17

Simplified Two-level Morphophonology

Kimmo Koskenniemi

17.1 Introduction

Writing two-level grammar rules and entire two-level grammars is sometimes consider difficult as in (Beesley and Karttunen, 2003, pp. 12–13):

> TWOLC rules conceptually apply simultaneously; this avoids the ordering problems but means that sets of twolc rules must be carefully written to avoid nasty and often mysterious conflicts among rules. Correct, non-conflicting TWOLC rule sets are notoriously difficult to write when some of the rules perform deletion or, especially, epenthesis.

This paper elaborates a simplified version of the two-level model which appears to avoid most of the sources of the complexity that some users of the standard model have experienced.[1] In the simplified model, separate morphophonemes are used more freely for alternations which have different causes or patterns. Generally, there is less need for creative or artistic solutions when using the new model. A technical difference is that in the present model, all rules are strictly conjunctive as they were in Koskenniemi (1983). Rules are also usually written for each morphophoneme separately.

In the simplified model, some tasks can be done automatically for the linguist. It is claimed here that building morphophonological grammars

[1] An earlier version of the simplified two-level morphology was presented in Koskenniemi (2013b). That version contained some of the features presented in this paper.

Tokens of Meaning: Papers in Honor of Lauri Karttunen.
Cleo Condoravdi and Tracy Holloway King (eds.).

is easier when using the simplified model. Even if one needs more rules, each rule is simpler.

The simplified model is expected to be best suited for languages which have morphophonemic alternations and use a phonemic or near-phonemic alphabet. Therefore, the terms 'letter' and 'phoneme' are sometimes used interchangeably in this paper.

The proposed method consists of a number of steps:

1. A linguist collects a relevant set of examples of inflected word forms from a grammar or a dictionary and lays them out as a table together with some grammatical information. The word forms that are collected are plain surface forms, see Section 17.2.
2. The linguist inserts boundary symbols manually between the morphs in the word forms, see Section 17.2.
3. An algorithm aligns the morphs belonging to each morpheme by inserting some zero symbols (Ø) as necessary. The letters in corresponding positions of the aligned allomorphs form the raw morphophonemes, see Section 17.3.
4. The linguist labels the raw morphophonemes implied by the alignment. Many raw morphophonemes are usable as morphophonemes as such. Some raw morphophonemes may be so similar that they can be unified by giving them the same label. In other cases, somewhat similar raw morphophonemes are better kept apart because the triggering conditions or alternation patterns are quite different, see Section 17.4.
5. The lexical representations of morphemes are mechanically established according to the alignment and the labeling of the morphophonemes,[2] see Section 17.4.
6. A set of positive examples is produced mechanically by combining the lexical representations and the aligned examples, see Section 17.4.
7. A tentative raw two-level rule is computed for each morphophoneme by using the set of examples and a set of negative examples produced for that morphophoneme. Such raw rules always work correctly with the given examples, but they may not be general enough to handle all other words and forms correctly, see Section 17.5.

[2]In (Beesley and Karttunen, 2003, p. 21) the importance of first setting up examples and only thereafter writing the rules is pointed out. It is not easy to establish the lexical representations in advance. The method presented here provides a method and guidelines for finding useful lexical representations in an objective manner.

8. Raw rules are refined into final rules by performing the following steps separately for each morphophoneme as defined above, see Section 17.6:
   (a) Tune the raw rule for this morphophoneme by generalizing its contexts as appropriate to simplify the rule.
   (b) Compile the rule into a finite-state transducer using a rule compiler and let a program check whether this rule accepts all positive examples.
   (c) Let a program produce the set of negative examples for this morphophoneme and check whether this rule rejects all these negative examples.
   (d) Proceed to the next rule if checks were successful. Improve the rule if some checks failed. Repeat with a revised version of the rule.

17.2 Examples and Morph Boundaries

Grammars and dictionaries often describe the inflection of words (i.e. declension and conjugation) by defining a number of classes of lexemes so that within each class the words are inflected in a similar way. The classes are presented using a selection of characteristic inflectional forms (so called principal parts) which determine the rest of the forms. In this way, all different forms of the stems and the affixes (i.e. their allomorphs) are explicitly included in the examples. Examples in this article are partly from the Reverse Dictionary of Modern Standard Finnish Tuomi (1980).

