Increasing Interaction and Support in the Formal Languages and Automata Theory Course*

[Extended Abstract]

Susan H. Rodger
Computer Science Department
Duke University
Durham, NC 27708
rodger@cs.duke.edu

Jinghui Lim
Computer Science Department
Duke University
Durham, NC 27708

Stephen Reading
Computer Science Department
Duke University
Durham, NC 27708

ABSTRACT

The introduction of educational software such as JFLAP into the course Formal Languages and Automata (FLA) has created a learning environment with automatic feedback on theoretical topics. In this paper we show how we further increase the interaction in the FLA course with the expansion of additional theoretical topics in JFLAP, and how we have added grading support into JFLAP for instructors.

Categories and Subject Descriptors

F.4.3 [**Theory of Computation**]: Mathematical Logic and Formal Languages Formal Languages; D.1.7 [**Software**]: Programming Techniques Visual Programming

General Terms

Theory

Keywords

JFLAP, automata, formal languages, pumping lemma, Moore machine, Mealy machine

1. INTRODUCTION

The teaching of formal languages and automata (FLA) is moving from a pencil and paper environment to an interactive learning environment with the introduction of educational software to experiment with the many concepts in this course. This approach has both advantages and disadvantages. The advantages allow students to experiment with concepts that would be difficult and tedious to do on paper, and to receive immediate feedback on problem solving. The disadvantages include the difficulty for the instructor choosing to move this route with a large number and variety of tools being developed, the added burden of grading homeworks in a format other than paper, and the difficulty on students learning several tools with different formats.

A large variety of tools have been developed to experiment with one or more concepts from the FLA course. We list a few of those tools here, there are many more[5]. Turings World[1] is for experimenting extensively with Turing machines. Forlan[10] is a toolset for experimenting with regular languages. Language Emulator[12] is a toolkit for regular languages and includes Moore and Mealy machines. The tool jFAST[13] allows experimentation with finite automata, pushdown automata and Turing machines. Grinder[4] focuses on finite state automata. RegEx[2] allows experimentation with regular expressions.

A single tool that covers a large number of topics takes enormous development time, but can be easier on instructors and students in moving from one concept to another. There are three efforts to create a large collection of FLA concepts in one tool. Taylor[11] explores many types of machines including Turing machines, finite automata, pushdown automata, and linear bounded automata using the software *Deus Ex Machina*, but does not include support for other topics such as grammars or transformations between grammars and automata. Ross[3] is in the process of developing an extensive hypertextbook called Webworks that will include many topics in automata theory, and incorporates text, sound, pictures, illustrations, slide shows, video clips and active learning models. This is a huge development effort in the works for several years now. JFLAP[8, 9], described in more detail in Section 2, is a software tool under development for over fifteen years to incorporate many topics in automata theory in one tool including automata, grammars and experimentation with proofs.

In this paper we show how we further increase the interaction in the FLA course with the expansion of additional theoretical topics in JFLAP, and how we encourage the creation of JFLAP files for grading by providing grading support for instructors within JFLAP. The main additional theoretical topics in JFLAP are described in Section 3 and include Moore and Mealy machines, and games for learning about regular and context-free pumping lemmas. The batch grading support is described in Section 4, and includes an

*The work of all three authors was supported in part by the National Science Foundation through grant NSF DUE CCLI-EMD 0442513

ITiCSE'07, June 23–27, 2007, Dundee, Scotland, United Kingdom.

interface for loading and grading a large number of student JFLAP files.

2. OVERVIEW OF JFLAP

JFLAP (Java Formal Languages and Automata Package) is instructional software for experimenting with automata and grammars, but goes further in allowing one to experiment with proofs and applications related to these topics. JFLAP's main feature is the ability to experiment with theoretical machines and grammars. With JFLAP one can build and run user-defined input on finite automata, pushdown automata, multi-tape Turing machines, regular grammars, context-free grammars (CFG), unrestricted grammars, and L-systems. After constructing the automaton or grammar, one can trace through a single input string or receive automatic feedback on multiple inputs.

