Guessing lexicon entries using finite-state methods

Koskenniemi, Kimmo Matti

Association for Computational Linguistics
2018-01

Koskenniemi , K M 2018 , Guessing lexicon entries using finite-state methods . in T Pirinen , M Rießler , J Rueter , T Trosterud & F M Tyers (eds) , Proceedings of the Fourth International Workshop on Computatinal Linguistics for Uralic Languages . Association for Computational Linguistics , Stroudsburg , pp. 59-77 , International Workshop for Computational Linguistics of Uralic Languages , Helsinki , Finland , 08/01/2018 .

http://hdl.handle.net/10138/233853

Guessing lexicon entries using finite-state methods

Kimmo Koskenniemi
University of Helsinki
Department of Modern Languages
kimmo.koskenniemi@helsinki.fi

Abstract

A practical method for interactive guessing of LEXC lexicon entries is presented. The method is based on describing groups of similarly inflected words using regular expressions. The patterns are compiled into a finite-state transducer (FST) which maps any word form into the possible LEXC lexicon entries which could generate it. The same FST can be used (1) for converting conventional headword lists into LEXC entries, (2) for interactive guessing of entries, (3) for corpus-assisted interactive guessing and (4) guessing entries from corpora. A method of representing affixes as a table is presented as well how the tables can be converted into LEXC format for several different purposes including morphological analysis and entry guessing. The method has been implemented using the HFST finite-state transducer tools and its Python embedding plus a number of small Python scripts for conversions. The method is tested with a near complete implementation of Finnish verbs. An experiment of generating Finnish verb entries out of corpus data is also described as well as a creation of a full-scale analyzer for Finnish verbs using the conversion patterns.

Tiivistelmä

Artikkelissa esitellään menetelmä, jonka avulla käyttäjä voi määrittää LEXC-leksikkoon sopivia uusia hakusanoja. Menetelmässä kuvataan kukin taivutusluokka säännöllisten lausekkeiden avulla. Samoja lausekkeita voidaan käyttää toisaalta tavanomaisen sanakirjan sanaluettelon konversioon ja toisaalta yksittäisten hakusanojen määrittämiseen siten, että käyttäjä antaa haluamansa hakusanan eri muotoja, kunnes hakusana on yksiselitteisesti määrätty. Arvaaminen voidaan suorittaa myös korpuksista kerättyjen tietojen avulla, jolloin oikea hakusana löytyy nopeammin. Myös pelkän sanalistan perusteella voidaan arvata hakusanoja. Menetelmä on toteutettu käyttäen HFST:n äärellistilaisten transduktorien työkaluja ja erityisesti käyttäen niitä Python-ohjelmointikielestä käsin. Lisäksi on tehty muutamia lyhyitä Python-skriptejä, joilla tietoja muunnetaan eri muodoista toisiinsa. Menetelmää on testattu soveltamalla sitä lähes kattavaan suomen kielen verbien taivutusmalliin. Menetelmiä on kokeiltu alustavasti toisaalta hakusanojen automaattiseksi muodostamiseksi tekstikorpuksen sanalistasta ja toisaalta täysimittaisen suomen kielen verbien morfologisen jäsentimen muodossa.

Proceedings of the 4th International Workshop for Computational Linguistics for Uralic Languages (IWCLUL 2018), pages 59–77,
Helsinki, Finland, January 8–9, 2018. ©2018 Association for Computational Linguistics

1 Introduction

Creating lexical entries is an important and time consuming task for any language. For lesser resourced languages with a rich morphology the task is particularly relevant. Building a lexicon requires often not only plenty of time and labour but also specific training. Thus, there is an obvious need for automating this task.

This paper describes the process of generating entries for computational morphological analysis in the framework of finite-state morphology and it uses the concepts of the Xerox/HFST LEXC lexicons, for more information see (Beesley and Karttunen, 2003). Inflection classes (the declinations and conjugations) refer to traditional dictionaries, where the inflection of lexemes is characterized by model words and numbers or other identifiers referring to those model words. Dictionaries often list many more inflection classes than there are different types of LEXC entries. LEXC can generalize the entries by relying on TWOLC or XFST rules which take care of the regular differences in the shapes of stems.

Several topics are discussed in this paper, including:

- How to describe inflection classes with regular expression patterns, i.e. how to formalize what kinds of syllable structures and phonological alternations are characteristic to each inflection class.

- How the regular expressions can be used for converting dictionary word lists with inflection class codes into lexical entries of a LEXC lexicon.

- How to reuse the same regular expressions for guessing all possible LEXC entries for a given single inflected word form.

- How to use such a mapping for selecting the correct LEXC lexicon entry by prompting the user for further forms of the same lexeme.

- How the same data for affixes and their sequencing can be reused for building ordinary morphological analyzers, lexicon converters and entry guessing.

- How to use the mapping for guessing in order to automatically deduce entries out of a corpus.

The idea here is to build a finite-state model of inflecting unknown lexemes roughly as was proposed by Ken Beesley and Lauri Karttunen (2003). For Finnish, their model could produce the following two results for a Finnish word form *puramme* ('we unpack', 'we disassemble'):

```
puramme --> purkaa V+PRES+ACT+PE2
            puraa+V+PRES+ACT+PE2
```

The first result would be the correct analysis and the second analysis proposes a nonexistent lexeme. But actually both results are ambiguous because neither of them tells how the lexemes are inflected. The final *a* of the stem has two possibilities in both results: either it alternates with *o* or it disappears in past tense. Thus there are four possible entries behind the analysis. One of the hidden entries is what we want.

For the purposes of lexicon entry guessing, we need an equally general method which is prepared to accept almost any surface word but would output lexical entries instead of the base forms and morphosyntactic features. A lexical entry consist of a lexical representation and a name of a continuation sub-lexicon. The lexical representation consist of phonemes and morphophonemes (here we always use braces for morphophonemes, e.g. *{aoe}*).[1] The name of a continuation sub-lexicon (e.g. */v*) determines the set of possible endins and possible less regular pieces of the stems. The mapping we are building could map e.g.:

```
puramme --> pur{kØ}{aoe} /v
            pur{kØ}{aØe} /v
            pur{aoe} /v
            pur{aØe} /v
```

The program would then prompt the user for further inflected forms of the same lexeme. In this way the user can soon narrow down the possibilities to the desired single lexicon entry without any detailed knowledge of the codes or conventions of the lexicon.

The work is done in the finite-state two-level framework in the spirit of the original (Koskenniemi, 1983) version and in particular the so called simplified two-level model as presented in (Koskenniemi, 2013b). Helsinki Finite-State Transducer Tools (HFST) were used for the implementations of the finite-state transducers (FST) described in this paper, for more information on HFST see (Lindén et al., 2011) and various sites in the net, e.g. `http://hfst.github.io`.

2 Previous and related research

The interactive method for guessing presented here was inspired by Aarne Ranta's Grammatical Framework (GF) system where a similar functionality was implemented, see (Ranta, 2011) and (Détrez and Ranta, 2012). They presented so called *smart paradigms* which have been implemented in GF. Smart paradigms perform a mapping that is similar to the mapping described in this paper but do it in a different way.

