Using JFLAP to Interact with Theorems in Automata Theory

Eric Gramond and Susan H. Rodger*
Duke University, Durham, NC
rodger@cs.duke.edu

Abstract

An automata theory course can be taught in an interactive, hands-on manner using a computer. At Duke we have been using the software tool JFLAP to provide interaction and feedback in CPS 140, our automata theory course. JFLAP is a tool for designing and running nondeterministic versions of finite automata, pushdown automata, and Turing machines. Recently, we have enhanced JFLAP to allow one to study the proofs of several theorems that focus on conversions of languages, from one form to another, such as converting an NFA to a DFA and then to a minimum state DFA. In addition, our enhancements combined with other tools allow one to interactively study LL and LR parsing methods.

1 Introduction

Traditionally the automata theory course has been taught without computers. Students work on assignments using pencil and paper with no immediate feedback. As a result, this course can be more difficult for some students than most of the other computer science courses which have hands-on interaction in the form of programming.

There are two reasons why many students have more difficulty with the automata theory course than other computer science courses. The first reason is that automata theory has more mathematics than most computer science subjects. Discrete mathematics is usually a prerequisite or taught as part of the course. The automata theory course consists of mathematical notation that must be grasped in order to understand the proofs of the theorems studied. The second reason is that students do not receive immediate feedback when working problems using pencil and paper. Weaker students especially need to work additional problems to understand concepts, and need to know if their solutions are correct.

The notation in the automata theory course is needed to describe theoretical representations of languages (automata and grammars). Tools such as JFLAP provide students an alternative visual representation and a chance to create and simulate these representations. As students interact and receive feedback on concepts with the visual representations provided in JFLAP, they may become more comfortable with the mathematical notation used in the formal representation.

Recently we have extended JFLAP to allow one to interact with proofs of several theorems that focus on conversions of languages from one form to another. In most of the conversions, the user creates the new representation with aid from JFLAP. The conversions in JFLAP are nondeterministic finite automaton (NFA) to deterministic finite automaton (DFA), DFA to minimum state DFA, NFA to regular grammar, regular grammar to NFA, nondeterministic pushdown automaton (NPDA) to context-free grammar (CFG), and three algorithms for CFG to NPDA. Two of the CFG to NPDA conversions are useful in studying LL and LR parsing.

In this paper, we describe related results in Section 2 and the original version of JFLAP briefly in Section 3. The modifications to JFLAP for experimenting with proofs of theorems on transformations for regular and context-free languages are described in Section 4. Section 5 describes how JFLAP can be used to study LL and LR parsing. We describe how JFLAP is used in the classroom in Section 6 and give concluding remarks in Section 7.

2 Related Work

In another theoretical area of computer science, algorithms and data structures, many tools including [5, 6, 9, 11] have been developed for creating animations of algorithms that can be used in teaching. Studies show [1] the need for students to participate in creating the animations as opposed to just passively watching the animations. Not nearly as many tools have been developed for automata theory that provide hands-on interaction. In this section we discuss some automata theory tools others have developed or are currently

*The work of this author is supported by the National Science Foundation's Division of Undergraduate Education through grant DUE-9555084 and by the National Science Foundations's Computer and Information Science & Engineering Directorate through grant CISE-9634475.

working on.

The project WebLab [4] includes concept animation, the activity of animating a particular concept, and it presents examples for a Java animator for deterministic finite state automata. Their goal is to develop a repretory of animations from the theory of computing including automata, grammars, languages and NP-completeness.

Turing's World [2] is a Macintosh program that focuses on Turing machines. One can create and run Turing machines. A special feature allows one to define submachines, and then use the submachines to build quite complex Turing machines. Turing's world also allows one to build finite automata and nondeterministic machines.

3 JFLAP

In this section we briefly describe the original version of JFLAP [10], and in the next section we describe the new modifications to JFLAP.