The information given in such sources is usually compact and useful. Such lists and tables are available for many languages which have a grammar or a dictionary, and they provide good starting point for building a morphophonological analyzer. Of course, one has to be prepared to augment the set of examples in order to cover all relevant morphophonemic alternations.

The linguist has to add some information into the plain example word forms in order to make the material suitable for the present method. Firstly, one must indicate what lexemes and what grammatical forms each example consists of. One could produce a two-column list where the left column has the inflected word form and the right column lists some names identifying the morphemes. Secondly, one must mark the boundaries between the morphs, e.g. as in Table 1.[3] Note

[3]Methods for automatically finding the places for the boundaries has been discussed at least in Theron and Cloete (1997). It is an interesting topic and would deserve more attention. The assumption that the boundaries are already placed makes the process much simpler.

katu	KATU
kadu+lla	KATU+ADE
kadu+i+lla	KATU+PL+ADE
katu+j+en	KATU+PL+GEN

TABLE 1 Segmented word forms with morpheme identifiers.

that only the morphemes that are actually present in the examples are segmented and listed by their name. In the examples, there is no overt morpheme for singular, thus neither a boundary nor an identifier for singular number is needed.

The dictionaries often give the inflectional information as a table so that each row represents a lexeme and each column represents a form. So, each cell in the table has one stem morph for that row and inflectional morphs for that column. Table 2 presents a small table of that type with appropriate column and row labels.

ID	STEM	STEM+ADE	STEM+PL+ADE	STEM+PL+GEN
KATU	katu	kadu+lla	kadu+i+lla	katu+j+en
HIIRI	hiiri	hiire+llä	hiir+i+llä	hiir+i+en
NIEMI	niemi	nieme+llä	niem+i+llä	niem+i+en
KALA	kala	kala+lla	kalo+i+lla	kalo+j+en
RIITA	riita	riida+lla	riido+i+lla	riito+j+en

TABLE 2 Example words as a table

One must include relevant examples of the phenomena to be described. Otherwise, it is unlikely that the rules will perform correctly when applied to further data. For Finnish, one ought to include example words not only for each inflectional class but also take care that there are examples of different types of consonant gradation and examples of stems and endings with front and back harmony. Typically, a few hundred examples are needed.

17.3 Phoneme by Phoneme Alignment of Allomorphs

From the segmented examples, one can automatically collect the different allomorphs (morphs representing the same morpheme). E.g. *katu*, *kadu* are the two stem allomorphs of the lexeme *KATU* and *lla* and *llä* are the allomorphs of *ADE*. In this step, the morphs for each morpheme will be aligned by adding zero symbols *Ø* where that is necessary or desirable to make the phonemes in corresponding positions match properly. It is assumed that consonant may not be aligned with vowels or vice versa, except that semivowels may correspond to some consonants

or vowels. Otherwise, identical phonemes match perfectly, and similar ones better than those differing in several features.

One may align the allomorphs of each stem and each affix using a finite-state algorithm which weights the similarity of different phonemes and the inserted zeros, and finds the optimal places where to insert the zeros, see e.g. Koskenniemi (2017) for the method and an experiment with aligning Estonian noun stems. As a result, we have the morphs aligned letter by letter so that the correspondences of letters in the same position is optimal in some sense. From Table 2 we would get the morphs for each morpheme as in Table 3

KATU:	katu, kadu
HIIRI:	hiiri, hiire, hiirØ
NIEMI:	niemi, nieme, niemØ
KALA:	kala, kalo
RIITA:	riita, riida, riido
ADE:	lla, llä
PL:	i, j
GEN:	en

TABLE 3 Aligned morphs of the morphemes.

The linguist might check the result of the alignment in order to find mistakes in the examples or in the positioning of the boundaries. Boundaries in wrong positions would show up as excessive zeros in the alignment.

