8

Twenty-Five Years of Finite-State Morphology

LAURI KARTTUNEN AND KENNETH R. BEESLEY

8.1 Introduction

Twenty-five years ago in the early 1980s, morphological analysis of natural language was a challenge to computational linguists. Simple cut-and-paste programs could be and were written to analyze strings in particular languages, but there was no general language-independent method available. Furthermore, cut-and-paste programs for analysis were not reversible, they could not be used to generate words. Generative phonologists of that time described morphological alternations by means of ordered rewrite rules, but it was not understood how such rules could be used for analysis.

This was the situation in the spring of 1981 when Kimmo Koskenniemi came to a conference on parsing that Lauri Karttunen had organized at the University of Texas at Austin. Also at the same conference were two Xerox researchers from Palo Alto, Ronald M. Kaplan and Martin Kay. The four Ks discovered that all of them were interested and had been working on the problem of morphological analysis. Koskenniemi went on to Palo Alto to visit Kay and Kaplan at Xerox PARC.

This was the beginning of Two-Level Morphology, the first general model in the history of computational linguistics for the analysis and generation of morphologically complex languages. The language-specific components, the lexicon and the rules, were combined with a runtime engine applicable to all languages.

Inquiries into Words, Constraints and Contexts.
Antti Arppe et al. (Eds.)

8.2 The Origins

Traditional phonological grammars, formalized by Chomsky and Halle (1968), consisted of an ordered sequence of REWRITE RULES that converted abstract phonological representations into surface forms through a series of intermediate representations. Such rewrite rules have the general form $\alpha \rightarrow \beta \ / \ \gamma \_ \ \delta$ where α, β, γ, and δ can be arbitrarily complex strings or feature-matrices. The rule is read "α is rewritten as β between γ and δ". In mathematical linguistics (Partee et al. 1993), such rules are called CONTEXT-SENSITIVE REWRITE RULES, and they are more powerful than regular expressions or context-free rewrite rules.

In 1972, C. Douglas Johnson published his dissertation, *Formal Aspects of Phonological Description*, wherein he showed that phonological rewrite rules are actually much less powerful than the notation suggests. Johnson observed that while the same context-sensitive rule could be applied several times recursively to its own output, phonologists have always assumed implicitly that the site of application moves to the right or to the left in the string after each application. For example, if the rule $\alpha \rightarrow \beta \ / \ \gamma \_ \ \delta$ is used to rewrite the string $\gamma\alpha\delta$ as $\gamma\beta\delta$, any subsequent application of the same rule must leave the β part unchanged, affecting only γ or δ. Johnson demonstrated that the effect of this constraint is that the pairs of inputs and outputs produced by a phonological rewrite rule can be modeled by a finite-state transducer. This result was largely overlooked at the time and was rediscovered by Ronald M. Kaplan and Martin Kay around 1980 . Putting things into a more algebraic perspective than Johnson, Kaplan and Kay showed that phonological rewrite rules describe REGULAR RELATIONS. By definition, a regular relation can be represented by a finite-state transducer.

Johnson was already aware of an important mathematical property of finite-state transducers established by Schützenberger (1961): there exists, for any pair of transducers applied sequentially, an equivalent single transducer. Any cascade of rule transducers can in principle be composed into a single transducer that maps lexical forms directly into the corresponding surface forms, and vice versa, without any intermediate representations.

These theoretical insights did not immediately lead to practical results. The development of a compiler for rewrite rules turned out to be a very complex task. It became clear that building a compiler required as a first step a complete implementation of basic finite-state operations such as union, intersection, complementation, and composition. Developing a complete finite-state calculus was a challenge in itself on the computers that were available at the time.

Another reason for the slow progress may have been that there were persistent doubts about the practicality of the approach for morphological ANAL-

YSIS. Traditional phonological rewrite rules describe the correspondence between lexical forms and surface forms as a one-directional, sequential mapping from lexical forms to surface forms. Even if it was possible to model the GENERATION of surface forms efficiently by means of finite-state transducers, it was not evident that it would lead to an efficient analysis procedure going in the reverse direction, from surface forms to lexical forms.

