Realizing the Promise of Visualization in the Theory of Computing

JOSHUA J. COGLIATI, AND FRANCES W. GOOSEY
Montana State University
MICHAEL T. GRINDER
Montana Tech of the University of Montana
BRADLEY A. PASCOE, ROCKFORD J. ROSS, AND CHESTON J. WILLIAMS
Montana State University

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 Key Words 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 is focused on algorithm animation.

The theory of computing lends itself naturally to visualization techniques: many models of computation, algorithms that convert one model to 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 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 dynamic processes. The prospect of software that would allow instructors to present such topics in an animated, error-free, and repeatable fashion in the classroom, and empower students to study these topics on their

This research was supported in part by National Science Foundation grant DUE-0127483/0353132.
Author's address: ross@cs.montana.edu.

own as often and as extensively as desired using the same software, has long held a strong allure. The primary hope is that students would learn the theory of computing better, or at least become 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 in the theory of computing have 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 to support 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 article [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.)

Despite 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 are probably not hard to understand in retrospect, and have been discussed in the literature [Ross 2002; Naps et al. 2003]. The reasons 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 integrated into the fabric of an existing course, which may use a different terminology and illustrations than 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 proposition 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. But later results show that students were helped when the visualization software required active participation by the students [Stasko 1997; 1999]. Indeed an important fact often overlooked in reports on student learning is that students certainly do not learn *worse* when using visualization software, but are 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 for developers of visualization software is clear. To be effective and used, visualization software must be designed for active learning, and must become an integrated part of a larger, comprehensive educational resource. In the rest of this article we discuss our approach to the design of visualization software and our effort to interweave it into the fabric of a hypertextbook. This work is the combined effort of many students and colleagues under the direction of Rocky Ross, the director of Webworks Laboratory at Montana State University where this work is underway. We restrict this discussion to just those components that would appear in a hypertextbook 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 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. Currently, our repertoire includes active learning applets for helping students learn the following:

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

We sometimes refer to the kinds of active learning applets above as active learning *models*, since the concepts visualized by them are models (e.g., of finite state automata, regular grammars). We also designed different sorts of active learning applets to support other aspects of teaching and learning, and sometimes refer to these applets as active learning *tools*; among them are the following:

--a slide show presentation system;
--a video clip display module; and
--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 passive instances of active learning models. 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 its use when students first encounter it. The initial exposure to the finite state automaton active learning model might, for instance, be in an embedded example that only requires students to continuously click a "step" button to observe the series of state transitions the illustrated finite state automaton performs as it processes a predefined input string (the students are not allowed to change the input string or modify the finite state automaton). Passive learning versions of an applet are intended for use when an instructor or hypertextbook author wants to convey a point without any of the distractions that might occur if students were allowed to simultaneously change aspects of the model.

An active learning example version. Most examples 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 his solution.

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

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 stand-alone 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 was 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 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; the following are among the most important:

--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 pop-up windows;
--undo, or reversal, of steps; and
--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. In this depiction we assume that a student was asked to construct a

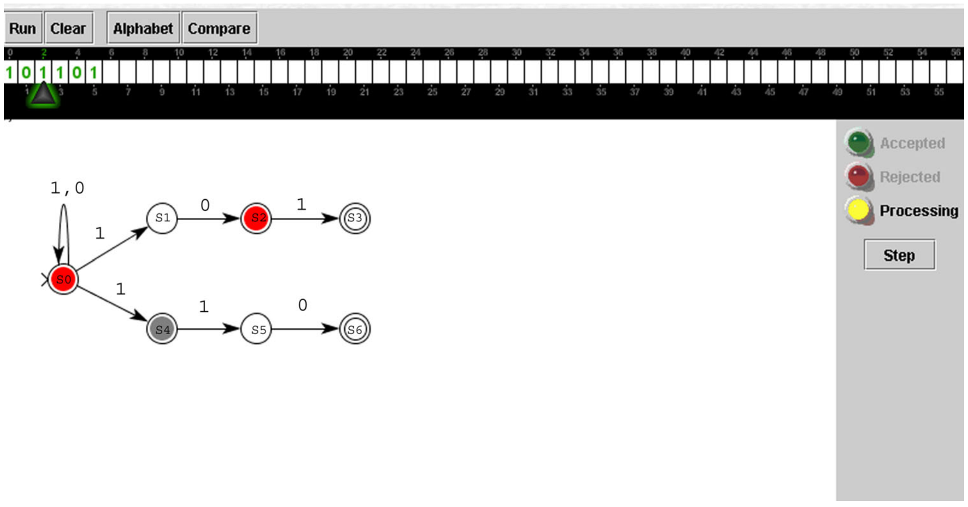


Fig. 1. Finite State Automaton.

finite state automaton to recognize any binary string that ends in 101 or 110. In this case, the student has constructed a nondeterministic finite state automaton and is testing it on the input string 101101.