JFLAP (Java Formal Languages and Automata Package) is a tool for creating and simulating several versions of automata, including finite automata, pushdown automata, 1-tape Turing machines and 2-tape Turing machines. The user creates a graph representing a transition diagram, labels the transitions, enters an input, and then steps through the execution of the machine. JFLAP allows one to create nondeterministic machines, with two choices for execution. In the "fast" mode, the user receives a message indicating either the input was not accepted, the input was accepted, or the execution is taking a long time. If the input was accepted, the user can select to step through an animation of the processing of the input to acceptance. In the "step" mode, the user starts in the start state and steps through the execution. All possible configurations reached are shown at each stage. However if the number of configurations exceeds 15, then the user must control the execution by freezing or removing some configurations.

At Duke we have used JFLAP with the textbooks [7, 8]. JFLAP has been designed with flexible definitions to allow for its use with most automata theory textbooks. For example, the transition for a pushdown automaton can have zero or more items in each field. That is, nothing is popped or one or more characters are popped, nothing is pushed or one or more characters are pushed, and input is ignored or one or more input characters are processed.

4 Modifications to JFLAP to study Proofs

The main modifications to JFLAP allow one to step through the proofs of several theorems representing the transformation of a language from one representation to another. In most cases, the user creates one representation using JFLAP and then JFLAP aids the user in creating an equivalent representation in another form. Transformations have been created for regular languages and context-free languages.

Other modifications made to JFLAP include an additional execution mode, and improved editing capabilities. A third execution choice is now available in which one can enter several input strings and receive the acceptance status of all the strings. The improved editing capabilities include the ability to move the labels on transitions, and to increase or decrease the size of loops.

In this section we give more detail and examples of the transformations.

4.1 Regular language transformations

There are four transformations for regular languages: converting an NFA to a DFA, a DFA to a minimum state DFA, an NFA to a regular grammar, and a regular grammar to an NFA.

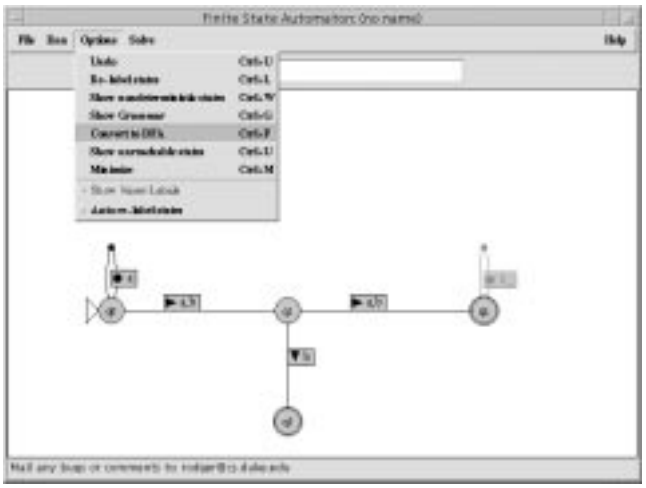

Figure 1: An NFA in JFLAP

In converting an NFA to a DFA, the user first creates an NFA in a JFLAP building window, and then selects the option to convert to a DFA. Figure 1 shows an NFA created with JFLAP and the options menu has selected the option to convert the NFA to a DFA. If the automaton created is already deterministic, the user is informed. Otherwise, a second building window appears. The user draws the corresponding DFA in this window. A state in the DFA may represent several states from the NFA. Thus, in the DFA, a state has an additional label in which the user enters the corresponding state numbers from the NFA. At any point during the construction of the DFA, the user can select a check feature which will report any errors made in the drawing. The user can also select an option to automatically expand a selected state, drawing the transitions out of this state. Another option can draw the complete DFA. Figure 2 shows the corresponding DFA to the NFA in Figure 1. There is an option to remove the labels attached to the states, revealing the state numbers.

