A Short History of Two-Level Morphology

Lauri Karttunen
Xerox Palo Alto Research Center
Kenneth R. Beesley
Xerox Research Centre Europe

September 28, 2001

Abstract

Twenty years ago 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 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. In this article we trace the development of the finite-state technology that Two-Level Morphology is based on.

1 The Origins

Traditional phonological grammars, formalized in the 1960s by Noam Chomsky and Morris Halle (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 rules have the general form `x -> y / z _ w` where `x`, `y`, `z`, and `w` can be arbitrarily complex strings or feature-matrices. 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* (Johnson, 1972), 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 of the string after each application. For example, if the rule `x -> y / z _ w` is used to rewrite the string "`uxxwv`" as "`uxywv`", any subsequent application of the same rule must leave the "`y`" part unchanged, affecting only "`ux`" or "`wv`". Johnson demonstrated that the effect of this constraint is that the pairs of inputs and outputs of any such rule can be modeled by a finite-state transducer. Unfortunately, this result was largely overlooked at the time and was rediscovered by Ronald M. Kaplan and Martin Kay around 1980 (Kaplan and Kay, 1981; Kaplan and Kay, 1994). 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 (Schützenberger, 1961): there exists, for any pair of transducers applied sequentially, an equivalent single transducer. Any cascade of rule transducers could in principle be composed into one transducer that maps lexical forms directly into the corresponding surface forms, and vice versa, without any intermediate representations. Later, Kaplan and Kay had the same idea, illustrated in Figure 1.

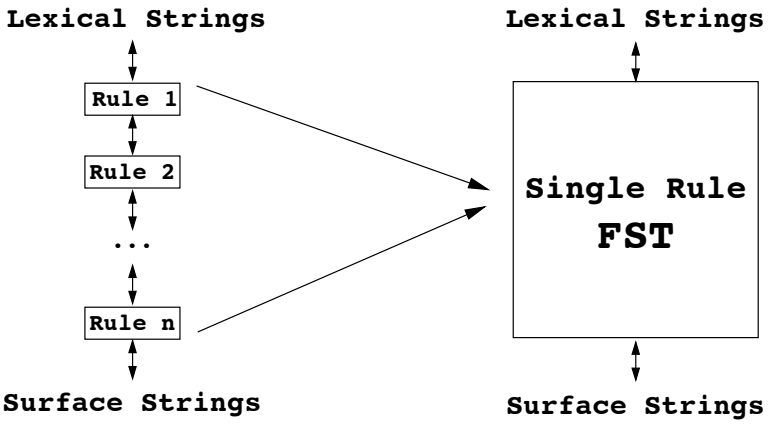

Figure 1: A Cascade of Rewrite Rules Composed into a Single **FST**

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 per-

sistent doubts about the practicality of the approach for morphological *analysis*. 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 2.

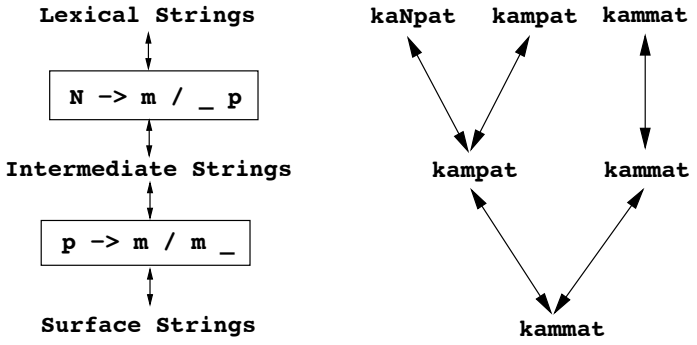

Figure 2: Rules Mapping *kammat* to *kaNpat, kampat, kammat*

The reason is that the surface form "`kammat`" has two potential sources on the intermediate level. The application of the `p -> m / m _` rule maps "`kampat`" and "`kammat`" to the same surface form. The intermediate form "`kampat`" in turn could come either from "`kampat`" or from "`kaNpat`" by the application of the `N -> m / _ p` rule. The two rule transducers are unambiguous in the downward direction but ambiguous in the upward direction.

