

A Visual and Interactive Automata Theory Course with JFLAP 4.0

Ryan Cavalcante^{*}
Dept of Computer Science
Duke University
Durham, NC 27708-0129

Thomas Finley[†]
Dept of Computer Science
Cornell University
Ithaca, NY 14853-7501

Susan H. Rodger[‡]
Dept of Computer Science
Duke University
Durham, NC 27708-0129
rodger@cs.duke.edu

ABSTRACT

We describe the instructional software JFLAP 4.0 and how it can be used to provide a hands-on formal languages and automata theory course. JFLAP 4.0 doubles the number of chapters worth of material from JFLAP 3.1, now covering topics from eleven of thirteen chapters for a semester course. JFLAP 4.0 has easier interactive approaches to previous topics and covers many new topics including three parsing algorithms, multi-tape Turing machines, L-systems, and grammar transformations.

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, pushdown automata, Turing machine, grammar, SLR parsing, LL parsing, L-system

1. INTRODUCTION

Many computer science students obtain only a superficial understanding of theory, even though theoretical concepts provide the fundamental basis for most areas of computer

^{*}The work of this author is supported in part by the National Science Foundation through grant NSF DUE-9752583.

[†]The work of this author is supported in part by the National Science Foundation through grant NSF DUE-9752583. This work was done while the author was at Duke University.

[‡]The work of this author is supported in part by the National Science Foundation through grant NSF DUE-9752583.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCSE'04, March 3-7, 2004, Norfolk, Virginia, USA
Copyright 2004 ACM 0-58113-798-2/04/0003 ...\$5.00.

science. In particular, a thorough understanding of the theory of formal languages and automata (FLA) is crucial in designing programming languages and compilers. However, the traditional FLA course is taught in a nonvisual and pencil-paper problem solving manner. Students find this approach frustrating as they have no visualization to relate to and they do not receive immediate feedback on problems. This contrasts starkly with the hands-on nature in most of their other computer science courses which contain programming assignments.

Over the past twelve years, several software tools have been developed that attempt to change the traditional FLA course from a textual presentation to a more visual and interactive approach. Many of these tools focus on one particular concept such as finite automata[9, 10] or Turing Machines[1]. One tool[11] allows experimentation with several machines, with a focus on Turing machines. A tabular approach [4] covers many topics but has no graphical representation. A web-based textbook [3] that is partially complete allows experimentation with concepts through applets. JFLAP [5] is an instructional tool that covers many FLA topics in a visual and interactive manner, from experimenting with machines and grammars to experimenting with related proofs.

In this paper we describe JFLAP 4.0[8], a new version of JFLAP that can be used with topics from eleven chapters of an FLA course, double the number of chapters of material JFLAP 3.1[5] covered. JFLAP 4.0 includes many new topics such as multi-tape Turing machines, L-systems, grammar transformations, and three types of parsing (LL(1), SLR(1), and brute force). JFLAP 4.0 has new approaches for converting NFA to regular expressions (RE), and for the conversion of a regular expression to an NFA. JFLAP 4.0 has new features such as combining two automata and comparing the equivalence of two automata. With these new additions, JFLAP 4.0 can be used with almost all topics in an FLA course. We describe JFLAP 4.0's differences from JFLAP 3.1, JFLAP 4.0's new additions, its use in a FLA course at Duke University and feedback from using JFLAP.

2. JFLAP 4.0 VS. JFLAP 3.1

This section compares JFLAP 4.0 and JFLAP 3.1's coverage of topics by referring to a typical FLA textbook [6]. A typical automata theory course would cover Chapters 1 through 11 of this textbook and possibly two additional chapters of material either from this book or outside of this textbook. We list those 11 chapters plus two extra topics.