JFLAP's second feature is the ability to construct in steps the proof of the transformation of one form to another form. For example, after constructing a nondeterministic finite automata (NFA), one can step through its conversion to a deterministic finite automata (DFA), then to a minimal state DFA, and then to regular grammar. As another example, one can build a CFG and construct in steps the equivalent nondeterministic PDA.

JFLAP's third feature is the experimentation with applications of theoretical material. One example is experimenting with parsing by constructing the SLR(1) parse table in steps, and then stepping through the parsing of input strings and the construction of the equivalent parse tree. The construction of the parse table includes building a special DFA that models the parsing process, thus seeing a use for a DFA. Another application is building an L-system grammar of a plant, and rendering it to watch a simulation of the plant growing.

3. NEW TOPICS IN JFLAP 6.0

In this section we describe new topics we have integrated into JFLAP. All these topics allow students to experiment with them and receive immediate feedback.

One new topic is transducers. A transducer is a finite automaton that generates output. JFLAP now provides experimentation with two types of transducers: Mealy machines for generating output based on transitions and Moore machines for generating output based on states.

Another new topic is the pumping lemma. We provide pumping lemma games for both the regular pumping lemma and the context-free pumping lemma. Given a language, the student plays against the computer to help categorize the language. For example, if the language is not regular, the game will prove it is not regular, and if the language is regular, a partition is found to show the pumping lemma holds for that language.

3.1 Mealy Machines

A Mealy machine is similar to a finite state automaton, but it does not have final states and it can produce output through its transitions. Adding Mealy machines to JFLAP expands on applications of automata as Mealy machines can model problems in many disciplines. We first provide a definition of a Mealy machine and then give an example of a Mealy machine implemented in JFLAP.

Definition: A Mealy machine is defined as a 6-tuple, $(Q,\Sigma,\Gamma,\delta,\omega, q_0)$, where

- Q is finite set of states
- Σ is input alphabet
- Γ is output alphabet
- q_0 is initial state
- δ is the transition function, δ:Q×Σ→Q
- ω is the output function, ω:Q×Σ→ Γ

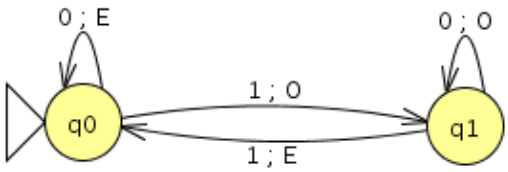


Figure 1: A Mealy Machine built with JFLAP

Input	Result
10	OO
110	OEE
101	OOE
1000	OOOO
1010	OOEE
1111	OEOE

Figure 2: Mealy Output from Several Inputs

Using JFLAP we build a Mealy machine to output information describing whether there is an odd number of 1's in a string. Since a Mealy machine doesn't have a final state, we will output the current status of a binary string based on how many digits have been viewed. Figure 1 shows the corresponding Mealy machine. There are two states q_0 and q_1, with q_0 indicated as the start state. The meaning of state q_0 is an even number of 1's have been seen and the meaning of state q_1 is an odd number of 1's have been seen. The transition $1; O$ from q_0 to q_1 means if a 1 is processed, output the letter O meaning an odd number of 1's have been processed, and the transition $1; E$ from q_1 to q_0 means if a 1 is seen output the letter E meaning there is an even number of 1's. The output for several inputs run in JFLAP is shown in Figure 2.

3.2 Moore Machines

A Moore machine is a transducer similar to a Mealy machine, but generates output based on states instead of transitions. Like the Mealy machine, the definition of a Moore machine is a 6-tuple, but with ω defined as $\omega : Q \rightarrow \Gamma$.

We build an example of a Moore machine in JFLAP shown in Figure 3 that divides binary numbers in half with truncation. One way to divide a binary number in half is to remove its last digit. However, Moore machines process the input string from left to right and do not have final states. Thus our algorithm keeps track of two symbols at a time (the previous and current symbol) and outputs the previous symbol.