17.4 Morphophonemic Representations

The alignment was made so that the phonemes in each position correspond to each other as closely as possible, e.g. for KATU:

```
k a t u
k a d u
```

The first, second and fourth phonemes are identical in the two morphs. The third position contains t and d and gives rise for a morphophoneme. The only task here is to give a name for the morphophoneme, and here we call it {td}. Similarly, the illative morpheme ILL consists of aligned morphs where the third letter is either a or ä which is a morphophoneme which we decide to denote with {aä}. After such decisions, we have morphophonemic representations for all

morphemes:

```
KATU:  k a {td} u
HIIRI: h i i r {ieØ}
NIEMI: n i e m {ieØ}
KALA:  k a l {ao}
RIITA: r i i {td} {ao}
ADE:   l l {aä}
PL:    {ij}
GEN:   e n
```

After the alignment and the adding of some zero symbols, our examples consist of equal length morphemes. Thus, we can combine the aligned examples with the morphophonemic representations and get examples which consist of pairs, where each of them consists of either identical phonemes or a morphophoneme and a phoneme as in Table 4. Note that pairs of identical symbols are abbreviated, e.g. *k* stands for *k:k*.

```
k a {td}:t u
k a {td}:d u l l {aä}:a
k a {td}:d u {ij}:i l l {aä}:a
k a {td}:t u {ij}:j e n
h i i r {ieØ}:i
h i i r {ieØ}:e l l {aä}:ä
h i i r {ieØ}:Ø {ij}:i l l {aä}:ä
h i i r {ieØ}:Ø {ij}:i e n
n i e m {ieØ}:i
n i e m {ieØ}:e l l {aä}:ä
n i e m {ieØ}:Ø {ij}:i l l {aä}:ä
n i e m {ieØ}:Ø {ij}:i e n
k a l {ao}:a
k a l {ao}:a l l {aä}:a
k a l {ao}:o {ij}:i l l {aä}:a
k a l {ao}:o {ij}:j e n
r i i {td}:t {ao}:a
r i i {td}:d {ao}:a l l {aä}:a
r i i {td}:d {ao}:o {ij}:i l l {aä}:a
r i i {td}:t {ao}:o {ij}:j e n
```

TABLE 4 Examples as sequences of pairs.

17.5 Raw Rules

The method presented here resembles somewhat that of Theron and Cloete (1997). In that article, there is a brief survey of other relevant approaches, and see also (Koskenniemi, 2013a, pp. 159–161) for a short survey of discovering morphophonological rules.

The examples which have been augmented with morphophonemes are positive examples which all rules must accepted. For each morphophoneme-symbol pair, one can start with a set of rules where each example (where that pair occurs) forms a context. A program then checks step by step whether the contexts can be truncated by one or more symbols without any loss of accuracy within the set of examples.

Rules with truncated context still accept the positive examples, but they might fail to reject some cases which they are expected to reject. A good approximation for the negative examples for a morphophoneme can be produced from the original examples by substituting each occurrence of the morphophoneme by all possible pairs of the morphophoneme and subtracting the original examples from these corrupted examples. Suppose we are working on a morphophoneme {*ao*} which can correspond to *a* or *o*, and there are (positive) examples as in Table 5.

```
k a l {ao}:a
k a l {ao}:a l l {aä}:a
k a l {ao}:o {ij}:i l l {aä}:a
k a l {ao}:o {ij}:j e n
r i i {td}:t {ao}:a
r i i {td}:d {ao}:a l l {aä}:a
r i i {td}:d {ao}:o {ij}:i l l {aä}:a
r i i {td}:t {ao}:o {ij}:j e n
```

TABLE 5 Positive examples for {*ao*}.

The program starts with rules as in Table 6.

```
{ao}:a =>
      .#. k a l _ .#. ;
      .#. k a l _ l l {aä}:a .#. ;
      .#. r i i {td}:t _ .#. ;
      .#. r i i {td}:d _ l l {aä}:a .#. ;
{ao}:o =>
      .#. k a l _ {ij}:i l l {aä}:a .#. ;
      .#. k a l _ {ij}:j e n
      .#. r i i {td}:d _ {ij}:i l l {aä}:a .#. ;
      .#. r i i {td}:t _ {ij}:j e n
```

TABLE 6 Initial raw rules.