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 from a psycholinguistic point of view, it appeared

that analysis was much harder computationally 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. With the lexicon included in the composition, all the spurious ambiguities produced by the rules are eliminated at compile time. The runtime analysis becomes more efficient because the resulting single transducer contains only lexical forms that actually exist in the language.

The idea of composing the lexicon and the rules together is not mentioned in Johnson's book or in the early Xerox work. Although there obviously had to be some interface relating a lexicon component to a rule component, these were traditionally thought of as different types of objects. Furthermore, rewrite rules were seen as applying to individual word forms; the idea of applying them simultaneously to a lexicon as a whole required a new mindset and computational tools that were not yet available.

The observation that a single finite-state transducer could encode the inventory of valid lexical forms as well as the mapping from lexical forms to surface forms took a while to emerge. When it first appeared in print (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, Kimmo Koskenniemi's TWO-LEVEL RULES (Koskenniemi, 1983).

2 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. (They weren't then aware of Johnson's 1972 publication.) **Xerox** 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 3.

Figure 3: Example of Two-Level Constraints

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 Figure 3 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 2. 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 4 illustrates the **y:i** and **0:e** constraints.

Figure 4: A Two-Level View of *y*∼*ie* Alternation in English

Note that the **e** in Figure 4 is paired with a **0** (= zero) on the lexical level. Formally this rule is expressed as `y:i <=> _ 0:e`. From the point of view of two-level rules, zero is a symbol like any other; it can be used to constrain the realization of other symbols. In fact, all the other rules must

"know" where zeros may occur. In two-level rules, zeros are not epsilons, even though they are treated as such 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 in fact ordinary symbols, a two-level rule represents an *equal-length relation*. This has an important consequence: Although transducers cannot in general be intersected, Koskenniemi's constraint transducers can be intersected. In fact, when a set of two-level transducers are applied in parallel, the apply routine simulates the intersection of the rule automata and composes the input string with the virtual constraint network.

Figure 5 illustrates the application of the **N:m** and **p:m** rules sketched in Figure 3 to the input "`kammat`". At each point in the process, all lexical candidates corresponding to the current surface symbol are considered one by one. If both rules accept the pair, the process moves on to the next point in the input. In the situation shown in Figure 5, the pair **p:m** will be accepted by both rules. The **N:m** rule accepts the pair because the **p** on the lexical side is required to license the **N:m** pair that has tentatively been accepted at the previous step. The **p:m** rule accepts the **p:m** pair because the preceding pair has an **m** on the surface side.

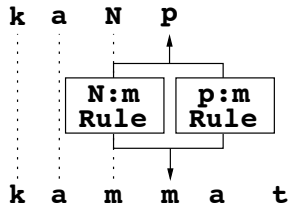

Figure 5: Parallel Application

When the pair in question has been accepted, the apply routine moves on to consider the next input symbol and eventually comes back to the point shown in Figure 5 to consider other possible lexical counterparts of a surface **m**. They will all be rejected by the **N:m** rule, and the apply routine will return to the previous **m** in the input to consider other alternative lexical counterparts for it such as **p** and **m**. At every point in the input the apply routine must also consider all possible deletions, that is, pairs such as **+:0** and **e:0** that have a zero on the input side.

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 is easy to manage in a system that has only two levels. The possible upper-side symbols are constrained at each step by consulting the lexicon. In Koskenniemi's two-level system, lexical lookup and the analysis of the surface form are performed in tandem. In order to arrive at the point shown in Figure 5, we must have traversed a path in the lexicon that contains the lexical string in question, see Figure 6. The lexicon acts as a continuous lexical filter. The analysis routine only considers symbol pairs whose lexical side matches one of the outgoing arcs of the current state.