Let us consider a simple illustration of the problem with two sequentially applied rewrite rules, `N -> m / _ p` and `p -> m / m _`. The corresponding transducers map the lexical form *kaNpat* unambiguously to *kammat*, with *kampat* as the intermediate representation. However if we apply the same transducers in the other direction to the input *kammat*, we get the three results shown in Figure 1.

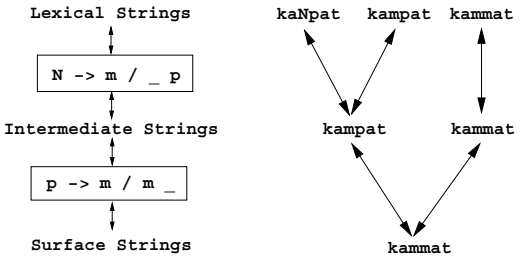

FIGURE 1 Deterministic Generation, Nondeterministic Analysis

This asymmetry is an inherent property of the generative approach to phonological description. If all the rules are deterministic and obligatory and if the order of the rules is fixed, each lexical form generates only one surface form. But a surface form can typically be generated in more than one way, and the number of possible analyses grows with the number of rules that are involved. Some of the analyses may turn out to be invalid because the putative lexical forms, say *kammat* and *kampat* in this case, might not exist in the language. But in order to look them up in the lexicon, the system must first complete the analysis. Depending on the number of rules involved, a surface form could easily have dozens of potential lexical forms, even an infinite number in the case of certain deletion rules.

Although the generation problem had been solved by Johnson, Kaplan and Kay, at least in principle, the problem of efficient morphological analysis in the Chomsky-Halle paradigm was still seen as a formidable challenge. As counterintuitive as it was, it appeared that analysis was computationally a much more difficult task than generation. Composing all the rule transducers into a single one would not solve the "overanalysis" problem. Because the resulting single transducer is equivalent to the original cascade, the ambiguity

remains.

The solution to the overanalysis problem should have been obvious: to formalize the lexicon itself as a finite state transducer and compose the lexicon with the rules. In this way, all the spurious ambiguities produced by the rules are eliminated at compile time. The resulting single transducer contains only lexical forms that actually exist in the language. When this idea first surfaced in Karttunen et al. (1992), it was not in connection with traditional rewrite rules but with an entirely different finite-state formalism that had been introduced in the meantime, called TWO-LEVEL RULES (Koskenniemi 1983).

8.3 Two-level Morphology

In the spring of 1981 when Kimmo Koskenniemi came to the USA for a visit, he learned about Kaplan and Kay's finite-state discovery.[1] PARC had begun work on the finite-state algorithms, but they would prove to be many years in the making. Koskenniemi was not convinced that efficient morphological analysis would ever be practical with generative rules, even if they were compiled into finite-state transducers. Some other way to use finite automata might be more efficient.

Back in Finland, Koskenniemi invented a new way to describe phonological alternations in finite-state terms. Instead of cascaded rules with intermediate stages and the computational problems they seemed to lead to, rules could be thought of as statements that directly constrain the surface realization of lexical strings. The rules would not be applied sequentially but in parallel. Each rule would constrain a certain lexical/surface correspondence and the environment in which the correspondence was allowed, required, or prohibited. For his 1983 dissertation, Koskenniemi constructed an ingenious implementation of his constraint-based model that did not depend on a rule compiler, composition or any other finite-state algorithm, and he called it TWO-LEVEL MORPHOLOGY. Two-level morphology is based on three ideas:

- Rules are symbol-to-symbol constraints that are applied in parallel, not sequentially like rewrite rules.
- The constraints can refer to the lexical context, to the surface context, or to both contexts at the same time.
- Lexical lookup and morphological analysis are performed in tandem.

To illustrate the first two principles we can turn back to the *kaNpat* example again. A two-level description of the lexical-surface relation is sketched in Figure 2.

