VISUALIZING THE PUMPING LEMMA FOR REGULAR LANGUAGES

by

Joshua Joseph Cogliati

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

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 2004

©COPYRIGHT

by

Joshua Joseph Cogliati

2004

All Rights Reserved

APPROVAL

of a thesis submitted by

Joshua Joseph Cogliati

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

Dr. Rockford Ross

Approved for the Department of Computer Science

Dr. Michael Oudshoorn

Approved for the College of Graduate Studies

Dr. Bruce McLeod

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Joshua Joseph Cogliati

TABLE OF CONTENTS

LIST OF FIGURES ... vi

GLOSSARY ... vii

ABSTRACT ... ix

1. INTRODUCTION ... 1

2. PREVIOUS WORK ... 4

   Webworks FSA Animator ... 4
   Dr. Susan Rodger's PumpLemma ... 6

3. PUMPING LEMMA FOR REGULAR LANGUAGES ... 10

4. PUMPING LEMMA ANIMATOR ... 13

   Pumping Classification Animator ... 14
   Substring Pump Classification Animator ... 17
   Loop Displayer ... 17
   Repeat Displayer ... 22
   Use of the Pumping Animator ... 22
      States and transitions visited ... 24
      States will be repeated with long input strings ... 24
      Loop-causing substring detection ... 25
      Loop effect on language ... 26
      Classification of an input string ... 26
      Applying the Pumping Lemma ... 27
   The Classify Algorithm ... 28
   Integrating the Animator into a Hypertextbook ... 30

5. EVALUATION ... 31

6. CONCLUSIONS ... 35

   Future Directions ... 35

REFERENCES CITED ... 37

   APPENDIX A – Pseudocode for the Classify Algorithm ... 39
   APPENDIX B – Mode Documentation ... 41
      Pumping Animator Modes ... 42

Secondary Mode Parameters 42
General Parameters 43
Using Parameters 43
APPENDIX C – Evaluation Test 44
Sample questions for Pre and Post-test 45

LIST OF FIGURES

Figure Page

1. Michael Grinder's FSA running 5
2. PumpLemma in action 7
3. FSA before a repeat has been found 14
4. FSA that has accepted 15
5. FSA with a substring being selected 18
6. FSA classifying a substring 18
7. FSA displaying a true loop 20
8. FSA displaying a false loop 20
9. FSA displaying all the loops 21
10. FSA in repeat mode 23

GLOSSARY

classified substring — A classified substring is a substring of an input string to an FSA that is being classified into x, y, and z portions according to the pumping lemma. For the standard pumping lemma the entire string is classified into x, y, and z portions. For the extended pumping lemma any substring of the input string of proper length may serve as a classified substring.

looping state — A looping state q in an FSA is a state such that for a classified string $r = xyz$ in the language recognized by the FSA, x takes the FSA to state q, y takes the FSA from state q back to state q (i.e. the processing of y causes the FSA to loop), and z takes the FSA from state q to an accept state.

looping transitions — Looping transitions are the sequence of transitions followed by an FSA as it processes the y portion of a classified string $r = xyz$ in the language recognized by the FSA.

loop portion — The loop portion of an input string is the symbols that take the FSA through a set of looping transitions. As such, if the string is accepted, new strings constructed by repeating or omitting the loop portion will also be accepted.

loop substring — Any substring s of a string r in a regular language that takes the FSA that recognizes that language from a state q back to state q is a loop substring.

u portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma. The u portion of r is the prefix in the factorization of r as uvw

v portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma.

w portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma. The w portion of r is the suffix in the factorization of r as uvw.

x portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the

substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma. If r is being classified directly, then u and w are the empty string. For a substring v that is to be classified according to the extended pumping lemma, v is classified directly as $v = xyz$, where y is the first substring of v that leads the FSA through a loop as v is processed. In this case, x is the prefix of v just in front of y.

y portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma. If r is being classified directly, then u and w are the empty string. For a substring v that is to be classified according to the extended pumping lemma, v is classified directly as $v = xyz$, where y is the first substring of v that leads the FSA through a loop as v is processed.

z portion — For a string r in a regular language that is to be classified according to the extended pumping lemma, r is first factored as $r = uvw$, where v is the substring in r that is classified, $|v| \geq p$, and p is the constant of the extended pumping lemma. If r is being classified directly, then u and w are the empty string. For a substring v that is to be classified according to the extended pumping lemma, v is classified directly as $v = xyz$, where y is the first substring of v that leads the FSA through a loop as v is processed. In this case, z is the suffix of v just after of y.

ABSTRACT

The pumping lemma for regular languages and its application are among the more difficult concepts students encounter in an introductory theory of computing course. The pumping lemma is used to prove that particular languages are not regular. Traditional methods of teaching the pumping lemma seem inadequate for helping average students learn this concept.

In this thesis we describe a set of software tools that help students visualize the pumping lemma for regular language. The Java programming language was used to create active learning animations of various aspects of the pumping lemma that run as applets in web browsers. Each of the steps in the proof of the pumping lemma is animated. The finite state automaton animations can be manipulated so that the concepts can be tried easily. With the feedback provided, student mistakes in understanding the concept can be discovered and quickly corrected.

New methods for teaching the pumping lemma are considered. These allow students to proceed at their own pace while learning about the pumping lemma through interactive animations. These methods can be used to augment or replace traditional teaching approaches. The pumping lemma animator will be included in an ongoing project designed to create animations and interactive tools for a complete course on theory of computing.

INTRODUCTION

The pumping lemma for regular languages provides a characterization of regular languages and is generally used to prove that certain languages are not regular. This lemma and its application are among the most challenging concepts students encounter in a theory of computing course. In this thesis we describe software that can be used by students to study the pumping lemma for regular languages in a visual, interactive fashion. We expect that students will be able to learn the pumping lemma and its application more readily through the use of this software.

Designed as applets that run in standard web browsers, the software models for the pumping lemma are also intended to be incorporated into a novel teaching and learning resource entitled *Theory of Computing: The Hypertextbook* [1]. From the paper *Hypertextbooks* by Dr. Ross comes the following description of a hypertextbook:

> A *hypertextbook* is a comprehensive, web-based teaching and learning resource that is intended to augment or supplant a traditional textbook for an academic subject. [...] Hypertextbooks extend the capabilities of traditional textbooks tremendously in that, beyond mere textual presentations and static illustrations, they can also incorporate video clips, audio files, and active links to other material on the web. They can be arranged (through the use of hyperlinks) to accommodate various teaching/learning needs and styles. Most unique, though, is their capacity for including active learning modules in the form of interactive applets that animate important concepts and engage students in exploratory learning. [2]

Theory of Computing: The Hypertextbook is proposed to be the first working example of the hypertextbook concept. In addition to traditional text presentations of the material, this hypertextbook will illustrate the key concepts of the theory of

computing through liberal use of interactive applets. Many of the concepts in a theory course have traditionally been poorly served by paper textbooks since many of these concepts are dynamic, but must be presented in static form in a traditional textbook. The pumping lemma animator software of this thesis will become part of *Theory of Computing: The Hypertextbook*.

Visualization software has been used successfully in other subjects in computer science [3, 4, 5] and in other disciplines. For example, the Virtual Labs Project in the Stanford University School of Medicine has been using learning modules in cardiovascular, gastrointestinal, respiratory, renal, visual, and neurophysiological systems for the past four years. These modules include simulations and interactive games to reinforce concepts being learned. Reaction from students and faculty has been positive [6].