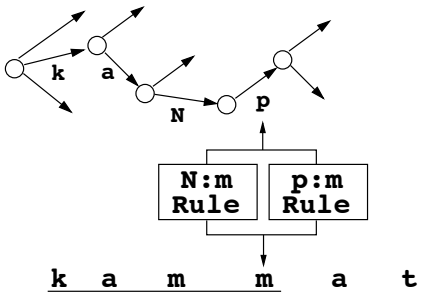

Figure 6: Following a Path in the Lexicon

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 in the forest.[1] Koskenniemi's lexicon can be thought of as a partially deterministic, unminimized simple network. In current finite-state systems the lexicon is a minimized network, typically a transducer, but the filtering principle is the same. The apply routine 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. The original implementation was primarily intended for analysis, but the model was in principle bidirectional and could be used for generation.

3 Linguistic Issues

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.

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).[2] In

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

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

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 help from universal principles makes the approach uninteresting from the OT point of view. [3]

4 Two-Level Rule Compilers

In his 1983 dissertation, Koskenniemi introduced a formalism for two-level rules. The semantics of two-level rules were well-defined but there was no rule compiler available at the time. Koskenniemi and other early practitioners of two-level morphology had to compile their rules *by hand* into finite-state transducers. This is tedious in the extreme and demands a detailed understanding of transducers and rule semantics that few human beings can be expected to grasp. A complex rule with multiple overlapping contexts may take hours of concentrated effort to compile and test, even for an expert human "compiler". In practice, linguists using two-level morphology consciously or unconsciously tended to postulate rather surfacy lexical strings, which kept the two-level rules relatively simple.

Although two-level rules are formally quite different from the rewrite rules studied by Kaplan and Kay, the basic finite-state methods that had been developed for compiling 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 was the encoding of contextual requirements 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, the same constraint is expressed by the rule `p => _ q`.

In the summer of 1985, when Koskenniemi was a visitor at the Center for the Study of Language and Information (CSLI) 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, based on Karttunen's 1989 Common Lisp implementation, was written by Lauri Karttunen, Todd Yampol and Kenneth R. Beesley in consultation with Kaplan at **Xerox PARC** in 1991-92 (Karttunen and Beesley, 1992). The landmark 1994 article by Kaplan and Kay on the mathematical foundations of finite-state linguistics gives a compilation algorithm for phonological rewrite rules and for Koskenniemi's two-level

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

rules.[4]

The **Xerox** two-level compiler has been used to create large-scale morphological analyzers for French, English, Spanish, Portuguese, Dutch, Italian and many other languages. In the course of this work, it soon became evident that the two-level formalism was difficult for the linguists to master. It is far too easy to write rules that are in conflict with one another. It was necessary to make the compiler check for, and automatically eliminate, most common types of conflicts.

For example, in Finnish consonant gradation, an intervocalic **k** generally disappears in the weak grade. However, between two high labial vowels **k** is realized as a **v**. Consequently, the genitive of "maku" 'taste' is "maun", but the genitive of "puku" 'suit, dress' is "puvun". The problem is that the **k** in "puku" is also an instance of an intervocalic **k**, and it should be deleted by the general rule. Figure 7 illustrates the conflict.

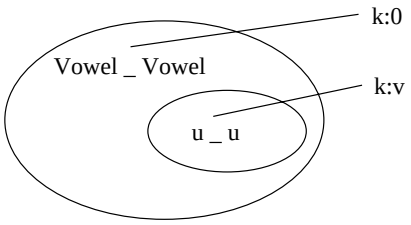

Figure 7: Conflict Between a General and a Specific Rule

This situation is not a problem for a derivational phonologist because the rule that turns **k** into **v** in the more specific context can be ordered before the deletion rule that applies in the more general environment. This is known as DISJUNCTIVE ordering. In Optimality Theory, cases of this sort are handled by constraint ranking. In a two-level framework, there is seemingly a problem. If one constraint requires the genitive of "puku" to be realized as "puun" and another accepts only "puvun", the result is that "puku" cannot be realized in the genitive at all.