Several other approaches have been proposed for the assisting or automating entry generation. Beesley and Karttunen (2003) presented a way to recognize unknown words using regular expressions in a LEXC lexicon as was mentioned above, and in this way cover an inflection class by each expression. The present paper elaborates this approach further and explains how one can generate such expressions in an principled way and how to connect the mechanism into the LEXC lexicon of normal morphological analysis and how to use such a generator in practice.

Huldén (2014) and Ahlberg et al. (2015) discuss how paradigms or inflectional tables can be used for finding or forming entries which is a topic beyond the scope of this paper where the inflection is assumed to be already known.

A recent paper (Esplà-Gomis et al., 2017) presents methods for a task quite relevant to that of this paper. In those papers, inflection classes (i.e. declinations and conjugations) are considered to consist of a set of affixes which are directly concatenated with the single stem of the lexeme. In contrast to this, GF is prepared to have lexemes with several stems, and so does the present approach. In addition, the present approach uses morphophonemes in order to describe regular variations within stems, and therefore a very small number of distinct classes is needed. In some languages,

[1] This morphophoneme indicates that in that position there may be either an *a* or an *o* or an *e* depending on the context. Inflection classes usually determine what kinds of phoneme alternations are present in lexemes in that class and what sets of affixes can be attached to them.

e.g. several Sami languages, a large portion of inflection is represented as stem alternations instead of and in addition to using suffixes. The approach presented in this paper is intended to be applicable even to languages with such characteristics. (Esplà-Gomis et al., 2017) present also methods for for optimizing the yes/no queries for the user. These or similar methods could be applied on top of the solutions in this paper but that is not discussed in this paper.

3 Regular expressions for inflection classes

In order to generate lexical entries interactively or from comprehensive word lists, we construct a model which characterizes the inflection classes by describing the common features and alternations in each class. It will be shown that with a single description, one one may solve two tasks : (1) converting a dictionary word list with base forms and class numbers into LEXC lexical entries and (2) guessing LEXC lexical entries out of inflected word forms as was discussed above.

The first mapping transforms dictionary headwords and their inflection class codes into LEXC entries (as sequences of symbols) e.g.:

```
p u r k a a V02*  -->  p u r {kØ} {aØe}  /v
```

The transformation can be represented equivalently as a sequence of symbol pairs where the left symbol is transformed into the right symbol:

```
p:p u:u r:r k:{kØ} a:{aØe} a:0 V02*:/v
```

or in an abbreviated form where pairs (e..g. *p:p*) of identical symbols are represented by a single symbol (*p*) without the colon:

```
p u r k:{kØ} a:{aØe} a:0 V02*:/v
```

The inflection code *V02** indicates that the entry is a verb of the second inflection class and that the stem is subject to consonant gradation. The example expresses the fact that the fourth phoneme of the dictionary word, *k*, must be replaced with a morphophoneme *{kØ}*. The morphophoneme tells that in that position *k* alternates with nothing (*Ø*).[2] At the end of the stem of the dictionary word, the final vowel *a* alternates with zero *Ø* and *e*. All these facts can be deduced by studying verbs with the inflection code *V02**, i.e. studying the shapes and what kinds of alternations occur in those verbs.

A LEXC lexicon consists of sub-lexicons containing entries for affixes and lexemes. The lexemes are in the sub-lexicon where everything starts and the guessing of such entries is the topic of this paper. Each entry typically corresponds to a morpheme. A morpheme is represented as a pair of its morphophonemic representation and a name of a sub-lexicon containing those morphemes (or entries) which may occur immediately after this morpheme. This name of the next sub-lexicon is often called the *continuation class* of a lexeme.

The inflection class also determines the continuation class, e.g. */v* (which indicates here that all verbal endings are attached directly to the stem). The association between a inflection code (e.g. *V02**) and a continuation class (e.g. */v*) could be included as a

[2] We use an arbitrary symbol (*Ø*) to denote deletion or epenthesis. In morphophonemes and within two-level rules it is always a concrete symbol, not an epsilon which would correspond to the empty string. In this way, one has a better control over epenthesis and deletions. The *Ø* symbols will be removed only after the rule component has been applied.

part of the regular expression but it proved to be better to represent as a separate two-column table, which is used both in building the converter and the guesser.

In order to generalize the patterns we need to define some common component expressions, e.g. vowels and consonants:

```
Vo = [a|e|i|o|u|y|ä|ö];
Co = [b|c|d|f|g|h|j|k|l|m|n|p|q|r|s|t|v|w|x|z];
```

For Finnish words, we need an expression for the mapping of gradating consonant clusters (as they appear in the dictionary words) into the corresponding morphophonemic representations (as they will be in the LEXC entry).

```
Gs = [(l|r|n) k k:{k∅}|(l|r) k:{k∅}|n k:{kg}|
       m p:{pm}|(l|r) p:{pv}|(l|r|m) p p:{p∅}|
       (h) t:{td}|l t:{tl}|n t:{tn}|r t:{tr}|(l|r|n) t t:{t∅}];
```

One might generalize the above transformation example by noticing that the initial part of the word is the same in the dictionary word and in the LEXC entry part. Near the end of the dictionary word there is a strong grade of a gradating consonant cluster but in the LEXC entry there is a corresponding morphophoneme. At the very end, there is the infinitive ending, (here) *a* which has to be removed, and the the code of the inflection class which has to be replaced by the corresponding continuation class. For our example, the following simple mapping could do the conversion:

```
[Co|Vo]* Gs a:{a∅e} a:0 V02*:/v
```

Such a simple expression might work correctly when converting dictionary words but usually one wants to describe the inflection classes in more detail according to the syllable structure and other characteristics shared by all words in that class. The more precisely the expression separates lexemes in its class from those in other classes, the better the expression serves its purpose. Precise characterizations help the building of the lexicon by finding atypical entries in the dictionary and possible mistakes in the data. Accurate expressions also help the guessing process to converge faster.

A short Python script was made for reading in expressions for all verbal inflection classes and for transforming the expressions into a converter LEXC lexicon. The converter lexicon implements the mapping from dictionary entries into actual Finnish LEXC entries. Each expression forms the first part of one converter lexicon entry and this part maps dictionary entries of that class into a morphophonemic representation. The second part of the converter lexicon entry is the inflection identifier used here as a name of a sub-lexicon. A small sub-lexicon of the transformer lexicon is produced out of the separate table that was mentioned above. Each line of that table is converted into a sublexicon to which the expression entry continues. This arrangement allows one to experiment with different types of converted lexicons e.g. one which is very permissive and useful for old or dialectal texts and others which are more normative by excluding less common ending allomorphs of each inflectional class. Below is a fraction of the generated LEXC for conversion:[3]

```
LEXICON Root
< Co* Vo+ (Sy1)* Co* [o|ö|u|y] [a:0|ä:0] > V01 ;
< Co* Vo+ (Sy1)* Gs [o|ö|u|y] [a:0|ä:0] > V01* ;
< Co* Vo+ (Sy1)* Co+ [a:%{a∅e%} a:0|ä:%{ä∅e%} ä:0] > V02 ;
```

[3] The curly brackets were used in the expressions as such but in LEXC they must be protected or quoted with a per cent sign (%). The Python script adds the per cent signs.

```
< Co* Vo+ (Sy1)* Gs [a:%{a∅e%} a:0|ä:%{ä∅e%} ä:0] > V02* ;
< Co VV t:%{tds%} [a:%{a∅e%} a:0|ä:%{ä∅e%} ä:0] > V03 ;
< Co VV t:%{tds%} [a:%{a∅e%} a:0|ä:%{ä∅e%} ä:0] > V04 ;
```

The above lexicon is then compiled into a FST, stored and used by another Python script which performs the conversion. This script can be applied to a test set of representative dictionary entries or to a full scale list of all dictionary list words. In order to have full control of possible failures of the expressions, the script uses the lookup mode so that it finds not only the appropriate result but also knows when the expressions fail to give any results. So, in addition to the resulting LEXC entries, also a control list is produced for verification and debugging. Potential errors in the source data (dictionary entries) as well as in the expressions can be found in this way.