In chapter 2 of the thesis, we discuss previous work upon which this thesis is based and similar efforts completed by others. This is followed in chapter 3 by a detailed discussion of the pumping lemma for regular languages. Then, in chapter 4 the software developed as part of this thesis for teaching and learning the pumping lemma is presented at length. This is followed by a discussion of evaluation in chapter 5. Finally, in chapter 6 we draw conclusions and outline future work.

We assume throughout that readers are familiar with theory of computing concepts, including the terminology and definitions of finite state automata, regular languages, regular expressions and the pumping lemma for regular languages. These

concepts can be found in introductory textbooks on the theory of computing (for example, see [7]). In this thesis literal symbols will be displayed with a typewriter font (e.g. `abcd`) to differentiate them.

PREVIOUS WORK

Webworks FSA Animator

The Webworks Finite State Automaton (FSA) Animator is the precursor to this project. Its purpose is to animate the running of a finite state automaton. It was created by Michael Grinder [8, 9, 10]. The FSA animator is a Java applet that allows a student to construct an FSA and then run that FSA on arbitrary strings. The applet animates all aspects of the operation of the FSA as it executes. The FSA animator also has an exercise feature. This allows a student's created FSA to be compared to a hidden, correct FSA given by an instructor. The comparison feature checks whether the FSA created by the student correctly recognizes the language of the exercise. The FSA animator supports both deterministic and nondeterministic FSAs.

The FSA animator allows the graphical construction of an FSA using the mouse. New states can be added and moved around. A transition is created by a click and drag operation from a source state to a target state. Once a transition has been created, it can be labeled with the symbols that cause the transition to occur.

As seen in figure 1, the FSA animator applet displays a virtual tape at the top of the applet window that can hold a string to be processed by the automaton. The FSA tape head is represented by a triangle that appears below the tape, underneath

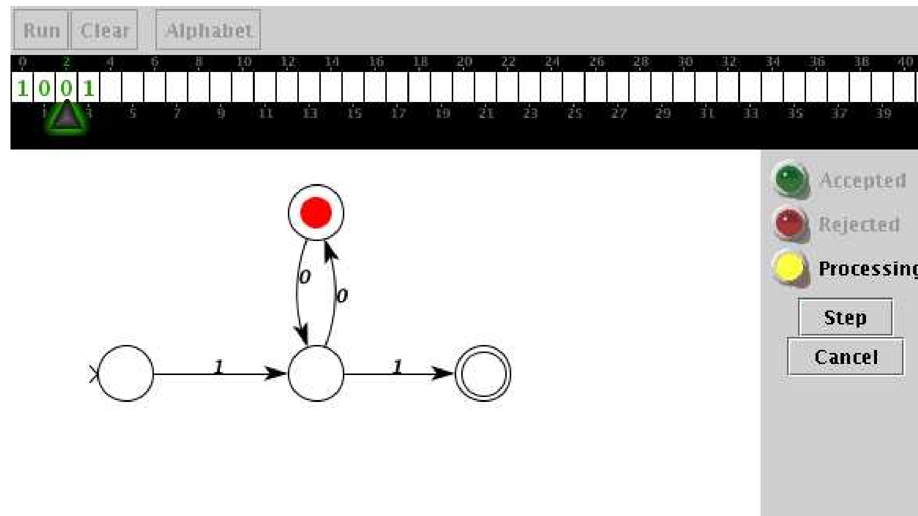


Figure 1. Michael Grinder's FSA running.

the current input symbol. Figure 1 shows the configuration of the pictured FSA after it has processed the leading 1 and 0 of the input string and is about to read the 0 directly above the input head triangle.

As the FSA runs, the current state is occupied by a red circle (there may be more than one current state if the FSA is nondeterministic). The red circle moves from state to state along the transition arrows as the machine steps through the symbols on the tape under user control.

The FSA animator has been used in experiments to see if it has any effect on student learning [8]. A control group was given FSA construction exercises without access to the FSA animator. A target group used the FSA simulator to complete the same set of exercises. The target group had a statistically significant higher percentage of students able to complete a set of exercises correctly. However, there was

no statistically significant difference in scores between the two groups in a subsequent test given later. The author concluded that although more evaluation of the FSA applet use was needed to determine the long-term effects on student learning, the benefits seen in student motivation when the FSA applet was used made its use worthwhile.

Grinder's FSA applet formed the basis for the pumping lemma visualization applets developed for this thesis. Many enhancements and modifications were made to the FSA applet so that it can illustrate the characteristics of FSAs upon which the pumping lemma is based. In the remainder of the thesis we use the terms "FSA animator" and "pumping animator" interchangeably as terms describing Grinder's FSA animator as enhanced by this author.

Dr. Susan Rodger's PumpLemma

Dr. Susan Rodger, of Duke University, created visualization software called PumpLemma to assist students in learning about the pumping lemma [11, 12]. It operates by allowing students to type in a language such as $\mathtt{a}^{\frac{n}{2}}\mathtt{b}^n$; it then assists students as they attempt to to check all the cases necessary (according to the pumping lemma for regular languages) to prove that the language is not regular. Possible input languages can be described by multiple base letters raised to various exponents.

Students use PumpLemma by specifying a language they wish to prove non-regular. First, they enter some language, such as $\mathtt{a}^{\frac{n}{2}}\mathtt{b}^n$ (see figure 2). Then, they

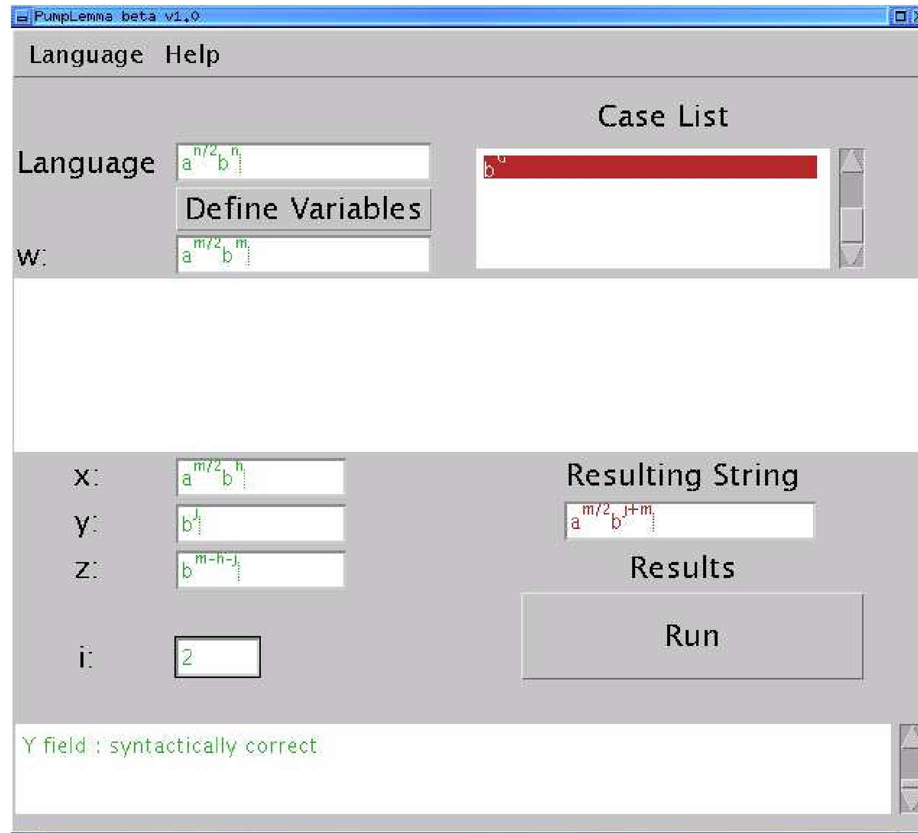


Figure 2. PumpLemma in action.

specify a range for n, such as $n > 0$. Next, the program generates a list of the possible values for the y portion of the string (recall that the pumping lemma requires breaking a selected string in the language into an x prefix, a y substring and a z suffix such that xy^iz is in the language for all $i \geq 0$). For the above language, there are three possibilities: $\mathtt{a}^G$ (where y lies entirely in the `a` portion of the string), $\mathtt{b}^G$ (where y lies entirely in the `b` portion of the string), $\mathtt{a}^G\mathtt{b}^J$ (where y lies across the `a`-`b` boundary in the string) where $G > 0$ and $J > 0$. Next, each of those possibilities must be checked by the student to see if it can be pumped.