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 (1). Note that the automaton is concurrently in states S0 and S2 (nondeterministically), as indicated by the red shading. 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 a transitionally smooth fashion helps students see how a particular state of the model is arrived at from a previous state [Saariluoma 2000].

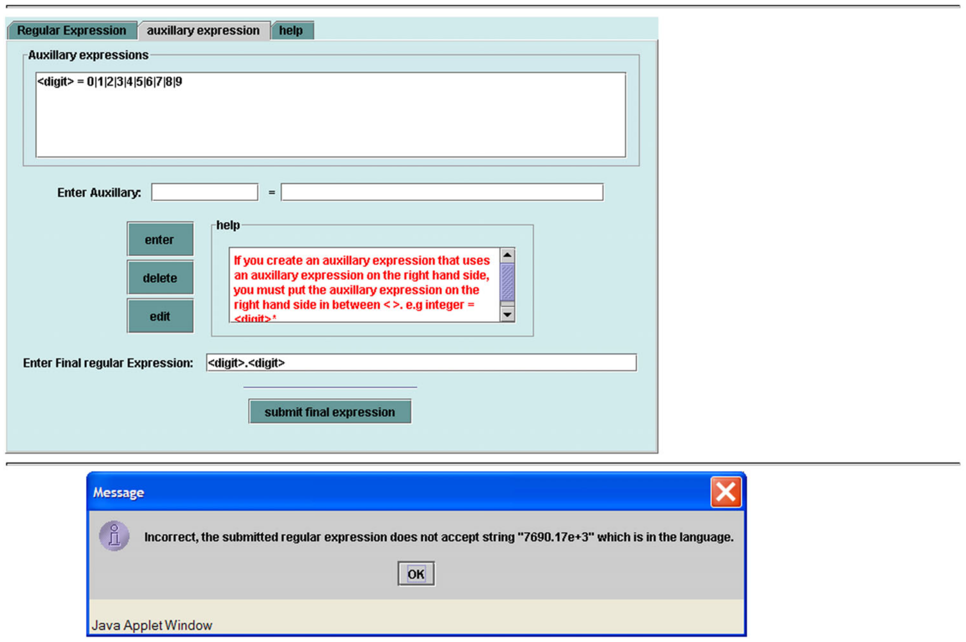
Note the three "indicator lights" on the right panel of the window. The yellow light indicates that the finite state automaton is in "processing" mode. When the automaton has finished processing the entire input string, the processing light will turn off and either the "accepted" or the "rejected" light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

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, that is, 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 that "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 holding 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.



<digit> = 0|1|2|3|4|5|6|7|8|9

If you create an auxillary expression that uses an auxillary expression on the right hand side, you must put the auxillary expression on the right hand side in between < >. e.g integer = <digit>*

<digit>.<digit>

Incorrect, the submitted regular expression does not accept string "7690.17e+3" which is in the language.

Fig. 2. The regular expression active learning model

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 is 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").

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 called <digit> that denotes the decimal digits in the top input window. The student then used this auxiliary regular expression to enter the following attempted solution to the exercise:

<digit>.<digit>

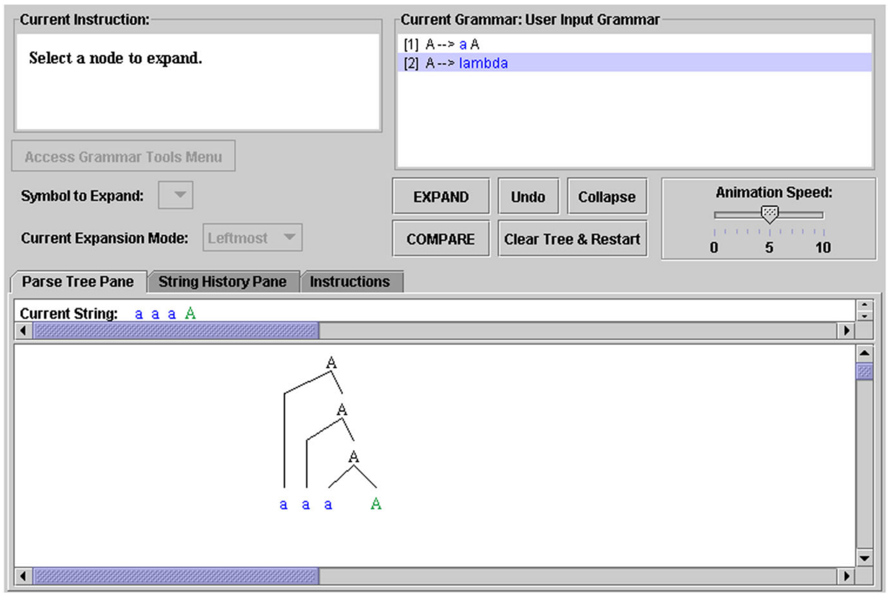
The student has clicked on the "submit final expression" button at the bottom of the applet window to see whether his or her attempted solution is correct. The pop-up window indicates to the student that the solution is flawed and the message provides feedback to indicate why. (As a technical note, the 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 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 is the work of Katie Walsh, 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 that 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."