As the lines indicate, each symbol in the lexical string *kaNpat* is paired with its realization in the surface string *kammat*. Two of the symbol pairs in Fig-

[1] They weren't then aware of Johnson's 1972 publication.

FIGURE 2 Example of Two-Level Constraints

ure 2 are constrained by the context marked by the associated box. The *N:m* pair is *restricted* to the environment having an immediately following *p* on the lexical side. In fact the constraint is tighter. In this context, all other possible realizations of a lexical *N* are *prohibited*. Similarly, the *p:m* pair requires the preceding surface *m*, and no other realization of *p* is allowed here. The two constraints are independent of each other. Acting in parallel, they have the same effect as the cascade of the two rewrite rules in Figure 1. In Koskenniemi's notation, these rules are written as `N:m <=> _ p:` and `p:m <=> :m _` , where `<=>` is an operator that combines a context restriction with the prohibition of any other realization for the lexical symbol of the pair. The colon in the right context of first rule, `p:`, indicates that it refers to a lexical symbol; the colon in the left context of the second rule, `:m`, indicates a surface symbol.

Two-level rules may refer to both sides of the context at the same time. The *y*∼*ie* alternation in English plural nouns could be described by two rules: one realizes *y* as *i* in front of an epenthetic *e*; the other inserts an epenthetic *e* between a lexical consonant-*y* sequence and a morpheme boundary (+) that is followed by an *s*. Figure 3 illustrates the *y:i* and *0:e* constraints.

FIGURE 3 A Two-Level View of *y*∼*ie* Alternation in English

Note that the *e* in Figure 3 is paired with a *0* (= zero) on the lexical level. In two-level rules, zero is a symbol like any other; it can be used to constrain the realization of other symbols, as in `y:i <=> _ 0:e`. In fact, all the other rules must "know" where zeros may occur. Zeros are treated as epsilons only when two-level rules are applied to strings.

Like rewrite rules, two-level rules describe regular relations; but there is an important difference. Because the zeros in two-level rules are ordinary symbols, a two-level rule represents an EQUAL-LENGTH RELATION. This has an important consequence: Although regular relations in general are not closed under intersection, equal length relations have that property. When a set of

two-level transducers are applied in parallel, the apply routine in fact simulates the intersection of the rule automata and composes the input string with the virtual constraint network.

Applying the rules in parallel does not in itself solve the overanalysis problem discussed in the previous section. The two constraints sketched above allow *kammat* to be analyzed as *kaNpat*, *kampat*, or *kammat*. However, the problem becomes manageable when there are no intermediate levels of analysis. In Koskenniemi's 1983 system, the lexicon was represented as a forest of tries (= letter trees), tied together by continuation-class links from leaves of one tree to roots of another tree or trees.[2] Lexical lookup and the analysis of the surface form are performed in tandem. In order to arrive at the point shown in Figure 4, the analyzer has traversed a branch in the lexicon that

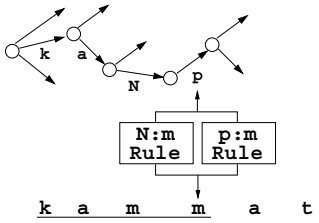

FIGURE 4 Following a Path in the Lexicon

contains the lexical string *kaN*. At this point, it only considers symbol pairs whose lexical side matches one of the outgoing arcs of the current state. It does not pursue analyses that have no matching lexical path.

Koskenniemi's two-level morphology was the first practical general model in the history of computational linguistics for the analysis of morphologically complex languages. The language-specific components, the rules and the lexicon, were combined with a universal runtime engine applicable to all languages.

8.4 A Two-Level Rule Compiler

In his dissertation, Koskenniemi introduced a formalism for two-level rules. The semantics of two-level rules was well-defined but there was no rule compiler available at the time. Koskenniemi and other early practitioners of two-level morphology constructed their rule automata *by hand*. This is tedious in the extreme and very difficult for all but very simple rules.

