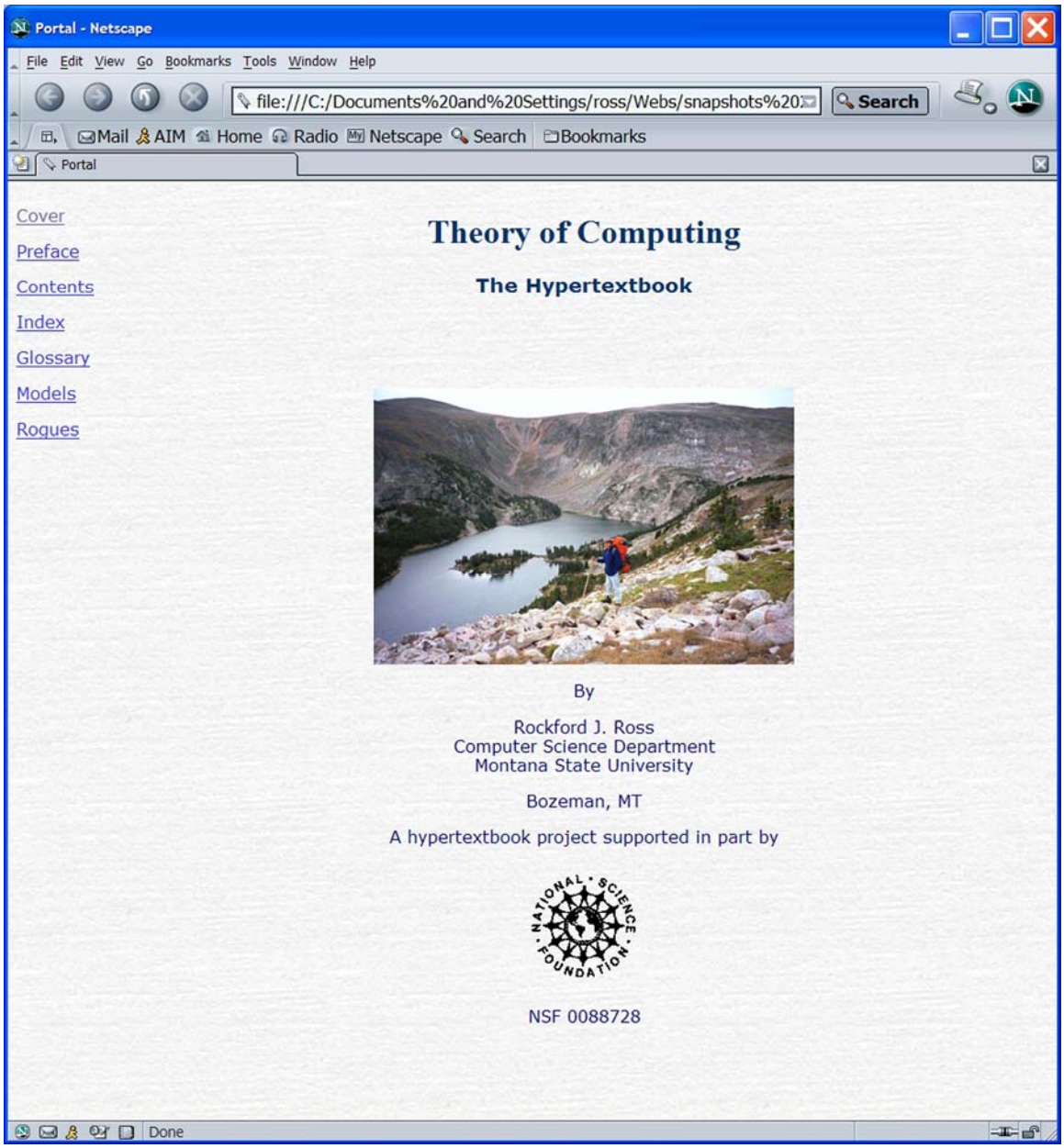


Figure 8. A Sample Hypertextbook Cover Page

Notice that the cover page appears in a Netscape Navigator window. All of the usual features of the browser with which all users are intimately familiar are available to the user, such as the forward, back, and refresh buttons as well as the bookmark option.

Along the left side of the window are links to important parts of the hypertextbook: the cover, preface, table of contents, index, glossary, models (which allow students access to the active learning models—embedded elsewhere throughout the hypertextbook as applets—in standalone mode as applications), and a rogues gallery of contributors to the project. This set of links is replicated on all pages of the hypertextbook for easy navigation.

