Frankie Robertson

Morphological Parsing with Lexical Transducers: A case study of OMorFi

Bachelor's Thesis in Information Technology

May 31, 2016

University of Jyväskylä

Department of Mathematical Information Technology

Author: Frankie Robertson

Contact information: frankie.r.robertson@student.jyu.fi

Title: Morphological Parsing with Lexical Transducers: A case study of OMorFi

Työn nimi: Morfologinen jäsentäminen transduktorien avulla: tapaustutkimus OMorFista

Project: Bachelor's Thesis

Page count: 34+0

Abstract: This thesis explores the task of morphological parsing, which is going from a written word to a representation of the units of meaning making up the word. The research objective is to investigate morphological parsing of Finnish with lexical transducers through a case study of OMorFi (Open Morphology for Finnish). The thesis also presents some linguistic and mathematical background as well as some techniques for constructing FSTs (Finite-State Transducers). The main results are an exposition and some analysis of OMorFi's paradigms, stubs & stems language model, some comparison with related work and ideas for potential future work.

Keywords: Natural Language Processing, Finnish Morphology, Morphological Parsing, Finite-State Transducers, Lexical Transducers

Suomenkielinen tiivistelmä: Tämän kandityön tarkoituksena on tutkia morfologista jäsentämistä, eli prosessia, jonka seurauksena kirjoitettu sana esitetään sanan muodostavina merkitysyksiköinä. Tutkimustehtävänä on tutkia Suomen kielen morfologista jäsentämistä transduktorien avulla hyödyntäen tapaustutkimusta, jonka kohteena on OMorFi (Open Morphology for Finnish). Lisäksi esitellään lingvististä ja matemaattista taustaa sekä äärellistilaisten transduktorien rakennustekniikoita. Pääasialliset tulokset ovat OMorFin kielimallin kuvaus ja analyysi sekä vertailu muihin vastaavanlaisiin projekteihin ja jatkotutkimusmahdollisuuksien hahmottelu.

Avainsanat: kieliteknologia, Suomen kielen morfologia, morfologinen jäsentäminen, äärellistilaiset transduktorit

List of Figures

List of Tables

Contents

1 Introduction

Finnish is a language with a rich morphology, which can prove a barrier in the creation of computer programs to process it. Incremental developments since the generative morphology of Chomsky & Halle (1968) have resulted in an efficient and computationally tractable model of morphology with lexical transducers. My research objective is to investigate how the task of morphological parsing of Finnish can be accomplished using lexical transducers through a case study of OMorFi (Open Morphology for Finnish). I will also investigate applications and related work. My choice of topic is partially motivated by an interest in the acquisition of Finnish in non-native speakers due to my own attempts at learning Finnish. I approach my research objective from the perspective of the following research questions:

- How can we go about constructing a lexical transducer for a language with a complex morphology? In particular, how does OMorFi do it for Finnish?
- What can we use lexical transducers for?
- What potential related future work is there in this area?

The thesis will take the form of a literature review focussed around a case study. A case study is research conducted on an individual rather than a population. It can be considered an inductive technique (Mills 2010, pp. 457-459) in that we might hope to learn things about the whole population from the individual. According to Schramm (1971, p. 6), a common feature of case studies is that they investigate sets of decisions, the reasons they were made, the manner in which they were implemented, and their results. Thus, in engineering disciplines, a case study of a system tells us about the particular design choices made by the system, but might also hint at fundamental trade-offs in the problem domain. In this case the individual system is a piece of software, namely, OMorFi. Conducting a case study on OMorFi therefore can tell us generally about the construction of lexical transducers, linguistic data processing and natural language processing software as well as something about the design choices and trade offs available with this type of system.

One reason that OMorFi is a good candidate for a case study is that it is an open source project. As such it is possible to study it not only through running it, but also by reading its source code. Studying OMorFi's source code will allow me to see how it fits together and get an idea of how it has been designed. In the process, I hope to identify the main stages, or the "hows", of constructing a lexical transducer, as well as look in detail so as to discover the key domain specific problems that drive the overall design of a system and explain the "whys".

