Increasing the use of JFLAP in Courses

S. H. Rodger, H. Qin, J. Su
Computer Science Department
Duke University
Durham, NC 27708

rodger@cs.duke.edu

1 Introduction

Automata theory courses have traditionally been taught with pencil and paper problem solving, resulting in small, tedious to solve problems that are likely to contain errors. In the past twenty years, a number of software tools have been developed. Most tools focus on a particular concept or a set of related concepts, while other tools focus on a wider variety of concepts. We list a few such tools (Barwise and Etchemendy, 1993; Cogliati et al., 2005; Taylor, 1998) that allow users to visualize and interact with concepts from this course.

We have developed JFLAP(Rodger and Finley, 2006; Rodger, 2011), an instructional tool for visualizing and interacting with many concepts in automata theory and formal languages including regular languages, context-free languages, and recursively enumerable languages. With JFLAP one can build an automaton and then step through a simulation on input strings. JFLAP focuses on several types of parsing including brute-force parsing, LL(1) parsing, SLR(1) parsing and CYK parsing. The SLR(1) parsing method involves building a DFA that models the parsing stack, building the parse table, and then parsing strings and stepping through the creation of parse trees. JFLAP also includes interaction with construction-type proofs, such as converting an NFA to a DFA to a minimal state DFA.

We have worked to increase the usability of JFLAP in courses in three ways, 1) by making JFLAP apply to a broad range of topics and general definitions of those topics, 2) by focusing on the interaction and usability of the tool by both students and instructors, and 3) by showing how JFLAP helps students solve problems that are too difficult to solve on paper.

2 Usability by range of problems and generality

We have defined items in JFLAP generally to fit with many automata theory textbooks, and have options for limiting the generality so a student can focus on a particular definition. JFLAP has several variations of the finite automaton (FA) including deterministic (DFA), nondeterministic (NFA), and other variants such as Moore and Mealy machines. In JFLAP the label on a transition for an FA has zero, one or multiple symbols. JFLAP will fit with books that have a simpler definition of length zero or one, and with books that allow multiple symbols. Labels in JFLAP now allow a regular expression of the form [m-p] to represent one of the characters in the range from m to p. This allows users to make more realistic FA easily such as an FA to recognize integers (using [0-9] and [1-9] on transitions) or an FA to recognize valid variable names using [a-zA-Z] on transitions.

The pushdown automaton (PDA) transition in JFLAP is defined to allow multiple symbols to read, pop or push. When creating a new PDA, JFLAP prompts for the transition definition to use, either all fields multiple characters (very general) or all fields a single character. By selecting single character mode, that forces the PDA to read exactly zero or one symbol from the tape, to push exactly zero or one symbol onto the stack and to pop exactly zero or one symbol from the stack. This is a standard definition for a nondeterministic PDA (NPDA) that many textbooks use. Later a textbook may generalize fields, such as allowing to push multiple symbols onto the stack. There is one more choice for a PDA, the type of acceptance. When running the PDA on an input string select either accept by final state or accept by

empty stack. Note that the NPDA starts with a bottom of stack marker. This marker and everything else must be off the stack to accept by empty stack.

The Turing machine is defined in several ways including one-tape, multi-tape and building blocks. In the definition of the one-tape Turing machine the user reads one symbol from the tape, writes one symbol to the tape and then moves the tape head left (L), right (R), or stays put (S). Before creating a Turing machine the user can now set several options to narrow the definition. Those options include allowing or disallowing transitions from final states, allowing or disallowing the stay option as a move, accepting by final state and accepting by halting.

Turing machine Building blocks allow a user to create a Turing machine, give it a name and then use the name as a building block within another Turing machine. This makes it easier to create larger and more interesting Turing machines. Building blocks can be connected to regular states or other building blocks. Generalized transitions allow for easier connections. For example, the transition labeled $a; a, S$ (meaning read an "a", write the same symbol "a" and do not move the tape head), can be shorted to a. Even more general, the transition $\sim$ means, read any symbol, write the same symbol and do not move the tape head.

In JFLAP one can also start with a grammar. After entering in the grammar and before parsing the grammar there is an option for listing the type of grammar. This is helpful to students to verify the type of grammar when an instructor has asked them to write a particular type of grammar. The types of grammars checked for include regular grammar (right-linear or left-linear), context-free grammar, context-sensitive grammar, unrestricted grammar, and Chomsky Normal Form (CNF).

Other preferences for generalization to fit with more textbooks include selecting the empty string as either ϵ or λ, and forcing the trap state to appear or making the trap state invisible. The trap state can make some automata very messy looking so the default is to not show the trap state, but some books require that a DFA include an arc out of every state for every symbol in the alphabet.

3 Usability by Students and Instructors

Students could use JFLAP in a number of problem solving ways. These problems may be homework problems, problems recreated from class that were worked during class by the instructor, or problems created by students in order to get additional practice with learning a concept or algorithm.

A usability study on JFLAP was run(Rodger et al., 2009) with feedback from students from several universities that showed several strong results in the usability of JFLAP and its use outside the classroom. For example, over 60% of the respondants thought the use of JFLAP in the course made learning course concepts somewhat easier or much easier.

We list several features of JFLAP to increase its usability by students.

- JFLAP has an editor for building a transition diagram for many of the different types of automata. Previously the user had no control of the location of transitions. JFLAP would attempt to best connect them between two states. We have increased the flexibility of the transitions by allowing the user to curve the transitions, and for loop transitions on a single state, to move them to another location on the edge of that state.