For example, for the $y = \mathtt{b}^G$ case, the student supplies $\mathtt{a}^{\frac{m}{2}}\mathtt{b}^h$ as the x portion, $\mathtt{b}^j$ as the y portion, and $\mathtt{b}^{m-h-j}$ as the z portion. Then, the student specifies an i value for the string xy^iz to try to find a conflict. In this case, if i is 2, a conflict results (the resulting string is not in the language since it has too many `b`'s), thus showing that $\mathtt{b}^G$ cannot be pumped.

After a conflict has been produced for each of the three cases $y = \mathtt{a}^G$, $y = \mathtt{b}^G$, and $y = \mathtt{a}^G\mathtt{b}^J$, the language has been shown to be non-regular. If a conflict is not produced for every case, no conclusions can be drawn.

PumpLemma still must be used with other programs to be an effective learning aid for the pumping lemma. PumpLemma is quite restricted with respect to the types of languages that it allows as we quote from the documentation "language selection is limited to ordered languages (i.e. $a^nb^n(ab)^n$, but not 'the number of a's equals the number of b's')" [12]. If the language is originally given as a grammar or as an informal description, the language must be converted by the student into the form that PumpLemma requires.

An example language L is $\{x$=y+$z|x$, y, and z are unsigned binary numbers and x is the sum of y and $z\}$ which has the alphabet {`0`,`1`,`=`,`+`} (for example, `10=1+1` and `111=101+10` are in the language, but `111` and `10=10+10` are not). This language L must be converted to a form that PumpLemma can recognize. As an example, a form for L that could be used by PumpLemma is $\mathtt{10}^n\mathtt{=10}^n\mathtt{+0}$ (actually, it is a little worse, since PumpLemma only allows symbols `a-f` in the alphabet; with that restriction, a

form for language L acceptable to PumpLemma could be $\texttt{ab}^n\texttt{cab}^n\texttt{db}$, where a="1", b="0", c="=", and d="+").

Consider string $\texttt{10}^*\texttt{=10}^*\texttt{+0}$, which is written as a regular expression, so it is regular. If language L is regular, then the intersection of languages L and $\texttt{10}^*\texttt{=10}^*\texttt{+0}$ is regular as well. The intersection of these two languages is the language $\texttt{10}^n\texttt{=10}^n\texttt{+0}$. So, if $\texttt{10}^n\texttt{=10}^n\texttt{+0}$ is not regular (and it is not regular), then language L is not regular. Using this approach, PumpLemma can be used to prove language L is not regular, but to do so requires extra knowledge.

PumpLemma is the only previously deployed software system known to the author that seriously attempts to visualize the pumping lemma. While PumpLemma could be quite useful for the purpose of teaching how to prove that a language is not regular, the system seems to have fallen into disuse. PumpLemma has not been updated since 1997, and it no longer runs or compiles under current Java implementations. Several modifications had to be made to allow it to be tested for this thesis. The modified version was sent to Dr. Rodger.

PUMPING LEMMA FOR REGULAR LANGUAGES

The pumping lemma for regular languages identifies some properties that all regular languages have. Since all regular languages have these properties, any language that does not have these properties is not regular. The proof of the pumping lemma is usually based on finite state automata (recall that any regular language can be implemented by an FSA and any FSA can be converted into a regular language) [7].

There are various versions of the pumping lemma for regular languages. We call the following the "standard version."

Lemma 1. Let L be a regular language. Then there is a constant p (usually referred to as the "pumping constant") depending only on L such that if s is a string in language L, and $|s| \geq p$, then s can be divided into three substrings x, y, and z such that $s = xyz$ and:

1. $|y| > 0$;
2. $|xy| \leq p$;
3. For each $i \geq 0$, $xy^iz \in L$.

The proof of this lemma follows from the fact that for each regular language, there is an FSA M that recognizes it. If we let p be the number of states in M, then for any string s in L of length greater than or equal to p, M must encounter some state at least twice while parsing s. That is, when processing any string s in L that

has at least as many symbols in it as the number of states, p, in M, M will be forced to visit at least one state at least twice. Some prefix x of s will lead M from the start state to some state q_j, some next portion y will lead M back to q_j, and the suffix z will lead M from state q_j to some accept state. If the y portion of s is repeated, as in $xyyz$, this new string will also be in the language since processing of the second y portion will leave the machine in state q_j. Furthermore, if the y portion is removed from the string, yielding $s' = xz$, s' is also in the language since the x portion leaves the machine in the q_j state and the z portion takes the machine to an accept state from q_j.

Students must have a clear grasp of these insights in order to understand and apply the standard version of the pumping lemma in a knowledgeable fashion. Unfortunately, it is likely that instructors of a theory of computing course usually do not have enough time to spend in class to ensure that the average student does have a clear understanding of the intuition behind the pumping lemma before moving on. The pumping animator described in the next section is intended to alleviate this problem.

We call the following the "generalized pumping lemma" for regular languages.

Lemma 2. Let L be a regular language. Then there is a constant p depending only on L such that for any string s in L and for each way of partitioning s as $s = uvw$ such that $|v| \geq p$, v can be further divided into three substrings x, y, and z such that $v = xyz$ and:

1. $|y| > 0$;

2. $|xy| \leq p$;

3. For each $i \geq 0$, $uxy^izw \in L$.

In order to help students grasp the intuition behind Lemmas 1 and 2, interactive animation applets based on the FSA animator of Michael Grinder were developed. We present these next.

PUMPING LEMMA ANIMATOR

The pumping lemma animation applet described in this section is based on the FSA animation applet of Michael Grinder [8]. The FSA Animator was modified to incorporate new modes. These modes allow the applet to be used for a variety of new purposes. Each mode emphasizes different properties of an FSA. Grinder's original mode as described in the chapter Previous Work, still runs, as do four other new modes.

- The first new mode is the pumping classification mode. When running in this mode, the animator will classify an input string into x, y, and z portions according to Lemma 1.
- The second new mode is the substring pumping classification mode. This mode classifies a selected substring of an input string into x, y, and z portions according to Lemma 2.
- The third new mode is the loop display mode. This mode requires that a student attempt to select a substring of the input string that leads the FSA through a loop; the animator will then check whether this selected portion really does lead the FSA through a loop.
- The last new mode is the repeat displayer, which displays states and transitions differently depending on how many times each state or transition has been visited as the input string is processed by the FSA.