Following the link to the table of contents brings up a page similar to the one shown in figure 9. Notice that below each chapter entry are three symbols: a green circle, a blue square, and a black diamond. These are the international ski industry symbols for "easy way," "intermediate way", and "challenging way" down the mountain. Clicking on these symbols leads the user to a presentation of the material described in the chapter heading for novice students, intermediate students, or advanced students, respectively. The green circle routes through the material will include many instances of the active learning models and tools to help students new to the material to learn well. The next two levels make increasingly less use of the models and tools and rely more increasingly on abstract mathematical presentations. Of course, students are free to move back and forth between the levels as needed.

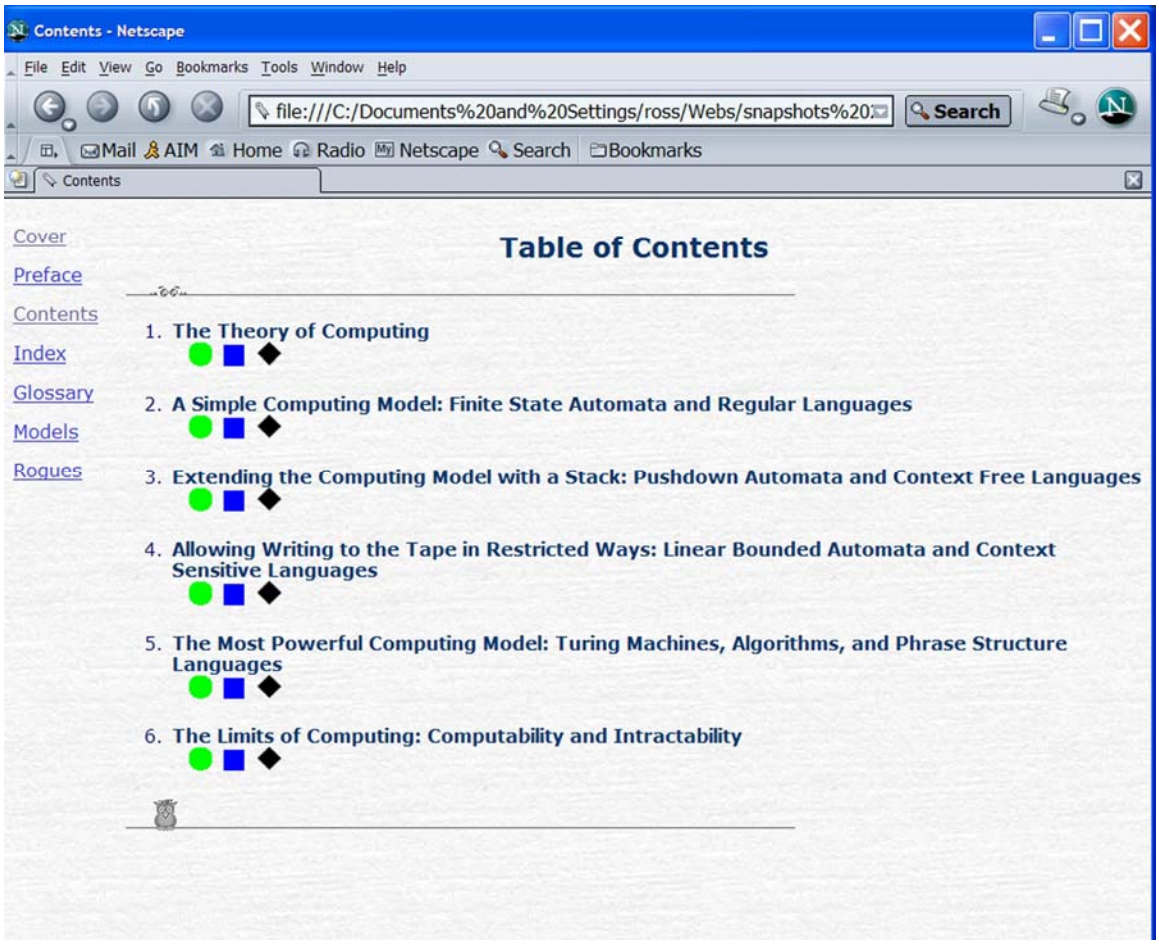
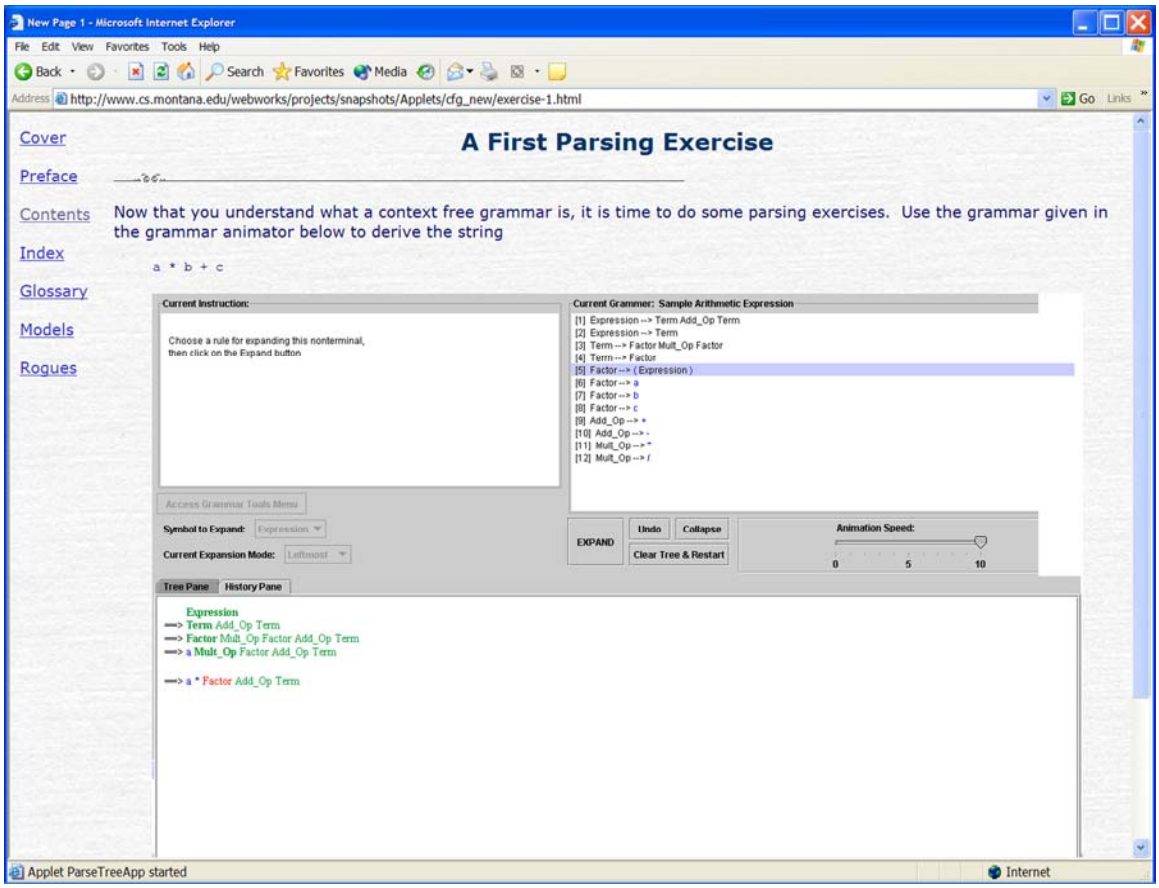