These rules clearly accept all positive examples but they would not accept any other words with the same inflection classes. Thus, the program tries to truncate the context parts in order to make the rules more general. In order to constrain the truncation, the program builds a set of negative examples for this morphophoneme. First, it creates all variations of the correspondences of the current morphophoneme as in Table 7.[4]

When the original correct examples are subtracted, the incorrect examples remain as in Table 8.

The program tries to truncate the left context parts which initially have a maximum length of four. By shortening them to three, two, one and zero lengths does not change the fact that even the truncated rules reject all of the negative examples of {*ao*}. Thus, we get a somewhat simpler raw rule as in Table 9.

We can eliminate identical copies of contexts and simplify the rule further as in Table 10.

When trying to truncate the right hand contexts, we notice that reducing the length to three, two and finally one, all negative examples are still rejected. There the program[5] stops and produces the following

[4]One might consider corrupting the correspondences for all morphophonemes, not only those of the current morphophoneme. A problem in that approach is that the raw rules for this morphophoneme would not discard all bad examples even if some other raw rule would. In that setup, the raw rules could not be processed separately.

[5]Producing the raw rules is simple, if we have the examples available in their morphophonemic form. No finite-state technology is necessary. One may represent both the positive and negative contexts as pairs of strings. Accepting negative examples is manifested as non-empty intersections of sets of truncated positive and negative contexts.

```
k a l {ao}:a
k a l {ao}:o
k a l {ao}:a l l {aä}:a
k a l {ao}:o l l {aä}:a
k a l {ao}:o {ij}:i l l {aä}:a
k a l {ao}:a {ij}:i l l {aä}:a
k a l {ao}:o {ij}:j e n
k a l {ao}:a {ij}:j e n
r i i {td}:t {ao}:a
r i i {td}:t {ao}:o
r i i {td}:d {ao}:a l l {aä}:a
r i i {td}:d {ao}:o l l {aä}:a
r i i {td}:d {ao}:o {ij}:i l l {aä}:a
r i i {td}:d {ao}:a {ij}:i l l {aä}:a
r i i {td}:t {ao}:o {ij}:j e n
r i i {td}:t {ao}:a {ij}:j e n
```

TABLE 7 Example with corrupted {*ao*}.

```
k a l {ao}:o
k a l {ao}:o l l {aä}:a
k a l {ao}:a {ij}:i l l {aä}:a
k a l {ao}:a {ij}:j e n
r i i {td}:t {ao}:o
r i i {td}:d {ao}:o l l {aä}:a
r i i {td}:d {ao}:a {ij}:i l l {aä}:a
r i i {td}:t {ao}:a {ij}:j e n
```

TABLE 8 True negative examples for {*ao*}.

```
{ao}:a =>
      _ .#. ;
      _ l l {aä}:a .#. ;
      _ .#. ;
      _ l l {aä}:a .#. ;
{ao}:o =>
      _ {ij}:i l l {aä}:a .#. ;
      _ {ij}:j e n
      _ {ij}:i l l {aä}:a .#. ;
      _ {ij}:j e n
```

TABLE 9 Left truncated raw rules for {*ao*}.

```
{ao}:a =>
      _ .#. ;
      _ l l {aä}:a .#. ;
{ao}:o =>
      _ {ij}:i l l {aä}:a .#. ;
      _ {ij}:j e n
```

TABLE 10 Left truncated raw rules for {*ao*} without duplicate contexts.

raw rules as in Table 11.

```
{ao}:a =>
      _ .#. ;
      _ l ;
{ao}:o =>
      _ {ij}:i ;
      _ {ij}:j ;
```

TABLE 11 Final raw rules {*ao*}.

One can readily see that the raw rules we get depend on the examples we have selected. If we would have had more case endings, we would have had more context parts in the first raw rule. The raw rules are not perfect but they serve as a useful starting points for refining them into a general and correct final rules.

The procedure succeeded well in the above example. Some other morphophonemes end up having plenty of long contexts. E.g. the raw rule for vowel harmony lists almost the full left context of each word where the morphophoneme occurs. Some linguistic knowledge is needed for refining such rules.