This is one of the many types of conflicts that the **Xerox** compiler detects and resolves without difficulty. It sees that the context of the **k:v** rule is subsumed by the context of the **k:0** rule. The conflict is resolved by compiling the more general rule in such a way that an intervocalic **k** can be either deleted or realized as **v**. The compiler takes advantage of UNDERSPECIFICATION, a very useful option in a declarative system of constraints. The general rule relies on the specific one to produce the correct result. In the two-level model, it is not the responsibility of any single rule to get everything right because the other rules will also have their say.

[4]The Kaplan and Kay article appeared many years after the work on the two-level compiler was completed but before the implementation of the so-called REPLACE RULES in the current **Xerox** regular expression compiler. The article is accurate on the former topic, but the compilation algorithm for replace rules (Karttunen, 1995; Karttunen, 1996; Kempe and Karttunen, 1996) differs in many details from the compilation method for rewrite rules described by Kaplan and Kay.

5 Two-Level Implementations

The first implementation (Koskenniemi, 1983) was quickly followed by others. The most influential implementation was by Lauri Karttunen and his students at the University of Texas (Karttunen, 1983; Gajek et al., 1983; Dalrymple et al., 1983). Published accounts of this project inspired many copies and variations, including those by Beesley (Beesley, 1989; Beesley, 1990). A copyrighted but freely distributed implementation of classic Two-Level Morphology, called **PC-KIMMO**, available from the Summer Institute of Linguistics (Antworth, 1990), runs on PCs, Macs and Unix systems.

In Europe, two-level morphological analyzers became a standard component in several large systems for natural language processing such as the British Alvey project (Black et al., 1987; Ritchie et al., 1987; Ritchie et al., 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. The MMORPH morphology tool (Petitpierre and Russel, 1995) built at ISSCO for MULTEXT is now available under GNU Public License.[5]

Some of these systems were implemented in Lisp (Alvey), some in Prolog (CLE, ALEP), some in C (MMORPH). They 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. Another difference is that morphological parsing could be constrained by feature unification. Because the rules are interpreted at runtime and because of the unification overhead, these systems are not very efficient, and two-level morphology acquired, undeservedly, a reputation for being slow. MMORPH solves the speed problem by allowing the user to run the morphology tool off-line to produce a database of fully inflected word forms and their lemmas. 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 compiler for it.

6 Lexical Transducers

The pioneers of finite-state morphology knew well that a cascade of transducers compiled from phonological rewrite rules could be composed into a single one (see Figure 1). It was also known from the beginning that a set of two-level transducers could be merged into a single one (Karttunen, 1983; Koskenniemi, 1983) by intersecting them, as in Figure 8. The problem is that in both cases the resulting single transducer is typically huge compared to the sizes of the original rule networks. Composition and intersection are exponential in the worst case. That is, the number of states in the resulting network may be the product of the sizes of the operands. Although the worst case is purely theoretical, in practice it turned out that intersecting large two-level rule systems was either impossible or impractical on the computers available in the early 90s.

