Contents

1 A Gentle Introduction
1.1 The Beginning .............................................. 1
1.2 Some Unavoidable Terminology ............................... 1
1.3 A Few Intuitive Examples .................................... 8
1.4 Sharing Structure ........................................... 16
1.5 Some Background in Set Theory .............................. 17
1.6 Composition and Rule Application ........................... 31
1.7 Lexicon and Rules ............................................ 32
1.8 The Big Picture ............................................. 36
1.9 Finite-State Networks are Good Things ....................... 37
1.10 Exercises .................................................. 37

2 A Systematic Introduction ........................................ 43
2.1 Where We Are ............................................. 43
2.2 Basic Concepts .............................................. 43
2.3 Simple Regular Expressions ................................. 45
2.4 Complex Regular Expressions ................................ 63
2.5 Properties of Finite-State Networks ........................ 74
2.6 Exercises .................................................. 78

3 The xsf Interface ................................................ 81
3.1 Introduction ................................................ 81
3.2 Compiling Regular Expressions ............................... 84
3.3 Interacting with the Operating System ....................... 109
3.4 Incremental Computation of Networks ....................... 121
3.5 Rule-Like Notations ........................................ 130
3.6 Examining Networks ....................................... 182
3.7 Miscellaneous Operations on Networks ...................... 190
3.8 Advanced xsf ............................................. 196
3.9 Operator Precedence ...................................... 202
3.10 Conclusion ................................................. 202
4 The lex Language
4.1 Introduction .............................................. 203
4.2 Defining Simple Automata ........................ 204
4.3 Defining Lexical Transducers ...................... 226
4.4 Useful lex Strategies .................................. 240
4.5 Integration and Testing ............................... 259
4.6 lex Summary and Usage Notes .................... 270

5 Planning and Managing Finite-State Projects .... 279
5.1 Infrastructure ............................................ 279
5.2 Linguistic Planning ..................................... 283
5.3 Composition is Our Friend ......................... 293
5.4 Priority Union for Handling Irregular Forms .... 300
5.5 Conclusions ............................................. 307

6 Testing and Debugging ................................. 311
6.1 The Challenge of Testing and Debugging .......... 311
6.2 Testing on Real Corpora .............................. 311
6.3 Checking the Alphabet ............................... 319
6.4 Testing with Subtraction ............................. 327

7 Flag Diacritics ............................................. 339
7.1 Features in Finite-State Systems .................... 339
7.2 What Are Flag Diacritics? ......................... 339
7.3 Using Flag Diacritics .................................. 341
7.4 Flag Diacritics and Finite-State Algorithms .... 356
7.5 Examples ............................................... 364

8 Non-Concatenative Morphotactics ................. 375
8.1 Introduction ............................................. 375
8.2 Formal Morphotactic Description .................. 378
8.3 Practical Non-Concatenative Examples ............ 383
8.4 Usage Notes for compile-replace .................. 412
8.5 Debugging Tips for compile-replace ............... 417
8.6 The Formal Power of Morphotactics .............. 419
8.7 Conclusion ............................................. 420

9 Finite-State Linguistic Applications ............... 421
9.1 Introduction ............................................. 421
9.2 The tokenize Utility ................................. 422
9.3 The lookup Utility .................................... 431
9.4 Using xfst in Batch Mode ............................ 439
9.5 Transducers for Morphological Analysis .......... 439
9.6 Spelling ............................................... 451

9.7 Beyond Tokenization and Morphological Analysis ... 453
9.8 Application Summary .................................. 455

A Graphing Quiz ........................................... 457
B Solutions to Exercises ................................. 459
B.1 Graphing Quiz ........................................ 459
B.2 The Better Cola Machine Diagram .................. 466
B.3 The Better Cola Machine Network ................. 466
B.4 Brazilian Portuguese Pronunciation ............... 470
B.5 The Monish Language ................................ 473
B.6 Esperanto Nouns ...................................... 476
B.7 Esperanto Adjectives ................................ 477
B.8 Esperanto Nouns and Adjectives ................. 477
B.9 Esperanto Nouns and Adjectives with Upper-Side Tags ... 479
B.10 Esperanto Nouns, Adjectives and Verbs ........ 480
A Word about this Book

Finite-State Programming Languages

This book will teach you how to use Xerox finite-state tools and techniques to do morphological analysis and generation. The tools include lex, a high-level language for specifying lexicons, and xfst, an interface that provides a regular-expression compiler and direct access to the XEROX FINITE-STATE CALCULUS, the algorithms for building and manipulating finite-state networks. Also included are tokenize and lookup, runtime applications that can be used for testing and building working systems. These tools have been used at the Xerox Research Centre Europe (XRCE), the Palo Alto Research Center (PARC) and elsewhere to build many practical linguistic applications including tokenizers, morphological analyzer/generators, part-of-speech taggers and even syntactic chunkers and shallow parsers.