- Ch. 1 Mathematical Preliminaries
- Ch. 2 Finite Automata
- Ch. 3 Regular Languages and Grammars
- Ch. 4 Properties of Regular Languages
- Ch. 5 Context-Free Languages
- Ch. 6 Simplification of Context-Free Grammars
- Ch. 7 Pushdown Automata
- Ch. 8 Properties of Context-Free Languages
- Ch. 9 Turing Machines
- Ch. 10 Other Models of Turing Machines
- Ch. 11 A Hierarchy of Formal Languages
- Extra 1 L-Systems
- Extra 2 LL and SLR Parsing

The following shows what part of the chapter (where 1 is most of the chapter) that JFLAP 4.0 and JFLAP 3.1 can be used with.

Chapter	JFLAP 3.1	JFLAP 4.0
Ch. 1		
Ch. 2	1	1
Ch. 3	3/4	1
Ch. 4		1/2
Ch. 5	1/2	1
Ch. 6		1
Ch. 7	1	1
Ch. 8		
Ch. 9	1/2	3/4
Ch. 10	1/4	1/2
Ch. 11		1/4
Extra 1		1
Extra 2		1

JFLAP 3.1 covered four chapters of material from this textbook, spread out over six chapters. JFLAP 4.0 covered nine chapters of material, spread out over eleven chapters.

3. JFLAP 4.0 NEW FEATURES

In this section we describe the many new topics and features of JFLAP 4.0. JFLAP is written in Java.

3.1 New Approach for NFA to RE

The new approach in JFLAP 4.0 for converting an NFA to a regular expression (RE) is more visual and easier to understand for the user than in JFLAP 3.1. In the new approach one first enters an NFA. For example, we have entered the NFA in Figure 1. The NFA must be in a particular format with exactly one final state, which cannot be the start state, and there must be exactly one transition from every pair of states. JFLAP helps the user in collapsing multiple transitions from pairs of states and in adding \emptyset between two states that did not previously have a transition.

All states except the initial and final state are removed one at a time. In our example we first remove state q_2 . When q_2 is removed, its meaning must be incorporated into all the other transitions. Figure 2 shows the regular expressions that will be placed on the arcs after state q_2 is removed, and Figure 3 shows the resulting NFA (actually a general transition diagram (GTG) since there are regular expressions on the arcs) after removing state q_2 . Similarly, state q_1 is removed and the resulting regular expression is shown in Figure 4.

3.2 New Approach for RE to NFA

In JFLAP 3.1, a bottom-up approach was used to convert an RE to an NFA. One constructed NFA submachines corre-

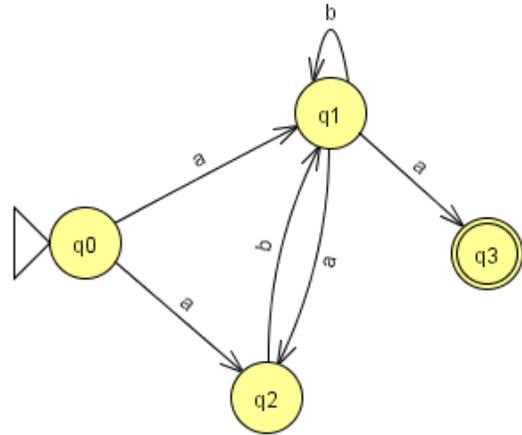


Figure 1: An NFA to convert

From	To	Label
0	0	\emptyset
0	1	a+ab
0	3	\emptyset
1	0	\emptyset
1	1	b+ab
1	3	a
3	0	\emptyset
3	1	\emptyset
3	3	\emptyset

Figure 2: Table in Removing state q_2

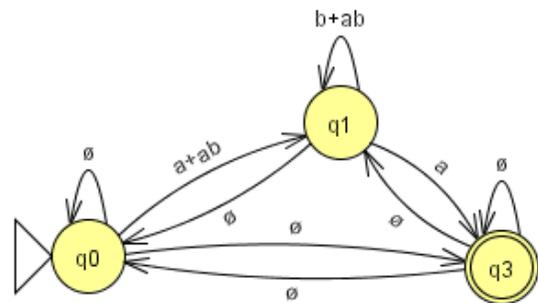


Figure 3: NFA after removing state q_2

$$\boxed{((a+ab)((b+ab)^*)a}$$

Figure 4: The equivalent regular expression

sponding to low level operands of the RE and then connected these submachines into larger and larger submachines until the final equivalent NFA was constructed. For example, to convert the RE $ab + (c + d)^*$ one first constructed two NFA that recognized the languages ab and $c + d$, then reformed the $c + d$ machine to recognize $(c + d)^*$, and then combined the two machines to recognize $ab + (c + d)^*$. How did JFLAP know to create the ab NFA, and then the $c + d$ NFA, and so forth? Implicitly JFLAP parsed the RE down to its lowest level operands. JFLAP 4.0 now makes this parsing more explicit without unduly complicating the conversion.