The patterns and definitions are given as a file of comma separated values (CSV) (possibly by editing with a spreadsheet and then saving in this format). The patterns and definitions are extracted from there using a short Python script which formats the data into the LEXC format and collects any multicharacter symbols needed for the definitions in the header part of the resulting LEXC file. This CSV file can then be reused for other purposes. The following are samples from a full-scale description of patterns for Finnish verbs according to the inflection class codes used in the *Reverse dictionary of Modern Standard Finnish* (Tuomi, 1980), see Figure 1.[4]

4 Reusable affix data

Converters and guessers are not meaningful in isolation. In order to know into what format the dictionary entries have to be converted, one needs to have at least a small test TWOLC lexicon and the associated rules. The test lexicon defines the target for the conversion and the guessing. It ought to include example lexemes from all inflection classes and define what affixes may be attached to the stems and in which combinations. The rules[5], in turn, make the test lexicon operational so that the morphophonemic alternations and the combinations of stems and affixes can be validated. The design of a LEXC lexicon and rules is beyond the scope of this paper except that parts of the test lexicon can be reused in the guesser. Thus, one would benefit from combining the writing the analyzing and the guessing lexicons.

Both the normal morphological analyzer and the guesser need a description of inflectional morphemes (affix entries), their shapes and the ways in which they may combine with each other. The structures of these two lexicon systems is rather isomorphic, i.e. the entries and the sub-lexicons correspond directly to each other, even if the entries are a bit different.

LEXC lexicons are technically a *collection of sub-lexicons* where each sub-lexicon has a *name* and a *set of entries*. For each lexeme (or root morpheme) and each affix (or inflectional morpheme) , there is an entry in some sub-lexicon. Each entry consists formally of three components: (1) *input string*, (2) *output string* and (3) the name of the *continuation sub-lexicon* from which the next entry is chosen:

[4] All Python scripts and the CSV files mentioned in this section are freely available at Github: `https://github.com/koskenni/twolex`. The HFST used in these Python scripts was loaded according to the instructions at `https://pypi.python.org/pypi/hfst`

[5] In this paper, two-level rules were used and the morphophonemes were established according to the principles of the simplified two-level model. The method for conversions, guessing and the reuse of affix data is independent of the kind of rules one uses.

```
ID,NEXT,MPHON,COMMENT
,V01,<Co* Vo+ Co+ [o|ö|u|y] [a:0|ä:0]>,PUNOA
,V01*,<Co* Vo+ Gs [o|ö|u|y] [a:0|ä:0]>,KUTOA
,V02,<Co* Vo+ (Co+ Vo+)* Co+ [a:{a∅e} a:0|ä:{ä∅e} ä:0]>,MUISTAA
,V02*,<Co* Vo+ (Co+ Vo+)* Gs [a:{a∅e} a:0|ä:{ä∅e} ä:0]>,HUUDAHTAA
,V03,<Co VV t:{tds} [a:{a∅e} a:0|ä:{ä∅e} ä:0]>,HUUTAA
,V04,<Co VV t:{tds} [a:{a∅e} a:0|ä:{ä∅e} ä:0]>,SOUTAA
...
,V09,<(Co) [Vo|VV] Co+ a:{aoe} a:0>,KAIVAA
,V09*,<(Co) [Vo|VV] Gs a:{aoe} a:0>,KATTAA
,V10,<Co [Vo|VV] Co+ a:{aoe} a:0>,HAASTAA
,V10*,<(Co) [Vo|VV] Gs a:{aoe} a:0>,MALTTAA
,V11,<Co a i s t a:{aoe} a:0 >,PAISTAA
,V11*,<Co [a|i|a a|a i] [Gsk|Gst] a:{aoe} a:0 >,VIRKKAA
,V12,<Co a a r t:{trs} a:{aoe} a:0>,SAARTA
,V13,<(Co) Vo+ Co+ e:{ei∅} [a:0|ä:0]>,LASKEA
,V13*,<(Co) Vo+ Gs e:{ei∅} [a:0|ä:0]>,KYLPEÄ
,V13*,<(Co) Vo+ Gsj e:{ei∅} [a:0|ä:0]>,SULKEA
,V14,<t u n t:{tns} e:{ei∅} a:0>,TUNTEA
,V15,<p o t:{tds} e:{ei∅} a:0>,POTEA
,V16,<l ä h:0 t:0 e:0 ä:0>,LÄHTEÄ
,V17,<Co* Vo+ Co+ i:{i∅} [a:0|ä:0]>,SALLIA
,V17*,<Co* Vo+ Gs i:{i∅} [a:0|ä:0]>,LEMPIÄ
,V17,<Co* Vo+ (Sy1)* [k s|p s] i:{i∅} [a:0|ä:0]>,KÄVELEKSIÄ
,V17*,<Co* Vo+ (Sy1)* [h t:{td}] i:{i∅} [a:0|ä:0]>,PUIKKELEHTIA
,V18,<n a i:{i∅} d:0 a:0>,NAIDA
,V18,<Co o i:{i∅} d:0 a:0>,VOIDA
,V18,<(p) u i:{i∅} d:0 a:0>,UIDA PUIDA
,V18,<Co* Vo+ (Sy1)* Co* [o|ö] i:{i∅} d:0 [a:0|ä:0]>,VOIDA
,V19,<s a a:{V∅} d:0 a:0>,SAADA
,V19,<j ä ä:{V∅} d:0 ä:0>,JÄÄDÄ
,V20,<m y y:{V∅} d:0 ä:0>,MYYDÄ
,V21,<Co [u:{u∅} o d:0 a:0|y:{y∅} ö d:0 ä:0]>,JUODA SYÖDÄ
,V22,<v i:{i∅} e d:0 ä:0>,VIEDÄ
,V23,<k ä y:0 d:0 ä:0>,KÄYDÄ
,V24,<Co* [Vo|VV] Co* (a|ä) i s 0:{e∅} t:0 [a:0|ä:0]>,NUOLAISTA
,V24,<Co* [Vo|VV] (Sy1) Co*  s 0:{e∅} t:0 [a:0|ä:0]>,NOUSTA SEISTÄ
,V24*,<Co* [Vo|VV] Gw (a|ä) i s 0:{e∅} t:0 [a:0|ä:0]>,LAUAISTA
,V25,<Co [Vo|VV] l 0:{e∅} l:0 [a:0|ä:0]>, TULLA NIELLÄ
,V26,<Co [Vo|VV] r 0:{e∅} r:0 [a:0|ä:0]>,PURRA PIERRÄ
,V27,<Co [Vo|VV] n 0:{e∅} n:0 [a:0|ä:0]>,PANNA MENNÄ
...
,V32,<j u o | p i e | s y ö>,JUOSTA juo-kse/v
,V33,<n ä | t e>,NÄHDÄ nä-ke/v
,V34,<Co* Vo+ (Sy1) Co+ [a|ä|o|ö|u|y|e] t:0 [a:0|ä:0]>,ALETA ale-ne/v
,V34*,<Co* Vo+ (Sy1) Gw [a|ä|o|ö|u|y|e] t:0 [a:0|ä:0]>,KYETÄ ale-ne/v
,V34*,<Co* Vo+ (Sy1) Gwj e t:0 [a:0|ä:0]>,KYETÄ ale-ne/v
,V35,<Co* Vo+ (Sy1)* Co* [a|ä] t:0 [a:0|ä:0]>,SALATA sala-V/v
,V35*,<Co* Vo+ (Sy1)* Gw [a|ä] t:0 [a:0|ä:0]>,AIDATA sala-V/v
,V35*,<Co* Vo+ (Sy1)* Gwj ä t:0 ä:0>,AIDATA sala-V/v
...
,V37,<Co* [Vo|VV] Co+ i t:0 [ä:0|a:0]>,SELVITÄ selvi-A/v
,V37*,<Co [Vo|VV] Gw i t:0 [ä:0|a:0]>,SIITÄ selvi-A/v
,V38,<Co* [Vo|VV] Co+ [o t:0 a:0|ö t:0 ä:0]>,KOHOTA koho-A/v
,V38*,<Co* [Vo|VV] Gw [o t:0 a:0|ö t:0 ä:0]>,TAUOTA koho-A/v
...
```