The software licensed with this book includes executable versions of xfst, lex, tokenize and lookup compiled for the Solaris, Linux (Intel-based), Windows (NT, 2000, Me, XP) and Macintosh OS X operating systems. The twolec compiler, used to compile alternation rules written in the traditional "two-level" format, is also included on the CD-ROM and is documented on the webpage (see below).

Linux is a trademark of Linus Torvalds. Windows NT, Windows 2000, Windows Me and Windows XP are trademarks of Microsoft Corporation. Intel is a trademark of Intel Corporation. Macintosh OS X is a trademark of Apple Corporation.

Webpages

Technical Information

For the latest technical information about this book and the Xerox finite-state software, direct your Internet browser to the URL

http://www.fsmbook.com/
The site will include

- Information on software updates
- Errata
- Clarifications
- New exercises, chapters, and other auxiliary material
- Frequently Asked Questions (FAQ)
- Downloadable examples
- Announcements of finite-state programming courses and workshops
- Commercial licensing information

In addition the webpage will include information about how to subscribe to the email distribution lists dedicated to Xerox finite-state software. These lists serve as channels for official announcements and as forums for questions and tips among users.

**Publisher Information**

For information about acquiring this and other books from CSLI Publications, see

http://cslipublications.stanford.edu/

**Acknowledgments**

Many people have contributed in one way or another to this book. We owe a general debt to pioneers in the field of finite-state linguistic theory, including C. Douglas Johnson (Johnson, 1972), now Professor Emeritus of Linguistics at the University of California at Santa Barbara. Ronald Kaplan and Martin Kay rediscovered the key insights of Johnson (Kaplan and Kay, 1981; Kaplan and Kay, 1994) and developed the first set of practical and robust algorithms, originally written in INTERLISP, for finite-state computing at the Xerox Palo Alto Research Center (PARC). Kimmo Koskenniemi’s influential Two-Level Morphology (Koskenniemi, 1983), based on work from PARC and popularized mainly by Lauri Karttunen (Karttunen, 1983), was an ingenious but limited implementation of finite-state morphology that worked without an automatic rule compiler or the algorithms for manipulating finite-state networks. Kaplan, Karttunen, Koskenniemi and other colleagues and students participated in the development of the first finite-state rule compilers (Koskenniemi, 1986; Karttunen et al., 1987) at CSLI at Stanford. Karttunen ported the algorithms into Common Lisp and, with Todd Yampol, reported them again into C. Karttunen and Kenneth Beesley wrote twole (Karttunen and Beesley, 1992), the two-level rule compiler, in C; and Karttunen and Yampol wrote lex (Karttunen, 1993), the lexicon compiler. Yampol also wrote an early interface to the finite-state algorithms called fsm, which was eventually replaced by fsc, written by Pasi Tapanainen. The newest interface, xfst, and the library of finite-state algorithms are currently maintained and expanded by Lauri Karttunen at PARC and by André Kempe and Tamás Gaál at the Xerox Research Centre Europe (XRCE) in Grenoble, France.

We also wish to thank our students and colleagues for their feedback during finite-state training and development. Some, including Ronald M. Kaplan of PARC and Mike Maxwell of the Linguistic Data Consortium, have gone out of their way to send thoughtful problem reports and suggestions that have led to better code and documentation. Gilles-Maurice de Schryver, Martine Pétrod and Christof Baeije found many typos in early drafts. The remaining errors and infelicities are of course our own.
Preliminaries

Target Audience

This book teaches linguists how to use the Xerox finite-state tools and techniques to build useful and efficient programs that process text in NATURAL LANGUAGES such as French, English, Spanish, Yoruba, Navajo and Mongolian. Most of the presentation will center around morphological analysis and generation, but many other applications are possible.

The presentation is based on years of practical development and training of computational linguists at the Palo Alto Research Center (PARC) and the Xerox Research Centre Europe (XRCE). It is aimed squarely at our trainees, who are typically field linguists or sophisticated native speakers of the language they wish to work on; most have at least a basic background in computing and formal linguistics. However, few trainees come to us with an adequate understanding of the formal properties of finite-state networks, and even our professional colleagues often need to have their eyes opened to the practical possibilities of computing with finite-state networks. Beyond learning the syntax and idioms of the various Xerox finite-state tools, the bigger challenge is to appreciate the possibilities and develop the finite-state mindset that is so different from traditional algorithmic or procedural computing.