JFLAP 4.0 takes a top-down approach. The user works in the same direction the parser does, and observes and interacts with the parser. This serves to make the steps of the algorithm less “magical” by dividing the previously wholly implicit parsing of the RE into more easily comprehensible small chunks. One starts with a GTG with one initial state, one final state, and a single RE transition from the initial to the final state on the RE to convert. Using a modified automaton editor, the user repeatedly reduces each RE transition on an RE R into many RE transitions consisting of R ’s operands, and the user adds lambda transitions to duplicate the functionality of R ’s lost operators. At each stage the GTG is equivalent to the initial RE. After enough transition reductions the GTG becomes an NFA.

3.3 LL and LR Parsing

In JFLAP 4.0 one can start with a CFG and build an LL(1) parse table or an SLR(1) parse table through a series of steps, and then simulate the parsing of input strings. These tools were modeled after the LL and LR parsing in the tool JeLLRap[2].

We give an example of building the SLR(1) parse table for the grammar $S \rightarrow aSb, S \rightarrow b$. In JFLAP, the user first enters the CFG and selects the option to build the SLR(1) parse table. The window in Figure 5 appears with the grammar (not shown) and the fields in the window would be blank. The user would first enter in the FIRST sets for the variables, followed by the FOLLOW sets for the variables. Then the user would construct the DFA that models the parsing stack. Each state has marked rules or items associated with it. These items can be easily selected from a menu; they would be too tedious to type in as users had to do in JeLLRap. JFLAP gives hints along the way in the construction of the DFA such as an error message if you try to add a transition that doesn’t exist, or if you have the wrong items for a state. Once the DFA is complete, the user enters in the entries in the parse table. Figure 5 shows the completed First and Follow sets, DFA and parse table.

Once the parse table is complete, the user can parse an input string. The parsing window is divided into four views: the grammar, the parse table, the parse tree and the input. The user enters an input string and steps through the parsing. For each step, the stack is adjusted, the current move in the parse table is highlighted, the new additions to the parse tree are added and a message such as “shifting a” is displayed. The user can view either a parse tree, an inverted parse tree or the derivation table.

If the grammar is not LL(1) or SLR(1), the user is allowed to continue, building a parse table that will have conflicts. If there is a conflict in the SLR(1) parse table, the user can choose one of the conflicts as valid and proceed to parse input strings.

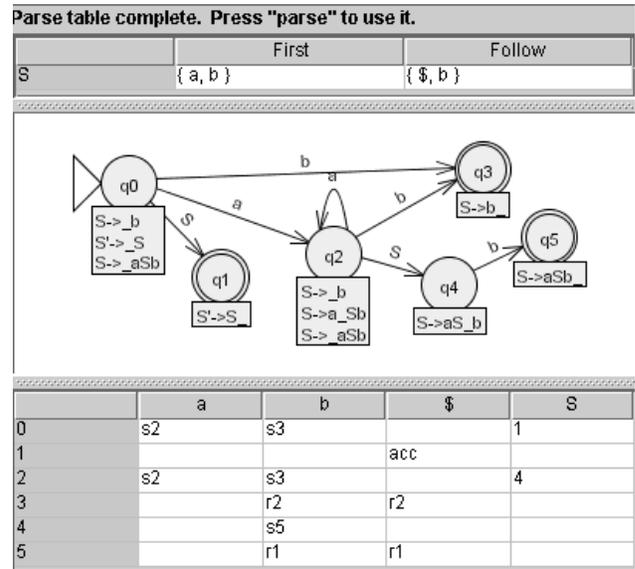


Figure 5: Building the SLR(1) Parse Table

3.4 Brute Force Parsing

The brute force parser in JFLAP 4.0 owes its heritage to Pâté’s brute force parser[2]. The brute force parser allows the derivation of strings in the language of restricted and unrestricted grammars through exhaustive search. JFLAP’s brute force parser improves over Pâté’s in speed, memory requirements, pruning strategies, and aesthetics.