In converting a DFA to a minimum state DFA, the user first creates a DFA in a JFLAP building window (or can use the resulting DFA from the conversion from NFA to DFA) and then selects the option to convert to a minimum state DFA. A second window appears in which a user will try to build two trees to determine which states are distinguishable from other states. Initially, all final states are grouped together as indistinguishable, and all nonfinal states are grouped together as indistinguishable (the roots of the two trees). A

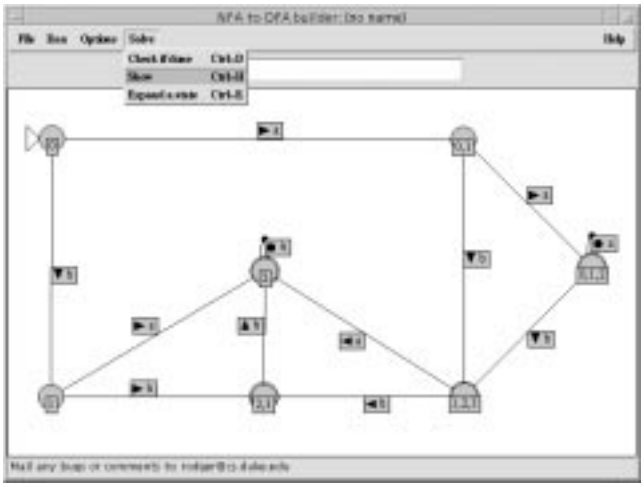

Figure 2: Corresponding DFA

node in the tree with more than one state is expandable to add two children if there is a letter of the DFA's alphabet to distinguish any of the two states. The user repeatedly selects letters that can distinguish states, defining the new children, until no nodes can be further expanded. The interaction in this window is similar to other windows. The user can create the tree themselves, have it partially expanded automatically, or have the complete trees generated automatically. Figure 3 shows the trees of distinguishable states for the DFA from Figure 2. The rightmost leaf node in the left tree contains two states 4 and 5 that are indistinguishable, meaning this leaf node cannot be further split. Once the correct trees are shown, the user selects an option to continue, and a third build window appears with states shown, each state is equivalent to a leaf in the trees. Using the information from the first build window, the user can complete the DFA by adding the appropriate arcs, or can select an option to automatically add them. Figure 4 shows the minimal state DFA for the DFA from Figure 2. Note that two of the states from Figure 2 were combined into one state in Figure 4.

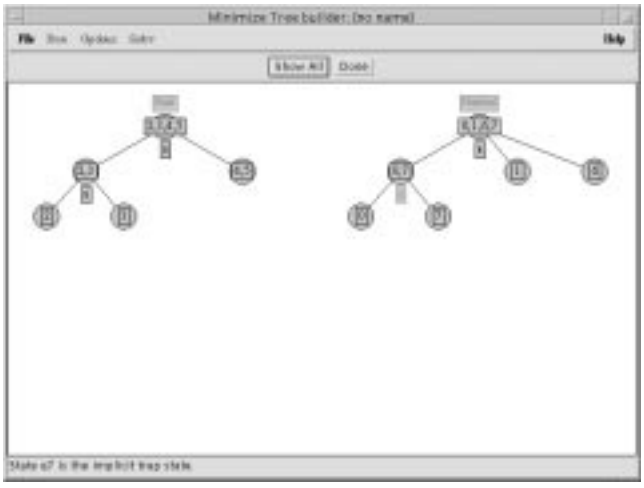

Figure 3: Tree of distinguishable states

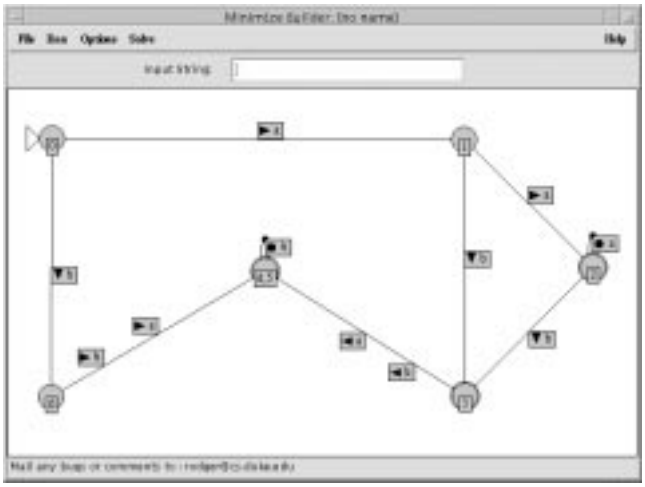

Figure 4: Minimum state DFA

In converting an NFA to a regular grammar, the user first creates an NFA in a JFLAP building window (or can use the resulting NFA or DFA from one of the other conversions) and then selects the option to convert to a regular grammar. The equivalent regular grammar appears in a new window.