Most publications in finite-state theory and computing are addressed to a small professional community, including expert teams at Xerox, PARC, AT&T, Paris VII University, The University of Pennsylvania, Groningen University, Sabanci University and the University of Tuebingen. Long experience has shown that the terse technical presentation appropriate for that audience is too often a closed book to the working linguist who is recruited to sit down and build a real system, for example, a finite-state morphological analyzer for Romanian or Zulu. This book is therefore a popularization, facing all the dangers of that medium, aimed not quite at the proverbial man on the street but at least at the working linguist. We try to follow the rules for good popularization, including the use of wordier definitions, plentiful practical examples and tutorial exercises, while at the same time staying honest and accurate. Suitable mathematical rigor must not and ultimately cannot be avoided, but we try to ease into it gradually.

Our training courses have repeatedly shown that certain key concepts, like the composition operation, are surprisingly difficult to grasp and yet must be under-
stood viscerally if a trainee is to catch on and be able to understand, write and especially debug rules. We therefore do not eschew repetition, sometimes explaining the same concept in several ways, hoping that one will click.

While it is fashionable for programming books to claim that no previous experience is required, we cannot honestly do that here. We assume that the reader has some kind of previous programming experience in a language like C, C++, Java, Perl, etc. Trainees without any programming experience often have a difficult time in our courses. In addition, the software was developed on Solaris, and the book often assumes an acquaintance with UNIX-like operating systems and the ability to use a text editor like emacs, pico, xemacs or vi. The port of the finite-state software to the Windows operating systems is relatively recent, and Windows users are urged to consult the webpage\(^1\) for the latest software versions and hints.

We also assume that our students are trained in formal linguistics and are capable of looking at their language objectively and explaining in some kind of formal terms what is going on. This book explains how to formalize linguistic facts in the various Xerox finite-state notations, but it cannot tell you what the facts are or train you to be a formal linguist. The tools cannot do the linguistics for you.

Having said that, we try to assume no acquaintance with finite-state networks or formal-language theory. We provide a whole chapter entitled "A Gentle Introduction" to finite-state networks, wherein we begin at the beginning, even explaining what we mean by "finite", "state", and "network"; and laying out in intuitive fashion what finite-state networks can do and why they are Good Things. Once the general concepts are established, we progress to a more formal introduction; and we present the subsequent material in a gradual progression of readings and examples, complete with practical exercises. Readers are strongly urged to do the exercises.

**Finite-State Tools**

**Development Tools**

Finite-state computing involves using development tools to specify various finite-state networks and combine them together algorithmically into larger networks that perform tokenization, morphological analysis, etc. Traditional grammar components such as lexicons, morphotactic rules, morphotactic filters, phonological and orthographical alternation rules and even some syntax rules are all implemented in this book as finite-state networks. Xerox has created an integrated set of software tools for creating finite-state networks:

- xfst is an interactive interface providing access to the basic algorithms of the Finite-State Calculus, providing maximum flexibility in defining and manipulating finite-state networks. xfst also provides a compiler for an extended metalanguage of regular expressions, which includes a powerful rule formalism known as REPLACE RULES.

- lex is a high-level declarative language used to specify natural-language lexicons. The syntax of the language is designed to facilitate the definition of morphotactic structure, the treatment of gross irregularities, and the addition of the tens of thousands of baseforms typically encountered in a natural language. lex source files are produced with a text editor such as xemacs, and the result of the compilation is a finite-state network.

**Runtime Code**

The finite-state networks created using lex and xfst are used in real software systems by runtime code. Two useful runtime applications are licensed with this book.

- tokenize is an application that applies a finite-state network to input text and can be used in a command-line pipe. It is typically used to divide a running text into word-like tokens.

- lookup is an application that applies a finite-state network, or a set of finite-state networks, to input tokens and can be used in a command-line pipe. lookup is often used to perform morphological analysis, perhaps as a component of a much larger application. The output from tokenize can be piped to lookup, and the output of lookup can in turn be piped to other applications that perform disambiguation, parsing, etc.

**Applications**

The mathematical properties of finite-state networks have been well understood for a long time, but it was once generally believed that finite-state grammars were too weak in descriptive power to model interesting phenomena in natural language.