Adams, Khan, Raeside & White (2007, pp. 56-57) lay out a framework consisting of three types of literature review: evaluative, exploratory, and instrumental. In terms of this framework, my thesis is primarily an exploratory review, in that its aim is to discover what knowledge exists in this area. It also shares something in common with an instrumental review, in that the specific application of OMorFi is used as a starting point to access the literature rather than starting with a research topic per-se. As it is primarily an exploratory literature review, the aim is to focus on a particular research topic. At the same time, it's important to make sure to show how the topic relates to the larger picture. A metaphor here is that of the human vision, where only the centre is totally in focus but there are more things in less detail around the edges. The vision around the edge of the eye helps to contextualise what is in focus. I identify the edges in this case as including exploration of applications and related and potential future work.

2 Linguistic background

In linguistics, morphology is the study of word forms and word formation (Matthews 1991, p. 3). Within the field of morphology, there are a number of different models corresponding to a variety of different types of analysis which are performed for different purposes or reflect different views of language. There are continuing fundamental debates in linguistics about how best to break down, abstract and analyse language; whether there exist some natural, universal set of definitions and what they might be (Aitchison, 2012, pp. 33-34; Lyons, 1968, pp. 196-197). However, for our purposes the definitions of Matthews (2007), form a reasonable starting point:

- A *morpheme* is a minimal unit of grammar into which a sentence or a word within a sentence can be divided. For example in Finnish we could analyse "puhun" as made up of the root morpheme "puhu" and the first person singular suffix morpheme "–n".
- A *word* is the smallest of the units that make up a sentence, and marked as such in writing. For our purposes, we take the definition of a word as that which is delimited by whitespace or punctuation.
- A *lexeme* is a word considered as a lexical unit, in abstraction from the specific forms it takes in specific constructions. One way of thinking of a lexeme is as being a set of all inflections of a dictionary form of a word. The lexeme is identified by its dictionary entry which consists of a *lemma*, also known as a headword, and a homonym ordinal.

Morphological parsing, morphological segmentation or morphological analysis is the task of going from a word to a more structurally rich representation within the above framework. In particular, we usually want to extract the lemma and a series of *tags*, describing the part of speech (e.g., verb) and a grammatical analysis of how the lexeme has been inflected. In an agglutinative language like Finnish, in contrast to a fusional language like Latin, there is a close relationship between the list of morphemes and the set of tags (Matthews 1991, pp. 107-114). The list of all tags

available for inclusion in an analysis and the rules for their formatting form a data format called a tag set. In Finnish, for example, a tag could represent the case (e.g., inessive "-ssa") or the presence of an enclictic particle (e.g., interrogative "-ko") if it is present. Morphological generation is the opposite process: that of going from a lexeme and some set of tags to a word form.

As well as inflection, morphological analysis might need to also consider processes of lexical derivation[1]. This is when new lexemes, with new distinct meanings, are formed from old ones. Here we consider two forms: morphological derivation and word compounding. Morphological derivation is the process of a new lexeme being formed by affixing a morpheme to an existing lexeme, possibly changing the part of speech in the process. A Finnish example is the verb "ajaa" (en: to drive), and the agent morpheme "-ja" (Karlsson 1999, pp. 232-243) forming the agent noun ''ajaja" (en: driver). Derivational morphemes are described as productive when they can be attached to more words of a particular part of speech, and result in a word which is understandable and considered valid by a native speaker (Matthews 1991, p. 69). Word compounding occurs when multiple words are attached to form new lexemes. In Finnish, both types of lexical derivation happen in a mutually recursive manner and can mix with inflection. For example, given a compound word, each internal word can be inflected, and can have already undergone lexical derivation itself (Karlsson 1999, pp. 232-243).