In converting a regular grammar to an NFA, the user first enters a regular grammar in a grammar window, and then selects the option to convert to an NFA. A build window appears, and the user constructs the equivalent NFA, or can use other options to have the NFA automatically built.

4.2 Context-free languages

There are four transformations for context-free languages: converting an NPDA to a CFG, and three algorithms for converting a CFG to an NPDA.

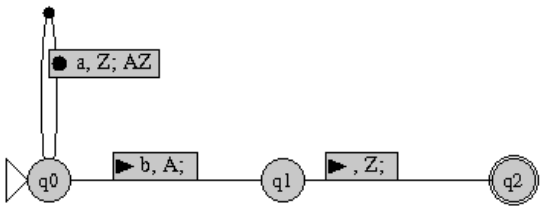

Figure 5: Pushdown Automaton Created with JFLAP

In converting an NPDA to a CFG, the user first creates an NPDA in a JFLAP building window, and then selects the option to convert to a CFG. The NPDA must have two additional requirements before conversion is allowed as described in [8]. For the first requirement, each transition must either increase or decrease the stack by one symbol. If the user's NPDA does not meet this requirement, the user can replace any transition by two or more transitions that all satisfy this requirement. For the second requirement, the NPDA must have only one final state, and the stack must be empty

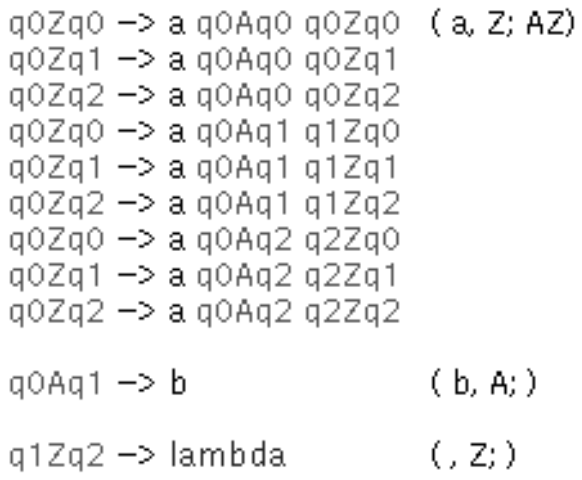

Figure 6: Corresponding CFG in JFLAP

for this state to be reached. Both of these requirements are presented in [8] as a build up to the theorem to convert an NPDA to a CFG. Once the user's NPDA is in the correct format, the conversion to CFG option will result in a new window with the corresponding CFG. The rules of the CFG that correspond to one transition from the NDPA are grouped together in one column and the corresponding transition from the NDPA shown in a second column.

Figure 5 shows an NPDA in the required format, and Figure 6 shows the corresponding CFG. Notice that for rules that increase the size of the stack by one character such as $(a, Z; AZ)$, there are several grammar rules generated. For rules that decrease the size of the stack by one character such as $(b, A;)$, there is only one corresponding grammar rule.

In converting a CFG to an NPDA, the user first enters a context-free grammar in the grammar window, and then selects the algorithm for conversion to an NPDA. Each algorithm assumes the grammar is in a specific form. One of the algorithms assumes the grammar is in Greibach Normal Form. The other two algorithms will work with any context-free grammar, but are used in illustrating LL and LR parsing, so one should enter an LL or LR grammar. After the algorithm has been selected, an NPDA building window appears with three states. The user must complete the NPDA with appropriate transitions based on the algorithm chosen. The user receives feedback as to whether or not their solution is correct. The solution can also be automatically generated.