Although two-level rules are formally quite different from the rewrite rules studied by Kaplan and Kay, the methods that had been developed for com-

[2] The TEXFIN analyzer developed at the University of Texas at Austin (Karttunen et al. 1981) had the same lexicon architecture.

piling rewrite rules were applicable to two-level rules as well. In both formalisms, the most difficult case is a rule where the symbol that is replaced or constrained appears also in the context part of the rule. This problem Kaplan and Kay had already solved by an ingenious technique for introducing and then eliminating auxiliary symbols to mark context boundaries. Another fundamental insight they had was the encoding of context restrictions in terms of double negation. For example, a constraint such as "*p* must be followed by *q*" can be expressed as "it is not the case that something ending in *p* is not followed by something starting with *q*." In Koskenniemi's formalism, `p => _ q`.

In the summer of 1985, when Koskenniemi was a visitor at Stanford, Kaplan and Koskenniemi worked out the basic compilation algorithm for two-level rules. The first two-level rule compiler was written in InterLisp by Koskenniemi and Karttunen in 1985-87 using Kaplan's implementation of the finite-state calculus (Koskenniemi 1986, Karttunen et al. 1987). The current C-version of the compiler, called TWOLC, was written at PARC in 1991-92 (Karttunen and Beesley 1992).[3]

Although the basic compilation problem was solved quickly, building a practical compiler for two-level rules took a long time. The TWOLC compiler includes sophisticated techniques for checking and resolving conflicts between rules whenever possible. Without these features, large rule systems would have been impossible to construct and debug. If two constraints are in conflict, some lexical forms have no valid surface form. This is a common problem and often difficult to remedy even if the compiler is able to detect the situation and to pinpoint the cause.

8.5 Two-Level Implementations

Koskenniemi's Pascal implementation was quickly followed by others. The most influential of them was the KIMMO system by Lauri Karttunen and his students at the University of Texas (Karttunen 1983, Gajek et al. 1983). This Lisp project inspired many copies and variations, including those by Beesley (1989, 1990). A free C implementation of classic Two-Level Morphology, called PC-KIMMO, from the Summer Institute of Linguistics (Antworth 1990), became a popular tool.

In Europe, two-level morphological analyzers became a standard com-

[3]The landmark 1994 article by Kaplan and Kay on the mathematical foundations of finite-state linguistics defines the basic compilation algorithm for phonological rewrite rules and for Koskenniemi's two-level rules. The article appeared years after the work on the two-level compiler was completed and just before the implementation of the so-called REPLACE RULES in the current PARC/XRCE regular expression compiler. The article is accurate on the former topic, but the algorithm for replace rules (Karttunen 1995, 1996, Kempe and Karttunen 1996) differs in many details from the compilation of rewrite rules as described by Kaplan and Kay.

ponent in several large systems for natural language processing such as the British Alvey project (Black et al. 1987, Ritchie et al. 1987, 1992), SRI's CLE Core Language Engine (Carter 1995), the ALEP Natural Language Engineering Platform (Pulman 1991) and the MULTEXT project (Armstrong 1996). ALEP and MULTEXT were funded by the European Commission. [4]

Some of these systems were based on simplified two-level rules, the so-called PARTITION-BASED formalism Ruessink (1989), which was claimed to be easier for linguists to learn than the original Koskenniemi notation. But none of these systems had a finite-state rule compiler.[5] Another difference was that morphological parsing could be constrained by feature unification. Because the rules were interpreted at runtime and because of the unification overhead, these systems were not efficient, and two-level morphology acquired, undeservedly, a reputation for being slow.

At XRCE and Inxight, the TWOLC compiler was used in the 1990s to develop comprehensive morphological analyzer for numerous languages. Another utility, called LEXC (Karttunen 1993b), made it possible to combine a finite-state lexicon with a set of two-level rules into a single LEXICAL TRANSDUCER using a special "intersecting composition" algorithm that simulates the intersection of the rules while simultaneously composing the virtual rule transducer with the lexicon. A lexical transducer can be considered the ultimate two-level model of a language as it encodes compactly:

- all the LEMMAS (canonical lexical forms with morphological tags)
- all the inflected surface forms
- all the mappings between lexical forms and surface forms.