One first enters a restricted or unrestricted grammar in JFLAP and selects the option to perform brute force parsing. The interface is similar to JFLAP’s interface for LL(1) and SLR(1) parsers. Figure 6 shows JFLAP’s brute force parser. The naïveté of the parser’s exhaustive search explains two differences in the interface: the parse table’s absence and the “pause” button’s presence. Additionally, unrestricted productions allow multiple symbols to be replaced rather than single variables; if the step from one sentential form to another requires an unrestricted production that replaces multiple symbols, the corresponding nodes are grouped together in a bracket, and the replacement symbols appear as children of that “bracket node” as shown in Figure 6 with aa replaced by b .

3.5 L-Systems

In JFLAP 4.0 one may create and render L-systems[7]. Figure 7 shows the editor for L-systems: from top to bottom are components for the axiom, rewriting rules, and initial values of graphical parameters (colors, line width, turn angles, etc.). Figure 8 shows JFLAP’s L-system renderer: the top of the window allows control of which derivation step to render (Figure 8 displays the 6th derivation), the large white panel shows derivation’s rendering (a tree), and controls on the bottom turn the L-system, useful for 3D renderings.

JFLAP’s L-systems have additional features beyond those standard in most L-systems tools. The turtle may turn in three dimensions, allowing the rendering of 3D structures. Some turtle commands take arguments, e.g. g will draw a line with default length, but $g(30)$ will draw a line 30 pixels long. Arguments may also be mathematical expressions. JFLAP also supports contextual rewriting rules that may

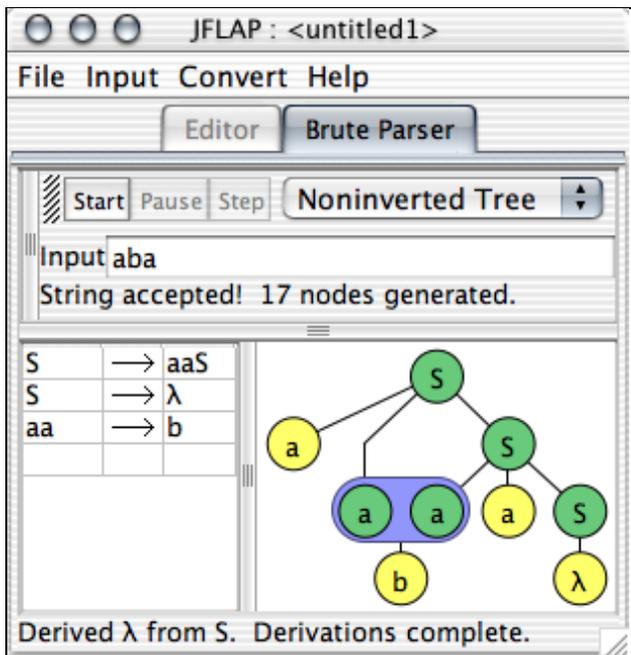


Figure 6: Brute Force Parser

rewrite a symbol only if it is prefixed and suffixed by certain symbols. JFLAP's L-systems are stochastic: if more than one rule may rewrite a symbol, a rule is uniformly randomly chosen from applicable rules to rewrite the symbol.

3.6 Transform Grammar

In JFLAP 4.0, one can transform a CFG to Chomsky normal form (CNF) in an easier to use interface than in Pâté[2]. One enters a CFG and then applies several algorithms to the CFG until the CNF is obtained. The algorithms are removing lambda productions, removing unit productions, removing useless productions, and converting the resulting grammar to CNF. Each algorithm has one or more interactive steps with some steps including a visual interpretation of the rules.