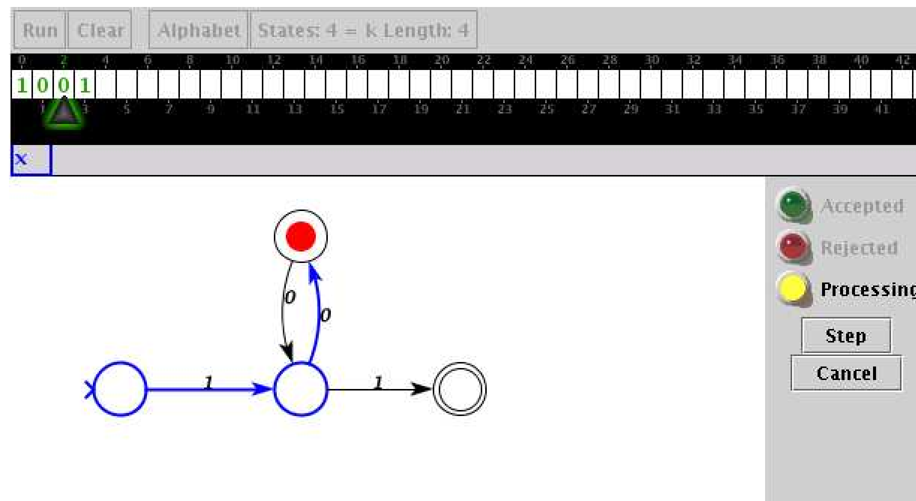


Figure 3. FSA before a repeat has been found.

Pumping Classification Animator

In the pumping classification mode, the animator automatically divides the input string into x, y, and z portions according to Lemma 1 as it runs. Students can observe where the first loop is encountered as the FSA being animated processes an input string.

As the animator starts, it first colors all states and transitions black to show that they are unvisited initially. As the input string is processed, the animator begins to color each state and transition visited with the color used to highlight the x portion (blue in figure 3) since these states and transitions are initially assumed to be part of the x portion of the string. This early configuration can be seen in figure 3.

At some point during processing, either the string on the tape will be completely processed, or a state will be repeated. The first time a repeated state is encountered

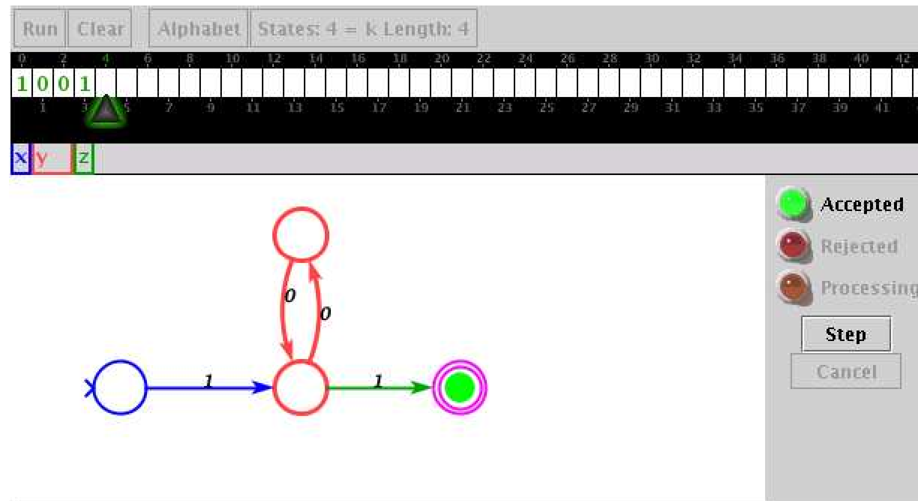


Figure 4. FSA that has accepted.

will be used to determine the y portion of the input string. The animator then properly recolors those states and transitions that form the loop traveled as the FSA processed y a different color (red in figure 4). Depending on how the instructor has initialized the program, the animator can either continue to check for repeated instances of the identified loop, or just classify the rest of the string beyond the y portion as the z portion.

After the machine has completed processing and accepting the input string, the FSA will be colored as in figure 4. The states and transitions associated with the x portion are colored blue, those with the loop corresponding to the y portion are colored red, and those associated with z are colored green. Students can watch the processing, including color changes, as the FSA processes the input string a symbol at a time.

In order to clarify the intuition behind Lemma 1, after processing of the input string has completed, the input tape can be modified to remove the y portion. When that is done, and the FSA is run again, the new string will be accepted as well. In the process, a new y portion in the new string might be uncovered. That is, it is possible that a substring causing a loop will be found in the new string from which the old y has been removed. In fact, this is guaranteed to happen if the new string is still at least as long as the number of states in the FSA. These steps can be repeated until all loops have been eliminated.

The input tape can also be modified by the student to include a repeat of the y portion (i.e., as $xyyz$), and the FSA can be run again to demonstrate that the newly constructed string is also accepted and hence is in the language recognized by the FSA.

In the standard pumping classification mode, the FSA animator always highlights the first, shortest loop that is encountered (this is the only loop guaranteed to exist by Lemma 1, although there may be others). Since the first, shortest loop can be determined algorithmically and is the only loop guaranteed to exist by the pumping lemma, it is the one displayed as the y portion. The full algorithm for doing this, `classify`, is described later in this thesis.

The pumping classification mode can also be augmented by specifying a "y repeat display parameter." This causes the FSA animator to highlight repeated y portions in the input string, rather than just the y portion causing the first loop. Thus when

the animator is being used to illustrate that a single loop must always exist, then the y portion repeats are hidden so as to make the output less confusing. When the desire is to show that the y portion may repeat, the repeat display mode parameter can be set to cause the animator to highlight all y repetitions.

Substring Pump Classification Animator

The substring pump classification animator mode uses the same general framework as the pump classification animator mode, except that it allows the user to select for classification a substring of the full string. This substring is then classified into x, y, and z portions. This mode allows for exploration of the ramifications of the generalized pumping lemma (Lemma 2) by checking that any selected substrings of the proper length will cause the FSA to loop.

The student selects a substring by clicking and dragging the mouse below the tape as shown in figure 5. The animation then will display a classification of the selected substring as the entire input string is processed by the FSA (see figure 6). The student-selected substring is treated as the v portion of the input string according to Lemma 2; thus the selected substring, v, is further classified into x, y, and z portions by the animator. By moving the substring selection around to different portions of the input string, a student can explore the concept that any substring of proper length in the input string will cause the FSA to loop.

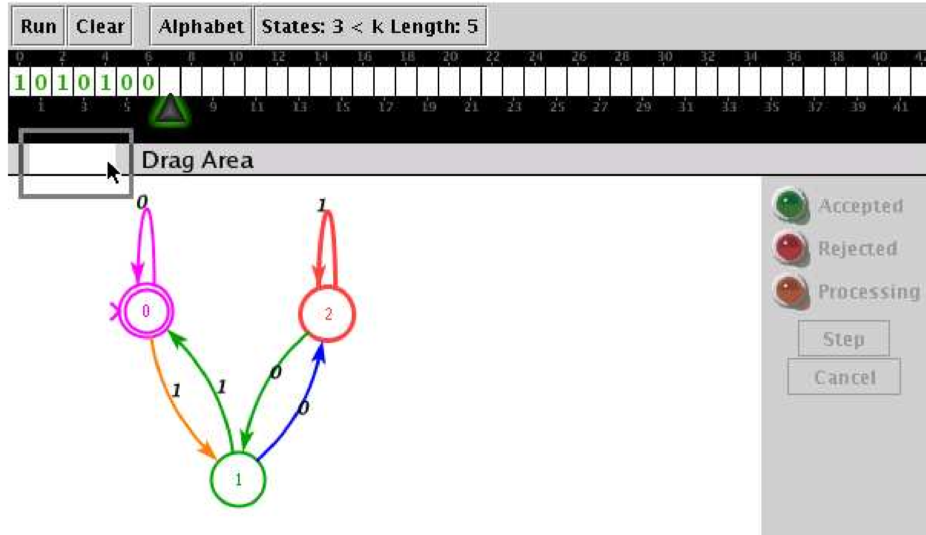


Figure 5. FSA with a substring being selected.