If the morphophoneme has free variation, one ought to prepare the collection of examples so that it contains examples of both variants. Otherwise the program cannot find useful raw rules. In general, free variation or optional alternations cause no problems when producing the raw rules. Raw rules are all right-arrow rules and in such cases, their contexts just happen overlap.

No raw left-arrow rules are proposed. Instead, right-arrow rules are proposed for all realizations of each morphophoneme. Often one of those right-arrow rules has an apparently complex contexts which often turns out to be a complement of the other context (or the union of other contexts). Note that in traditional two-level grammars, one uses mostly double-arrow rules.

17.6 Refining Raw Rules into Two-level Rules

The strategy in refining the raw rules is to generalize some contexts to accept also other similar examples. Each raw context consists of the left context and the right context and initially both of these parts consist of sequences of symbol pairs. The first step is to generalize the context by replacing some symbol pairs with expressions which stand for several distinct pairs of symbols as long as the rule rejects all negative examples. A common expression can represent a set of symbol pairs e.g. by using brackets parenthesis and vertical bars, e.g. [l|n|s|{Øt}:t]. Typically, vowels are combined with vowels and consonants with consonants. Generalizing one context may make it include some other contexts. Such other context are obviously redundant and they can be discarded.

The alphabets or lexical and surface characters and their feasible pairs are not explicitly declared in the present approach. The examples define them implicitly. Let *Pair* denote the expression which allows any of those feasible symbol pairs (but just one). In two-level rule expressions a colon *:* (separated by space from neighboring symbols) denotes *Pair*.

Quite often, one can generalize a symbol pair, e.g. {*ao*}*:a* into an expression that corresponds to all pairs which have the second symbol i.e. *a* as their second part. Let *:a* denote the set of such pairs and the set is defined as the composition *Pair .o. a*. Similarly, {*aä*}*:* denotes the set of all pairs which have e.g. {*aä*} as their first part and the set is defined as {*aä*} *.o. Pair*.

With a slightly larger set of examples one could get the following two raw rules:

```
{ao}:a =>
      _ {Øt}:Ø ;
      _ .#. ;
      _ {Øh}:Ø ;
      _ {Øt}:t ;
      _ {i}:i ;
{ao}:o =>
      _ {ij}:i ;
      _ {ij}:j ;
```

One could generalize the second rule into the following by substituting {*ij*}*:i* and {*ij*}*:j* both with {*ij*}*:* and removing the second context as

an identical copy of the first:

```
{ao}:o => _ {ij}: ;
```

Because this context is not overlapping with the contexts in the first raw rule and there are just these two pairs, one can generalize the contexts of the first rule so that {*ao*}:*a* can occur anywhere except before {*ij*}::

```
{ao}:a /<= _ {ij}: ;
```

Together these two refined sub-rules results in a final double-arrow rule:

```
{ao}:o <=> _ {ij}: ;
```

Let us study a slightly more complicated raw rule which accounts for the form of the plural ending:

```
{ij}:j =>
    e _ e ;
    e _ {Øt}:Ø ;
    {iieØ}:e _ {Øt}:Ø ;
    {iiØeØ}:e _ {Øt}:Ø ;
    {Øie Ø}:e _ {Øt}:Ø ;
    {ao}:o _ {Øt}:Ø ;
    o _ e ;
    {ao}:o _ e ;
    o _ {Øt}:Ø ;
    {aoØ}:o _ {Øt}:Ø ;
    u _ e ;
    u _ {Øt}:Ø ;
```

One could substitute the pairs with a vowel with surface vowels and remove duplicate contexts:

```
{ij}:j =>
    :e _ :e ;
    :e _ {Øt}:Ø ;
    :o _ :e ;
    :o _ {Øt}:Ø ;
    :u _ :e ;
    :u _ {Øt}:Ø ;
```

From the symbols of our original examples we have a vowels $Vowel = [a \mid e \mid i \mid \ldots]$, consonants $Consonant = [b \mid c \mid d \mid f \mid \ldots]$ and semivowels, e.g. $Glide = [j \mid \ldots]$. Two-level rules operate in the domain of symbol pairs. Therefore all relevant sets must be expressed using symbol pairs and as regular expressions which construct sets out of

elementary pairs or other sets. As opposed to the traditional two-level formalism, the simplified model uses no definitions for the alphabet nor any character sets.