Current Instruction:

Select a node to expand.

Current Grammar: User Input Grammar

[1] A --> a A

[2] A --> lambda

Current String: a a a A

Fig. 3. The regular grammar active learning model.

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

As with the finite state automaton and the 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.

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 is the work of Teresa Lutey, and was reported in Grinder et al. [2002]. Extensions were made by Brad Pascoe, including the compare feature.[1]

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.

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 not regular. Thus the first order of business is to help students learn what the pumping lemma is and why it is true; the following facts are among those to be learned:

--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, 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);

[1] It should be noted that the stand-alone 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.

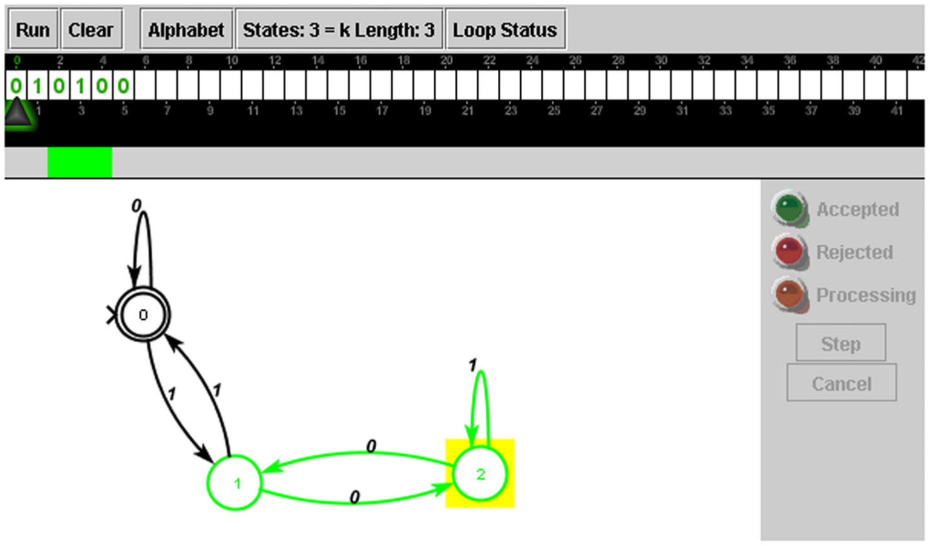


Fig. 4. The pumping lemma active learning model.

-- 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; and

--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 do not 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 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.

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 the usual fashion as the input string is read by the automaton;

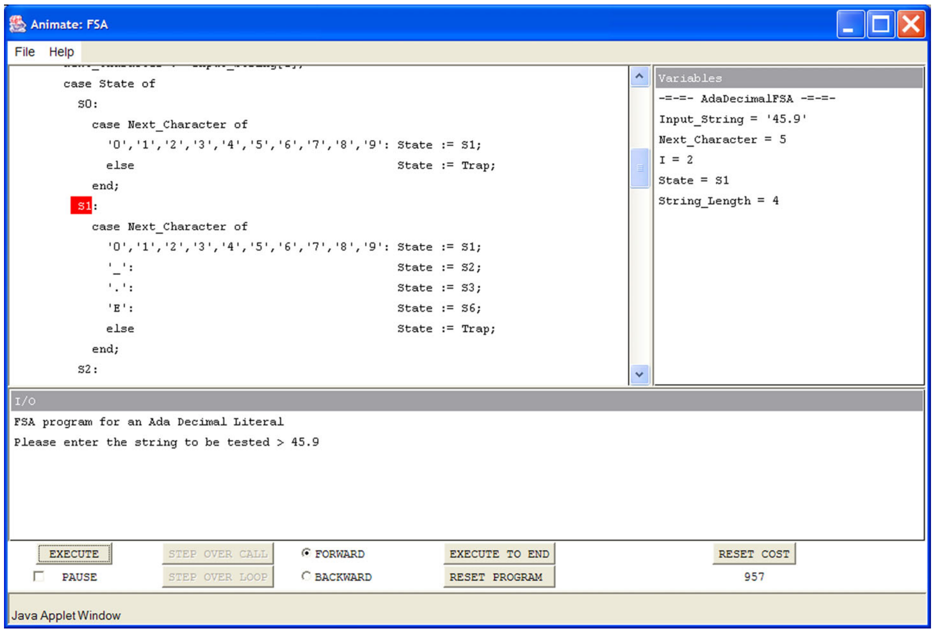
-- *active learning examples*. Students can be required to supply strings for input and/or to select any substring of the input string of length greater than or equal to the number of states in the finite state automaton and watch the automaton identify the first encountered loop in the selected substring.

