Increasing Visualization and Interaction in the Automata Theory Course

Ted Hung and Susan H. Rodger*
Computer Science Department
Duke University
Durham, NC 27708-0129
rodger@cs.duke.edu

Abstract

In this paper we describe how to increase the visualization and interaction in the automata theory course through the use of the tools JFLAP and Pâté. We also describe new features in these tools that allow additional visualization and interaction. New features in JFLAP include the addition of regular expressions and exploring their conversion from and to nondeterministic finite automata (NFA), and increasing the interaction in the conversion of automata to grammars. New features in Pâté include the display of a parse tree while parsing unrestricted grammars, and improved interaction with parsing and the transformation of grammars.

1 Introduction

Many of the concepts and proofs studied in an automata theory course (or the early foundations in a compiler course) can easily be visualized and interacted with. The concepts include drawing and simulating theoretical machines and showing the derivations and parse trees for strings accepted by grammars. The proofs include construction type proofs where one representation of a language is converted to another representation, such as proving that context-free grammars (CFG) and pushdown automata (PDA) both represent context-free languages.

*The work of this author is supported by the National Science Foundation's Division of Undergraduate Education through grant DUE-9752583 and by the National Science Foundation's Computer and Information Science & Engineering Directorate through grant CISE-9634475.

Visualization provides students with an alternative view in addition to the theoretical representation that is usually presented in textbooks. Furthermore, interaction allows students to experiment with the concepts and proofs and to receive feedback. Studies in the area of algorithms [1] show the need for students to have an alternative visual representation they can interact with.

It is common for automata theory textbooks to start by visualizing the simple concepts, and then not visualize the more complicated concepts. For example, almost all such textbooks visualize the finite automaton, but fewer visualize the Turing Machine and even fewer visualize the pushdown automaton. The textbooks all visualize parse trees for context-free grammars. There appears to be no similar visualization for unrestricted grammars.

We have used several tools in the automata theory course at Duke to convert the course from a lecture only format with written exercises to a more interactive lecture format with interactive lab and homework exercises. In this paper we describe the new features of tools JFLAP [3] and Pâté [2], and how these tools can be integrated into the automata theory course.

In Section 2 we describe JFLAP and its new features in Section 3. In Section 4 we describe Pâté and its new features in Section 5. Section 6 describes how these tools and others are used to transform the automata theory course into an interactive and visual course. We conclude in Section 7.

2 JFLAP

JFLAP (Java Formal Languages and Automata Package) is a tool for creating and simulating several versions of automata and for converting representations of languages from one form to another. The versions of automata supported include 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 three choices for execution, a fast mode that gives the answer, a step mode that steps through an animation, and a multiple input mode to test several strings at the same time in fast mode.

In JFLAP's conversion mode, one can convert a representation of a language into another representation of the language. The regular language transformations supported are 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. The context-free language transformations supported are converting an NPDA to a CFG, and three algorithms for converting a CFG to an NPDA.

3 New Features in JFLAP

The new features in JFLAP include regular expressions, the conversion of regular expressions to NFA, the conversion of NFA to regular expressions, steps added to the conversions of automata to grammars that previously just gave the answer, and an expanded help section now in html format.

With these new features in JFLAP, one can now convert any representation of a regular language into another representation and either create the new representation with help or watch an animation step through its creation. Figure 1 shows the flow of possible conversions for regular languages.

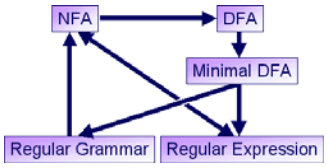

Figure 1: Regular Language Conversions

3.1 Regular Expressions

In the regular expression to NFA conversion, the user first enters a regular expression and selects the *Create FSA* option. An NFA drawing window appears and the user has three choices. The user can either build the complete NFA, build the NFA in stages with help, or have the NFA shown. In building the NFA, the user must follow the algorithm explained in the help section of JFLAP, as this particular algorithm is checked for correctness. In the stage building choice, the user is given pieces of the NFA and asked to connect them. For example, Figure 2 shows that at one stage in the regular expression to NFA conversion for $ab^*a(bb^*a)^*$, the user would be given an NFA for b^* and an NFA for a and would modify them to create the NFA for b^*a. The NFA built following the algorithm is most likely far from the minimal solution since many lambda arcs are added. However, once built other conversions can convert the NFA into a DFA and then a minimal state DFA.

