Reference

Stockwell, Robert P. & Donka Minkova 2001. *English words: history and structure*. Cambridge: Cambridge University Press.

Author's address:
(Andrew Carstairs-McCarthy)
Department of Linguistics
University of Canterbury
Private Bag 4800
Christchurch 8140
New Zealand
E-mail: andrew.carstairs-mccarthy@canterbury.ac.nz

Kenneth R. Beesley & Lauri Karttunen, *Finite State Morphology*. Stanford, CA: CSLI Publications (distributed by the University of Chicago Press), 2003. xviii + 505pp. and CD-ROM. ISBN hardbound 1-57586-433-9, paperbound 1-57586-434-7.

Reviewed by ERWIN CHAN & CHARLES YANG
DOI: 10.3366/E1750124508000263

At hand is an unusual book, at least for most readers of the present journal. In *Finite State Morphology* (henceforth *FSM*), Kenneth Beesley and Lauri Karttunen provide a detailed introduction to the Finite State approach to morphology developed at the Xerox Corporation, with the associated software on a CD-ROM.[1] While these tools are typically deployed for morphological analysis in natural language processing applications, the authors are right to claim linguists as their core audience: this book can be viewed as a work of linguistic theory in the guise of a programming language. Linguists stand much to gain from this book and may even develop a fuller appreciation of themselves, as we shall explain.

An itemized summary is neither the most exciting nor the most informative format for a review of a book of this nature. In this review, we will provide a background introduction to the Finite State approach to computational morphology, and then turn to the specific offerings in *FSM*, while relegating some more technical points to footnotes. We conclude with a general discussion on how *FSM* contributes to the theory of morphology.

1 What is Finite State Morphology?

The descriptive devices of morphology traditionally include the system developed in *The sound pattern of English* (Chomsky & Halle 1968):[2] an ordered list of rewrite rules of the form A→B/C_D, which maps the underlying representation of a lexical form to its surface representation through a sequence of intermediate steps. To use an example from *FSM*, consider two rules that are responsible for some nasal assimilation process:

(1) N → m / __ p(where N stands for an underspecified nasal)
(2) p → m / m __

Since (1) applies before (2), the input string *kaNpat* turns, via *kampat*, into *kammat*, the desired output string.

Ordered rules, however, are not particularly conducive to computational applications as there is no appropriate formal framework in which rules can be readily formulated and implemented. In a nutshell, *FSM* provides the tools for turning rules into practical morphological analyzers. The theoretical groundwork was laid out by Johnson (1972) and, independently in the early 1980s, by Kaplan & Kay (1994). They show that rewrite rules are equivalent in power to Finite State transducers, which are a variant of Finite State automata that linguists are more familiar with. Instead of accepting or rejecting a single string, as in the case of Finite State automata, a Finite State transducer accepts or rejects two strings whose letters are pair-matched, while still retaining the Markovian property of Finite State transitions. As a result, Finite State transducers are simple, well understood and easy to implement computationally. Moreover, it is also found that an ordered cascade of rewrite rules can in principle be automatically COMPILED into a single Finite State transducer, thus capturing the mapping from the underlying form to the surface form in terms of paired strings. Figure 1 gives the Finite State transducer for the rules in (1) and (2), where the letters above and below the arrows represent the input and output strings, and the circles represent the states that the Finite State transducer traverses in scanning the string pair.

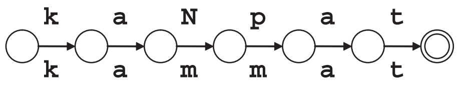

Figure 1. A Finite State transducer, which expresses the relationship between *kaNpat* and *kammat*, can mimic the effect of the ordered rules in (1) and (2).

The automatic compilation of rules into Finite State transducers (unlike the one in Figure 1, which we constructed manually) promises an advantage over testing rules by hand, a tedious and error-prone process for large natural language processing applications. A compiler would also allow a linguist to focus on the WHAT question, the development of linguistic descriptions for languages, rather than the HOW question, which concerns the implementation and execution details of the resulting system. But this promise was not delivered until *FSM* and related technologies well into the 1990s. Instead, computational morphology saw the development of Two-Level Morphology (Koskenniemi 1983), where contextual constraints are expressed in parallel directly between lexical and surface levels, rather than as rules applied in serial order.