This example also illustrates a feature of JFLAP that allows one to change the name of states to more meaningful names. The states in this machine have been renamed to be the previous and current symbol seen. Here the start

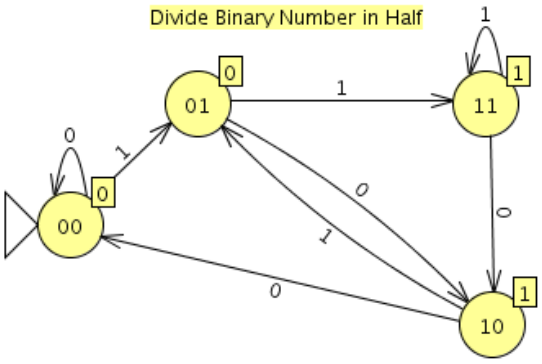


Figure 3: Moore Machine - Dividing Binary Numbers

Input	Result
10	001
110	0011
101	0010
1000	00100
1010	00101
1111	00111

Run Inputs | Clear | Enter Epsilon | View Trace

Figure 4: Moore Machine Output Runs in JFLAP

state (indicated by the triangular arrow) has been renamed from the default q_0 to 00. Assuming 0's preceeding any binary number are ignored, this start state represents the two previous and current nonexistant symbols as both 0 before processing any input. The output associated with each state is shown as a box attached to the top-right side of the state. The output for each state in this example is the previous symbol processed, which also happens to be the leftmost symbol of the state name. Thus, the output for state 00 is 0 and the output for state 10 is 1.

Let's process the string 101. Starting in state 00 automatically outputs the output for that state, 0. Processing the 1 moves to state 01 and outputs 0. Processing the 0 moves to state 10 and outputs 1. Processing the final 1 moves to state 01 and outputs 0. Thus the resulting output is 0010, which is equivalent to 10. Results of several inputs for this problem run in JFLAP are shown in Figure 4.

3.3 Regular Pumping Lemma Game

The pumping lemma is a more abstract concept than an automaton. In the FLA course, students experiment with a regular language in the form of an automaton, which is a concrete item that can be built and traced through with input. Then they are asked to identify a language, whether or not it is regular. If they can build a DFA for the language, they can easily answer the question. If they cannot, then they must consider proving the language is not regular. One such procedure is to use the pumping lemma. In this section we describe how we have created a pumping lemma game students can play to learn this concept. We modeled our pumping lemma game after the descriptions in [6].

First, we give the definition of the pumping lemma for regular languages.

Pumping Lemma: Let L be an infinite regular language. Then there is a constant $m > 0$ such that for any $w \in L$, with $|w| \geq m$, w can be partitioned into three parts $w = xyz$ with

$$|xy| \leq m$$
$$|y| \geq 1$$
$$xy^i z \in L \quad \text{for all } i \geq 0$$

The pumping lemma can be used to prove a language is not regular.

The game approach with JFLAP involves two players, the student and JFLAP. JFLAP's goal is to prove the language is not regular. The student's goal is to make it as hard as possible. The game proceeds as follows.

First the student selects a language for the game (one of nine choices) and selects an integer for m, the constant in the pumping lemma. JFLAP then picks a string in the language such that the string is of length greater than or equal to m. The student then has to partition the string into the three parts x, y, and z such that $xy^i z$ is in the language for all $i \geq 0$.

Here is an example shown in Figure 5 for the language $\{a^n b^n | n \geq 0\}$. In step 1, the user selected 7. In step 2 JFLAP shows the string $a^7 b^7$ and asks the user to select a decomposition. Step 3 starts with the slider bars all the way to the left. The user slid the first bar over two spaces to indicate the selection of $x = aa$, the second bar over an additional two spaces to indicate the selection of $y = aa$ and z becomes the rest of the string, $z = a^3 b^7$. JFLAP provides helpful messages to guide the user. When the user selects x, the second slider bar moves along also, and if the user slides the second bar back to the left, an error message is shown. If the user slides the bar too far to the right, making the condition $|xy| \leq m$ not true, JFLAP displays a message showing this condition has been violated and the user must make another selection. Once the user has a correct partition, in step 4, JFLAP shows a choice of i that gives a contradiction, 0 in this case and the generated string that is not in the language. Step 5 provides further explanation by providing an animation of building the pumped string that is not in the language.