In the course of this work it became evident that lexical transducers are easier to construct with sequentially applied replace rules than with two-level rules. Large systems of two-level rules are notoriously difficult to debug. Most developers of morphological analyzers at XRCE and at companies such as Inxight have over the years switched to the sequential model and the XFST tool that includes a compiler for replace rules. The ordering of replace rules seems to be less of a problem than the mental discipline required to avoid rule conflicts in a two-level system, even if the compiler automatically resolves most of them. From a formal point of view there is no substantive difference; a cascade of rewrite rules and a set of parallel two-level constraints are just two different ways to decompose a complex regular relation into a set of simpler relations that are easier to understand and manipulate.

[4] The MULTEXT morphology tool (Petitpierre and Russel 1995) built at ISSCO is available at `http://packages.debian.org/stable/misc/mmorph.html`

[5] A compilation algorithm has been developed for the partition-based formalism Grimley-Evans et al. (1996), but to our knowledge there is no publicly available implementation.

The Beesley and Karttunen (2003) book *Finite State Morphology* describes the XFST and LEXC tools and offers a lot of practical advice on techniques for constructing lexical transducers.[6]

8.6 Reflections

Although the two-level approach to morphological analysis was quickly accepted as a useful practical method, the linguistic insight behind it was not picked up by mainstream linguists. The idea of rules as parallel constraints between a lexical symbol and its surface counterpart was not taken seriously at the time outside the circle of computational linguists. Many arguments had been advanced in the literature to show that phonological alternations could not be described or explained adequately without sequential rewrite rules. It went largely unnoticed that two-level rules could have the same effect as ordered rewrite rules because two-level rules allow the realization of a lexical symbol to be constrained either by the lexical side or by the surface side. The standard arguments for rule ordering were based on the *a priori* assumption that a rule could refer only to the input context (Karttunen 1993a).

But the world has changed. Current phonologists, writing in the framework of OT (Optimality Theory), are sharply critical of the "serialist" tradition of ordered rewrite rules that Johnson, Kaplan and Kay wanted to formalize (Prince and Smolensky 1993, Kager 1999, McCarthy 2002).[7] In a nutshell, OT is a two-level theory with *ranked* parallel constraints. Many types of optimality constraints can be represented trivially as two-level rules. In contrast to Koskenniemi's "hard" constraints, optimality constraints are "soft" and violable. There are of course many other differences. Most importantly, OT constraints are meant to be universal. The fact that two-level rules can describe orthographic idiosyncrasies such as the *y*∼*ie* alternation in English with no appeal to universal principles is a minus rather than a plus. It makes the approach uninteresting from the OT point of view.[8]

Nevertheless, from the OT perspective, two-level rules have some interesting properties. They are symbol-to-symbol constraints, not string-to-string relations like general rewrite rules. Two-level rules enable the linguist to refer to the input and the output context in the same constraint. The notion of FAITHFULNESS (= no change) can be expressed straight-forwardly. It is possible to formulate constraints that constrain directly the surface level. These ideas were ten years ahead of their time in 1983.

[6] The book includes a CD that contains TWOLC, XFST, LEXC and other finite-state tools. See also `http://www.fsmbook.com`. The documentation for TWOLC, missing from the book, is included on the CD.

[7] The term SERIAL, a pejorative term in an OT context, refers to sequential rule application.

[8] Finite-state approaches to Optimality Theory have been explored in several recent articles (Eisner 1997, Frank and Satta 1998, Karttunen 1998).

It is interesting to observe that computational linguists and "paper-and-pencil linguists" have historically been out of sync in their approach to phonology and morphology. When computational linguists implemented parallel two-level models in the 1980s, paper-and-pencil linguists were still stuck in the serialist Chomsky-Halle paradigm. When most of the computational morphologists working with the Xerox tools embraced the sequential model as the more practical approach in the mid 1990s, a two-level theory took over paper-and-pencil linguistics by a storm in the guise of OT.