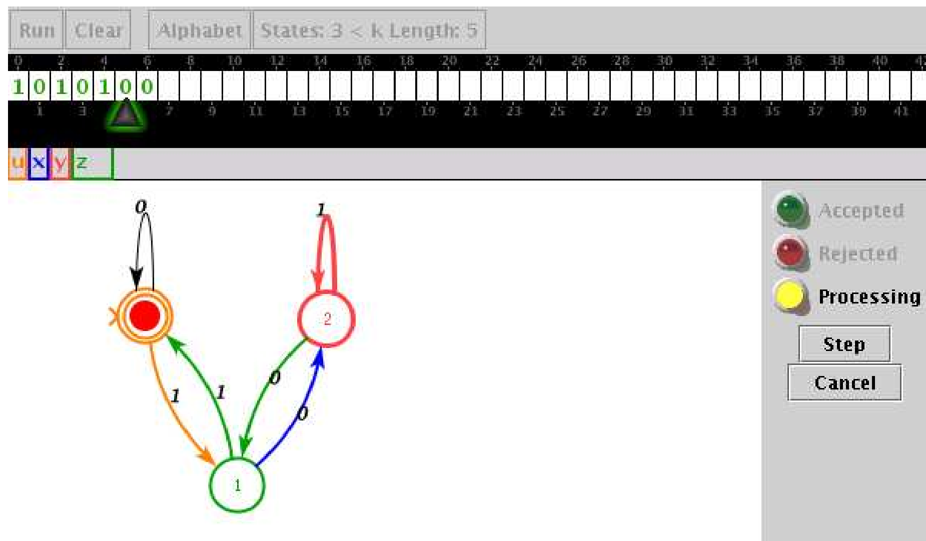


Figure 6. FSA classifying a substring.

Loop Displayer

For most input strings the FSA processing those strings will encounter more loops than just the first, shortest loop. The loop display mode is used for exploring all the substrings of an input string that will cause the FSA to loop. A loop displayer component keeps track of loop-causing substrings in the input string.

Like the substring pumping classification mode, the loop display mode allows arbitrary substrings of the input string to be selected, but this time for a different purpose. When a substring is selected in loop display mode, the animator will check to see if that substring does indeed cause a loop. If the substring causes a loop, the animator will underline it with green, and if not, the animator will underline it with red. As the FSA processes the selected substring, the animator colors the states and transitions visited green. This allows the path taken by the FSA as it processes the selected substring to be easily seen. If this path is a true loop, the highlighted path beginning and ending states will be the same, as shown in figure 7. If the highlighted path beginning and ending state are different, this implies that the selected substring does not represent a loop in the FSA. The coloring will make this obvious, as shown in figure 8.

The loop displayer can be set to uncover all the loops in a given string, as displayed in figure 9. Unlike the pumping classification modes, the loop display mode will find all the loops in an input string, not just the first, shortest loop. The algorithm for doing this first generates the sequence of states that are visited as the input string is

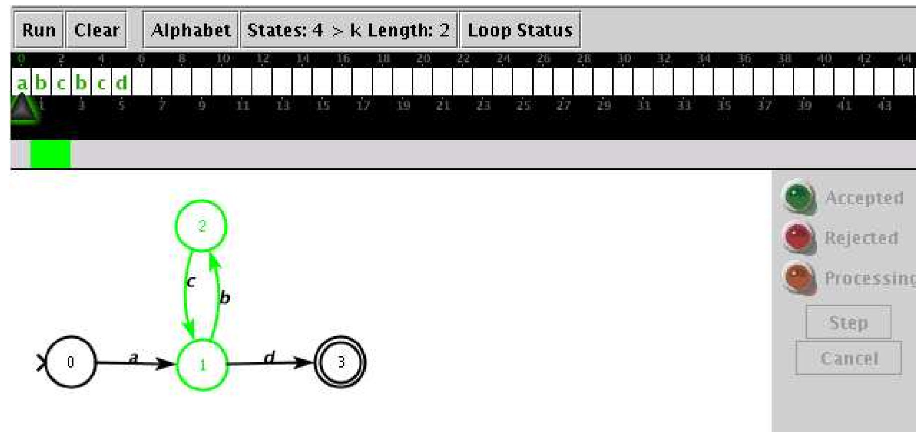


Figure 7. FSA displaying a true loop.

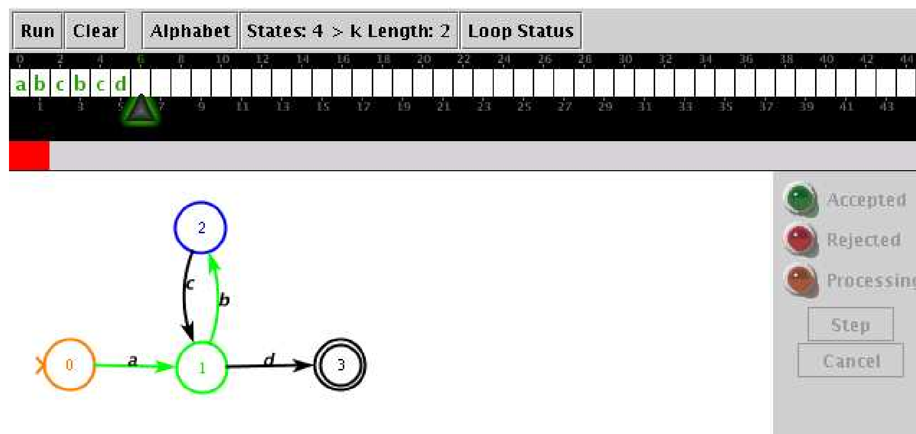


Figure 8. FSA displaying a false loop.

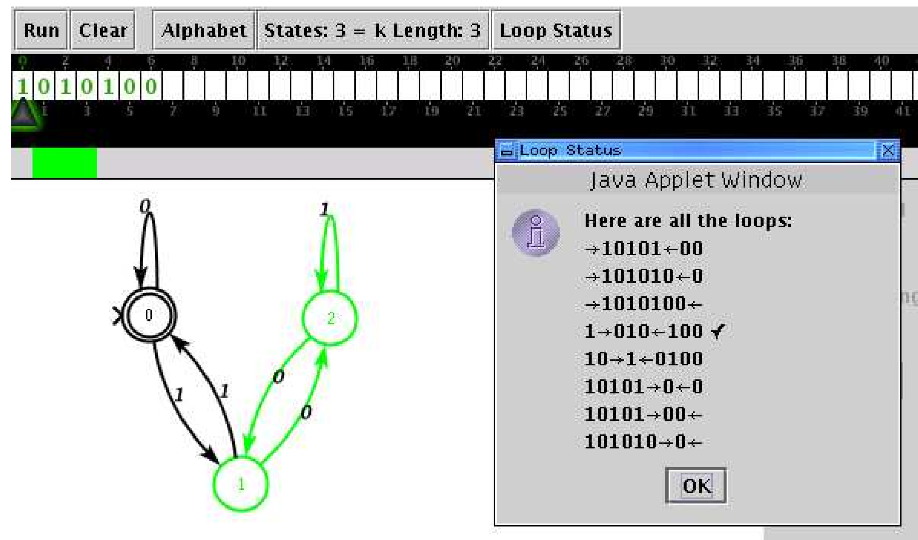


Figure 9. FSA displaying all the loops.

processed. Then, for each state in the sequence the algorithm examines all the states after that state in the sequence. If the selected state reappears later in the sequence, the substring that takes the FSA from the selected state to the repeated occurrence of that state is added to the list of loop-causing substrings. Thus, all the loop-causing substrings for the input string are found.