Derivation is traditionally considered distinct from inflection: inflection is said to be a grammatical phenomena, whereas derivation is said to occur within the lexicon. It should be mentioned that the distinction is based on multiple criteria and may not always be entirely clear cut. Matthews (1991, p. 43) gives a number of edge cases and arguments against the distinction, gradually bringing in more criteria to deal with the various edge cases. Given the distinction is made on not entirely solid ground, alternative models have been formulated, such as Booij (1996), who presents a tripartite model. An example of an edge case is the English word "sands", which has the sense *desert*, and thus here "-s" behaves more like a derivational morpheme

[1]Also referred to as word formation.

than an inflectional morpheme. In applied linguistics, the distinction can be made on a pragmatic basis: we shouldn't expect to find inflected forms in a normal dictionary[2], but we would expect to find at least the most common derived forms.

The rest of this chapter deals with the limits of the usefulness and the range of applicability of tagging word derivation in morphological parsing. A major restriction on usefulness is that knowing the derivation of a word isn't sufficient on its own to reliably predict the word sense (Matthews 1991, pp. 67-69). We might be able to get some idea of word sense, but we are not able to obtain the exact word sense in the general case. The primary force restricting the range of applicability is the difficulty of encoding the rules that govern when word derivations are possible. These can include phonological and semantic restrictions, but it is also possible to construct cases that need an appreciation of subtle contextual cues which probably reaches the level of being AI-hard (Raymond 1991).

One reason to consider lexical derivation in morphological parsing is that it may allow us to recognise and parse new words by guessing a possible derivation. For example, if we have no knowledge of the word "ajaja", and don't know what part of speech it is, but know "ajaa" is a verb and "-ja" makes verbs into nouns, then we can guess that "ajaja" is in our lexicon and that it might be a noun. In the absence of a perfect set of rules for when a certain derivation is allowed, the degree to which we're willing to allow derivations might depend on our application. For example, for Optical Character Recognition (OCR) and spell checking applications we might prefer to guess less, so as to avoid the possibility of introducing false positives.

Another reason to take account of lexical derivation is that it can provide useful information for our application. For example, since an agent morpheme also exists in English ("-er"), rule-based machine translation from Finnish to English could parse "ajaja" into "aja" and "-ja", look them up in a bilingual dictionary to convert them into "drive" and "-er" and then generate "driver", rather than having to explicitly have ajaja $\mapsto$ driver in our bilingual dictionary. However, the problem of predicting word sense prevents this from working in the general case. Unlike the previous,

[2]For Finnish, however, we might expect to find their inflection class.

rather convenient example, many derivational processes produce a lexeme which is distinct from the sum of its parts, for example, in Finnish "lohikäärme" refers to the same concept as dragon in English but is formed from the lexemes "lohi", (en: salmon) and "käärme", (en: snake). While it's a reasonable enough way to form a new word, in terms of semantics the combination of salmon and snake is a fairly crude approximation of the idea of a dragon. It's quite unlikely that an English speaker with no knowledge of Finnish would get the sense "dragon" from the literal translation "salmon snake".

In addition to the word sense problem, the evolution of the lexicon can introduce structural irregularities. Meanings can diverge or lexemes can disappear from usage entirely (Matthews 1991, pp. 54-59). Extending the agent noun example further, in English[3] the affixes which make agent nouns are "-er", "-or" and "-ist". However, for the word "author", the root "auth" is no longer in the English lexicon, but rather we have to look at the word's etymology to find that in Latin it was formed from "augeō", meaning to increase, spread or enrich, and the Latin agent morpheme "-tor".

The rules under which certain forms of morphological derivation can occur can be extremely irregular, and the attachment of less productive derivational morphemes can be governed by complex, context dependent rules. An irregular example which would probably be impossible to encode in a rule-based system is the English derivational suffix "-age" which forms nouns as in roughage, baggage and wordage. In normal current English usage, this suffix isn't particularly productive to the point where many less frequently used dictionary words ending "-age" now sound quite archaic, but in 1970s surfer slang it can be extremely productive, for example, "suckage" or "grindage". There are many difficult to interpret, subtle contextual cues a computer program would have to interpret to know when this type of derivation is allowed. Thus, being able to account for all types of derivation completely accurately is probably either prohibitively difficult or AI-hard.