In recent years, although finite-state power and natural languages have not changed, the descriptive possibilities of finite-state grammars have been reexamined more positively, and the availability of practical tools like xfst and lex has made possible the development of

- Finite-state tokenizers that divide a running input text into tokens, i.e. words and various multi-word strings, for further morphological or syntactic processing;

- Finite-state morphological analyzer/generators (sometimes called LEXICAL TRANSDUCERS at Xerox) for English, French, German, Spanish, Portuguese, Dutch, Italian, Arabic and other languages;
Lexical transducers have many advantages. As well as being mathematically beautiful, bidirectional and highly efficient, these finite-state applications can have wide lexical coverage, take up little memory space, and be robust, commercial-viable products. In 1996 the Xerox Spanish Lexical Transducer, for example, contained over 46,000 baseforms, analyzed and generated over 3,400,000 inflected wordforms, and yet occupied only 3349 kbytes of memory, or about 1 byte per inflected wordform. For commercial applications, it can also be further compressed, and yet run directly in the compressed form, using Xerox compression algorithms and runtime code. The same language-independent runtime code is used for all languages.

Large lexical transducers and taggers now exist for English, French, German, Spanish, Portuguese, Dutch, Italian and Japanese, and the extension of finite-state techniques to process new languages has become almost routine at Xerox and among our research and business clients. For example, significant work has already been done on the morphological analysis of Basque, Turkish, Arabic, Finnish, Swedish, Norwegian, Danish, Sámi, Zulu and several Eastern European languages.

Requirements

Software

Xerox Finite-State Tools and Licensing

This book is a hands-on tutorial in the use of xfst and lexc, so the student should have these tools installed and running on his or her computer from the start. The versions of the Xerox software licensed with this book are limited to non-commercial use as explained in the license agreement.

For information on commercial licensing, see the webpage.²

Text Editor

The source files for xfst and lexc are best created using commonly available text editors such as emacs, xemacs, pico or vi. On PCs, the Notepad editor also saves files as plain text. If you use a word processor such as Microsoft Word, you must be careful to save the buffers as text-only files.

The Xerox finite-state networks have been defined to accommodate 16-bit UNICODE characters, which would have distinct advantages in many applications. However, the difficulties of UNICODE text-editing, display and printing make UNICODE processing awkward at this time. Check the webpage periodically for up-to-date information on the Xerox software and Unicode.

²http://www.fsmbook.com/
Hardware

The finite-state tools are provided in object form, compiled for Solaris (SUN workstations), for Microsoft, for Linux and for Mac OS X. Although interesting experiments can be performed on modest machines, large-scale linguistic development can often require serious computer crunch and large amounts of RAM. Although the final results of most morphological analyzers, for example, are under five megabytes in size, and while the Xerox finite-state algorithms are the fastest and most memory-efficient available, intermediate stages of various operations can often explode in size before minimization operations can be invoked. While 24 or 36 MBytes of RAM might be considered a minimum for full-scale work on most languages, XRCE maintains SUN workstations with up to 2 GBytes of RAM to handle particularly intensive finite-state computations.

Other Documentation

This book replaces previous documentation, some of which is terse, lacking in examples, and increasingly out of date. Where appropriate, sections of the following documents have been recycled:


Online documentation is also available at www.fsmbbook.com.

Chapter 1

A Gentle Introduction

1.1 The Beginning

This book shows how to use the Xerox finite-state programming languages xfst and lex to create finite-state networks that perform morphological analysis and related tasks. So let's begin at the beginning: What are FINITE-STATE NETWORKS and why should anyone care about them?

The following gentle introduction will attempt to paint the big picture, avoiding technical vocabulary and mathematical definitions, but conveying the core concepts necessary to make sense of all the formalism that must follow. We'll try to present a vision of where we're headed, showing how finite-state networks, created using various Xerox tools, are combined together into practical working applications, and in particular into morphological analyzers. Finally, we will try to convey the general notion that finite-state networks are Good Things, useful for doing many kinds of linguistic processing. Compared to the alternatives, applications based on finite-state networks are usually smaller, faster, more elegant, and easier to maintain and modify.

Analogies and explanations in this gentle introduction are not to be taken too far or too seriously; some will certainly not stand up to rigorous scrutiny. Experienced computational linguists and computer scientists can safely skip this section and move on to the more formal introduction in the next chapter.

1.2 Some Unavoidable Terminology

First of all, what is meant by the terms finite, state and network? Let's start with the term state, which has everyday meanings that are closely related to the formal one we will need.