-- *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, showing 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, not yet completed, is to design and implement an active learning model that will lead 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.,



```
    case State of
      S0:
        case Next_Character of
          '0','1','2','3','4','5','6','7','8','9': State := S1;
          else                                      State := Trap;
        end;
      S1:
        case Next_Character of
          '0','1','2','3','4','5','6','7','8','9': State := S1;
          '_':                                      State := S2;
          '.':                                      State := S3;
          'E':                                      State := S6;
          else                                      State := Trap;
        end;
      S2:
```

```
-=-=- AdaDecimalFSA -=-=-
Input_String = '45.9'
Next_Character = 5
I = 2
State = S1
String_Length = 4
```

```
FSA program for an Ada Decimal Literal
Please enter the string to be tested > 45.9
```

Fig. 5. The program animator active learning tool.

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).

This applet executes Pascal programs in a highly visual fashion (note that Pascal serves as a convenient pseudo-language for expressing algorithms). The program animation applet allows a student to select a program 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 the analysis of the time complexity 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 of 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 at the Webworks Laboratory, and has been discussed in the literature numerous times (e.g., 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 at the Webworks Laboratory, that is, the

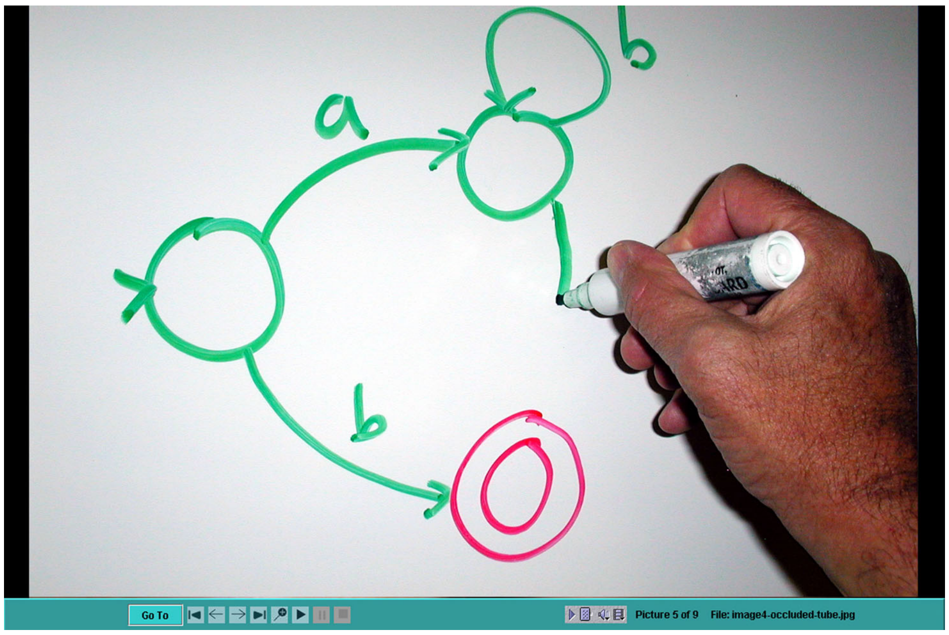


Fig. 6. The slide show applet.

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.

In 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 leads 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 may 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 an arbitrary amount, where desired; (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 having to leave their position on the current page. Indeed, multiple slide shows could be included in one applet. Students who need additional help could select as many different shows as desired from a pulldown menu, each showing, for example, the construction by hand of a different finite state automaton. In this way slide shows can help alleviate the instructor’s time limits. For instance, students can observe the instructor present the hand-construction of a finite state automaton in class a few times and then by means of slide shows in a hypertextbook study as many additional presentations as needed on their own time.