Figure 1: Regular expression patterns for Finnish verbs

```
INPUT:OUTPUT CONT;
```

Either the input or output string can be empty, and also here, only one of them needs to be given if they are identical. If both are empty strings, even the colon may be omitted.

In the framework used in Beesley and Karttunen (2003), the input string is the final base form of the lexeme or for affixes, it consists of the morphosyntactic features of each affix. The output string is the (morph)phonological shape of the affix. In order to reduce the number of entries for morphemes, morphophonemic forms are used. Then one may let the rules take care of the different shapes the affixes have when combined with different stems or other affixes.

The present approach uses the three components of an entry in different ways depending on whether an analyzer or a guesser is made. E.g. an entry for the ending for conditional mood in analysis could be:

```
+COND+ACT:isi Person;
```

where the INPUT consists of the morphosyntactic features *+COND* and *+ACT*, the OUTPUT (or morphophonemic representation) is *isi* and the next morpheme is in a lexicon *Person*.

In affixes for guessing the components OUTPUT and CONT are the same as for the analysis but the INPUT component is an empty string, e.g.:

```
:isi Person;
```

Let us move to *lexeme entry classes*. An entry class corresponds to one or several similar inflection classes (as used in the dictionaries). For lexemes of such a class, stems are not fully related to each other via simple phonological rules. E.g. a Finnish verb *salata* (‘hide’, ‘keep in secret’) belongs to a common inflection class where the stems are e.g. *salaa-*, *salas-*, *sala-*, *salat-*, *salan-*. One may simplify the rule component by splitting such lexemes in two parts: (1) a truncated stem which is constant or phonologically regular for a lexeme and (2) stem endings which are common to all lexemes in this lexeme entry class. Both parts may themselves contain regular phonological alternations such as vowel harmony or consonant gradation.

For each lexeme entry class, a sub-lexicon is established and it lists the end parts of the stems in such a class. A sub-lexicon corresponding to a lexeme entry class is common to all lexeme entries of this class and the lexeme entries all continue to this sub-lexicon. The actual sub-lexicon used in the analysis of words inflected like *salata* could be as follows:

```
LEXICON sala-A/v
{nt}{dlnrtØ}{aä}:{VØ} v0;
{nt}{dlnrtØ}{aä}:s    v1;
{nt}{dlnrtØ}{aä}:{VØ} v2;
{nt}{dlnrtØ}{aä}:{nt} v3;
```

In this example, *sal-A/v* is the name of this sub-lexicon, *v0* is the sub-lexicon for present tense forms, *v1* is the sub-lexicon for the past tense morpheme, *v2* for conditional morpheme and *v3* is the sub-lexicon where infinitives and participal morphemes reside. The base form for verbs is traditionally the first infinitive which is in sub-lexicon *v3*. The INPUT component (for analysis) consists of whatever must be added to the truncated stem (i.e. *{nt}* is added) in order to form the v3 stem. It is given here followed by the common morphophonemic form of the infinitive ending *{dlnrtØ}{aä}*.[6] The OUTPUT component consists of the final parts of morphophonemic representations of the different stems.

In the guesser version, all this information is not needed. Instead, the INPUT that is needed here, consists of the name of the continuation class itself (defined as a multicharacter symbol and preceded by a space that has been quoted with a per cent sign).

```
LEXICON sala-A/v
% sala-A/v:{V∅} v02;
% sala-A/v:s v1;
% sala-A/v:{nt} v3;
```

Some linguists think that complex lexicons in LEXC format are not convenient for humans to edit.[7] Therefore, it is a practical idea to create and edit those parts of the lexicon in a simple tabular form by using e.g. some spreadsheet calculator. From the internal format of spreadsheet calculators, one may store the data as comma separated values (CSV) which are easy to process with small Python scripts. One may use slightly different scripts in order to produce either normal LEXC entries for morphological analysis or entries modified for the guesser or other purposes. Below is the source CSV data for the above examples for *sal-A/v* sub-lexicons:

```
ID       , NEXT, MPHON, FEAT, BASE
sala-A/v, v0   , {V∅} ,     , {nt}{dlnrt∅}{aä}
        , v1   , s    ,     , {nt}{dlnrt∅}{aä}
        , v2   , {V∅} ,     , {nt}{dlnrt∅}{aä}
        , v3   , {nt} ,     , {nt}{dlnrt∅}{aä}
```

In a similar manner, the CSV entries of actual affixes can be in the same tabular format, e.g. the conditional ending:

```
ID, NEXT       , MPHON, FEAT      , BASE
v2, Person neg , isi   , V COND ACT,
```

For analysis, INPUT comes from the FEAT column, and OUTPUT from the MPHON column. There are two sub-lexicon names in the NEXT column and therefore two separate entries are produced into the LEXC lexicon. For guessing, in inflectional endings, INPUT will be empty but otherwise the entry will be similar whereas the lexicons for inflection classes will be slightly different. In them, the INPUT is the name of the sub-lexicon (and OUTPUT as in analysis). Short Python scripts perform all simple conversions that are needed.[8]

5 Producing the FST for guessing

Now we know how to make the sub-lexicons for affixes and the special sub-lexicons for each lexeme entry class. When converting dictionary headwords into lexeme en-

[6] The infinitive ending may be in several shapes (*-a, -ä, -ta, -tä, -da, -dä, -la, -lä, -na, -na, -ra, -rä*), but the forms are fully determined by the phonological properties of the preceding stem and easily handled by a rule.