Ever since gaining prominence in the 1980s, Two-Level Morphology has become a staple in computational linguistics. But it is not the easiest tool to use (or to teach, in our experience). The two-level commitment forces one to directly manipulate input-output letter strings, and represent serial rules as parallel constraints. This can be a highly unintuitive and labor-intensive process, even for experienced programmers. Moreover, the insistence on only two levels raises questions about the validity, or efficiency, of such an approach when issues of opacity and long distance dependencies are taken into account (Barton et al. 1987, Anderson 1988).

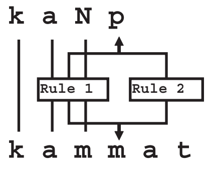

Figure 2. Model of transduction process in Two-Level Morphology (Koskenniemi 1983).

2 The *Xerox Toolkit*

The main accomplishment of *FSM* is to make computational morphology far more accessible to linguists; in doing so, it finally delivers the promise of automatic rule compilation. The salvation comes in the form of *XFST*, a program for compiling and executing rules.[3] It is now possible to specify linearly ordered rules very much in the style of SPE, and the system will compile the rules, behind the scenes, into a Finite State transducer.

Mastering the syntax of *XFST*, like any other programming language, will no doubt take time. Even though the authors made a real effort to make the materials accessible to non-specialists, we doubt that a linguist without any computational background will find this book an easy read. After a general introductory chapter, the reader is confronted with an exhaustive but tedious treatment of the Finite State formalism of the type we touched upon in section 1. Our advice is to skim this and jump directly to the chapter that introduces *XFST*.[4] The presentation is generally effective thanks to the large number of real linguistic examples ranging from reduplication in Malay to agreement in Monish (a fictional language invented for pedagogical purposes). Once you make the effort, the transition from linguistic analysis to computational implementation can be quite straightforward, as some actual code illustrates:

(3)
```
define Rule1 [N -> m || _ p];
define Rule2 [p -> m || m _];
read regex Rule1 .o. Rule2
```

Behind the scenes, the *XFST* system first translates the rules into Finite State transducers that are formally equivalent to those we constructed earlier. For instance, the rule in (1), [N -> m || _ p], is converted to a set of string pairs (x, y) for which y is the result of the application of rule (1) to x. The operation that follows is that of COMPOSITION (.o.), which is discussed in depth throughout the book: it takes two sets of string pairs (x, y) and (y, z) and converts them into a new pairing (x, z), thereby achieving the effect of rule ordering.[5]

There is a diverse range of rule formats that one can conveniently evoke in *XFST*, and it is clear that these are designed by linguists, for linguists, and to handle widely attested linguistic phenomena. For example, deletion can be handled by the use of the null string ([ ]) on the right hand side of rules. Epenthesis gets its own treatment, with the necessary restrictions so that the system does not keep on inserting symbols ad infinitum. One of our favorites is a rule that allows one to specify separate restrictions on underlying

December 1, 2008 Time: 09:40am word023.tex

and surface levels, which proves handy for modeling harmony processes. But having so many options for specifying rules isn't necessarily a good thing. Like grammatical formalisms, programming languages ought to limit the degree of expressive freedom availed to the user, which also makes for a smooth learning experience. We find several forms of rule writing that may prove convenient in some conceivable cases. But these are probably too rare to warrant independent treatments while alternative expressions are possible. Invariably, these rule formats come with a disclaimer 'have not been widely used in practice'.

Later chapters describe ways to handle non-concatenative phenomena. One of the most useful augmentations to the *XFST* system is flag diacritics. Flag diacritics encode the usual set of morphological features (e.g., case, tense, gender, number, etc.): they serve the purpose of representing morphosyntactic information but can also work as constraints on morphological analysis. To use one of the examples in the book, consider the cliticization of the French definite articles *la* and *le* (page 364). The requirement here is that the noun or adjective be singular and vowel initial, and the cliticized form is represented as *l'* in orthography. A straight-up Finite State implementation would essentially require the system to 'remember' the occurrence of *l'* with the expectation of finding a singular form later on in the word, so that *l'arbre* 'the tree' is acceptable while *l'arbres* is not. This sort of restriction usually leads to more rule writing for the user, and larger Finite State networks for the computer. With flag diacritics, by contrast, one can specify SINGULAR as a feature for *l'* in the lexicon: as soon as *l'* is read from the input, the system can switch to a network that accepts only singular adjectives or nouns. The use of feature matching and unification is familiar from contemporary grammatical formalisms, and the insights here are fundamentally the same: complex dependencies on surface representations can be factored into simple relations on more abstract representations.