One has a choice of nine languages to choose from, one of which is regular. In this case, the student can find a partition that allows the string to be pumped.

3.4 Context-Free Pumping Lemma

The pumping lemma game for context-free languages (CFL) is more complicated as there may be many cases to consider. We first give the definition of the pumping lemma for CFL's.

Pumping Lemma for CFL's Let L be an infinite context-free language. Then there is a constant $m > 0$, such that for any $w \in L$, with $|w| \geq m$, w can be partitioned into five parts $w = uvxyz$ with

$$|vxy| \leq m, \text{ (limit on size of substring)}$$
$$|vy| \geq 1, (v \text{ and } y \text{ not both empty})$$
$$\text{For all } i \geq 0, uv^i xy^i z \in \text{L}$$

In this pumping lemma game, the user starts by selecting a language from a list of nine possible languages. The window is partitioned into two parts. The left side looks similar

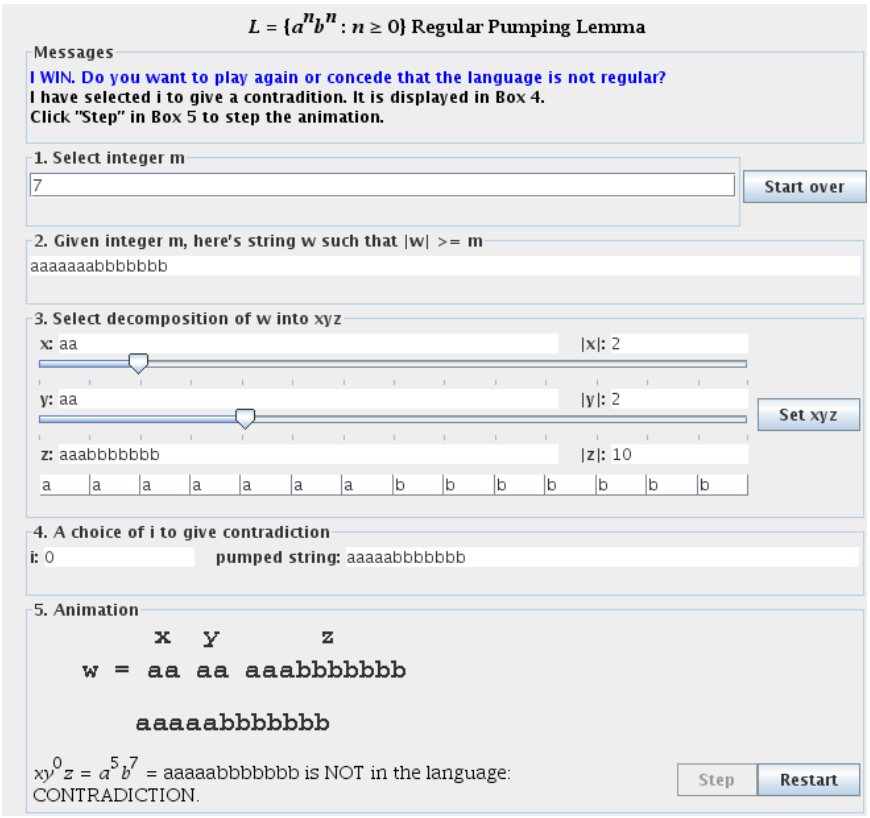


Figure 5: Regular Pumping Lemma Game

to the window for the regular pumping lemma. The right side of the window is for categorizing all the cases describing the choices for v and y (explained below), and starts as empty. The user plays the game by selecting the choice of m in step 1. JFLAP shows a string in step 2 that is in the language and whose length is greater than or equal to m. In step three the user moves the slider bars to partition the string into five parts u, v, x, y and z. In step 4 JFLAP finds an i to give a contradiction and step 5 shows an animation of building the string that provides the contradiction. At this point, the user can add the case to the right side of the window, and then start over to find another case.