[5] http://packages.debian.org/stable/misc/mmorph.html

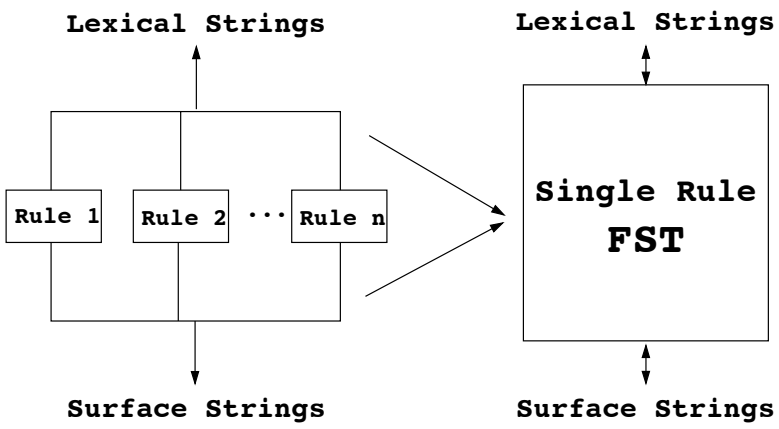

Figure 8: A Set of Two-Level Rules Intersected into a Single **FST**

It was at that time that the researchers at **Xerox** (Karttunen et al., 1992) realized what should have been obvious to them much earlier. The intersection of two-level rules blows up because it constrains the realization of all the strings in the universal language. But in fact we are typically interested only in the strings of a particular language. If the lexicon is composed with the rules, it filters out all the spurious strings. This method had not been tried earlier because it seemed that the composition of a large lexicon with a large rule system would result in something even larger. It soon became evident that the result of composing a source lexicon with an intersected two-level rule system was never significantly larger than the original source lexicon, and typically much smaller than the intersection of the rules by themselves. Figure 9 illustrates this interesting phenomenon.

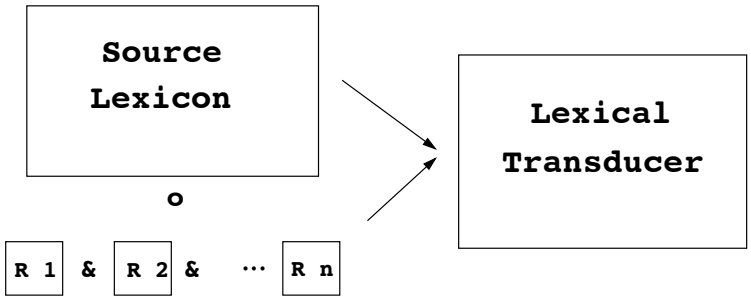

Figure 9: Intersecting and Composing Two-Level Rules with a Lexicon

The resulting single lexical transducer contains all the lexical forms of the source lexicon and all of their proper surface realizations as determined by the rules. The construction of a special INTERSECTING COMPOSITION algorithm made it possible to carry out the construction even in cases where the rules could not be intersected by themselves.

Including the lexicon at compile time obviously brings the same benefit in the case of a cascade of rewrite rules. The result does not grow significantly

if the composition is done "top-down", starting with the lexicon and the first rule of the cascade.

7 The Future

A considerable amount of work has been done, and continues to be done, in the general framework of two-level morphology. However, at **Xerox** the newer **xfst** tool that includes an extended regular expression compiler with replace rules has largely supplanted two-level rules in many applications.

If the lexical forms are very different from the corresponding surface forms, it appears that for most computational linguists a cascade of replace rules is conceptually easier to master than an equivalent set of two-level rules. The ordering of the 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. Finite-state optimality theory can be thought of as yet another way to accomplish this task.

When two-level rules were introduced, the received wisdom was that morphological alternations should be described by a cascade of rewrite-rules. Practitioners of two-level morphology used to write papers pointing out that a two-level account of certain phenomena was no less adequate than a serialist description (Karttunen, 1993). It is interesting to note how linguistic fashions have changed.

From the current point of view, two-level rules have many interesting properties. They are symbol-to-symbol constraints, not string-to-string relations like general rewrite rules. Two-level rules make it possible to directly constrain deletion and epenthesis sites because the zero is an ordinary symbol. Two-level rules enable the linguist to refer to the input and the output context in the same constraint. The only anachronistic feature is that two-level constraints are inviolable. Perhaps we will see in the future a new finite-state formalism with weighted and violable two-level constraints.