3.7 Other Features

Other new features in JFLAP 4.0 include the following. One can now create multi-tape Turing machines for up to five tapes. With three tapes, one can now create a Universal Turing machine. One can compare two finite automaton for equivalence. This allows students to check their design against another solution that may appear to be different. In particular, students can construct an automaton using properties (for example, reversal) and compare the result against a solution automaton. One can combine several automaton in the same window. This allows one to build a new automaton faster by using parts from other automaton.

4. JFLAP'S USE IN COURSE

We describe how we use JFLAP in teaching the FLA course CPS 140 at Duke University. JFLAP was designed to be general and to be used with most automata theory textbooks, such as [6]. We teach in a classroom with a computer whose monitor is displayed via a projector to a large screen. We use JFLAP during class in a number of ways.

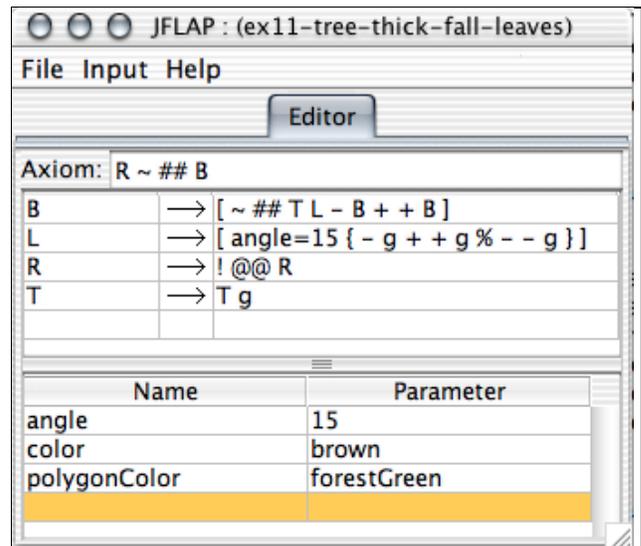


Figure 7: L-System Editor

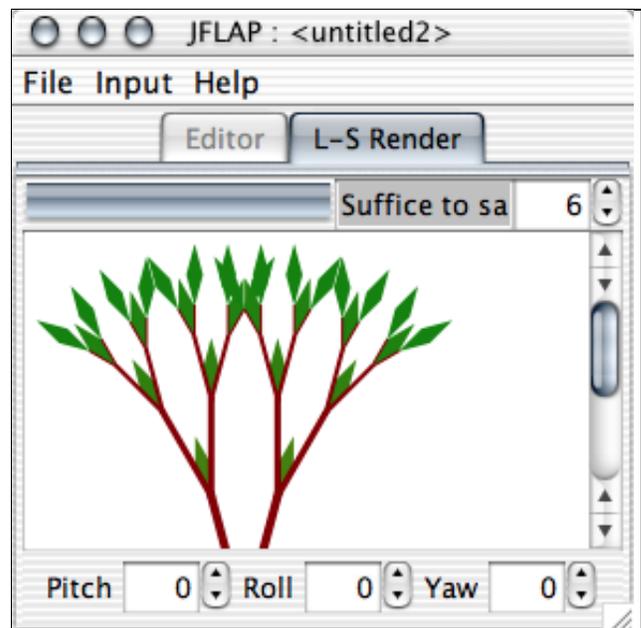


Figure 8: Rendering of Figure 7's L-system

- In the beginning we use JFLAP to show students how to use the editor to construct an automaton. For example, we demonstrate that states can be moved after they are created. In earlier years when students used JFLAP and we did not demo it in class, their designs were cluttered as they did not realize they could move states to make a prettier picture.
- We use JFLAP to solve problems during class. For example, we ask students to build an NPDA for the language ww^R (an even length palindrome). This NPDA cannot be built deterministically, and students are not used to thinking nondeterministically. They want to find the middle first. We build this NPDA with student input and test its correctness with several input strings.
- We use JFLAP to debug. For example, we will build an incorrect Turing machine for $a^n b^n c^n$ and ask students if it is correct and if not, for them to describe how to fix it. The machine we give them works for strings of the form $a^n b^n c^n$, however it also accepts incorrect strings. We ask for input strings that work and do not work and run them in class. Students then tell us how to fix the machine. We fix it and retry our input strings.
- We use JFLAP for illustrating proofs. For example we step through the conversion of a DFA to an RE.
- We use JFLAP to relate to other computer science concepts, such as running time. We compare the running time of an algorithm for $a^n b^n c^n$ on a one-tape Turing machine with an algorithm for it on a two-tape Turing machine.
- We use JFLAP to show how a PDA is used for LR parsing. We convert a CFG to a PDA and show it is nondeterministic. Then we ask the students to suggest lookaheads so that we can step through the parsing in linear time.