We now explain the cases for a language. Consider the language $a^n b^n c^n$. The condition $|vxy| \leq m$ limits the size of this substring, but doesn't restrict where it can start. (Note that in the regular pumping lemma, the substring xy always started on the left end of the string.) To prove a language is not CFL, every possible partition must be considered and proven that it fails. Without a starting restriction for vxy we must consider all possible starting positions. We have done this by focusing on the contents for v and y and describing what type of letter they contain. For example, v has $a's$ then there are three cases for y: 1) y is all a's, 2) y is a's followed by b's and 3) y is b's. Note that y cannot have c's or $|vxy| > m$. In this example, there are 11 cases that the student must find. The user can find as many cases as they want, and at some point they can select "Find" to show all the cases.

3.5 Other New Features of JFLAP 6.0

We would like to mention a few other features that have been added to JFLAP. Annotations (called notes) can now be added in the editor pane at any location. In Figure 3 we added a note with a title to the Moore machine. Copying portions of a drawing is now easier by allowing one to select a bounding box area to copy.

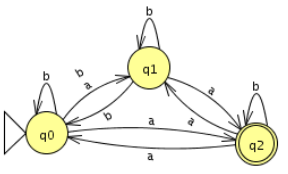
Multi-run input has been added to Grammars. In previous versions of JFLAP, only one input could be tested at a time on a grammar, with the result being either a derivation or a parse tree. Now one can run several inputs at the same time, and then select one at a time from the results to view the derivation or parse tree.

One change to make JFLAP fit with more textbooks has been a new preference to select the representation of the empty string, either *lambda* or *epsilon*. JFLAP stores your preference in a file and displays the empty string from any file with your preference.

4. BATCH GRADING SUPPORT

In previous versions of JFLAP, running multiple inputs at once for an automaton was added to aid in grading and to provide students with faster testing. In particular, one machine would be loaded and run on several inputs. The next machines loaded would use the same inputs, so the grader would not have to reenter them. However, grading could still take a long time for a large course as each machine has to be loaded and run.

In JFLAP 6.0, batch grading support has been added that



File	Input	Result
fa1.jff	aabbaabb	Reject(Accept)
fa1.jff	aaaa	Reject(Accept)
fa1.jff	aaab	Reject(Accept)
fa1.jff	aaabaaab	Reject(Accept)
fa2.jff	aabbaabb	Reject(Accept)
fa2.jff	aaaa	Reject(Accept)
fa2.jff	aaab	Reject(Accept)
fa2.jff	aaabaaab	Reject(Accept)
fa5.jff	aabbaabb	Accept
fa5.jff	aaaa	Accept
fa5.jff	aaab	Accept
fa5.jff	aaabaaab	Accept

Figure 6: Batch Grading Three Automata

allows one to load and produce output for multiple machines at once. Before using, the grader should place all the files in one directory and create an input text file with accept or reject status for each string. From the main JFLAP menu one selects *Batch* and then selects the automata files to run followed by selecting the input file. A window then appears with a picture of one of the automaton in the left side of the window, and the right side of the window lists the name of the files and the input strings to test. After selecting *Run Inputs* the results are displayed. For example, Figure 6 shows a run with three automata named fa1.jff, fa2.jff and fa5.jff. The machine *fa5.jff* has been selected and highlighted, and its graphic picture is shown on the left side of the window. On the right side are shown the results of running four input strings on each of these three files. The results for file *fa1.jff* are shown first, then the results for file *fa2.jff* and then the results for *fa5.jff*. The file *fa5.jff* accepts all four strings, and the other two files reject all four strings.