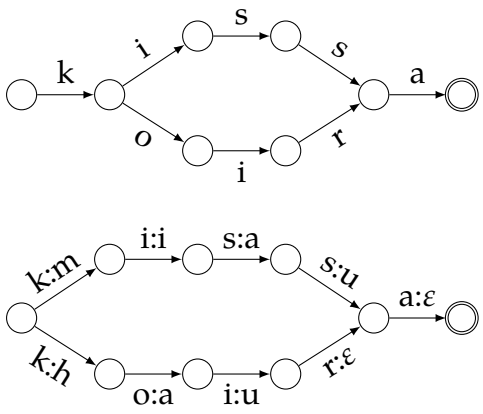
[3]This is due to my limited knowledge of Finnish, but these phenomena occur across languages.

3 Finite-state transducers

A Finite-State Transducer (FST) can be understood as an extension of a Finite-State Automaton (FSA). Each construction has an equivalent graphical form. Figure 1 shows an analogy between an FSA and an FST, and should give some intuition for those familiar with FSA. Formally, as in Roche & Scheabes (1997, p. 14), an FST can be defined as 6-tuple $(\Sigma_1^*, \Sigma_2^*, Q, i, F, E)$ such that:

- Σ_1 is a finite set, the input alphabet;
- Σ_2 is a finite set, the output alphabet;
- Q is a finite set of states;
- $i \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final states; and
- $E \subseteq Q \times Q \times \Sigma_1 \times \Sigma_2$ is the set of edges (or state transitions).

Here, the Kleene star (*) is a function from sets of symbols (also called alphabets) to sets of strings (also called languages). This definition admits non-determinism since E is a general relation rather than a function, and since this definition includes ε-transitions – transitions which have the empty string as their input. We can view an FST as representing a relation $L(T)$ between Σ_1^* and Σ_2^*. One way to define this relation is by taking the transitive closure of the FST's edges, simultaneously accumulating



An FSA defining the regular language: { koira, kissa } (a set of strings)

An FST defining the regular relation: { (koira, hau), (kissa, miau) } (a relation of strings)

Figure 1. An example of, and an analogy between, an FSA and an FST.

input and output strings with concatenation, and then considering the resulting input and output strings on the resulting edges which connect the initial state to a final state. This relation is called a regular relation, by analogy with regular expressions. By definition, every regular relation has an equivalent FST.

Here, rather than the string transducer defined above, we consider primarily letter transducers, which can be defined as the 6-tuple $(\Sigma_1 \cup \{\varepsilon\}, \Sigma_2 \cup \{\varepsilon\}, Q, i, F, E)$. There is no loss of power from using a letter transducer rather than a string transducer since there is an algorithm which transforms an arbitrary string transducer to a letter transducer representing the same relation $L(T)$. Weighted versions of FSTs and letter transducers can be defined, where each edge and final state is additionally assigned a weight. As in Ésik & Kuich (2009, p. 70), given a semiring, we can define the weight of a pair in $L(T)$ in terms of its paths as:

$$\bigoplus_{\text{path} \in \text{paths}} \left(\bigotimes_{\text{edge} \in \text{path}} \text{edgeWeight}(\text{edge}) \right) \otimes \text{finalStateWeight}(\text{path})$$

In morphological applications, often the min-plus Tropical semiring, where $\oplus \mapsto \min$ and $\otimes \mapsto +$ (Droste & Kuich 2009, p. 7-8), is used. In this case, if edges are weighted with negative log probabilities, the weight of a pair in $L(T)$ is the negative log of the joint probability of the path with the greatest joint probability.