Nothing good comes for free; the *FSM* toolkit certainly doesn't. The only way to obtain the software is to purchase the book, and the only way to use it, for non-commercial purposes, is to first agree to a legalese-laden single user license. This may have limited *FSM*'s pedagogical reach: one would have to think twice before passing the cost, $40 retail, onto every student in an introductory computational linguistics class, where morphology may only take up two weeks. Furthermore, the source code is not available, so an advanced programmer will not be able to add new capabilities to the software. While we cannot complain about Xerox's interests in protecting their intellectual property (after all, they footed the bill) the all too frequent (and boldface) mentions of the Xerox Corporation leads to sensory overload. And there are non-Xerox Finite State systems available for free, such as the *FSA* system (van Noord 1997) and the *AT&T Finite-State Tools* (Mohri et al. 1998), even though these general purpose tools may be less convenient to use for morphological analysis.

More technically, the convenience of *XFST* does not come for free either. Lost in translation from an ordered sequence of rules to a composed transducer are the operational details of the morphological system, which is highly useful during the testing and debugging stage. In the Two-Level Morphology system, where each rule has its own Finite State transducers (constructed by hand, admittedly), it is usually quite easy to locate the misbehaving rule when an error has been detected. We have not found a way to do so under *XFST*; we suspect that the automatic compilation system cannot retain the transparency of the rules once they are collapsed into a larger Finite State transducer.[6]

An important issue that deserves more discussion is the computational complexity of Finite State Morphology. It is well known that, theoretically, the two-level approach to morphology is fundamentally intractable (Barton et al. 1987); this result applies to the *XFST* system, which is two-level at its core. Having to simulate the long distance relationships such as harmony and reduplication with Finite State implementations, the resulting networks may grow exponentially in the worse case. Of course, whether the worse case surfaces in practice is a separate issue, and some empirical results from the 1980s suggest that the situation may not be quite as dire (Barton et al. 1987, Koskenniemi & Church 1988). The authors do issue frequent warnings about the size explosion in network compilation,[7] though some quantitative discussion would have been helpful.

Finally, it should be noted that *FSM* transducers are more properly called LEXICAL TRANSDUCERS than MORPHOLOGICAL ANALYZERS, as they produce all possible analyses of a word, rather than the one appropriate to a word's usage in a sentence. If one wishes to construct a morphologically annotated corpus, one will need to perform disambiguation, a problem more appropriately addressed through statistical methods.

3 *Finite State Morphology* and Morphology

There are plenty of useful lessons that linguists can take away from *FSM*.

First, who are the real winners of *FSM*? The REAL WORLD needs linguists, who will be pleased to learn from this book that their analytical skills are essential in the computational implementation of morphology. The authors are quite explicit (page 283):

> 'The lesson is this: study the language and do some old-fashioned pure linguistics modeling before jumping into coding. Your programs will never be better than the linguistic model behind them'.

There is no replacement for knowledge, not even powerful computers.

Second, which is the real winner among linguistic theories? For *FSM* the successful LINGUISTIC MODEL would seem to be a particular kind, namely the SPE-style system that assembles words out of pieces with ordered rules. Indeed, Karttunen (1998) has had some very critical remarks about other formal systems such as Optimality Theory. It has been shown (Frank & Satta 1998) that counting the number of constraint violations, which can go arbitrarily high (McCarthy 2003), places OT beyond the descriptive power of Finite State models, and thus deprives it of the simplicity and efficiency associated with Finite State systems. In other words, OT is descriptively more powerful but computationally less attractive than classical generative phonology. While it is possible to FINITIZE OT by placing a cap on the number of constraint violations (Karttunen 1998), the complexity of the resulting network (as measured by size) still compares unfavorably to that of derivational systems (Idsardi, to appear). However, we get a feeling that linguists will not take these complexity arguments seriously.