The loop display mode can be set so that the loop displayer presents one of three different levels of information to students using the animator:

Show loops — In this mode the displayer shows all the substrings that cause loops that exist for the displayed FSA and the current input string.

Show number — In this mode the displayer only shows how many substrings there are that cause loops and those that have been discovered already (i.e., students can be required to identify loop-causing substrings of the input string on their own).

Show nothing — In this mode the displayer only shows the loop-causing substrings that have been already discovered by a student and whether or not all such substrings have been found.

Repeat Displayer

The FSA Animator can be set to repeat display mode. In this mode, states and transitions are highlighted in one color as they are first encountered. When a state or transition is repeated, it is highlighted differently. This is implemented by keeping a "touch" count for each transition and state. When the FSA first starts processing, the touch count is set to zero for all states and transitions. Each time a state or transition is encountered during processing of the input string, the touch count for that state or transition is incremented by one. The highlighting of states and transitions is updated based on the current touch count.

States and transitions are colored in three different ways. When they are untouched, the transition arrows and states are thin and black. As they are touched, the lines used to draw them get thicker and the color changes (the change in thickness should help students who are color blind). This is shown in figure 10.

Use of the Pumping Animator

There are a variety of ways that the pumping animator can be used for teaching and learning depending on the needs of the instructor or student. Some of them are not even be directly related to the pumping lemma. For example, the loop display mode might be used for a lesson about how an FSA recognizes an infinite language.

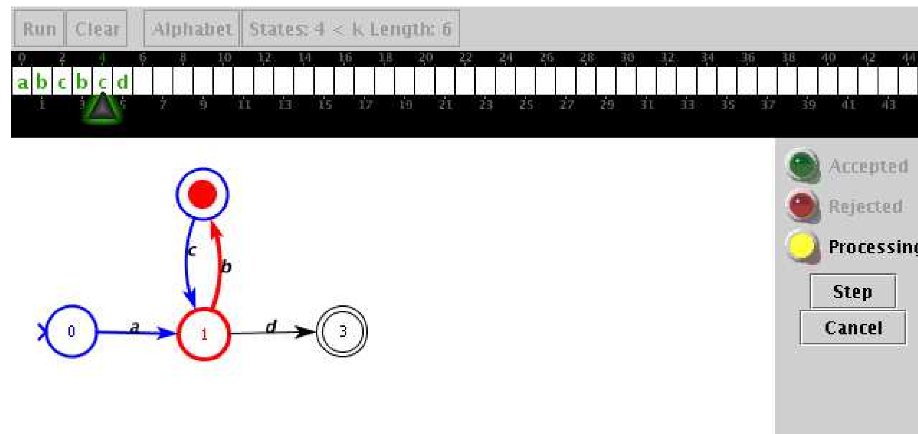


Figure 10. FSA in repeat mode.

With respect to learning the pumping lemma, a variety of concepts must be mastered. The pumping animator modes are designed to help students learn these concepts as active participants in the process. These concepts include:

- How states and transitions are visited, or touched, as an input string is processed.
- That sufficiently long input strings guarantee that the FSA will have to touch at least one state more than once.
- How to identify substrings of an input string that cause the FSA to loop while processing the input string.
- Recognizing that loop substrings of the input string can be removed or repeated, resulting in new strings that the FSA accepts.
- How to identify the x, y, and z portions of an input string of sufficient length according to Lemma 1.

- How to identify the u,v, and w portions of an input string, and then the x, y, and z portions of v, in an input string of sufficient length according to Lemma 2.

How these concepts can be learned through the use of the pumping animator is explored below.

States and transitions visited

A concept to learn early is that states and transitions are touched as input string processing occurs. Repeat mode can demonstrate this. The student runs a provided FSA in repeat mode and as the FSA processes the input string, touched states are colored differently (see figure 10). This visually shows which states and transitions are visited. Students can enter different input strings for processing. Examples can be given that demonstrate that not all states will necessarily be touched for certain FSA-input string pairs. Exercises that have students supply input strings for provided FSAs that cause certain states to be touched can also be assigned to help students learn this concept.

States will be repeated with long input strings

The observation that sufficiently long input strings will force an FSA to loop can be demonstrated with the repeat mode by using a variety of strings with different lengths and FSAs that recognize infinite languages. The longer input strings (longer

than the number of states) will demonstrate that for sufficiently long strings, states will be repeated as the FSA processes the string.

For certain kinds of FSAs, students will also learn that input strings shorter than the number of states *may* force the FSA to loop. For other kinds of FSAs, students will learn that only strings as long as the number of states *guarantee* that the FSA will loop. Students will also learn that there is always at least one string that will cause the FSA to accept without any loops (as long as the FSA recognizes more than the empty language). These observations will help students understand that for every FSA which accepts an infinite language, there is some length of input string that will guarantee that the FSA has a loop. This length may be less than the number of states, but it will never be more than the number of states.

Students can try many input strings on their own to explore all of these issues in assigned exercises. Since all activity of the FSA is being shown in the applet, students will be able to gain a clear understanding of these issues.

Loop-causing substring detection

The third concept useful for understanding the pumping lemma is identification of substrings of an input string that cause an FSA to loop, which loop mode can demonstrate. Recall that the loop mode of the pumping animator has modes that can (1) list for the student all substrings of an input string that cause loops, (2) give the number of substrings that cause loops (but not list them), and (3) not list any information about the loop-causing substrings or their number. Students can

use version 1 to see how loop substrings are identified and then versions 2 and 3 to try to find the substrings that cause loops on their own. When the students select substrings in this mode they can watch as the FSA processes the whole input string to see whether their selected substring really does cause a loop. This demonstrates the idea of loops, preparing students to examine the effect of loops on the language that an FSA accepts.

Loop effect on language

The loop mode and the pumping mode together can be used to learn how loop-causing substrings can be used to construct new strings that an FSA accepts. First, a loop-causing substring can be identified by a student using the animator either in loop mode or pumping mode. Then, the student can create new input strings that are identical to the original input string except that the loop-causing substring is either omitted or repeated one or more times in the new string (e.g. if the loop-causing substring is `bc` in the input string `abcd` then new strings accepted by the FSA include `ad`, `abcbcd`, `abcbcbcd`, . . .). These new strings can be input to the same FSA and the student can verify that the FSA accepts these new strings.

Classification of an input string

The last concept that is helpful for students learning the pumping lemma is how to classify an input string into the x portion, y portion, and z portion according to Lemma 1. Once the concept of loops is understood, this is relatively easy to learn.

The pumping mode is used, which allows the student to see how—in any sufficiently long string—the x portion, the y portion and the z portion are found. Similarly, with respect to Lemma 2, students can identify any substring of sufficient length in the input string and determine how to find the x, y, and z portions of that substring.

Applying the Pumping Lemma

Exploring these five concepts helps students achieve an intuitive understanding of the pumping lemma. However, students must still learn how to apply this lemma. For example, consider the language L $=\{\mathtt{a}^n\mathtt{b}^n | n \geq 0\}$. This language is not regular, so there is no FSA that recognizes it. There are several methods that can be used to *try* to build an FSA that recognizes the language L. The instructor can show various attempts and explain why they fail, and the students can make their own attempts and discover why they fail.

There are two categories of attempts. The first category consists of FSAs that do not have any cycles in the graph of the transitions reachable on a path from the start state to an accept state (i.e., there are no input strings that will cause a loop). These attempts will fail because these types of FSA only recognize finite languages.