Weighted and unweighted FSTs are widely used in the field of natural language processing, where they prove to be a versatile tool for tasks including information extraction (Hobbes, et al. 1997), speech recognition (Pereira & Riley 1997), spelling correction (Pirinen 2014), spelling to phonetics which is used in text to speech applications (Laporte 1997), named entity recognition (Kokkinakis, et al. 2014), and our focus here, morphological analysis and morphological generation. Transducers which can perform these last two tasks are known as lexical transducers. They relate strings containing natural language words, which are said to be in surface form, to strings representing an analysis in what's referred to as lexical form.

As a concrete example, OMorFi's FinnTreeBank 3.1 (FTB3.1) lexical transducer relates the Finnish language to a set of strings containing tags in the FTB3.1 tag set. The

transducer's relation includes the following pair:

Surface form	Lexical form
puhun	puhua V Prs Act Sg1

We can then interpret the lexical form as a space delimited list of tags as defined in Voutilainen, Purtonen & Muhonen (2012), like so:

Tag	Interpretation
puhua	The lexeme is puhua
V	The part of speech is verb
Prs	The tense is present
Act	The voice is active
Sg1	The person is first person singular

One application of a lexical transducer is at the first and last stages of shallow-transfer Rule Based Machine Translation (RBMT)[1]. At the first stage, morphological analysis is performed in the source language, say, Finnish. The shallow-transfer stage then operates on the words in lexical form, performing tasks such as looking up lemmas in a bilingual dictionary, converting case endings to prepositions and reordering words. At the beginning of the final stage we might have a string in lexical form in the target language, say, English. We can then perform morphological generation in this language. Since lexical analysis might output more than one result for a particular word form, there is generally a disambiguation step where the most likely analysis is chosen. This step corresponds to part of speech tagging and can use statistical and/or machine learning techniques, for example the hidden Markov model based Trigrams'n'Tags tagger (Brants 2000)), or rule-based techniques such as Constraint Grammar (Karlsson 1990), or a mixture of both (Hulden 2011). Lexical transducers can also be used as part of Statistical Machine Translation (SMT)[2].

There are a few different ways to perform spell checking with finite-state techniques.

[1]For example, Apertium (Forcada, et al. 2011).

[2]For example, Google Translate or Moses (Koehn, et al. 2007).

One reasonably simple way, as outlined in Pirinen (2014) and used in Voikko v4.0.2[3], is to make an FSA from a lexical transducer by discarding its output side. At this point weights can be added according to word probability, which can be estimated using word frequency statistics. Another transducer which corrects errors, referred to as an error model, is then constructed. The idea is to model the process of errors as a regular relation and then invert the corresponding FST to get an error correcting FST. A simple error model relates strings separated by a Levenshtein distance (a.k.a. edit distance) of one (Levenshtein 1966). A word is correctly spelt only if it is accepted by the FSA. We can then generate a list of suggestions for a misspelt word in order of likelihood by feeding the output of the error correcting FST into the FSA. This set up can be improved using more sophisticated error models such as a model with weights representing the probability of the error occurring, which could for example, take into account keyboard layout. The error model and FSA are not composed in advance since this will result in a prohibitively large transducer (Pirinen 2014, p. 69).

Another application area is hyphenation. Liang (1983) presents this problem and a solution in depth. Placing hyphens in certain places when splitting a word over two lines in justified text, such as on a compound boundary, a morpheme boundary, or a syllable boundary, results in more readable text. This is a relatively early application of FSTs, and Liang's implementation may also be one of the most widespread applications in terms of number of users, since it is part of the TEX typesetting system.

One simple but extremely practical pair of applications are the closely related tasks of lemmatisation and stemming (Lovins 1968). A word form is lemmatised by extracting the lemma or stemmed by discarding any inflection data leaving a stem which identifies the lexeme. The task of lemmatisation can be seen as a subset of morphological parsing. This is immediately useful in the field of Information Retrieval (IR) for performing the task of full-text search. Before a document is indexed, it is preprocessed by either lemmatising or stemming all the words. Then, before performing a query, each keyword is stemmed. As a result, an end user will obtain results equivalent to having searched for all combinations of all inflected forms of their keywords.