Students can use JFLAP for homework either to create an automaton to submit for grading, or to go through the steps in an algorithm. They can also use JFLAP to load examples done in class and to create additional problems to help them in understanding. JFLAP can be used outside of class by the instructor to grade students JFLAP submissions.

5. FEEDBACK

We have some feedback from JFLAP's use in CPS 140 at Duke University. In the spring of 2003 we used JFLAP for 6 of 9 homeworks. A questionnaire was given after their first use of JFLAP. In response to the question "Was JFLAP easy to use?", all 33 students responded yes. In response to the question "Did you look at the help at all? If so, what part did you look at and was it helpful?", 27 students did not look at the help. The 6 students who looked at the help said it was helpful. In response to the question "Do you prefer creating FA using JFLAP or drawing them on paper?", 17 students preferred to use JFLAP, 12 students preferred to use paper first, then JFLAP for testing and 2 students preferred paper.

JFLAP is being used around the world. A search of JFLAP (which appears to be a unique name) on Google

showed over 2000 web pages have the word JFLAP on them. Many courses use JFLAP in some way and have it listed on their course web page. Our download site of JFLAP 4.0 has over 3800 downloads since January 2003. From a combination of course web pages and download feedback, we have determined that JFLAP is being used in over 40 countries.

6. CONCLUSION

JFLAP 4.0 can be used along with an automata theory textbook to create a hands-on FLA course. JFLAP 4.0 allows one to interact with concepts in eleven of thirteen chapters for an FLA course, doubling the amount of material JFLAP 3.1 covered. The instructor can use JFLAP during class to solve problems. Students can use JFLAP outside of class for homework and additional problems to aid in understanding of theory concepts. JFLAP is available for free [8].

7. REFERENCES

- [1] J. Barwise and J. Etchemendy. *Turing's World 3.0 for the Macintosh*. CSLI, Cambridge University Press, 1993.
- [2] 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. In *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, pages 15–19. SIGCSE, March 1997.
- [3] C. Boroni, F. Goosey, M. Grinder, and R. Ross. Engaging students with active learning resources: Hypertextbooks for the web. In *Thirty-second SIGCSE Technical Symposium on Computer Science Education*, pages 65–69. SIGCSE, February 2001.
- [4] D. Hannay. Hypercard automata simulation: Finite state, pushdown and turing machines. *SIGCSE Bulletin*, 24(2):55–58, June 1992.
- [5] T. Hung and S. H. Rodger. Increasing visualization and interaction in the automata theory course. In *Thirty-first SIGCSE Technical Symposium on Computer Science Education*, pages 6–10. SIGCSE, March 2000.
- [6] P. Linz. *An Introduction to Formal Languages and Automata, 3rd Edition*. Jones and Bartlett, Sudbury, MA, 2001.
- [7] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.
- [8] S. H. Rodger. Jflap web site, 2003. www.cs.duke.edu/~rodger/tools/.
- [9] M. Stallman, R. Cleaveland, and P. Hebbbar. Gdr: A visualization tool for graph algorithms. In *Proceedings of Computational Support for Discrete Mathematics*, pages 17–28. American Mathematical Society, 1994.
- [10] K. Sutner. Implementing finite state machines. In *DIMACS Workshop on Computational Support for Discrete Mathematics*, pages 347–363. DIMACS, March 1992.
- [11] R. Taylor. *Models of Computation and Formal Languages*. Oxford University Press, New York, 1998.