If one views the mapping from lexical forms to surface forms as a regular relation, the choice between different ways of decomposing it has practical consequences but it is not a deep theoretical issue for computational linguists. No brand of finite-state morphology has ever been promoted as a theory about language. Its practitioners have always been focused on the practical task of representing the morphological aspects of a language in a form that supports efficient analysis and generation. They have been remarkably successful in that task.

Paper-and-pencil morphologists in general are not interested in creating complete descriptions for particular languages. They design formalisms for expressing generalizations about morphological phenomena commonly found in all natural languages. But if it turns out, as in the case of REALIZATIONAL MORPHOLOGY (Stump 2001), that the theory can be implemented with finite-state tools (Karttunen 2003), perhaps the phenomena are not as complex as the linguist has imagined.

Acknowledgments

Much of the material in this article comes from an unpublished paper, "A Short History of Two-Level Morphology", that we presented at a Special Event celebrating "Twenty Years of Two-Level Morphology" at ESSLLI-2001 in Helsinki (`http://www.helsinki.fi/esslli/`).

References

Antworth, Evan L. 1990. *PC-KIMMO: a two-level processor for morphological analysis*. No. 16 in Occasional publications in academic computing. Dallas: Summer Institute of Linguistics.

Armstrong, Susan. 1996. Multext: Multilingual text tools and corpora. In H. Feldweg and E. W. Hinrichs, eds., *Lexikon und Text*, pages 107–112. Tuebingen: Max Niemeyer Verlag.

Beesley, Kenneth R. 1989. Computer analysis of Arabic morphology: A two-level approach with detours. In *Third Annual Symposium on Arabic Linguistics*. Salt Lake City: University of Utah. Published as Beesley, 1991.

Beesley, Kenneth R. 1990. Finite-state description of Arabic morphology. In *Proceedings of the Second Cambridge Conference on Bilingual Computing in Arabic and English*. No pagination.

Beesley, Kenneth R. 1991. Computer analysis of Arabic morphology: A two-level approach with detours. In B. Comrie and M. Eid, eds., *Perspectives on Arabic Linguistics III: Papers from the Third Annual Symposium on Arabic Linguistics*, pages 155–172. Amsterdam: John Benjamins.

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite State Morphology*. Palo Alto, CA: CSLI Publications.

Black, A., G. Ritchie, S. Pulman, and G. Russell. 1987. Formalisms for morphographemic description. In *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, pages 11–18.

Carter, D. 1995. Rapid development of morphological descriptions for full language processing systems. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics*, pages 202–209.

Chomsky, Noam and Morris Halle. 1968. *The Sound Pattern of English*. New York: Harper and Row.

Eisner, Jason. 1997. Efficient generation in primitive Optimality Theory. In *Proceedings of the 35th Annual ACL and 8th EACL*, pages 313–320. Madrid: Association for Computational Linguistics.

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

Gajek, Oliver, Hanno T. Beck, Diane Elder, and Greg Whittemore. 1983. LISP implementation. *Texas Linguistic Forum* 22:187–202.

Grimley-Evans, Edmund, George Anton Kiraz, and Steven G. Pulman. 1996. Compiling a partition-based two-level formalism. In *COLING'96*. Copenhagen. cmp-lg/9605001.

Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description*. The Hague: Mouton.

Kager, Reni. 1999. *Optimality Theory*. Cambridge, England: Cambridge University Press.

Kaplan, Ronald M. and Martin Kay. 1981. Phonological rules and finite-state transducers. In *Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting*. New York. Abstract.

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

Karttunen, Lauri. 1983. KIMMO: a general morphological processor. *Texas Linguistic Forum* 22:165–186.

Karttunen, Lauri. 1993a. Finite-state constraints. In J. Goldsmith, ed., *The Last Phonological Rule*. Chicago, Illinois.: University of Chicago Press.

Karttunen, Lauri. 1993b. Finite-state lexicon compiler. Tech. Rep. ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center, Palo Alto, CA.