All the systems that have been used for the description of morphological alternations, rewrite rules, two-level rules, and optimality constraints are UNIDIRECTIONAL. They have a generative orientation, viewing surface forms as a realization of the corresponding lexical forms, not the other way around. In the two-level formalism, the left-arrow part of a rule such as `N:m <=> _ p:` constrains the realization of a lexical **N**, not the interpretation of a surface **m**. This is an arbitrary choice, it would be easy to have another operator just like `<=>` but oriented in the other direction, constraining the potential lexical sources of a surface **m**. In fact, both types of rules could co-exist within the same BIDIRECTIONAL two-level rule system. This is an interesting possibility, especially for weighted constraints.

References

Antworth, E. L. (1990). *PC-KIMMO: a two-level processor for morphological analysis*. Number 16 in Occasional publications in academic computing. Summer Institute of Linguistics, Dallas.

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

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

Black, A., Ritchie, G., Pulman, S., and Russell, G. (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, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper and Row, New York.

Dalrymple, M., Doron, E., Goggin, J., Goodman, B., and McCarthy, J., editors (1983). *Texas Linguistic Forum, Vol. 22*. Department of Linguistics, The University of Texas at Austin, Austin, TX.

Eisner, J. (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, R. and Satta, G. (1998). Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24(2):307–316.

Gajek, O., Beck, H. T., Elder, D., and Whittemore, G. (1983). LISP implementation. In Dalrymple, M., Doron, E., Goggin, J., Goodman, B., and McCarthy, J., editors, *Texas Linguistic Forum, Vol. 22*, pages 187–202. Department of Linguistics, The University of Texas at Austin, Austin, TX.

Grimley-Evans, E., Kiraz, G. A., and Pulman, S. G. (1996). Compiling a partition-based two-level formalism. In *COLING'96*, Copenhagen. cmp-lg/9605001.

Johnson, C. D. (1972). *Formal Aspects of Phonological Description*. Mouton, The Hague.

Kager, R. (1999). *Optimality Theory*. Cambridge University Press, Cambridge, England.

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

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

Karttunen, L. (1983). KIMMO: a general morphological processor. In Dalrymple, M., Doron, E., Goggin, J., Goodman, B., and McCarthy, J., editors, *Texas Linguistic Forum, Vol. 22*, pages 165–186. Department of Linguistics, The University of Texas at Austin, Austin, TX.

Karttunen, L. (1993). Finite-state constraints. In Goldsmith, J., editor, *The Last Phonological Rule*. University of Chicago Press, Chicago, Illinois.

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

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

Karttunen, L. (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, L. and Beesley, K. R. (1992). Two-level rule compiler. Technical Report ISTL-92-2, Xerox Palo Alto Research Center, Palo Alto, CA.

Karttunen, L., Kaplan, R. M., and Zaenen, A. (1992). Two-level morphology with composition. In *COLING'92*, pages 141–148, Nantes, France.

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

Karttunen, L., Uszkoreit, H., and Root, R. (1981). Morphological analysis of Finnish by computer. In *Proceedings of the 71st Annual Meeting of SASS*. Albuquerque, New Mexico.

Kempe, A. and Karttunen, L. (1996). Parallel replacement in finite-state calculus. In *COLING'96*, Copenhagen. cmp-lg/9607007.

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

McCarthy, J. J. (2002). *The Foundations of Optimality Theory*. Cambridge University Press, Cambridge, England.

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

Petitpierre, D. and Russel, G. (1995). MMORPH — the multext morphology program. Technical report, Carouge, Switzerland. Multext Deliverable 2.3.1.

Prince, A. and Smolensky, P. (1993). Optimality Theory: Constraint interaction in generative grammar. Technical report, Rutgers University, Piscataway, NJ. RuCCS Technical Report 2. Rutgers Center for Cognitive Science.

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

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

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

Ruessink, H. (1989). Two level formalisms. In *Utrecht Working Papers in NLP*, volume 5.

Schützenberger, M.-P. (1961). A remark on finite transducers. *Information and Control*, 4:185–196.