9. THE VIDEO TOOL

The purpose of the video tool is similar to that of the slide show tool. It provides a way to include video clips of processes important to the subject. Small “lecturelets” on a topic are prime candidates for a video clip.

The snapshot of the video applet 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, an 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.

Both the slide show and video applets were a tremendous benefit in the ongoing construction of the hypertextbook on the theory of computing. They also serve a useful and effective purpose in their own right, and will be used liberally in the hypertextbook. The slide show and video applets can also be used to fill the gaps with new active learning models. That is, the ultimate goal of our project is to provide effective active learning models of the many important concepts in the theory of computing; but these take a great deal of time to develop and refine. In the interim, we resort to slide shows and videos 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.

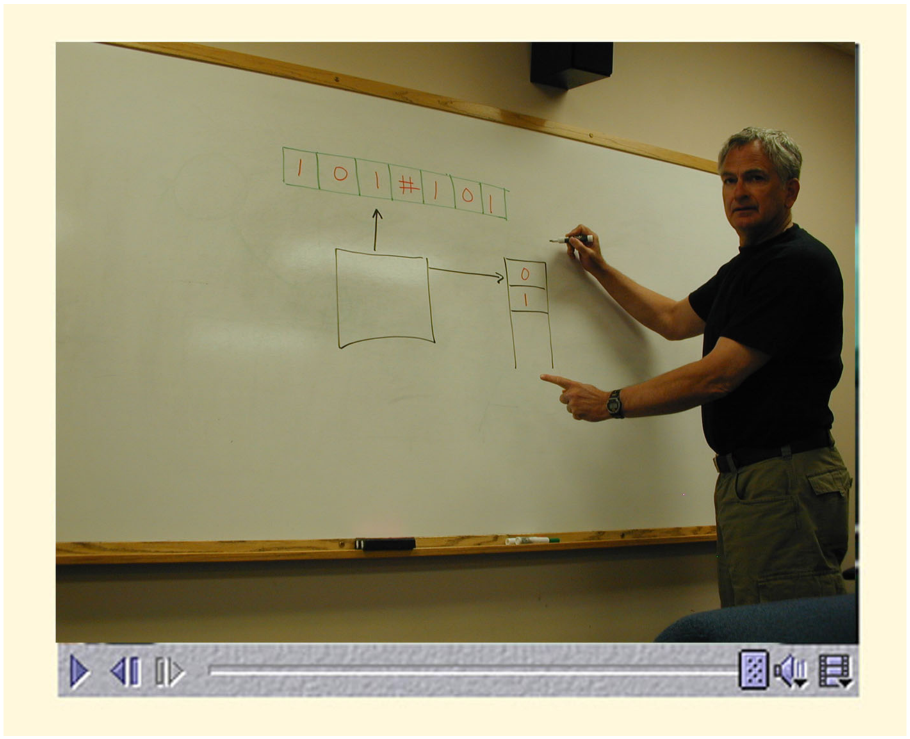


Fig. 7. The video applet tool.