- The image in the JFLAP editor pane can now be saved to an image file in a number of formats: png, jpg, gif or bmp, or export to svg format. This feature will be used to include JFLAP pictures in textbooks and papers, for figures in class notes and by students for pictures for homework.

- The layout of a transition diagram is a graph layout problem. This problem is studied in many graph drawing tools and even an annual conference called Graph Drawing. In JFLAP the user places states where they want them and can then move them around.

JFLAP includes about a dozen different graph layout algorithms for the user to try to find one that looks right for their automaton. The user can save a layout so they can come back to it if they do not like the results from other layouts.

- There is now an undo and redo capability that stores and revives editing commands, a request from many users.

Instructors use JFLAP in many ways. They use it to prepare slides for class, and to prepare homework problems or problems to work on in class or lab. For some problems they may create an example that students would either further develop or debug. Some instructors use JFLAP during lecture to illustrate new concepts or proofs.

For instructors to use JFLAP during a lecture or in presentations, we have recently modified the tool to include a zoom functionality. For example, in the automata editor pane, there is now a slider bar that can be used to increase the size of the automaton. Similar zooming slider bars appear in the grammar pane and many of the parts of more complex windows that are part of the parsing or construction proofs.

4 Usability by Complexity of problems

We give examples to show how one can explore more complex problems with JFLAP.

4.1 Example: Universal Turing machine

Using JFLAP we created a Universal Turing machine with just over 30 states that is available for download. It is a 3-tape Turing machine in which tape 1 is the encoding of a Turing machine M, tape 2 represents the current tape of M starting with the encoding of its input and tape 3 holds the current state M is in. A recent change in JFLAP makes it easier to run the Universal Turing machine. JFLAP now allows an input string to be loaded from a file and the input for the Universal Turing machine can be quite large.

4.2 Example: parsing equivalent, yet different grammars

This example illustrates the simulation of two equivalent grammars in order to compare them. The grammar on the left has a λ-rule. The grammar on the right is the equivalent grammar after removing the λ-rule and adding rules needed to represent the missing rules.

G1:	$S \to aB$	$B \to b$		G2:	$S \to aB$	$S \to a$	$B \to B$
	$B \to BB$	$B \to \lambda$			$B \to BB$	$B \to b$	$B \to aa$
	$B \to aBa$				$B \to aBa$		

Running the brute-force parser in JFLAP on the left grammar for the string *aabbab* generates over 1800 steps and takes about 8 seconds. Running the same string on the grammar on the right is much faster, about 40 steps and less than a second. The difference between the two grammars is impressive to students when demonstrating the runs during a lecture.

4.3 Conversion of DFA to regular expression

This example illustrates that a proof conversion can be too complex and tedious to enter in all the steps and the interaction with the user needs to be reduced to a more reasonable amount. The algorithm JFLAP uses for converting a DFA to a regular expression removes one state at a time until the DFA has just two states and the regular expression is easily obtainable. This algorithm starts by creating an equivalent DFA with only one final state, and creating arcs between all combinations of states (adding $\emptyset$ if there is no transition). Next the user selects one state at a time to remove, picking any state except the initial state or final state. For the state removed, every remaining arc must incorporate information from the removed state in the form of a regular expression. With regular expressions on arcs we now call the transition

diagram of the DFA a Generalized Transition Graph (GTG). This part was very complex to create as an interactive tool. We limited the interaction, because it was too tedious for the user to enter in each regular expression. However, we wanted the user to be able to focus in on any regular expression generated. When a user selects a state to remove, JFLAP lists all the transitions not involving that state in a table with the new regular expression that will become the new label for that transition. The user can click on any transition in the table and the transitions from the DFA that form this new transition are highlighted. Once the user has seen enough, the table disappears, the state is removed and the GTG updated with the new regular expression labels. This step is repeated until there are just two remaining states, and it is easy to compute the regular expression.

4.4 Comparison between NPDA and SLR Parsing

Our last example shows how multiple parts of JFLAP can be used to study a new concept. We focus on SLR(1) parsing. With JFLAP one can convert a CFG into an NPDA that models the SLR(1) parsing process. One can run the NPDA with input strings and observe the nondeterminism. The user has to guide the parsing by examining the lookahead. In another approach, the same grammar can be converted to an SLR(1) parse table. In this case the SLR(1) parsing is deterministic. It automatically uses the lookaheads since that information is contained in the SLR(1) parse table. Showing the processing of both the NPDA and the SLR(1) parser for the same grammar and string helps reinforce the idea behind the parsing that SLR(1) is about an NPDA that uses lookaheads to choose the next rule.

5 Conclusions

We have developed JFLAP to cover a wide variety of topics in the area of automata theory. We have shown how we have made JFLAP more widely usable by making the definitions of its parts general, yet also allow restrictions on the general definitions. We have made changes to JFLAP to make its interface easier to use by students and have made changes that make it easier for instructors to use in the classroom. Finally we have given several examples that show the advantages of using JFLAP to illustrate more complicated examples.

6 Acknowledgements

The work of the authors was supported in part by the National Science Foundation through NSF grants DUE-0442513 and DUE-1044191.

References

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

J. Cogliati, F. Goosey, M. Grinder, B. Pascoe, R. Ross, and C. Williams. Realizing the promise of visualization in the theory of computing. *JERIC*, 5:1–17, 2005.

S. H. Rodger. Jflap web site, 2011. `www.jflap.org`.

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

Susan H. Rodger, Eric Wiebe, Kyung Min lee, Chris Morgan, Kareem Omar, and Jonathan Su. Increasing engagement in automata theory with jflap. In *Fourtieth SIGCSE Technical Symposium on Computer Science Education*, pages 403–407. SIGCSE, March 2009.

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