There are several options with batch grading. By selecting a different automaton on the right, the picture on the left changes to correspond to that automaton. One can add an additional automaton, and it is added with all the current inputs. One can add an input string, adding a line for each automaton. Files can be selected and edited to fix a bug. A trace for a specific automaton and string can be viewed. Most important, the results can be saved to files. Selecting *Save Results* writes each input string and its result for each file to a file of the same name with a ".txt" extension added. These files can be returned to students.

Batch grading is available for finite automata, pushdown automata, Turing machines and grammars.

5. USE OF JFLAP

JFLAP is used world wide in over 160 countries in formal languages courses, compiler courses, discrete math courses and artificial intelligence courses. The JFLAP web site [7] tracks usage of JFLAP. This web site allows users to download JFLAP for free and has had over 35,000 downloads since 2003. This site provides many resources including sample files, tutorial, slides, and papers.

JFLAP is used at Duke University in the course CompSci 140 and informal feedback from students over the past ten years has been very positive. JFLAP is currently undergoing a formal two-year study from 2005-2007 with twelve universities participating to evaluate JFLAP's effectiveness in learning automata theory topics.

6. CONCLUSION

We continue to develop JFLAP into an extensive tool covering many of the topics in a formal languages course including experimentation with automata, grammars, and proofs involving these concepts. We have recently added the experimentation of transducers to JFLAP, Moore and Mealy machines, and a new approach of games for learning the pumping lemmas. We also address concerns of grading JFLAP files by providing an interface for grading a large number of JFLAP files.

7. ACKNOWLEDGMENTS

Thanks to Peter Linz for suggestions on improving the pumping lemma interface.

8. REFERENCES

[1] J. Barwise and J. Etchemendy. *Turing's World 3.0 for the Macintosh*. CSLI, Cambridge University Press, 1993.

[2] C. W. Brown and E. A. Hardisty. Regexex: An interactive system providing regular expression exercises. In *Thirty-eighth SIGCSE Technical Symposium on Computer Science Education*, page (to appear). SIGCSE, March 2007.

[3] J. Cogliati, F. Goosey, M. Grinder, B. Pascoe, R. Ross, and C. Williams. Realizing the promise of visualization in the theory of computing. *JERIC*, 5, 2005.

[4] M. T. Grinder. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. In *Thirty-fourth SIGCSE Technical Symposium on Computer Science Education*, pages 157–161. SIGCSE, February 2003.

[5] V. J. Harvey and S. H. Rodger. Editorial for the special issue on software support for teaching discrete mathematics. *Journal on Educational Resources in Computing*, 5(2):1–16, June 2005.

[6] P. Linz. *An Introduction to Formal Languages and Automata, 4th Edition*. Jones and Bartlett, Sudbury, MA, 2006.

[7] S. H. Rodger. Jflap web site, 2006. www.jflap.org.

[8] S. H. Rodger, B. Bressler, T. Finley, and S. Reading. Turning automata theory into a hands-on course. In *Thirty-seventh SIGCSE Technical Symposium on Computer Science Education*, pages 379–383. SIGCSE, March 2006.

[9] S. H. Rodger and T. W. Finley. *JFLAP - An Interactive Formal Languages and Automata Package*. Jones and Bartlett, Sudbury, MA, 2006.

[10] A. Stoughton. Experimenting with formal languages. In *Thirty-sixth SIGCSE Technical Symposium on Computer Science Education*, page 566. SIGCSE, February 2005.

[11] R. Taylor. *Models of Computation and Formal Languages*. Oxford University Press, New York, 1998.

[12] L. F. M. Vieira, M. A. M. Vieira, and N. J. Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. In *Thirty-fifth SIGCSE Technical Symposium on Computer Science Education*, pages 135–139. SIGCSE, March 2004.

[13] T. M. White and T. P. Way. jfast: A java finite automata simulator. In *Thirty-seventh SIGCSE Technical Symposium on Computer Science Education*, pages 384–388. SIGCSE, March 2006.