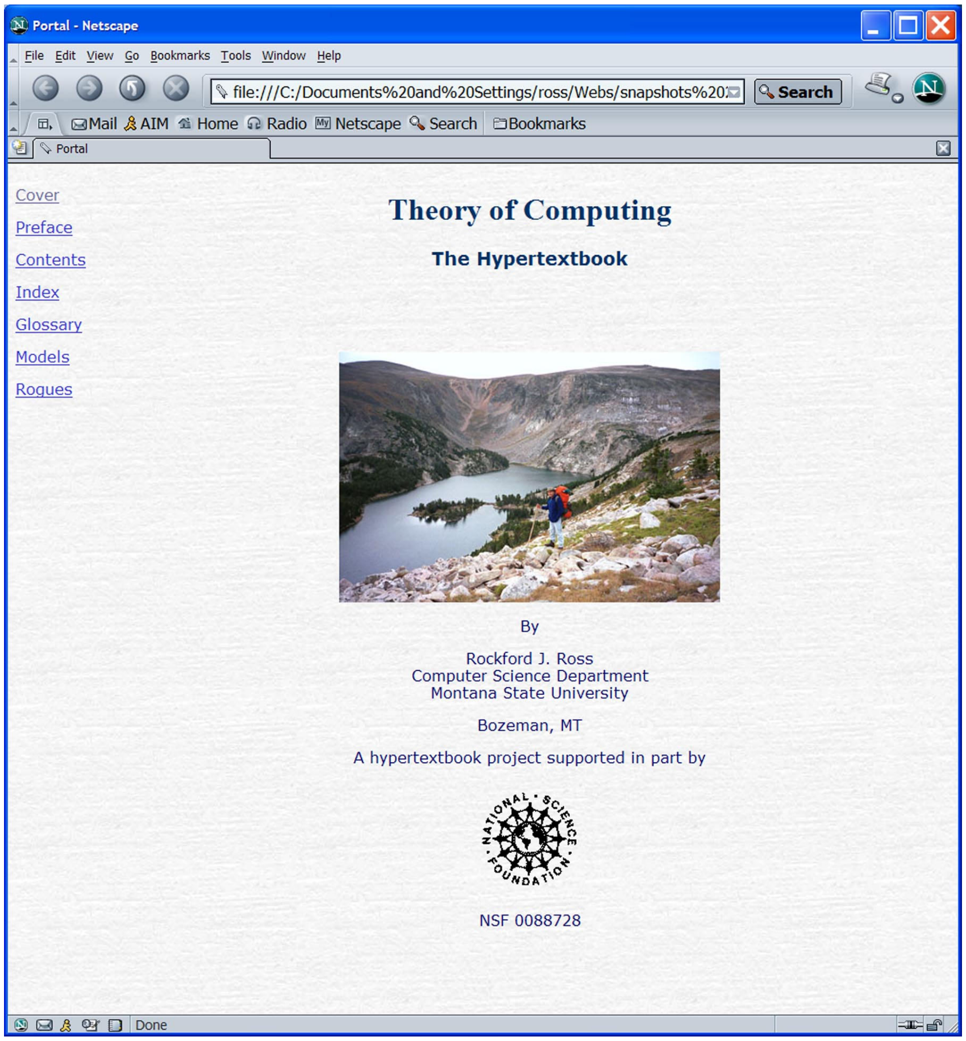
10. TYING IT ALL TOGETHER—A HYPERTEXTBOOK ON THE THEORY OF COMPUTING

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 article, for many reasons stand-alone applets do not seem to be widely used in the classroom. So it is 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]; and 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.


Portal - Netscape
File Edit View Go Bookmarks Tools Window Help
file:///C:/Documents%20and%20Settings/ross/Webs/snapshots%20
Search
Mail AIM Home Radio Netscape Search Bookmarks
Portal
Done

Cover
Preface
Contents
Index
Glossary
Models
Rogues

Theory of Computing

The Hypertextbook

By

Rockford J. Ross
Computer Science Department
Montana State University

Bozeman, MT

A hypertextbook project supported in part by

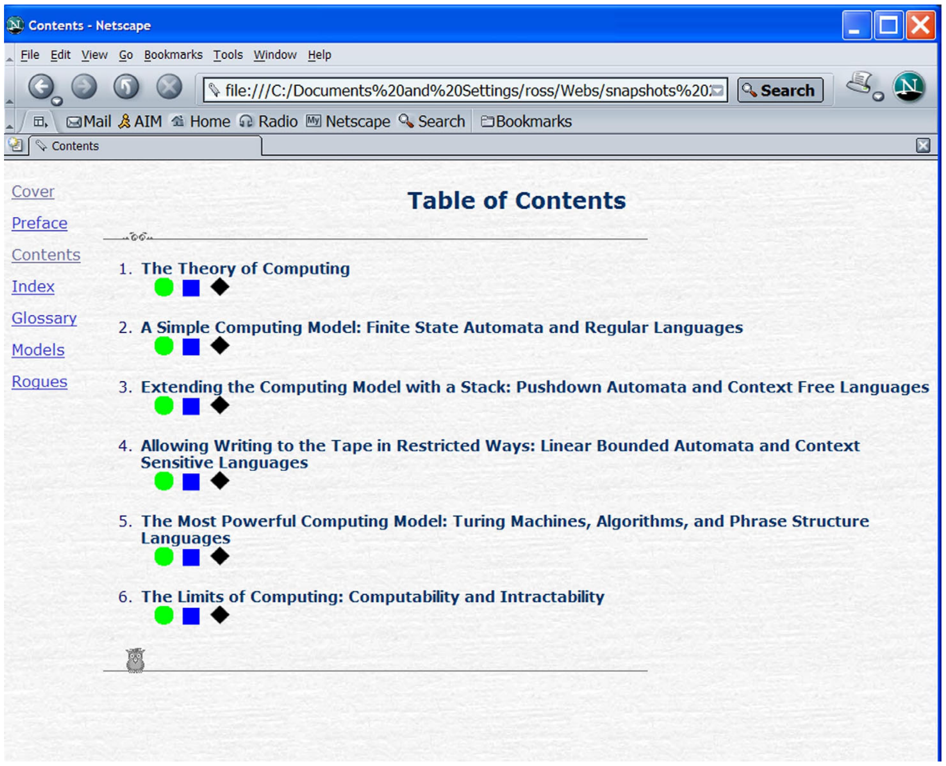
NSF 0088728

Fig. 8. A sample hypertextbook cover page.