Third, it is interesting to note that for certain morphological phenomena, the computational approach and the theoretical accounts dovetail fairly nicely. This is particularly clear in the discussion of reduplication, Templatic Morphology, and other cases that do not readily fall under the Finite State approach. The solution to these problems presented in *FSM*, much like the use of flag diacritics, follows the general principle of using abstract representations to simplify surface-level descriptions. For instance, reduplication is handled by the postulation of reduplicative morphemes (Marantz 1982), and the treatment of Arabic morphology follows quite directly the insights of the tier-based approach (McCarthy 1981). These examples highlight the unifying theme of computation between computational and theoretical linguistics, that clean computational models are clean linguistic models.

Fourth, working with *FSM* may force the linguist to confront theoretical problems more broadly than pencil and paper analysis. A broad coverage morphological system should handle both the general patterns of the language as well as the more idiosyncratic cases: how to strike a balance between these is a matter of considerable theoretical interest. Unfortunately, *FSM* has no original insight to contribute. For instance, one of the ways in which exceptions are handled here is simply by listing: *swim-swam* is coded by directly lexical lookup. The authors do not provide guidelines on when productive rules and morphological decomposition should be used and when to resort to holistic storage. These decisions have direct implications on the economy of description and efficiency of processing. Of course, theoretical and experimental research in morphology faces the same range of issues, and it may be fruitful if these two lines of work can find a point of convergence (Yang 2002, 2005).

Fifth, the sole focus of the book is the implementation of morphological systems, but that is only one of the many areas in computational morphology. Current research topics include other approaches to non-concatenative languages (Cohen-Sygal & Wintner 2006), induction of morphological segmentation (Goldsmith 2006), and joint morphological and syntactic disambiguation (Cohen & Smith 2007). And this is not to mention the study of morphological acquisition and cognitive processing, where many problems can, and have been, studied in a computational framework. It remains to be seen whether the Finite State approach can make connections with these many sides of morphological research.

Finally, a philosophical point: in what sense is morphology Finite State when one is constantly confronted with the linguistic facts that are obviously not? Even in English, hardly an exotic morphological specimen, we find pig latin, *shm*-reduplication (*fancy-schmancy*), and other cases where long distance dependencies are at odds with the doctrine of Finite State. On our view, the insistence on Finite-Stateness (or not) stems from the commitment to the weak generative capacity of linguistic systems. At some level, this sentiment is understandable: for practical problems in computational morphology, one needs to deal with letters and strings, and the emphasis is inevitably placed on the algorithmic process that manipulates these units. A strong generative capacity perspective shifts the attention to structural descriptions. Even in syntax, where the adjacency relations at the word level gets you nowhere, meaningful syntactic relations can be defined on adjacent structural units such as heads, specifiers, and complements. The theoretically motivated treatment of non-Finite-State phenomena in *FSM* shows that if the morphological representation is sufficiently abstract, the core engine of a Finite State process may well be maintained.

Notes

1. The authors maintain a website, http://fsmbook.com, that supplies documentation, updates, and other resources.
2. We will not be concerned with larger theoretical questions such as the place of morphology in the architecture of the grammar. We use the term MORPHOLOGY to refer to the description and analysis of lexical forms, and we take the goal of computational morphology to be to assign the correct structural descriptions to these forms, including the rejection of illicit ones.
3. Chapter 4 describes LEXC, an alternative formalism for finite state networks that is useful for the construction of the lexicon. Although classical generative phonology is concerned primarily with rules, incorporation of a lexicon helps to prevent spurious overanalysis in a finite state system. The final *FSM* networks are produced by composing a lexicon transducer with a rule transducer, so that only the legitimate lexical roots and their inflections are represented in the network. LEXC represents lexical items as sequences of underlying morphemes, which is especially useful for agglutinative languages.

   The main drawback to LEXC is that it uses a different set of conventions and syntactic forms than XFST. It is also possible to write rules within LEXC. On page 386 there is an example of code that produces equivalent grammars, one written in LEXC code, the other in XFST. This amount of flexibility is not considered to be a good way to design a programming language.
4. In order to obtain a deep level of understanding of Finite State Morphology, we recommend study of automata theory beyond what *FSM* has space to present.
5. Composition is often performed on entire networks, and the clever use of it can considerably ease system development. For example, one can construct a network that overgenerates possible structures and subtract from it a network that filters invalid structures, instead of writing intricate rules to exactly specify the final language.
6. There is a command (*inspect net*, page 189) that allows the user to traverse through the network during execution. But the size of the compiled network would provide only limited feedback from network operations to linguistic descriptions. Noting that this tool is only useful to experts or Xerox developers, the authors do not provide further details.
7. The authors also offer several solutions to particular problems that may reduce the problem of network size. For example, interpreted flag diacritics express constraints that could also have been accomplished through composition of filtering expressions, and the lookup program pipes input through a sequence of transducers at runtime, achieving the same effect as a single composed transducer. These strategies demonstrate the trade-off between storage and