Karttunen, Lauri. 1995. The replace operator. In *ACL'95*. Cambridge, MA. cmp-lg/9504032.

Karttunen, Lauri. 1996. Directed replacement. In *ACL'96*. Santa Cruz, CA. cmp-lg/9606029.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In *FSMNLP'98. International Workshop on Finite-State Methods in Natural Language Processing*. Bilkent University, Ankara, Turkey. cmp-lg/9804002.

Karttunen, Lauri. 2003. Computing with realizational morphology. In A. Gelbukh, ed., *Computational Linguistics and Intelligent Text Processing*, vol. 2588 of *Lecture Notes in Computer Science*, pages 205–216. Heidelberg: Springer Verlag.

Karttunen, Lauri and Kenneth R. Beesley. 1992. Two-level rule compiler. Tech. Rep. ISTL-92-2, Xerox Palo Alto Research Center, Palo Alto, CA.

Karttunen, Lauri, Ronald M. Kaplan, and Annie Zaenen. 1992. Two-level morphology with composition. In *COLING'92*, pages 141–148. Nantes, France.

Karttunen, Lauri, Kimmo Koskenniemi, and Ronald M. Kaplan. 1987. A compiler for two-level phonological rules. In M. Dalrymple, R. Kaplan, L. Karttunen, K. Koskenniemi, S. Shaio, and M. Wescoat, eds., *Tools for Morphological Analysis*, vol. 87-108 of *CSLI Reports*, pages 1–61. Palo Alto, CA: Center for the Study of Language and Information, Stanford University.

Karttunen, Lauri, Hans Uszkoreit, and Rebecca Root. 1981. Morphological analysis of Finnish by computer. In *Proceedings of the 71st Annual Meeting of SASS*. Albuquerque, New Mexico.

Kempe, André and Lauri Karttunen. 1996. Parallel replacement in finite-state calculus. In *COLING'96*. Copenhagen. cmp-lg/9607007.

Koskenniemi, Kimmo. 1983. Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Koskenniemi, Kimmo. 1986. Compilation of automata from morphological two-level rules. In F. Karlsson, ed., *Papers from the Fifth Scandinavian Conference on Computational Linguistics*, pages 143–149.

McCarthy, John J. 2002. *The Foundations of Optimality Theory*. Cambridge, England: Cambridge University Press.

Partee, B. H., A. ter Meulen, and R. E. Wall. 1993. *Mathematical Methods in Linguistics*. Dordrecht: Kluwer.

Petitpierre, Dominique and Graham Russel. 1995. MMORPH — the multext morphology program. Multext Deliverable 2.3.1, ISSCO, Carouge, Switzerland.

Prince, Allan and Paul Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Tech. rep., Rutgers University, Piscataway, NJ. RuCCS Technical Report 2. Rutgers Center for Cognitive Science.

Pulman, S. 1991. Two level morphology. In H. Alshawi, D. Arnold, R. Backofen, D. Carter, J. Lindop, K. Netter, S. Pulman, J. Tsujii, and H. Uskoreit, eds., *ET6/1 Rule Formalism and Virtual Machine Design Study*, chap. 5. Luxembourg: CEC.

Ritchie, G., A. Black, S. Pulman, and G. Russell. 1987. The Edinburgh/Cambridge morphological analyser and dictionary system (version 3.0) user manual. Tech. Rep. Software Paper No. 10, Department of Artificial Intelligence, University of Edinburgh.

Ritchie, G., G. Russell, A. Black, and S. Pulman. 1992. *Computational Morphology: Practical Mechanisms for the English Lexicon*. MIT Press, Cambridge, Mass.

Ruessink, H. 1989. Two level formalisms. Utrecht Working Papers in NLP 5, Utrecht University.

Schützenberger, Marcel-Paul. 1961. A remark on finite transducers. *Information and Control* 4:185–196.

Stump, Gregory T. 2001. *Inflectional Morphology. A Theory of Paradigm Structure*. Cambridge, England: Cambridge University Press.