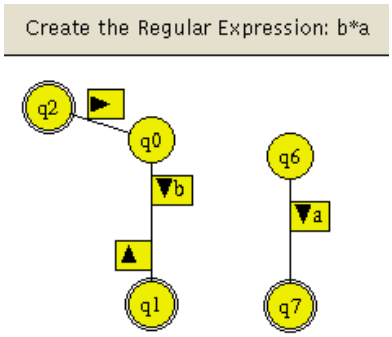

Figure 2: Regular Expression to NFA in JFLAP

In the NFA to regular expression conversion, a recursive formula which is a slight modification from [4] is used. The user starts with an NFA, and then enters the starting recursive formula. A table is then displayed showing all the formulas that must be calculated. Since filling out the formulas is a bit tedious and straight forward, the user can choose to either have the formulas all filled out, or step through the filling out of the formulas. At the bottom of recursion and on the way back out of recursion, the formulas are replaced by regular expressions. The first expressions are fairly simple, and the user can either type them in or have them displayed. Later formulas can result in rather long regular expressions. The user is asked to simplify these regular expressions.

Here are some of the rules that are used in the simplifications of regular expressions. Not all rules are shown. The ! and {} represent lambda and the empty set.

```
a + {} = a
a{} = {}
a! = a
(!+a)* = a*
(!+a)a* = a*
a + a = a
b + a*b = a*b
```

For example, Figure 4 shows part of the working window of the conversion of the NFA shown in Figure 3 to a regular expression. The notation *R(a,b,c)* represents the regular expression between states a and b without going through a state number higher than c. In Figure 4, the regular expressions for the bottom four formulas *R(2,1,1)* through *R(0,1,0)* have already been calculated. Currently the formula *R(2,2,1)*, the first item highlighted, is under calculation. Since the components of its right-hand side have already been calculated, the line immediately below fills them in with their regular expressions. Other highlighted lines show where these

regular expressions come from. At the bottom of the window, the user types in the simplification for the regular expression listed under *R(2,2,1)*. Figure 5 shows the final resulting expression several steps later that corresponds to the DFA in Figure 3.

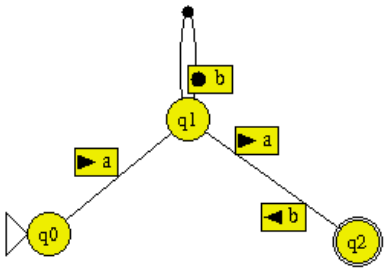

Figure 3: A DFA in JFLAP

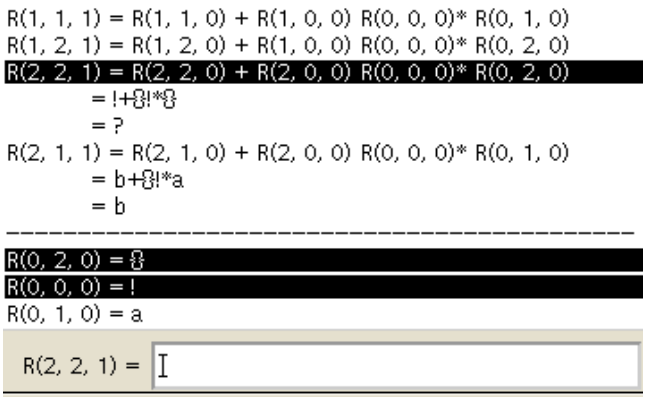

Figure 4: Regular Expression Conversion in JFLAP

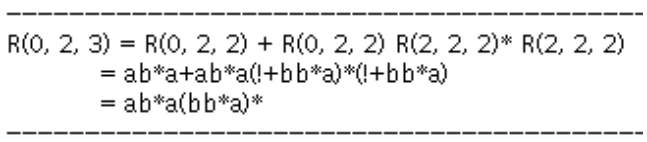

Figure 5: Regular Expression Result in JFLAP

3.2 Conversions to Grammars

In JFLAP, the user can convert an NFA to a regular grammar, and an NPDA to a context-free grammar. Previously, the conversions of automata to grammars showed the resulting grammar with no intermediate steps. In the new version, the user steps through the conversion. The conversion highlights each arc in the automaton, one by one, showing the corresponding grammar rules for that arc. At the completion, the complete grammar is then shown in one window unless it is too large. The user can then either save the grammar or continue to convert the grammar to other representations.

4 Pâté

In this section we briefly describe the previous version of Pâté, and in the next section describe the new features added to Pâté. Pâté is a parser for restricted and unrestricted grammars and a grammar transformer from a context-free grammar to Chomsky Normal Form (CNF). Given a grammar and an input string, the parser is an exhaustive search parser that builds a derivation tree (not displayed) of all possible derivations in a breadth-first manner. Some pruning of nodes is done to speed up the search. Once a derivation is found, the user can choose to display the derivation in textual format or in the form of a parse tree (for restricted grammars only). Alternatively, a message may indicate that the string is not in the language of the grammar.