[3]This information was obtained by inspection of its source code.

4 Languages and tools for transducers

This chapter surveys the available tools and techniques for the construction of lexical transducers. Just as regular expressions are a language for describing regular languages, there are languages to describe regular relations. Languages describing regular relations can be compiled into finite-state transducers analogously to the compilation of regular expressions into finite-state automata. Roughly, there are three types of constructs upon which tools and languages to generate FSTs can be based:

- context-sensitive replacement rules,
- a textual description of the structure of an FST, and
- unary and binary operations on regular relations or FSTs.

Context-sensitive replacement rules were introduced in Chomsky & Halle (1968), where they were used to transform written English into phonetic English. There, the rules are written as follows:

$$A \quad \rightarrow \quad B \quad / \quad X \text{——} Y$$

This can be read as "A is replaced with B when it has left context X and right context Y". For example, applying this rule would transform XAY to XBY. A set of context-sensitive rewrite rules can be interpreted as accepting a language (Partee, et al. 2012, p. 450). This language will be a context-sensitive language, which, in terms of the Chomsky hierarchy (Chomsky 1956), is strictly more powerful than a context-free language, which is in turn strictly more powerful than a regular language. At first glance then, it seems impossible to compile context-sensitive replacement rules to an FST, which is a strictly less powerful formalism.

Douglas (1970) discovered that the way Chomsky & Halle and subsequent phonologists had used the rewrite rules in phonological derivations was actually more restricted. They assumed that once a rewrite rule has been applied, its output can not be affected by subsequent reapplication. It turns out that this restriction is sufficient

to allow a context-free rewrite rule to admit an equivalent regular relation[1]. The algorithm for compiling a Chomsky & Halle style replacement rule into an FST was first presented in Kaplan & Kay (1981), but the first publication available to the general public beyond an abstract was Kaplan & Kay (1994).

Concurrently with these developments, Koskenniemi (1984) formulated a different type of context-sensitive replacement rule, with clear operational semantics from the start. In Chomsky & Halle's scheme, rules are applied in series, with the output of each rule acting as the input to another: they are combined by composition (Roche & Scheabes 1997, pp. 21-23). Two level replacement rules operate in parallel: they are combined by intersection (Roche & Scheabes 1997, pp. 23-25)[2]. Another difference is that in Chomsky & Halle rules the context can only depend on the input string whereas in two level rules the context can also depend on the output string.

There have been a series of compilers for Koskenniemi's formalism. Probably the two most advanced are the implementation in the Xerox/PARC finite-state toolkit[3] which, along with the rest of the toolkit, is closed source but freely available for non-commercial use, and hfst-twolc which is part of the Helsinki Finite-State Toolkit (HFST) and, along with the rest of HFST, is licenced under the GPLv3. Chomsky & Halle style rules are implemented in xfst (Xerox Finite-State Tool) with the additional feature of the context having access to the output string.

The next way to construct an FST is to describe its structure directly by giving an encoding of the 6-tuple specified in chapter 3. Lexc (Lexicon Compiler) is a tool using this technique. The body of a lexc input file consists of a number of named *lexicon* sections, which correspond to states in an FST. Each lexicon contains a series of records corresponding to state transitions. Each record contains an input string, an output string and the name of another lexicon, which corresponds to the out-state. There are

[1]Conversely, Roche (1997, pp. 244-247) shows how the task of top down parsing, which corresponds to accepting a context-free language, can be performed by iterating the function realised by an FST.

[2]In general, letter transducers are not closed under intersection, but ε-free letter transducers are. The two levels move in lockstep, letter by letter, and differences in length are handled by adding padding characters called zeros.