The the content of following sets are implicitly defined by the collection of examples:

1. The set *SurfV* consisting of symbol pairs that are called *surface vowels* is defined by the composition: *[[:] .o. Vowel]* (where *.o.* stands for composition operator as in the XFST regular expressions). Note that *Ø* was not a vowel and therefore pairs such as *{ieØ}:Ø* do not belong to *SurfV*.
2. The set *SurfC* consisting of symbol pairs that are called *surface consonants* is the composition: *[[:] .o. Consonant]*.
3. The set *Cons* consisting of symbol pairs which are called *morphophonemic or underlying consonants* is defined as *[SurfC.u .o. [:]]* (where *.u* is the unary operator for the input, upper or left side of the relation or transducer).

Using such sets one can still generalize the above rule without losing any accuracy:

```
{ij}:j =>
    SurfV _ SurfV ;
    SurfV _ {Øt}:Ø ;
```

This is how far one can go blindly without looking the examples and using some linguistic knowledge. One notes that *{tØ}:Ø* always precedes a vowel. So the final form for the rule could be:

```
{ij}:j => SurfV _ (:Ø) SurfV ;
```

17.7 Rule Compilation

Simplified two-level grammars can be compiled using the Xerox TWOLC or HFST-TWOLC rule compiler. The larger number of morphophonemes causes some more work when maintaining various sets and keeping them consistent with the increasing set of examples. Writing yet another two-level rule compiler is not difficult any more, especially because the finite-state calculus can now be used also from higher level languages like Python. In order to integrate the rule compiler with the processing of positive and negative examples the author wrote an entirely new two-level rule compiler using the formulas presented in Yli-Jyrä and Koskenniemi (2006). The linguist can compile and test single rules (or bunches of rules) against the examples and get immediate feedback after the compilation of each rule. The compiler reports any rejected positive examples or accepted negative ones.

The compiler is simpler than the Xerox TWOLC or HFST-TWOLC by omitting definitions of the alphabet and character sets. Only definitions are used. Four types of rules (=>, <=, <=> and / <=) are supported but neither *where* clauses nor conflict resolution is included. They are not needed when following the principles of simplified two-level grammars where one proceeds morphophoneme by morphophoneme and each morphophoneme gets its own rule(s).

One could expect that with such tools, even less experienced linguists could have a fairly smooth start and a reasonable learning curve when writing comprehensive morphological analyzers.

17.8 Testing Rules

Not only the writing of two-level rules is claimed to be difficult. Also the testing of them has been considered difficult: (Beesley and Karttunen, 2003, p. 13):

> A system of replace rules is relatively easy to check and modify because each rule can be applied individually. The output from one rule can be typed as input to the next rule and the effect of a whole cascade of replace rules can be checked step by step. When we move to TWOLC rules, however, the semantics of the rules demand that we conceive of them always as simultaneously applied constraints on the relation between the lexical language and the surface language. Because twolc rules are designed to apply in parallel, it is difficult to test them individually.

In the simplified model, the compilation of a rule and its testing are integrated so that after the compilation of a rule, the program tests the rule against all positive examples as a FST, then builds a FST which corresponds to the negative examples for this rule, and finally tests those against the compiled rule.

One can see that corrupting one morphophoneme at a time when producing negative examples might not be sufficient. Let us, thus, assume that the set of examples contains all possible surface correspondences that can be correctly generated out of the lexical representations of the examples. Under this assumption, all multiply corrupted negative examples will be rejected by some rule, i.e. if not a rule for every morphophoneme, anyway at least by some. Building such a multiply corrupted set of negative examples with finite-state calculus is easy, and the test can be made by intersecting that FST with all rules. Admittedly, this final check can be made only after we have all rules available.