computation: an increase in network size due to composition is avoided at the expense of an increased amount of processing time. In this regard, we can see how a computational implementation of linguistic theory introduces new and extremely important problems in addition to descriptive adequacy.

Reference

Anderson, Stephen R. 1988. Morphology as a parsing problem. *Linguistics* 26: 521–544.

Barton Jr., G. Edward, Robert C. Berwick & Eric Sven Ristad 1987. *Computational complexity and natural language*. Cambridge, MA: MIT Press.

Chomsky, Noam, & Morris Halle 1968. *The sound pattern of English*. New York: Harper and Row.

Cohen, Shay B. & Noah A. Smith 2007. Joint morphological and syntactic disambiguation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Prague*. 208–217. (Available on-line.)

Cohen-Sygal, Yael & Shuly Wintner 2006. Finite-state registered automata for non-concatenative morphology. *Computational Linguistics* 32(1): 49–82.

Frank, Robert & Giorgio Satta 1998. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics* 24(2): 307–316.

Goldsmith, John 2006. An algorithm for the unsupervised learning of morphology. *Natural Language Engineering* 12(3): 1–19.

Idsardi, W. To appear. Calculating metrical structure. In C. Cairnes & E. Raimy (eds.) To be announced. Cambridge, MA: MIT Press.

Johnson, C. Douglas 1972. *Formal aspects of phonological description*. The Hague: Mouton.

Kaplan, Ronald M. & Martin Kay 1994. Regular models of phonological rule systems. *Computational Linguistics* 20(3): 331–378.

Karttunen, Lauri 1998. The proper treatment of optimality in computational phonology. In *Proceedings of the International Workshop on Finite-State Methods in Natural Language Processing*. Ankara. 1–12.

Koskenniemi, Kimmo 1983. *Two-level morphology: a general computational model for word-form recognition and production*. Helsinki: Department of General Linguistics, University of Helsinki.

Koskenniemi, Kimmo & Kenneth Ward Church 1988. Complexity, two-level morphology and Finnish. In *COLING-88*: *Proceedings of the 12th International Conference on Computational Linguistics*, volume 1 (Budapest, 1988). 335–339.

Marantz, Alec 1982. Re reduplication. *Linguistic Inquiry* 13: 435–448.

McCarthy, John 1981. A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry* 12: 373–418.

McCarthy, John 2003. OT constraints are categorical. *Phonology* 20: 75–138.

Mohri, Mehryar, Fernando C. N. Pereira & Michael Riley 1998. A rational design for a weighted finite-state transducer library. In *Automata Implementation. Second International Workshop on Implementing Automata, WIA '97*. (= Lecture Notes in Computer Science 1436). Dordrecht: Springer.

Noord, Gertjan van 1997. FSA utilities: a toolbox to manipulate finite-state automata. In Darrell Raymond, Derick Wood & Sheng Yu (eds.), *Automata implementation*. (= Lecture Notes in Computer Science 1260). Dordrecht: Springer.

December 1, 2008 Time: 09:40am word023.tex

Yang, Charles 2002. *Knowledge and learning in natural language*. New York: Oxford University Press.

Yang, Charles 2005. On productivity. *Yearbook of Language Variation* 5: 333–370.

Author's addresses:

(Erwin Chan)
University of Arizona
Department of Linguistics
Douglass Hall 200E
1100 E. University Blvd
Tucson, AZ 85721
E-mail: echan3@seas.upenn.edu

(Charles Yang)
Department of Linguistics
University of Pennsylvania
608 Williams Hall
Philadelphia, PA 19104-6305
E-mail: charles.yang@ling.upenn.edu

Author's address:

(Bogdan Szymanek)
Department of Modern English
John Paul II Catholic University of Lublin
Al. Raclawickie 14
20–950 Lublin, Poland
Email: szymanek@kul.lublin.pl