In the grammar transformer part of Pâté, one enters a CFG and then through a series of steps converts the grammar into CNF. The steps include removing lambda productions, unit productions, and useless productions. At each step an equivalent grammar is created.

5 New Features in Pâté

The new features in Pâté include the visualization and animation of a parse tree for unrestricted grammars, improved interaction in both the parsing and grammar transformation, and an expanded help section now in html format.

5.1 Parse Tree for Unrestricted Grammar

It is common for automata theory textbooks to show a parse tree for restricted grammars, but after looking through over a dozen textbooks we were not able to locate one that followed up by showing a parse tree for an unrestricted grammar. The difficulty in displaying such a parse tree results from the fact that the left-hand side of a rule in an unrestricted grammar contains both variables and terminals and can replace them or switch their ordering. In Figure 6 rules 3 through 6 are in this format. In drawing such a parse tree, terminals produced in a rule may not be part of the final string, or they may be moved to another location in the tree.

Our parse tree works in a step mode in the following way. The components of the left-hand side must be adjacent, however they may be on different levels of the current parse tree. Our tool drops these components down to the same level by extending their branches, and highlights them in a box to indicate they are being replaced. The right-hand side of a rule works the same as in a parse tree for a restricted grammar. For example, Figure 7 shows the partial parse tree for the derivation of the string `aabbcc` from the grammar in

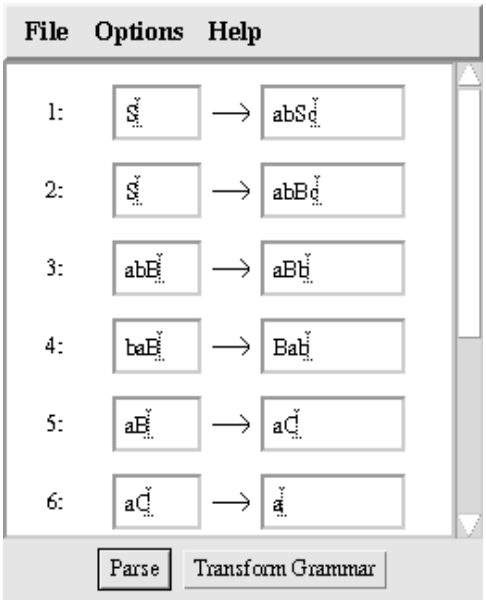

Figure 6: An Unrestricted Grammar in Pâté

Figure 6. In this case the first left-hand side with multiple items `abB` has all the items on the same line. A shaded box is drawn around them and they are replaced with `aBb`. Figure 8 shows the completed parse tree three steps later. Both the `a` and `b` from the second row of Figure 7 have been extended down to be used in the rules transforming `aB` and `baB` respectively.

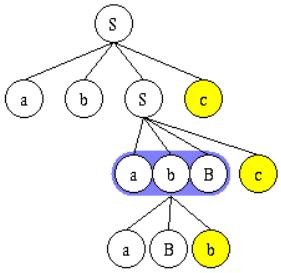

Figure 7: Partial Parse Tree in Pâté

5.2 Improved Interaction in Pâté

In Pâté, both the parser and the grammar transformer now allow one to step through the derivations in forward and reverse directions. In the parser, both the textual output and the parse tree output can either show the answer or step through the construction of the answer. In both cases, the answer is determined first and then the starting point of the derivation or first node in the parse tree is shown. Figure 9 shows the partial textual derivation of the string `aabbcc` corresponding to Figure 7.

In the grammar transformer, many of the transformations include deleting a few of the existing rules, and adding replacement rules. In the previous version of Pâté, the user had to type in the complete new grammar, a bit tedious since many rules are typed in again unchanged. In the new version, the existing grammar is shown and the user selects rules to delete by highlighting them and then types in the new replacement rules. A copy of the previous grammar is now shown beside the grammar the user is constructing in the same window. Figure 10 shows a portion of the Lambda Removal window from Pâté. The grammar on the left is the modified grammar with no lambda productions. The grammar on the right is the previous grammar. The symbol ∅ is lambda.