Note that the cover page appears in a Netscape Navigator window. All of the usual features of the browser, intimately familiar to users, are available, 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 throughout the hypertextbook as applets, in stand-alone 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 in Figure 9. Notice that there are three symbols below each chapter entry: a green circle, a blue square, and a black diamond. These are the international ski symbols for "easy way," "intermediate way," and "challenging way" down the mountain. Clicking on these


Contents - Netscape
File Edit View Go Bookmarks Tools Window Help
file:///C:/Documents%20and%20Settings/ross/Webs/snapshots%20
Search
Mail AIM Home Radio Netscape Search Bookmarks
Contents

Cover
Preface
Contents
Index
Glossary
Models
Rogues

Table of Contents

1. The Theory of Computing
   ● ■ ◆
2. A Simple Computing Model: Finite State Automata and Regular Languages
   ● ■ ◆
3. Extending the Computing Model with a Stack: Pushdown Automata and Context Free Languages
   ● ■ ◆
4. Allowing Writing to the Tape in Restricted Ways: Linear Bounded Automata and Context Sensitive Languages
   ● ■ ◆
5. The Most Powerful Computing Model: Turing Machines, Algorithms, and Phrase Structure Languages
   ● ■ ◆
6. The Limits of Computing: Computability and Intractability
   ● ■ ◆

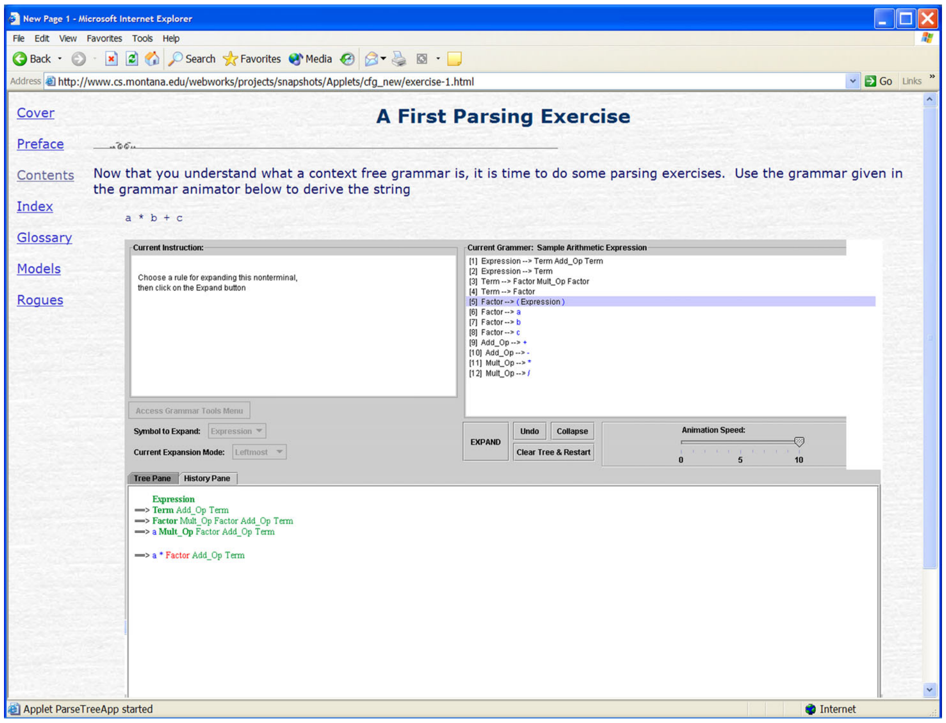
Fig. 9. Table of contents of the hypertextbook.

symbols leads the user to a presentation of the material described in the chapter heading for novice, intermediate, or advanced students, respectively. The green circle routes through the material include many instances of the active learning models and tools to help students new to the material. The next two levels make increasingly less use of the models and tools, and increasingly rely on abstract mathematical presentations. Of course, students are free to move back and forth between the levels.

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 is embedded seamlessly into the surrounding text that describes an exercise. The left side of the page has the usual links to other parts of the hypertextbook, as described earlier.

11. SUMMARY

We described a number of existing active learning models and tools that can be used in stand-alone mode or included seamlessly in a hypertextbook on the theory of computing; and elaborated 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.