5 LL and LR parsing

JFLAP in combination with the tool jeLLRap can be useful for examining LL and LR parsing. The tool jeLLRap (a Java version of LLparse and LRparse [3]) is an instructional tool for building LL(1), LL(2) and LR(1) parse tables from grammars, and then simulating the parsing of input. The underlying foundation of these parsing methods is a pushdown automaton combined with lookaheads to remove the nondeterminism. Modifications to JFLAP allow one to enter an LL or LR grammar and then construct the equivalent NPDA. Since this NPDA is most likely nondeterministic (and exponential in execution time on input), the student must guide the execution of the NPDA by choosing the appropriate transition using lookaheads. That is, the student must understand the LL and LR algorithms well in order to choose the appropriate transition.

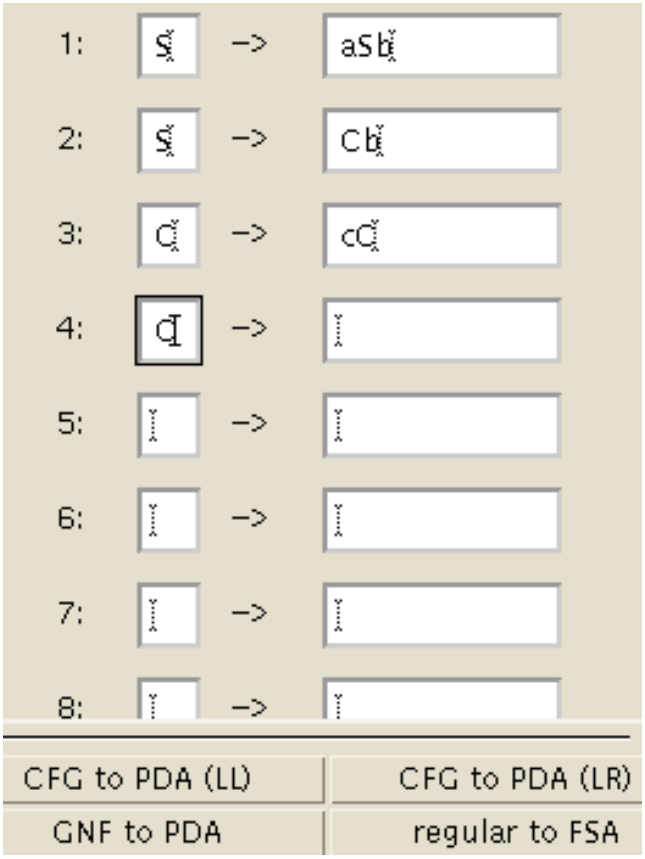

Figure 7: Context-free grammar

Figure 7 shows an LR(1) grammar with 4 rules has been entered. Figure 8 shows the corresponding npda for LR parsing. Note that this NPDA is nondeterministic as there are several places where one can either shift or reduce. If the naive user enters a small input string of size 6 and tries the fast run, the user will be informed that the execution is taking too long. In this case, the user must execute using the step run, and must understand how lookaheads are used in order to choose the correct configuration to continue tracing to reach acceptance.

6 Use of JFLAP in the classroom

The tool JFLAP can be used both in and out of the classroom. We use JFLAP during lecture to introduce topics, to work examples, and to illustrate how easy it is to build and run machines. For example, in lecture we ask students to create an NPDA for a specific language. We ask them to work in a group and give them five to ten minutes for

