Preface

When I first started teaching automata theory in 1989, I taught it the traditional way as it had been taught to me. Students "programmed" finite automaton, pushdown automaton and Turing machines by writing them down using pencil and paper, either in formal notation or as a transition diagram. Their answers were almost always wrong, since it was too tedious for them to check. I wanted students to be able to check their answers and interact with these automata. I wanted them to be able to design their own additional problems and be able to receive feedback on them. I began designing a tool for experimenting with pushdown automata, and over the years with the help of many undergraduate and graduate students, FLAP was created and eventually evolved into JFLAP.

Susan Rodger

This book is a hands-on guide through the Java Formal Language and Automata Package (JFLAP), an interactive visualization and teaching tool for formal languages. This book is intended as a supplement to an undergraduate automata theory course or an undergraduate compiler course. This book is not a textbook! We assume the user is using a textbook along with our book. Our book guides users interactively through many of the concepts in an automata theory course or the early topics in a compiler course, including descriptions of algorithms JFLAP has implemented and sample problems for reinforcement of concepts. However, our book assumes that the user has read briefly about these concepts first in an automata theory textbook or compiler textbook.

JFLAP allows users to create and operate on automata, grammars, L-systems, and regular expressions; the term *structure* is used to refer to any single automaton, grammar, L-system, or regular-expression. JFLAP offers the following major groups of operations to apply to structures:

Explore the Language of Structures JFLAP has the ability to simulate input strings on nondeterministic automata, build parse tables and parse trees for grammars, and render successive expansions of L-systems. The automata represented in JFLAP are finite automata (FA), pushdown automata (PDA) and multi-tape Turing machines. The parsing algorithms in JFLAP are brute-force parsing, LL(1) parsing and SLR(1) parsing.

Convert between Equivalent Structures A wide range of conversion abilities are available, e.g., regular expression to FA, nondeterministic FA to deterministic FA (DFA), PDA to grammar, grammar to PDA, DFA to minimized DFA, context-free grammar to Chomsky normal form grammar, and others.

Miscellaneous Analysis of Structures JFLAP also offers a few sundry analysis tools to display properties of structures, like highlighting λ-transitions, highlighting nondeterministic states, and determining the equivalence of two FAs.

In addition to designing the structures listed above, our book guides the user through interactive alternative perspectives that would be difficult to do with pencil and paper. Here we list several examples from the book.

- In Chapter 2 users convert a deterministic finite automaton (DFA) to a minimal DFA. During the conversion, users must determine if two states p and q are distinguishable or not. With JFLAP they make two copies of the DFA, one with p a start state and the other with q a start state and run both DFA's on the same set of input strings to determine if the states are distinguishable.

- In Chapter 6 users parse context-free grammars using a brute-force parser. To determine the language of a grammar, users enter each variable's productions separately (when possible) and determine the language of each variable first, then put them together for the language of the grammar.

- In Chapter 8 users parse a grammar with an SLR(1) parser. Then they build an NPDA that models the SLR(1) parsing process for this grammar and run the NPDA with the same strings. The NPDA is likely nondeterministic, so the students guide the run with lookaheads.

JFLAP uses general definitions of its structures to allow it to fit with a range of textbooks. We mention some of these definitions here. Definitions for each structure are in the corresponding chapter. Instructors might prefer to require students to use a subset of the JFLAP definition if that fits with their textbook.

- Finite automata allow users to enter strings of length zero or greater. Instead, an instructor might want to require students to enter strings of length zero or one.

- Pushdown automata can pop zero or more symbols from the stack in each transition. An instructor might want to require students to always pop one symbol.

- Turing machine movements for the tape head are Right, Left and Stay. An instructor might want to require students to use only Right and Left.

Organization

This book covers topics from a formal languages and automata theory course, with additional topics on parsing and L-systems. We find that undergraduate students need to see an application to understand why this material is important. With SLR(1) parsing, students see the use of DFAs and PDAs. L-systems are included as an alternative grammar that results in interesting visual pictures.

We recommend chapters for an automata theory course and for the first part of a compiler course. Both courses should have students look at the JFLAP Startup section following the preface first before reading the chapters.

For an automata theory course, the minimal coverage would be Chapters 1, 2, 3, 4, 5, 6 and 9. Our book covers automata before grammars, so if your textbook covers grammars before automata then for regular languages cover the sections in the order 3.1-3.2, 1, 2, 3.3, 3.4, If context-free grammars are covered in your textbook before PDA, then cover them in the order 6.1, 5, 6.2 and 6.3. Optional topics include Chapters 7, 8, 10 and 11. If instructors want to cover SLR(1) parsing only and not LL(1) parsing, then cover 8.1 and 8.3.