[7] Especially the handling of so called flag diacritics requires duplication and results in less readable LEXC source files. Moreover, there are no convenient ways to parametrize the LEXC files so that one could compile different versions out of the same source file.

[8] All Python scripts and the CSV files mentioned in this section are freely available at Github: `https://github.com/koskenni/twolex`.

tries in Section 3, each pattern had an inflection class code and there was a separate table associating the inflection class and the lexeme entry class so that the pattern and the sub-lexicons could produce a lexeme entry with the proper continuation class. We use the same pattern data when building the regular expression entries for guessing.

In the conversion, the expressions themselves were transformations from the dictionary head word into a morphophonemic representation of the lexeme entry. For the guesser, we only need the output part of this mapping. For transforming the conversion patterns into guessing patterns, a small Python script was made. The script changed the regular expressions in the definitions and in the regular expressions patterns so that any symbol pair was replaced by the output part only, e.g.:

```
p:{pv}  -->  {pv}
a:0     -->  0
```

The result was an output projection of the initial transduction. It was formatted according to the conventions required by LEXC and compiled together with the affixes in a respective format. The compiled lexicon FST was then compose-intersected with the two-level rules. The inverse of this was then minimized and optimized for lookup so that it could be used for looking up possible entries for any verb form, e.g.:

```
$ hfst-lookup -i guesser.fst
> hakkeroiden
hakkeroiden     hakkero     haravo-i/v  0,000000
hakkeroiden     hakkero{i∅} /v          0,000000
```

The hfst-lookup program reads a word form (*hakkeroiden*) at a time and looks the FST for any matches and prints them (*hakkero haravo-i/v* and *hakkero{i∅} /v*) together with the input word. In addition, the program prints a weight of the results (which is not yet used in the guessing but probably one can find useful ways to incorporate weights into the process).

6 Selecting the correct entry interactively

In Section 5 we ended up with a FST which maps any inflected word form into a set of possible LEXC entries. Neither the XFST scripts not the HFST command line tools lend themselves for the kind of looping and testing that one would need for interfacing the guessing FST with a human user in a natural way. Fortunately, this is quite easy when using the HFST that is embedded in Python 3.

A very simple script can read in the FST produced as above. In a loop, the program can read in a word form and search the FST for any entries which were associated with it. Searching can be done with an efficient lookup function which produces the results in a form that the script can easily test. If there still are several entries remaining, the script asks for another form of the same word. An intersection of the new and the previous set of results is calculated. If only one result remains, that is the answer. If several results remain, then the user must enter a further form.

The above procedure solves the problem in most cases but not in all. Two tentative lexical entries can overlap so that one generates all forms which the other one does but it generates some additional forms which the other does not. The user can then find the solution if the entry which she is searching has a lager set of forms. If the correct entry would have only forms also acceptable for the other entry, the problem cannot be solved by just entering more forms. In this situation, the user needs to enter a negative example, i.e. a form which would be allowed by the wrong candidate lexeme but not by the correct. The program, then, subtracts the entries corresponding to such a word-form.

The following is an example where the user enters *brassata* and *brassasin* in order to narrow down the possible LEXC entries. The negative example *brassajaa* (prefixed with a minus sign) resolves the problem that the two tentative entries (*sala-A/v* and *pala-V/v*) are overlapping so that the former is included in the latter.

```
ENTER FORMS OF A WORD
brassata
    {'brassa ale-ne/v', 'brassa sala-A/v', 'brassa pala-V/v',
      'brassat{t∅}{a∅e} /v', 'brassat{a∅e} /v'}
brassasin
    {'brassa sala-A/v', 'brassa pala-V/v'}
-brassajaa
    {'brassa sala-A/v'}
RESULT:
brassa sala-A/v ;
```

As the reader can readily see, the script is just a starting point which can be made more sophisticated, c.f. (Esplà-Gomis et al., 2017). Instead of just accepting correct or incorrect forms from the user the program might generate critical forms and ask the user whether they are correct or not. Forms which are valid inflected forms of one but not all tentative entries are useful in this respect. There are several possibilities in selecting which forms to ask.

One could make the guesser more helpful by restricting the inflectional forms to so called principal parts i.e. a minimum set of forms which is still sufficient for determining the correct entry. With this restriction, one can use the guesser FST together with its inverse. The given word form goes through the guessing FST and results in a set of tentative entries. Each of the entries is fed to the inverse FST. In this way one gets a set of principal forms for each entry candidate. These lists could be shown to the user who then can select one list and thus the underlying entry.

One may also process the sets of forms of each lexeme candidate and hide common word forms. One idea is that a sequence of word forms would be presented in a particular order. The sequence would only contain forms which are not common to all tentative entries. At the top of the list would be those forms which belong to the least number of entries. The user would then respond by telling which is the first acceptable form of the target entry. If all forms in front of that one are marked as negative examples then the interaction between the program and the user might converge even more rapidly.

7 Corpus-assisted guessing of entries

A list of all word forms occurring in a large corpus is quite valuable when choosing among different possible entry candidates. The correct entry is more likely to have some word forms in the corpus than the entries for non-existing lexemes. After the first step when the user has given a word form to the guesser, the program has a set of alternative entries which could generate that word form, e.g.:

```
rääkkäsi
     {'rääkkä sala-A/v', 'rääkkäs{ä∅e} /v',
      'rääkkäs{e∅} /v', 'rääk{k∅}ä sala-A/v'}
```

Let us consider each of these entries in turn. The set of word forms occurring in the corpus which are also generated by the entry is interesting.[9] What happens if one feeds these word forms into the algorithm described above. The algorithm may find a unique solution if the set contains enough forms. Such solutions are likely to be the correct ones we are looking for. Sometimes there may several possibile answers because the corpus contains forms of other (similar looking) lexemes and even typing errors. In case there are many (unique) solutions, one can choose the one having the largest set of word forms or the program might ask the user to choose the correct one.

The plain algorithm would require another word form in order to proceed. The corpus-assisted algorithm would find the solution right away from the first word form:

```
rääkkäsi
        CORPUS CONTAINS: { rääkättiin, rääkkään,
            rääkkäävät, rääkäten, rääkännyt, rääkätty,
            rääkättäisi, rääkätään, rääkkäisi, rääkätä,
            rääkkäsi, rääkkää }

==================
rääk{k∅}ä sala-A/v ;
==================
```

Using the HFST finite-state tools, it is easy to implement the enhanced guesser. We already have G, a FST which maps a word form into the set of possible entries $\{e_1, ..., e_k\}$. We must prepare the corpus data in advance in order to support the interactive guessing of the entries. The distinct word forms (types) occurring in a corpus can be easily produced as a list. This list can be converted into a FSA, say W using the *hfst-strings2fst* command line tool. The composition $C = W \circ G$ maps each word form in the corpus into the entries which could generate them. The inverse of this mapping, $H = C^{-1}$ gives us the word forms in the corpus that an entry could generate. This mapping H is used in the corpus-assisted version of the entry guesser. See Appendix A for some examples of corpus-assisted for Finnish verbs guessing are given. It appears that such methods could be used for building lexicons for lesser resourced languages.