[3]Distributed with, and described extensively in Beesley & Karttunen (2003).

two special lexicons named *Root* and #, which correspond to the starting state and an accepting state respectively. Lexc compiles this text description of a string transducer to an encoding of a letter transducer. Implementations of lexc exist in the Xerox and Helsinki toolkits. The typical usage of lexc is to encode the lexicon by listing lemmas and other morphemes, and to express the morphotactical matter of which pairs of morphemes can attach to each other. The resulting transducer is then composed with context-sensitive replacement rules, which express the morphophonemic matter of how attached morphemes cause sound changes in each other.

Another way to directly express the structure of an FST is using the extended regular expressions first introduced in xfst. These are an extension of POSIX style regular expressions (IEEE 2013). A character, such as “a”, will produce an edge with “a” as its input and output. A colon separated pair, “a:b”, produces an edge taking “a” to “b”. HFST includes extended regular expressions in its implementation of xfst, and as an independent tool: hfst-regex.

Finally, we come to unary and binary operations on regular relations or FSTs. Intersection and composition have already been mentioned as methods of combining different types of rule transducers and composition as a way of combining the lexicon transducer and the rules transducer. For two level rules, the intersection of rules and composition with the lexicon can be performed with a single compose-intersect operation to save memory (Karttunen, et al. 1992). Of the remaining operations, many are best understood as operations between regular relations, such as the invert and union/disjunct operations, or sometimes as combining two regular languages into a regular relations, such as the Cartesian product. Some operations operate at the level of FSTs, such as the minimise operation, which outputs an FST with a smaller or equal number of edges and states compared to the input FST, without affecting the corresponding regular relation. These operations can be performed from within xfst and are also implemented within HFST as independent command line tools where they can be combined by using Unix-style pipes.

5 Case study: OMorFi

OMorFi (Open Morphology for Finnish) uses the tools introduced in the previous chapter, together with linguistic rules and data, to build a number of lexical transducers (Pirinen 2015)[1]. Python, bash shell scripts and the GNU autotools are used for automation. OMorFi's databases, however, represent the majority of the work which has gone into OMorFi. OMorFi can be understood as a lexical data processing pipeline, as visualised in figure 2 (p. 15). This chapter analyses OMorFi in terms of this pipeline, following it from its input to its final product: from upstream sources of linguistic data to a number of lexical transducers performing different analyses.

5.1 Lexical data processing pipeline

Finnish, like most natural languages, has a large, varied and evolving lexicon. Even native speakers of a natural language only know a fraction of the total lexicon. In fact, it is impossible to fully enumerate every lexeme in a natural language since new words can always be introduced – it's possible to coin a new word that will be readily understood, for example by using lexical derivation, as explained in chapter 2.

For the purposes of morphological analysis we need a list of lemmas and other morphemes and an *inflection class* for each lemma to determine how it combines with other morphemes. OMorFi acquires this information by *screen scraping* from existing sources of linguistic data. Screen scraping is the process of transforming data meant for human consumption to structured data in a machine readable format. Since OMorFi is distributed under the GNU Public Licence (GPL) version 3, each source of data must be licenced under a compatible licence.

[1]Here I have cited the most recent publication related to OMorFi, but this chapter operates at a closer level of detail than is available in the literature. Most of the content of this chapter comes from reading the OMorFi's source code and experimenting with it. The version of OMorFi used was its git master branch as of the 19th of February 2016, available at this URL: `https://github.com/flammie/omorfi/commit/e832e931ea4b530e56b12b968c443f3e24e44df1`.