Figure 9. Table of Contents of the Hypertextbook

Finally, individual "pages" in the hypertextbook integrate text, graphics, and the active learning model and tool applets into a seamless presentation of the material. Figure 10 shows a truncated sample page from the hypertextbook. Notice that the context free grammar active learning model applet shown is embedded seamlessly into the surrounding text that describes an exercise to be performed. The left side of the page has the usual links to other parts of the hypertextbook as described earlier.



11. Summary

We have described a number of existing active learning models and tools that can be used in standalone mode or included seamlessly within the fabric of a hypertextbook on the theory of computing. We further elaborated on the concept of a hypertextbook. The eventual goal of the project is the construction of a hypertextbook that serves as a complete teaching and learning resource for an undergraduate or graduate course on the theory of computing. As the project progresses we expect to release chapters of the hypertextbook as they are completed.

It should be clear that there are many more active learning models that should be constructed as this project progresses, including ones that illuminate such advanced concepts as problem reductions and NP-completeness. Indeed, such a project is likely never to see true completion. There will always be room for new and improved active learning applets and for inclusion of fresh material. We are now at the point where a first chapter on finite state automata, regular expressions, and regular grammars is doable, and we expect the others to follow in short order.

Finally, we expect the hypertextbook to solve the problem of the lack of use of visualization software in the classroom. Since the visualization software is part and parcel of the hypertextbook in the form of seamlessly interwoven active learning applets, the visualization tools will be used as a matter of course. Progress can be monitored on the project website [Ross 1999].

References

AKINGBADE, A., FINLEY, T., JACKSON, D., PATEL, P., AND RODGER, S.H. 2003 Jawaa: Easy Web-Based Animation from CS 0 to Advanced CS Courses. In *Thirty-fourth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 35, pages 162-166, March 2003.