There is also another way of making this total test. One can extract the lexical side of the examples and compose-intersect it with the combination of all rules. The surface side of that operation ought to be

equivalent with the surface side of the original examples.

One can reduce the need for such testing of all rules simultaneously by choosing context expressions which refer to the lexical symbols alone, or sets which can be expressed as unions of such, e.g. use *Cons* rather than *SurfC*.

Epenthesis does not exist in a technical sense within simplified two-level grammars. The lexical representations never contain plain zeros. Where a linguist sees epenthesis, the method sees an alternation between zero and e.g. *e*. This ends up as a morphophoneme {*Øe*} which must be present in the lexicon entry of that morpheme.

One may speculate that the most crucial part of building a two-level rule grammar will be the selection and preparation of relevant examples. While a set of examples is established, the later steps such as the alignment and building raw and refined rules probably are fast and easy but they are likely to trigger revisions in the set of examples and in the positioning of the morph boundaries.

Moving any boundaries slightly left or right might cause significant changes in the morphophonemes and the results. Similarly important are the initial decisions about grammatical morphemes, e.g. whether one postulates one affix for a grammatical form or perhaps several if the morphs are substantially different. Comprehensive testing is probably needed in order to find guidelines for such decisions.

17.9 Future

One could establish a measure of the goodness of a collection of morphophonemes as they are produced by the insertion of the morph boundaries. This measure would evaluate the success of those steps. One could test different choices for the boundary positions and choose the best of them. Ultimately, this might lead to the automation of that step.

One could establish a measure of the complexity of individual rules on the basis of the number of contexts, the lengths of the context and naturalness of the sets present in the contexts. A measure could be used for guiding the refinement of raw rules more or less automatically. The total sum of such measures could be used as a further measure of the success of the whole grammar. It could also be used for deciding between the earlier alternative steps of the procedure including the naming or merging of morphophonemes.

A permissive full scale morphological analyzer for Early Modern Standard Finnish ought to be built. Such an analyzer could be applied to classical literature and 19th century newspapers etc. In addition and more importantly, it could be used as a component in comparing mod-

ern language and historical texts, and also when comparing Finnish with other Uralic languages, cf. Koskenniemi (2013a).

The simplified two-level model could be applied to build a morphological analyzer for Inari Saami, a language with particularly complex morphophonemics including vowel and consonant quantity alternations.

References

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Two-Level Rule Compiler*. Xerox PARC. https://web.stanford.edu/~laurik/.book2software/.

Koskenniemi, Kimmo. 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production*. No. 11 in Publications. University of Helsinki, Department of General Linguistics.

Koskenniemi, Kimmo. 2013a. Finite-state relations between two historically closely related languages. In *Proceedings of the workshop on computational historical linguistics at NODALIDA 2013; May 22-24; 2013; Oslo; Norway*, no. 87 in NEALT Proceedings Series 18, pages 53–53. Linköping University Electronic Press; Linköpings universitet.

Koskenniemi, Kimmo. 2013b. An informal discovery procedure for two-level rules. *Journal of Language Modelling* 1(1):155–188.

Koskenniemi, Kimmo. 2017. Aligning phonemes using finite-state methods. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 56–64. Gothenburg, Sweden: Association for Computational Linguistics.

Theron, Pieter and Ian Cloete. 1997. Automatic acquisition of two-level morphological rules. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, ANLC '97, pages 103–110. Stroudsburg, PA, USA: Association for Computational Linguistics.

Tuomi, Tuomo. 1980. *Suomen kielen käänteissanakirja / Reverse Dictionary of Modern Standard Finnish*. No. 274 in Toimituksia. Suomalaisen Kirjallisuuden Seura, 2nd edn.

Yli-Jyrä, Anssi and Kimmo Koskenniemi. 2006. Compiling generalized two-level rules and grammars. In T. Salakoski, F. Ginter, S. Pyysalo, and T. Pahikkala, eds., *Advances in Natural Language Processing, Proceedings of the 5th International Conference on NLP, FinTAL 2006, Turku, Finland, August 2006*, vol. 4139 of *Lecture Notes in Computer Science*, pages 174–185. Springer.