New Page 1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.cs.montana.edu/webworks/projects/snapshots/Applets/cfg_new/exercise-1.html
Go Links
Cover
Preface
Contents
Index
Glossary
Models
Rogues
Access Grammar Tools Menu
EXPAND
Undo
Collapse
Clear Tree & Restart
Animation Speed:
0
5
10
Tree Pane
History Pane
Applet ParseTreeApp started
Internet

A First Parsing Exercise

Now that you understand what a context free grammar is, it is time to do some parsing exercises. Use the grammar given in the grammar animator below to derive the string

a * b + c

Current Instruction:

Choose a rule for expanding this nonterminal,
then click on the Expand button

Current Grammer: Sample Arithmetic Expression

```
[1]  Expression --> Term Add_Op Term
[2]  Expression --> Term
[3]  Term --> Factor Mult_Op Factor
[4]  Term --> Factor
[5]  Factor --> ( Expression )
[6]  Factor --> a
[7]  Factor --> b
[8]  Factor --> c
[9]  Add_Op --> +
[10] Add_Op --> -
[11] Mult_Op --> *
[12] Mult_Op --> /
```

Symbol to Expand: Expression

Current Expansion Mode: Leftmost

```
    Expression
==> Term Add_Op Term
==> Factor Mult_Op Factor Add_Op Term
==> a Mult_Op Factor Add_Op Term

==> a * Factor Add_Op Term
```

Figure 10 shows a truncated sample page from the hypertextbook.

It is clear that many more active learning models should be constructed as this project progresses, including those that illuminate such advanced concepts as problem reductions and NP-completeness. Indeed, such a project is never likely to see 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 lack of use of visualization software in the classroom. Since, in the form of seamlessly interwoven active learning applets, visualization software is part and parcel of the hypertextbook, visualization tools will be used as a matter of course. Progress can be monitored on the project website http://www.cs.montana.edu/webworks/projects/theoryportal/ [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 *Proceedings of the 34 SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull.* 35, 162-166.

BEN-ARI, M. 2001. Constructivism in computer science education. *J. Comput. Math. Sci. Teaching 20*, 1,45-73.

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 *Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull. 31*, 1 (March), 7-11.

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 *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull. 33*, 1(March), 65-69.

BYRNE, M. D., CATRAMBONE, R., AND STASKO, J.T. 1996. Do algorithm animations aid learning? Tech. Rep. GIT-GVU-96-18, Georgia Institute of Technology, Atlanta, GA, Aug. The results here are not negative, but they are inconclusive.

BYRNE, R. 2000. Mental models website. May. 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 Bull.* (June), 33-37.

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

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

GREENING, T. (ED.) 2000. *Computer Science Education in the 21st Centur.* Springer Verlag, 173-193.

GRINDER, M.T. 2003. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. In *Proceedings of the 24th SIGCSE Technical Symposium on Computer Science Education. .SIGCSE Bull. 35* (March),157-161,

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 *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull.* 34, 1 (Feb.), 371-375.

HUNG, T. AND RODGER, S.H. 2000. Increasing visualization and interaction in the automata theory course. In PROCEEDINGS OF THE 31ST *SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull. 32,* 1 (March), 6-10.

MAYER, R. E. AND ANDERSON, R.B. 1991. Animations need narrations. *J. Edu. Psychol. 83*, 4, 484-490, 1991.

MAYER, R.E. AND ANDERSON, R.B. 1992. Helping students build connections between words and pictures in multimedia learning. *J. Edu.Psychol. 84*, 4.

NAPS, T. ET AL. 2003. Evaluating the educational impact of visualization. ITiCSE 2003, working group report.

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

ROSS, R. J. 2003. Snapshots of slime. *SIGACT News (Education Forum) 34*, 4 (Dec.), 78-83.

ROSS, R. J. 2002. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. In *Software Visualization.* Lecture Notes in Computer Science, LNCS 2269, S. Diehl (ed.), Springer Verlag, 269-283.

SAARILUOMA, P. 2000. Image and iInterface: Some psychological aspects of visualisation. Report at PVW 2000. http://cs.joensuu.fi/pages/pvw/saariluoma.htm.

STASKO, J. T., DOMINGUE, J., BROWN, M. H., AND PRICE, B. A. 1997. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA.

STASKO, J. T. 1999. Evaluating animations as student aids in learning computer algorithms. *Comput. Edu.33*,4, 253-278.

STASKO, J. T. 1997.Using student-built algorithm animations as learning aids. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education. SIGCSE Bull.* 29, 1 (March), 25-29.

SUTINEN, E. (ED.) 2001. Hypertextbooks for the web. In *Proceedings of the First Program Visualization Workshop* (University of Joensuu), 221-233.

Received July 2004; revised March 2005; accepted January 2006