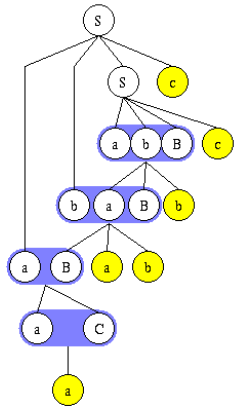

Figure 8: Complete Parse Tree in Pâté

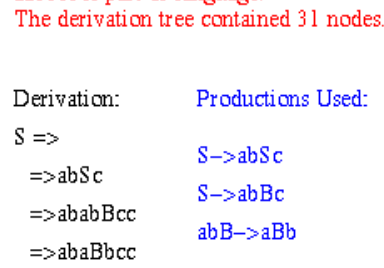

Figure 9: Partial Derivation in Pâté

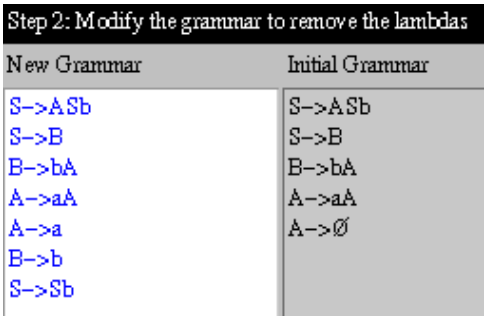

Figure 10: Part of Lambda Removal in Pâté

5.3 Other Improvements

File format for JFLAP and Pâté is now the same. Restricted grammars created in one tool can also be used in the other. Many of the interfaces in Pâté such as the grammar input window and the graphical windows in the grammar transformations are now similar to those in JFLAP, making it easier to switch between these tools.

6 Using Tools To Teach

For many years now we have used JFLAP, Pâté and other tools in teaching the automata theory course CPS 140 at Duke to increase the visualization and interaction in this course. In [3] we give student comments showing the effectiveness of these tools. We have used these tools with the textbooks [4, 5], but they are designed to work with most automata theory textbooks. Here we describe how to integrate these tools and the impact the new additions to these tools will have on this course.

We currently use JFLAP and Pâté during lectures to introduce topics, to work examples, and to illustrate the easy use of the tools. For example, we have found that some students did not realize that they could move states around in JFLAP unless they saw us do it. By working examples in class and saving them in files, students can reproduce the same examples later. Students have used these tools in labs, for homework assignments, to try additional examples and to study for exams.

JFLAP and Pâté are instructional tools that work best with small examples (dfa with 12 or fewer states, grammar with 8 or fewer rules). One can construct much larger examples, however those examples can become tedious to work with. Examples should be constructed so the student can do enough to understand the concept or algorithm, and then use the *show* features to complete the rest of the algorithm. One especially has to be careful with example sizes in the transformations as the new representations can be quite large. For example the equivalent DFA from an NFA could grow to an exponential number of states.

As another example of size restrictions, many students do not understand that some things are not computable on a computer due to their solution growth rate. The exhaustive search parser in Pâté is simple to understand and one can tell students that it can take a long time, but when you type in a grammar and input string during lecture and select parse and wait, they really begin to question why it is taking so long. Pâté informs them of the size of the derivation tree and tells them this string may take too long to derive. This interaction with Pâté has much more impact then words from an instructor.

The additional step interaction added to both Pâté and JFLAP makes the tools more useful by the instructor during lecture. They can now step through an algorithm and ask for feedback on what is going to happen next. For example, in the conversion of an NFA to a regular grammar, the instructor can ask the students what the rules for a particular arc are, and then have JFLAP display the answer. The new interface in Pâté in the grammar transformation windows makes it simpler to change a grammar, and thus easier to demo in class.

7 Conclusion

JFLAP and Pâté are tools for integrating visualization and interaction into an automata theory course. The addition of regular expressions to JFLAP allows one to now take one representation of a regular language and convert it to any of the other representations and even back again. The addition of a parse tree for unrestricted grammars in Pâté allows one to continue the visualization of parse trees as they learn different types of grammars. The increased interaction in both of these tools allows one to focus on the steps of algorithms.

JFLAP, Pâté and other tools we have developed are available free. The software and more information about them are available on

`http://www.cs.duke.edu/~rodger/tools`

Acknowledgement JFLAP and Pâté would not be possible without the work of Dan Caugherty, Mark Losacco, Madga Procopiuc, Tavi Procopiuc, Eric Gramond, Anya Bilska and Jason Salemme.

References

[1] Badre, A., Lewis, C., and Stasko, J. Empirically evaluating the use of animations to teach algorithms. *Proceedings of the 1994 IEEE Symposium on Visual Languages* (1994), 48–54.

[2] Bilska, A. O., Leider, K. H., Procopiuc, M., Procopiuc, O., Rodger, S. H., Salemme, J. R., and Tsang, E. A collection of tools for making automata theory and formal languages come alive. *Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (1997), 15–19.

[3] Gramond, E., and Rodger, S. H. Using jflap to interact with theorems in automata theory. *Thirtieth SIGCSE Technical Symposium on Computer Science Education* (1999), 336–340.

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

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