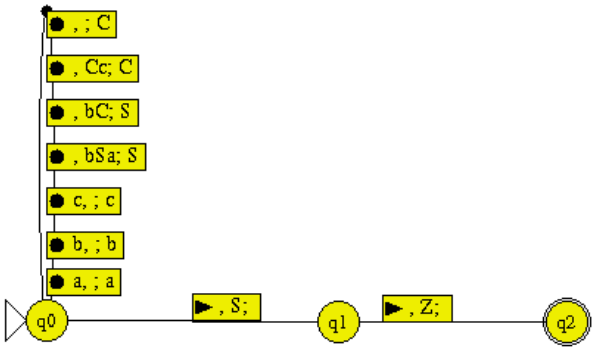

Figure 8: NPDA for LR parsing

this problem. Then with input from one group, we build the NPDA using JFLAP and run the machine on sample input. As an alternative problem, we give the students a language and an incorrect NPDA and ask them to determine if the NPDA is correct, and if it is incorrect, to figure out how to fix it. We can then show the NPDA is incorrect by running input it should accept, and then fix the NPDA based on the student suggestions. Outside of the classroom, students use JFLAP for homework assignments, to work additional problems, and to study for exams.

The original version of JFLAP has been used at several universities around the world. At Duke we have had positive results using JFLAP for several years. Student comments from the spring 1998 semester overwhelmingly stated that JFLAP was easy and helpful to use. Some students stated that with the first JFLAP lab they preferred to write the automaton on paper first and then draw it in JFLAP. But they were still happy to use JFLAP because it allowed them to test and debug their automaton. By the time of the third lab, they were comfortable enough to create the automaton using JFLAP instead of on paper. The main complaint with JFLAP was that the transition labels could not be moved. In our modifications to JFLAP this label is now slideable making it easier to layout the transition diagram.

7 Conclusion and Future Work

New modifications to JFLAP guide the user through several proofs of theorems that focus on transformations of languages from one form to another, where the forms are automata and grammars. Students are given a visual representation they can interact with in addition to the formal textual representation. This alternative hands-on representation provides a means for an interactive automata theory course.

We plan to continue to expand JFLAP to allow for the experimentation of other proofs of theorems in this course. More information on JFLAP can be obtained from the web address `http://www.cs.duke.edu/~rodger`

Acknowledgement JFLAP is a project that began in 1990 as NPDA and later FLAP and would not be possible without the work of many students including Dan Caugherty, Mark Losacco, Madga Procopiuc, and Tavi Procopiuc.

References

[1] A. Badre, C. Lewis, and J. Stasko, Empirically Evaluating the Use of Animations to Teach Algorithms, *Proceedings of the 1994 IEEE Symposium on Visual Languages*, p. 48-54, 1994.

[2] J. Barwise and J. Etchemendy, Turing's World 3.0 - An Introduction to Computability Theory, 1993.

[3] A. O. Bilska, K. H. Leider, M. Procopiuc, O. Procopiuc, S. H. Rodger, J. R. Salemme and E. Tsang, A Collection of Tools for Making Automata Theory and Formal Languages Come Alive, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 15-19, 1997.

[4] C. Boroni, F. Goosey, M. Grinder, R. Ross and P. Wissenbach, WebLab! A Universal and Interactive Teaching, Learning, and Laboratory Environment for the World Wide Web, *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, p. 199-203, 1997.

[5] M. Brown, ZEUS: A System for algorithm animation and multi-view editing. *Proceedings of the IEEE 1991 Workshop on Visual Languages*, p. 4-9, Kobe, Japan, Oct. 1991.

[6] P. Gloor, *AACE - Algorithm Animation for Computer Science Education*, IEEE Workshop on Visual Languages, p. 25-31, 1992.

[7] H. Lewis, C. Papadimitriou, *Elements of the Theory of Computation, Second Edition*, Prentice Hall, 1998.

[8] P. Linz, *An Introduction to Formal Languages and Automata, Second Edition*, D. C. Heath and Company, 1996.

[9] W. Pierson and S. H. Rodger, Web-based Animations of Data Structures Using JAWAA, *Twenty-ninth SIGCSE Technical Symposium on Computer Science Education*, p. 267-271, 1998.

[10] M. Procopiuc, O. Procopiuc, and S. Rodger, Visualization and Interaction in the Computer Science Formal Languages Course with JFLAP, *1996 Frontiers in Education Conference*, Salt Lake City, Utah, p. 121-125, 1996.

[11] J. Stasko, Tango: A Framework and System for Algorithm Animation, *IEEE Computer*, p.27-39, September 1990.