8 Guessing entries from a corpus

One can modify the computer-assisted guessing so that it works without human intervention. The input side (projection) of the transducer H accepts all entries that we need to consider. It is a finite-state machine and it can easily be converted into a plain list (of strings). Once this is done, the algorithm may proceed by considering each entry at a time and test whether the entry ought to be accepted or not.

The mapping H which was defined above, maps every entry e into those word forms occurring in the corpus which the entry would accept. As in the previous section, we evaluate the goodness of an entry e in E by using the set of word forms in

[9] One could simply choose the entry with the longest list but here we wish to stress the correctness of guesses.

$H(e_i)$ and by feeding them into the algorithm and see whether the list makes the algorithm to converge into exactly one entry. If successful, we have a good candidate for an entry. If unsuccessful, we have not enough evidence to exclude the other candidate entries because some of them also generate all word forms in the list.

An experiment of this method is described in Appendix B. The results were encouraging although the guesser only covered verbs and all noun and adjective forms presented harmful noise to the procedure. For a random sample of word form types taken out from a large text corpus of Finnish, the method provided some 77 % correct results when using very simple criteria. Further research on the topic is clearly needed. One could assign weights to the affixes and use them when excluding less likely entries.[10]

One could easily combine the information from a corpus with the interactive guessing. A word form given by the user would first be expanded to a set of tentative entries. Then, each tentative entry would be tested against the corpus in order to see whether the corpus would give conclusive evidence for exactly one of the entries. In such cases, the entry could be directly selected and the interaction would be faster. Even partial evidence could be utilized byt that would probably need some further research and testing.

9 Experiment with Finnish verbs

The examples presented in the preceding sections were taken from an experiment with Finnish verb morphology. There was a long term interest to deal with older Finnish texts and therefore the *Reverse Dictionary of Modern Standard Finnish* (RDMSF) which reflects the language in the first half of the 20th century was taken as the basis rather than *Kielitoimiston sanakirja* (KS) which reflects the present day use. RDMSF allows more liberal use of ending allomorphs and stem variants than the KS. The extra forms are readily understood even by present day speakers but seldom used any more although they are quite commonly found in earlier texts.

The examples in the preceding sections were made using a the RDMSF conjugation tables and example words of the dictionary and two-level rules and a lexicon with verbal ending which had been prepared earlier for other purposes. A couple days were spent in establishing 78 regular expression patterns for the 45 conjugations used in RDMSF. A Unix makefile was prepared to control the use of a number of small command line and Python scripts. In this way, it was convenient to rebuild the FSTs for conversion, analysis and guessing.

The test set of selected entries was converted using the conversion FST. The resulting entries were then combined with verbal affixes and compose-intersected with the rule FSTs. The string pairs represented by this FST was produced in a human readable form. The list consisted of pairs of base form plus features and and the corresponding surface form:

```
...
iätä+V+INF1+NOM:iätä
iätä+V+INF2+ACT+INE:iätessä
iätä+V+INF2+ACT+MAN:iäten
```

[10] HFST-LEXC has a facility for weighted entries and these would automatically propagate to the mappings that were described above. The same applies to the FSA for word form types in the corpus whose frequencies could be utilized when assigning weights to them.

```
iätä+V+INF2+PSS+INE:iättäessä
iätä+V+PAST+ACT+1PL:ikäsimme
iätä+V+PAST+ACT+1SG:ikäsin
iätä+V+PAST+ACT+2PL:ikäsitte
...
```

The list was checked manually and some errors were detected in the affix tables and one in the rules. After modifications, the test data appeared to be clean of errors.

The conversion was tested against the full list of 16,000 verb entries in the dictionary. The test revealed some points where the patterns had to be made more general in order to accept less typical verb entries in some conjugations. The analysis was tested superficially by entering word forms randomly picked up from *Nykysuomen sanakirja* (Sadeniemi, 1951–1961) and verifying that the results were correct. The same kind of testing was done with the guesser.

10 Further work

There is a plan to continue the present work and produce a full scale Finnish morphological analyzer and guesser which could be used for various purposes, including the analysis of Finnish texts from the 19th century. The present approach makes such an analyzer quite flexible for extending and tuning. One could easily add and change inflectional patterns so that the historical endings and stem patterns would be better covered. The kind of a morphophonemic lexicon which is used in this approach lends itself also to applications within historical linguistics and comparing related languages with each other, cf. (Koskenniemi, 2013a). Whereas the existing open source morphological analyzer OMORFI of Pirinen (2015) is designed for a wide coverage lexicon and is normative, the proposed one OFITWOL would be permissive and descriptive. OMORFI aims at excluding old and dialectal inflections whereas OFITWOL aims at including them, cf. the arguments in Koskenniemi and Kuutti (2017).

Handling Finnish dialects by using the morphophonemic lexical representations would also be an interesting topic to study. It is not always possible to relate word forms in standard Finnish with forms in dialects because a word form alone does not contain the relevant information. Morphophonemes combine the information of the various stems of a lexeme and various forms of affixes. These morphophonemic forms might contain the sufficient information for generating old or dialectal forms of Finnish out of the morphophonological representations of OFITWOL.

References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2015. Paradigm classification in supervised learning of morphology. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Denver, Colorado, pages 1024–1029. http://www.aclweb.org/anthology/N15-1107.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. Studies in Computational Linguistics, 3. University of Chicago Press. Additional info, see: `www.stanford.edu/~laurik/fsmbook/home.html`.

Grégoire Détrez and Aarne Ranta. 2012. Smart paradigms and the predictability and complexity of inflectional morphology. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Avignon, France, pages 645–653. http://www.aclweb.org/anthology/E12-1066.

Miquel Esplà-Gomis, Rafael C. Carrasco, Víctor M. Sánchez-Cartagena, Mikel L. Forcada, Felipe Sánchez-Martínez, and Juan Antonio Pírez-Ortiz. 2017. Assisting non-expert speakers of under-resourced languages in assigning stems and inflectional paradigms to new word entries of morphological dictionaries. *Language Resources and Evaluation* 51(4):989–1017.

Måns Huldén. 2014. Generalizing inflection tables into paradigms with finite state operations. In *Proceedings of the 2014 Joint Meeting of SIGMORPHON and SIGFSM*. Association for Computational Linguistics, Baltimore, Maryland, pages 29–36. http://www.aclweb.org/anthology/W14-2804.

Kimmo Koskenniemi. 1983. *Two-level Morphology: A General Computational Model for Word-Form Recognition and Production.* Number 11 in Publications. University of Helsinki, Department of General Linguistics.

Kimmo Koskenniemi. 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*. Linköping University Electronic Press; Linköpings universitet, number 87 in NEALT Proceedings Series 18, pages 53–53. http://www.ep.liu.se/ecp/article.asp?issue=087&article=004&volume=.

Kimmo Koskenniemi. 2013b. An informal discovery procedure for two-level rules. *Journal of Language Modelling* 1(1):155–188. http://jlm.ipipan.waw.pl/index.php/JLM/article/view/62.