For a compiler course, related chapters on parsing are Chapters 1, 2, 3.1-3.3, 5, 6.1, 6.2, 8, and 11. Skip 8.2 if you are not covering LL(1) parsing. Chapter 11 is optional but provides some interesting thoughts on how one parses other types of grammars, including the "parse tree" for unrestricted grammars in JFLAP which is not really a tree, but rather a directed acyclic graph.

We now give a brief description of the highlights of each chapter.

Chapter 1, Finite Automata, guides the user through editing and simulating both deterministic and nondeterministic automata.

Chapter 2, NFA to DFA to Minimal DFA, guides the user through transforming an NFA into a DFA, and then a minimal state DFA. In the NFA to DFA algorithm, a DFA state represents corresponding states reachable from the NFA. In the DFA to minimal DFA algorithm, states are grouped into sets of indistinguishable states using a tree method.

Chapter 3, Regular Grammars, guides the user through editing and parsing grammars. Its conversion algorithms between FA and right-linear grammars are standard.

Chapter 4, Regular Expressions, has conversion algorithms that use generalized transition graphs (GTG), wherein a transition label can be a regular expression. The regular expression to NFA algorithm starts with a GTG with one transition containing the regular expression and de-expressionifies one operation at a time until the result is an NFA. The FA to regular expression algorithm starts with an FA represented as a GTG with a transition between every pair of states (using $\emptyset$ if there was no transition). The algorithm removes one state at a time, replacing transitions with regular expressions, until there are only two states remaining.

Chapter 5, Pushdown Automata, introduces the editing and simulation of NPDAs.

Chapter 6, Context-Free Grammars, revisits the brute-force parser and goes into detail of how it works. Two algorithms are presented. The CFG to NPDA algorithm uses the model for LL parsing. Students do not need to know LL parsing for this chapter. The NPDA to CFG algorithm can easily result in a rather large grammar with many useless productions, and thus seems magical to students. After the conversion students test the same strings in both the original NPDA and the resulting CFG.

Chapter 7, Transforming Grammars, covers the transformation of a CFG to Chomsky Normal Form. Several transformations occur including removing λ-productions, unit productions and useless productions. Students test the same strings in the transformed and original grammars.

Chapter 8, LL and SLR Parsing, describes FIRST and FOLLOW sets which are needed in parsing, and the two parsing methods LL and SLR. For both LL and SLR parsing, students are shown how to convert a CFG to an NPDA for that method, how to build the parse table and how to compare the NPDA with parsing. In parsing a string, students can view either the derivation or a parse tree.

Chapter 9, Turing Machines, shows how to edit and simulate a one-tape Turing machine first, then expands to multi-tape Turing machines, including an example of a universal Turing machine.

Chapter 10, L-systems, describes how to edit L-systems and use them to produce plants, fractals, and other graphical structures.

Chapter 11, Other Grammars in the Hierarchy, describes how to use the brute force parser to parse strings with unrestricted and context-sensitive grammars, and some insight about how to structure these grammars so brute force parsing is not totally intractable.

JFLAP software

JFLAP and all files referenced in this book are available at `www.jflap.org` and are free! We recommend that files are loaded and interacted with as the text is read.

Acknowledgments

This book is the result of many years spent on developing JFLAP and we would like to thank several people. Thanks to Peter Linz for many conversations and ideas on JFLAP. Thanks to Erich

Kaltofen and Mukkai Krishnamoorthy for encouraging me in the early days of JFLAP, and Owen Astrachan in the later days of JFLAP. Thanks to family members for putting up for long absences including Thomas, Markus and Erich Narten, and John and Susan Finley. We especially want to thank John and Susan Finley for many hours of proofreading.

Thanks to the many students who have worked on FLAP, JFLAP or related tools over the years. These include Dan Caugherty, Mark LoSacco, Magda Procopiuc, Octavian Procopiuc, Greg Badros, Ryan Cavalcante, Ted Hung, Eric Gramond, Lenore Ramm, Robyn Geer, Alex Karweit, Anna Bilska, Jason Salemme, Ben Hardekopf, Steve Wolfman, Eric Luce, Ken Leider, Bart Bressler and Stephen Reading. Thanks to all the people who attended the JFLAP workshop in June 2005 and gave valuable feedback on JFLAP. This includes Michelangelo Grigni, Kelly Shaw, Dwight House, Sandria Kerr, Christopher Brown, Jacquelyn Long, Dawit Haile, Sanjoy Baruah, Costas Busch, Rakesh Verma, Joseph Bergin, Rockford Ross, and Eric Wiebe. And finally thanks to the over 17,000 users of JFLAP, many of you who have given us feedback over the years.

Susan H. Rodger and Thomas Finley, June 2005.