The second category consists of FSAs with loops, which will fail to recognize language L. If one gives such an FSA purported to recognize language L a sufficiently long input string, say $\mathtt{a}^p\mathtt{b}^p$ where p is the number of states in the FSA, a loop will occur as the FSA processes the a's. For example, if the FSA has ten states, then $p = 10$ for this FSA and the string $\mathtt{a}^p\mathtt{b}^p$ has ten a's, so the FSA will loop while still

processing the `a`'s. Since the substring causing the loop can be omitted to create a new string that the FSA must accept, the FSA will accept some string $\mathtt{a}^{p-k}\mathtt{b}^{p}$ (where k is the length of the loop-causing substring), which is not in language L.

Once students begin to see that any attempt to construct an FSA to recognize language L (or similar languages that are not regular) will fail (and why), they will have a firm foundation for understanding and applying the two versions of the pumping lemma (Lemma 1 and Lemma 2). That is, once the intuition acquired by exploring the pumping lemma is internalized by students through use of the various modes of the pumping animator, proving that a language is not regular using the abstract versions of the pumping lemma (Lemma 1 and Lemma 2) will be more easily understood.

The Classify Algorithm

Algorithm `classify` treats all input strings s, as having the form $s = uvw$. `Classify` then classifies the v portion of an input string into $v = xyz$, where y is the substring that causes the corresponding FSA to loop according to either Lemma 1 or Lemma 2. For a string s being classified according to Lemma 1, u and w are considered to be empty (i.e., $s = v = xyz$). For a string s being classified according to Lemma 2, u and w may be nonempty and the x, y, and z portions are in the v part of the string (i.e., $s = uvw$ and $v = xyz$). The `classify` algorithm takes as its input a list of the states that have been encountered, the length of the u portion of the string and the length of the v portion of the string.

`Classify` is called each time the FSA processes a symbol in the input string. The next state is passed to `classify`, which maintains a list that contains a "classification" for each state encountered. When substring v is reached, `classify` factors v into x, y, and z portions. It does this by creating a dictionary that is keyed on the state and has as a value the index (i.e., the location in the input string) at which the state was touched. `Classify` thus continues to processes the v portion of the input string as long as no state is repeated (which can be checked by searching the dictionary). That is, each symbol of v is classified as part of the x portion, until a repeated state is found or the v portion is done. If a repeated state is found, then the index associated with the previous location when the state was touched is looked up in the dictionary. The substring consisting of the symbols in the input string from that index location to the current index location is reclassified to be the y portion.

Next, the algorithm checks for successive repeats of the y portion in the input string. If all the states encountered when processing the y portion immediately repeat in the same order as processing of v continues, a "y repeat" has been found (more than one y repeat may be found). When a mismatch is found, the rest of the v portion of the string is classified as z.

After the z portion has been determined, the rest is of the input string is labeled as w. The pseudocode for `classify` appears in appendix A.

As classification of the input string occurs, all the states and transitions are recolored as the new classification dictates. The y portion of the input string is given

highest priority and once the states and transitions of the loop traversed as y is processed are colored, this coloring remains even if any of these states or transitions are also traversed later as a different part of the string is processed by the FSA.

Integrating the Animator into a Hypertextbook

The pumping animator has been designed to be integrated into *Theory of Computing: The Hypertextbook*. The pump animator works well together with the other animators already designed for use in a chapter on regular languages and finite state automata, including deterministic and nondeterministic FSA applets, regular grammar applets, and regular expression applets. Exercises and examples using the pumping animator will be written for use in chapter 1 of the hypertextbook on regular languages and finite state automata, primarily in a section on the pumping lemma for regular languages.

EVALUATION

The pump animator has yet to be evaluated. Formative evaluation is needed to enhance the animator to better meet student needs, and summative evaluation is needed to determine how much students benefit from using the animator.

Formative evaluations are most important for determining how lessons and animation tools can be improved. Formative evaluations can be done by soliciting feedback from students and from observations made as students use the animators and associated lessons. These evaluations can be used to improve the animators' usability and effectiveness.

Summative evaluation requires different methods than formative evaluations. The most important requirement is that there be a control group and a target group of students to evaluate. For the purposes of this thesis the control group would be a group of students who learn the pumping lemma through traditional lectures and textbooks. The test group would use the pumping animator to learn the pumping lemma. Comparison of learning between the control and test group would help establish whether the active learning pumping animator helped students learn the pumping lemma better. Without the control group, it would be difficult to determine whether the pumping animator aided student understanding of the pumping lemma.

It is important to have control and test groups of students who exhibit similar capabilities. If the sample sizes are not large, it is probable that the students in the

control and target groups will not be equal in average ability. If, for example, more of the students in the control group already understand the concept being taught than the students in the target group, learning test results will be biased. Pretests can help resolve this problem if they clearly identify important differences between the test and control groups. If significant differences are detected between the two groups *prior* to the experiment, then a measure like average scores on the post-test cannot be used. Instead, measurement of student improvement between pre-test and post-test results can be used. This will measure whether the student's understanding of the concept being examined was increased *during* the experiment.

There are two problems with this approach, however. The first, for example, is that two different one-point improvements may not be the same. For instance, it might be easy to get from a score of 5 to a score of 6, but it might be hard to get from a score of 9 to a score of 10. This needs to be considered when analyzing the score improvements. The second is that the pre-test itself may prime students to focus on the concepts that they are expected to learn in the experiment, as pointed out by Hundhausen, Douglas and Stasko [4]. However, the advantage remains that testing for improvement more accurately measures whether learning takes place during an experiment. Experiments that have incorporated this approach have identified differences in learning between the control and target groups more often [4]. When carefully done, summative evaluations will return useful results.

Summative evaluations are needed to find answers to some interesting questions about the use of the pumping animator. A key question that can be partially answered is this: does use of an active learning animator for the pumping lemma (as described in this thesis) improve student comprehension of the pumping lemma over traditional approaches? To do an evaluation of this question a control group learning with traditional resources would need to be compared to a test group that uses the animation software in addition to traditional resources. To carry the experiment out, a pretest on the pumping lemma would be given to both groups. Then, the control group would be given a traditional lesson on the pumping lemma and would have access to a traditional textbook. The experimental group would be given similar treatment but, in addition, be required to use the pumping animator to learn about the pumping lemma. Afterwards, both groups would be given a post test on pumping lemma concepts (see appendix C for a sample test). The scores on the pre- and post-test for each group would then be compared to see if there is a statistically significant difference in learning by one group over the other.

There are many other interesting questions that could be explored through summative evaluation, including:

- Do students use the animator software for self-learning during their free time?
- Does use of the animator help augment lectures?
- Does use of the animator work for learning by independent study?

- Are students more excited and motivated about learning the pumping lemma when active learning applets are used?

CONCLUSIONS

As far as the author knows, the pumping animator presented in this thesis is the first of its kind. It is expected to become a useful resource for teaching and learning the pumping lemma for regular languages. The animator will play an important role in *Theory of Computing: The Hypertextbook*.

This thesis has also identified a set of five concepts necessary for understanding the pumping lemma. It was shown that the various modes of the pumping animator can be used to help students learn these concepts. The pumping animator will be integrated into the hypertextbook on the theory of computing to help students learn the pumping lemma.

Future Directions

Two major things remain to be completed based on the work of this thesis: (1) an extension to an active learning applet for helping students learn how to apply the pumping lemma to show that a language is not regular, and (2) formal formative and summative evaluations of the pumping animator. The first task will likely be the subject of another thesis and will present interesting technical and intellectual challenges. The second task can be carried out by instructors teaching the theory of computing courses. Formative evaluations are certain to point out improvements needed to the current pumping animator software.

Some minor features that need to be added include an exercise mode to the pumping classification mode and creating some tools to generate the HTML code needed for embedding the applet in a webpage.