Kimmo Koskenniemi and Pirkko Kuutti. 2017. *K + K = 120: Papers dedicated to László Kálmán and András Kornai on the occasion of their 60th birthdays*, Research Institute for Linguistics, Hungarian Academy of Sciences, chapter Indexing Old Literary Finnish text, page 32 p.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A. Pirinen, and Miikka Silfverberg. 2011. Hfst – framework for compiling and applying morphologies. In Cerstin Mahlow and Michael Piotrowski, editors, *Systems and Frameworks for Computational Morphology 2011 (SFCM-2011)*. Springer-Verlag, volume 100 of *Communications in Computer and Information Science*, pages 67–85.

Tommi A. Pirinen. 2015. Development and use of computational morphology of finnish in the open source and open science era: Notes on experiences with omorfi development. *SKY Journal of Linguistics* 28:381–393.

Aarne Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars.* CSLI Publications, Stanford. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

Matti Sadeniemi, editor. 1951–1961. *Nykysuomen sanakirja*, volume 1–6. Werner Söderström Osakeyhtiö, 4 edition.

Tuomo Tuomi. 1980. *Suomen kielen käänteissanakirja / Reverse Dictionary of Modern Standard Finnish.* Number 274 in Toimituksia. Suomalaisen Kirjallisuuden Seura, 2 edition.

A Test of corpus-assisted guessing of entries

The assisting corpus used here was a list of word forms starting with *r* from the SKTP collection of texts[11]. The lists used here consists of some 115,000 word form types. Only a small portion (less than 1/10) of them was actually forms of verbs. The word forms that were tested as input were manually selected from another list, the Finnish PAROLE corpus. Some forms occurring six times in the Parole corpus were picked up and fed to the program. Nouns, nominal derivations and also verb forms with clitic particle were excluded from the selection. This unsystematic, small and biased test gave very promising results, i.e. the correct solution was found directly:

```
ryöstäen
    (1) << ryöst{ä∅e} /v >> ryöstää, ryöstivät, ryöstetä,
    ryösti, ryöstävät, ryöstä, ryöstetään, ryöstetty,
    ryöstänyt, ryöstäen, ryöstettiin, ryöstettävä, ryöstäessä,
    ryöstän, ryöstämme, ryöstäisi
===============
ryöst{ä∅e} /v ;
===============

rakasti
    (1) << rakast{a∅e} /v >> rakastettava, rakastettaisi,
    rakastaisitte, rakastaisit, rakasteta, rakastettu, rakastatte,
    rakastaisivat, rakastaen, rakastakaamme, rakastaisimme,
    rakastivat, rakastit, rakasta, rakastat, rakastin, rakastamme,
    rakastetaan, rakastettiin, rakasti, rakastanut, rakastaisin,
    rakastakaa, rakastavat, rakastan, rakastaisi, rakastimme,
    rakastaa, rakastettaisiin
================
rakast{a∅e} /v ;
================

roihuaa
    (1 ) << roihu halu-A/v >> roihuttiin, roihunnut, roihusivat,
    roihuta, roihusi, roihua, roihutessa, roihuten, roihuaa,
    roihuavat
    (2) << roihua{k∅}{a∅e} /v >> roihuaa, roihuakin
================
roihu halu-A/v ;
================

rikkoontuivat
    (1) << rikkoon{tn}u /v >> rikkoontuessa, rikkoontua,
    rikkoontuivat, rikkoontuisi, rikkoontuu, rikkoontui,
    rikkoontunut, rikkoontunee, rikkoontuvat, rikkoonnu
=================
rikkoon{tn}u /v ;
=================

ryhtyvät
```

[11] The Downloadable Version of the Finnish Text Collection, “sktp-dl, ftc-dl”, ID: http://urn.fi/urn:nbn:fi:lb-2016050206, The resource is available in FIN-CLARIN Kielipankki - the Language Bank of Finland at http://urn.fi/urn:nbn:fi:lb-2014052719

```
    (1) << ryhty /v >> ryhtyi, ryhtyne, ryhtykäämme, ryhtykööt,
    ryhtyköön, ryhtyy, ryhtyen, ryhtyvät, ryhtyisivät, ryhtyisi,
    ryhtykää, ryhtyisimme, ryhtynevät, ryhtynyt, ryhtynen,
    ryhtyivät, ryhty, ryhtyessä, ryhtyisin, ryhtyä, ryhtynee
    (2) << ryh{td}y /v >> ryhtyi, ryhtyne, ryhtykäämme, ryhtykööt,
    ryhtyköön, ryhtyy, ryhtyen, ryhtyvät, ryhdyimme, ryhdyttäne,
    ryhdyttiin, ryhdyttäköön, ryhdyttäkö, ryhdymme, ryhdyn,
    ryhtyisivät, ryhdyin, ryhtyisi, ryhtykää, ryhdyitte,
    ryhtyisimme, ryhtynevät, ryhdy, ryhdyttäneen, ryhdytään,
    ryhdyttäessä, ryhtynyt, ryhtynen, ryhtyivät, ryhdyttäisiin,
    ryhdytte, ryhdytty, ryhtyessä, ryhtyisin, ryhtyä, ryhdytä,
    ryhdyttäisi, ryhdyttävä, ryhdyit, ryhtynee, ryhdyt
    (3) << ryhtyv{ä∅e} /v >> ryhtyvän, ryhtyvää, ryhtyvät, ryhtyvä
=============
ryh{td}y /v ;
=============

repeäisi
    (1) << re{pv}e katke-A/v >> repeävät, revennyt, repesin,
    repeän, repesi, revetessä, repeäisi, repeää, repee,
    repeäisivät, repesivät, revetä, repeä
===================
re{pv}e katke-A/v ;
===================

rajasi
    (1) << rajas{e∅} /v >> rajasta, rajasivat, rajasi, rajasimme,
    rajastaan
    (2) << raja sala-A/v >> rajaat, rajaavat, rajattu, rajaan,
    rajattaneen, rajaten, rajatkaa, rajata, rajaamme, rajasivat,
    rajaisimme, rajattiin, rajattaisi, rajaisivat, rajasimme,
    rajattaisiin, rajasi, rajaisi, rajattaessa, rajattava, rajaisit,
    rajaisin, rajannut, rajaa, rajataan
===============
raja sala-A/v ;
===============
```

B Test of guessing entries from a corpus

The evaluation of the method sketched in Section 8 was based on the same word form list out of SKTP as in the Appendix A. Python scripts were written to implement the method and the corpus of word form types beginning with *r* was processed. The following is a list of a sample of 30 proposed lexicon entries out of that list. Not all results proposed by the algorithm were taken because there was much noise in those based on just a few word forms.[12] Thus only entries which covered at least eight distinct word forms were considered here. They are taken out of a total list of some 350 proposed entries. through equal interval sampling. Seven out of the 30 appear to be incorrect (marked with -), others are OK (marked with +).