BEN-ARI, M. 2001 Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching*, 20(1):45-73, 2001.

BORONI, C.M., GOOSEY, F.W., GRINDER, M.T., LAMBERT, J.L., AND ROSS, R.J. 1999 Tying it all Together Creating Self-Contained, Animated Interactive, Web-Based Resources for Computer Science Education. In *Thirtieth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 31, number 1, pages 7-11, March 1999.

BORONI, C.M., GOOSEY, F.W., GRINDER, M T., AND ROSS, R.J. 2001 Engaging Students with Active Learning Resources: Hypertextbooks for the Web. In *Thirty Second SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 33, number 1, pages 65-69, March 2001.

BYRNE, M.D., CATRAMBONE, R., AND STASKO, J.T. 1996 Do Algorithm Animations Aid Learning? Technical Report GIT-GVU-96-18, Georgia Institute of Technology, Atlanta, GA 30332-0280, August 1996. The results here are not negative, but they are inconclusive.

BYRNE, R. *Mental Models Website*. May 2000
http://www.tcd.ie/Psychology/Ruth_Byrne/mental_models/index.html

CHESNEVAR, C.I., COBO, M.L., AND YURCIK, W. 2003 Using Theoretical Computer Simulators for Formal Languages and Automata Theory. In *ITiCSE 2002 Working Group Reports (SIGCSE Bulletin)*, volume 35, pages 33-37, June 2003.

CRAIK, K. 1943 *The Nature of Explanation*. Cambridge University Press.

GENTNER, D. AND STEVENS, A.L. (Eds) 1983 *Mental Models*. Lawrence Erlbaum Associates, Hillsdale, NJ.

GREENING, T. (Ed.) 2000 *Computer Science Education in the 21st Century*, chapter Shifting Paradigms: Teaching and Learning in an Animated, Web-Connected World, pages 173-193. Springer Verlag, 2000. Invited chapter by Rockford J. Ross.

GRINDER, M.T. 2003 A Preliminary Empirical Evaluation of the Effectiveness of a Finite State Automaton Animator. In *Twenty-fourth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 35, pages 157-161, March 2003.

GRINDER, M.T., KIM, S.B., LUTEY, L. ROSS, R.J. AND WALSH K.F. 2002 Loving to Learn Theory: Active Learning Modules for the Theory of Computing. In *ThirtyThird SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 34, number 1, pages 371-375, February 2002.

HUNG, T. AND RODGER, S.H.. Increasing Visualization and Interaction in the Automata Theory Course. In *Thirty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 32, number 1, pages 6-10, March 2000.

MAYER, R.E. AND ANDERSON, R.B. 1991 Animations Need Narrations. *Journal of Educational Psychology*, 83(4):484-490, 1991.

MAYER, R.E. AND ANDERSON, R.B. 1992 Helping Students Build Connections Between Words and Pictures in Multimedia Learning. *Journal of Educational Psychology*, 84(4):444, 1992.

NAPS, T. and et al. 2003 Evaluating the Educational Impact of Visualization. A working group report from ITiCSE 2003, July 2003.

ROSS, R.J. 1999 Webworks web site. http://www.cs.montana.edu/webworks, 1999.

ROSS, R.J. 2003 Snapshots of Slime. In *SIGACT News (Education Forum)*, volume 34, number 4, pages 78-83, December 2003.

ROSS, R. 2002 Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web. In *Software Visualization (LNCS 2269)*, pages 269-283. Springer Verlag, 2002. Diehl S. (Ed.).

SAARILUOMA, P. 2000 *Image and Interface: Some Psychological Aspects of Visualisation*. Report given at PVW 2000. http://cs.joensuu.fi/pages/pvw/saariluoma.htm

STASKO, J., DOMNGUE, J., BROWN, M.H., AND PRICE, B.A. 1997 Software Visualization: Programming as a Multimedia Experience. MIT Press.

STASKO, J. Evaluating Animations as Student Aids in Learning Computer Algorithms. *Computers & Education*, 33(4):253-278, 1999. This paper shows more positive results than the 1996 paper.

STASKO, J.T. Using Student-Built Algorithm Animations as Learning Aids. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education (SIGCSE Bulletin)*, volume 29, number 1, pages 25-29, March 1997.

SUTINEN, E. (Ed.). *Proceedings of the First Program Visualization Workshop*, chapter Hypertextbooks for the Web, pages 221-233. University of Joensuu, 2001. Chapter by Rockford J. Ross.