This work will continue with the ongoing construction of *Theory of Computing: The Hypertextbook*, as it is extended into a complete teaching and learning resource covering a standard course on the theory of computing.

REFERENCES CITED

[1] Joshua J. Cogliati, Frances W. Goosey, Michael T. Grinder, Bradley A. Pascoe, and Rockford J. Ross. Realizing the promise of visualization in the theory of computing. submitted to Journal of Educational Resources in Computing, July 2004.

[2] Rockford J. Ross. Hypertextbooks: Animated, active learning, comprehensive teaching and learning resources for the web. *Software Visualization*, pages 269–283, 2002.

[3] Colleen Kehoe, John Stasko, and Ashley Taylor. Rethinking the evaluation of algorithm animations as learning aids: an observational study. *Int. J. Human-Computer Studies*, 54:265–284, 2001.

[4] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, 2002.

[5] Ayonike Akingbade, Thomas Finley, Diana Jackson, Pretesh Patel, and Susan H. Rodger. Jawaa: Easy web-based animation from cs 0 to advanced cs courses. *Thirty-fourth SIGCSE Technical Symposium on Computer Science Education*, 35:162–166, 2003.

[6] Camillan Huang. Virtual labs: E-learning for tomorrow. *PLoS Biology*, 2(6):734–735, June 2004.

[7] Michael Sipser. *Introduction to the Theory of Computation*, chapter 1, pages 31–83. PWS Publishing Company, 1997.

[8] Michael Thomas Grinder. *Active Learning Animations for the Theory of Computation*. PhD thesis, Montana State University, Comupter Science Department, December 2002.

[9] Michael T. Grinder. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. *Twenty-fourth SIGCSE Technical Symposium on Computer Science Education*, 35:157–161, March 2003.

[10] Michael T. Grinder, S. B. Kim, Teresa L. Lutey, Rockford J. Ross, and Katie F. Walsh. Loving to learn theory: Active learning modules for the theory of

computing. *ThirtyThird SIGCSE Technical Symposium on Computer Science Education*, 34(1):371–375, February 2002.

[11] Susan Rodger. PumpLemma. FTP: ftp://ftp.cs.duke.edu/pub/rodger/tools/-PumpLemma.tar.2-97.gz, February 1997.

[12] Anna O. Bilska, Kenneth H. Leider, Magdalena Procopiuc, Octavian Procopiuc, Susan H. Rodger, Jason R. Salemme, and Edwin Tsang. A collection of tools for making automata theory and formal languages come alive. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, pages 15–19, 1997.

APPENDIX A

Pseudocode for the Classify Algorithm

This is a pseudocode description of the classify algorithm. It is presented as pseudocode instead of Java code because the pseudocode is more compact and easier to understand.

```
classify(states, lengthSymbols, lengthU, lengthV, checkYRpeats):
        classification <- new list size lengthSymbols
        classification[0 .. min(lengthSymbols,lengthU)] <- u
        statesSeen <- new dictionary
        lengthV <- min(lengthV,lengthSymbols - lengthU)
        index <- lengthU
        while(index < lengthU + lengthV
                 and not statesSeen.contains(states[index])):
              classification[index] <- x
              statesSeen.add(states[index],index)
              index <- index + 1
        if(index < length(states)
                 and statesSeen.contains(states[index])):
              startIndex <- statesSeen[states[index]]
              classification[startIndex .. index] <- y
              afterIndex <- index
              if(checkYRepeats):
                    yLength = index - startIndex
                    count = 1
                    while(statesSeen[startIndex .. index]
                          = statesSeen[startIndex + count*yLength
                                .. index + count*yLength]):
                           statesSeen[startIndex + count*yLength
                                .. index + count*yLength] <- y count
                           count <- count + 1
                    afterIndex <- startIndex + count*yLength
              classification[afterIndex .. lengthU + lengthV] <- z
          classification[lengthU + lengthV .. lengthSymbols] <- w
          return classification
```

APPENDIX B

Mode Documentation

Pumping Animator Modes

The pumping animator uses various modes to run. The primary modes are specified in the Mode parameter passed to the applet, which can take on the following values:

compare — This is the mode of Grinder's original FSA animator. In this mode, the FSA that a student builds can be compared to a hidden, correct FSA that an instructor created.

pumping — This mode shows the pumping classification on the input string.

pumpingsubstring — This mode shows the pumping classification on a substring of the input. In contrast to the pumping mode, this mode allows different substrings to be selected for pumping analysis.

repeat — This mode will color the states and the transitions based on the number of times that the state or transition has been touched as the input string is processed.

loop — This mode will identify the substrings of the input string that cause the FSA to loop.

Secondary Mode Parameters

Some of the primary modes have secondary modes. When using loop mode there are several possible ways of doing loop mode that are selected with the parameter LoopHints. LoopHints can have the following values:

no — No loop information is displayed, except whether all substrings causing loops have been found by the student.

number — The number of substring causing loops is displayed, but not the substrings themselves until they are identified by the student.

show — All substrings causing loops are displayed.

The pumping modes also have a YRepeats parameter that determines whether or not y-repeats are to be displayed. If the value of YRepeats is true, then y-repeats will be displayed.

General Parameters

There are several setup parameters for the pumping animator. These setup parameters are:

FSAFile — The filename of the FSA XML file that is initially loaded when the applet starts.

TargetFile — The filename of the FSA XML file that is used to compare against the FSA the student creates. This FSA is hidden from the student.

Tape — The value of the initial string that is placed on the tape.

Using Parameters

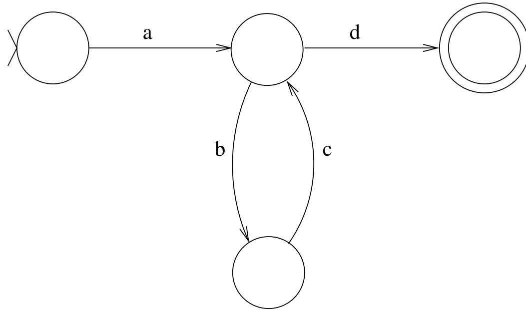
The parameters can be transmitted to the pumping animator in three different ways. First, they can be passed (with the exception of FSAFile and TargetFile) on the command line to the application (e.g., `Mode=pumping`). Second, they can be passed to the applet as attributes in the embed tag (e.g., `Mode=pumping`). Third, when using the applet tag, they can be passed with the tag param (e.g., `<param name="Mode" value="pumping" />`).

APPENDIX C

Evaluation Test

Sample questions for Pre and Post-test

1. Notice that the string `abcd` is accepted by the FSA depicted below. What portion of the string `abcd` can be repeated such that the resulting string will also be accepted by this FSA:



2. Find all the substrings of the string `abcbcd` that force a loop when it is run on the above FSA.

3. Formally prove that the language $\{\mathtt{a}^n\mathtt{b}^m\mathtt{c}^n | n, m \geq 0\}$ is not regular.

4. How would you prove that the language of strings that have the same number of `01`'s as `10`'s is regular or not?

5. Prove that the language in the previous question is regular, or prove that it is not regular.

6. Prove that the language $\{\mathtt{a}^n\mathtt{b}^m | n, m \geq 0\}$ is regular.

7. You meet a person named Bill on internet chat who claims to have a four-state FSA that can recognize the language $\{\mathtt{a}^n\mathtt{b}^{2n} | n > 0\}$. Bill will not show the

FSA to you, but will give the FSA to Trent (whom both Bill and you trust), and Trent will run any strings that you send him and report whether Bill's FSA accepts or rejects the strings. What strings will you give to Trent and why?