1. + RAAPUSTAA `raapust{a∅e} /v` raapusti raapustin raapustivat raapustaa raapustaisi raapustaisin raapustan raapustanut raapustavat raapusteta raapustetaan raapustettava raapustettiin raapustettu
2. + RAATAA `raa{td}{aoe} /v` raada raadan raadat raadatte raadetaan raadettava raadettiin raadettu raadoin raataa raataen raataessa raataisi raataisivat raatanut raatavat raatoi

[12] The mapping proposes some entries for most word forms. Certain forms of nouns happen to be similar to some verb forms and occasionally a couple of such misleading forms uniquely determine an entry.

raatoivat

3. + RAIKUA `rai{k∅}u /v` raiu raikua raikuen raikuessa raikui raikuisi raikuisivat raikuivat raikukoot raikunut raikuu raikuvat
4. - (*RAKASTA as juosta) `rakas{e∅} /v` rakasta *rakastaan* rakastaisi *rakastako* rakastaman rakastaneen rakastava *rakasten* rakastu
5. - (*RANKATA without gradation) `ranka sala-A/v` rankaisi rankaisin rankaisivat *rankaa rankaan* rankannut *rankasi* rankata rankataan rankattava rankattiin rankattu
6. + RAPISTELLA `rapistel{e∅} /v` rapisteli rapistelivat rapistella rapistellaan rapistellen rapistellessa rapistelee rapistelen rapistelevat
7. + RASKAUTTAA `raskaut{t∅}{a∅e} /v` raskauta raskauteta raskautettiin raskautettu raskautti raskauttaa raskauttaisi raskauttanut raskauttavat
8. - (*RATKAA as kaivaa) `ratk{aoe} /v` ratkaisi ratkaisimme ratkaisin ratkaisivat ratketa ratketaan ratkettava ratkettiin ratkoi ratkoimme ratkoivat
9. + RAVATA `rava sala-A/v` ravaisin ravaa ravaan ravaatte ravaavat ravannee ravannut ravasi ravasimme ravasivat ravata ravataan ravaten ravatessa ravattaisiin ravattava ravattiin ravattu
10. + REAGOIDA `reago haravo-i/v` reagoi reagoimme reagoin reagoisi reagoisin reagoisit reagoisitte reagoisivat reagoit reagoivat reagoi reagoida reagoidaan reagoiden reagoidessa reagoimme reagoin reagoinevat reagoinut reagoit reagoitaisi reagoitaisiin reagoitava reagoitiin reagoitu reagoivat reagoinnut
11. + REKISTERÖITYÄ `rekisteröity /v` rekisteröity rekisteröityi rekisteröityisi rekisteröityisivät rekisteröityivät rekisteröitynyt rekisteröityvät rekisteröityy rekisteröityä
12. - (*REPEÄ as kylpeä and without gradation) `rep{ei∅} /v` repi REPIN repisi repisimme repisin repisivät repivät repee REPET repeä repien repiessä
13. + REVETÄ `re{pv}e katke-A/v` repee repesi repesin repesivät repeä repeäisi repeäisivät repeän repeävät repeää revennyt revetessä revetä
14. + RIEPOTELLA `riepot{t∅}el{e∅} /v` riepotella riepotellaan riepotellessa riepotellut riepoteltava riepoteltiin riepoteltu riepotteli riepottelisi riepottelivat riepottele riepottelee riepottelevat
15. + RIITAUTUA `riitau antau-TU/v` riitauduin riitauduta riitauta riitauttaneen riitautua riitautui riitautuivat riitautunut riitautuu riitautuvat
16. + RIKKOONTUA `rikkoon{tn}u /v` rikkoonnu rikkoontua rikkoontuessa rikkoontui rikkoontuisi rikkoontuivat rikkoontunee rikkoontunut rikkoontuu rikkoontuvat
17. - (*RISKIÄ as sallia) `risk{i∅} /v` *riski riskimme riskin riskisi riskit riskien riskimme* RISKINE *riskinen riskiä*
18. - (*RIVIÄ as sallia) `riv{i∅} /v` *rivi rivimme rivin rivisi rivit rivien* RIVIESSÄ *rivimme rivin rivinen rivit riviä*
19. + ROIHUTA `roihu halu-A/v` roihua roihuaa roihuavat roihunnut roihusi roihusivat roihuta roihuten roihutessa roihuttiin
20. (*ROKOTAA as muistaa without gradation) - `rokot{a∅e} /v` rokotimme ROKOTIVAT rokota rokotamme rokoteta rokotetaan rokotettaessa rokotettaisi rokotettaisiin rokotettava rokotettiin rokotettu
21. + ROSKATA `roska sala-A/v` roskaisivat roskaa roskaan roskaavat roskanne roskannut roskasi roskasivat roskata roskataan
22. + RUKSIA `ruks{i∅} /v` ruksi ruksin ruksit ruksivat ruksi ruksia ruksien ruksii ruksin ruksit ruksitaan ruksittu ruksivat
23. + RUNTATA `runt{t∅}a sala-A/v` runtannut runtata runtataan runtaten runtattava runtattiin runtattu runttaisi runttaa runttaavat runttasi runttasin runttasivat

24. + RUSTATA `rusta sala-A/v` rustaisi rustaa rustaamme rustaavat rustannut rustasi rustasin rustasivat rustata rustataan rustatessa rustattaessa rustattava rustattiin rustattu

25. + RYHMITTYÄ `ryhmit{t∅}y /v` ryhmity ryhmityin ryhmitymme ryhmityttiin ryhmittyen ryhmittyessä ryhmittyi ryhmittyisi ryhmittyisivät ryhmittyivät ryhmittynyt ryhmittyvät ryhmittyy ryhmittyä

26. + RYNNIÄ `rynn{i∅} /v` rynni rynnin rynnivät rynni rynnien rynniessä rynnii rynnin rynninyt rynnittiin rynnittävä rynnitä rynnitään rynnivät rynniä

27. + RYYDITTÄÄ `ryydit{t∅}{ä∅e} /v` ryyditettiin ryyditetty ryyditetään ryyditti ryydittivät ryydittäen ryydittäessä ryydittäisi ryydittänyt ryydittävät ryydittää

28. + RYÖVÄTÄ `ryövä sala-A/v` ryöväisi ryövännyt ryöväsi ryöväsivät ryövättiin ryövätty ryövätä ryövätään ryövää ryöväävät

29. + RÄKSYTTÄÄ `räksyt{t∅}{ä∅e} /v` räksytetty räksytetään räksytä räksytän räksytti räksyttivät räksyttäessä räksyttänyt räksyttävät räksyttää

30. + RÖKITTÄÄ `rökit{t∅}{ä∅e} /v` rökitimme rökitettiin rökitetty rökitettävä rökitämme rökitti rökittivät rökittäisi rökittänyt rökittävät rökittää

For results 4, 5, 8, 12 and 20 also a better solution is present in the list of all solutions. The incorrect entries passed the test because there happened to be some misspelled tokens (shown as small caps) forms of other verbs (shown in san serif) or nominals (shown as *emphasized*) which fitted those entries but not to the correct ones. The false results 12 and 20 would have been avoided if there were no typos in the corpus. In the absence of the misspelled words, the fase entries would have failed because the correct one also generates the same set of word forms (plus many others). The list of word forms in false results 17 and 18 are almost exclusively nouns. The false results 4, 5 and 8 contain each a set of forms from two different common verbs. The data hints that one ought to have some method of weighing the goodness of competing candidate entries. If some entry convincingly accounts a word form, that word form could be excluded from the lists of other entries.