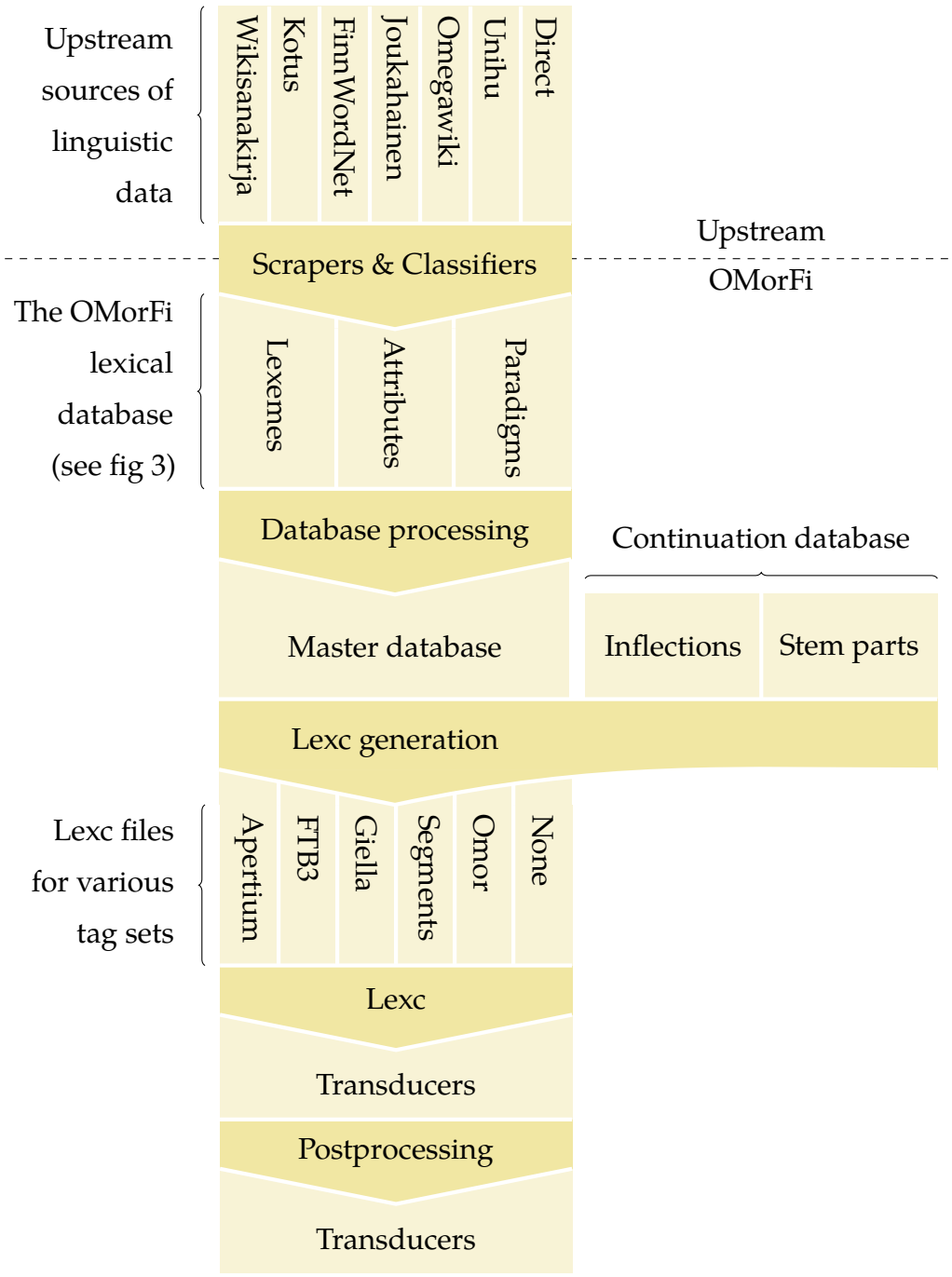


Figure 2. The data processing pipeline leading from upstream (external to OMorFi) sources of lexical data to lexical transducers

Kotimaisten Kielten Keskus, Kotus, (en: Institute for the Languages of Finland) have published *Nykysuomen Sanalista*, NSSL, (en: The new Finnish word list) (Kotus 2007). In NSSL each word is given one of 79 inflection classes. NSSL classes 1-51 are for nominals, the part of speech including nouns and adjectives, and classes 52-78 are for verbs. Class 99 is for words which don't inflect, including the adverbs[2]. If a word participates in consonant gradation, it is labelled with one of 13 types. NSSL doesn't distinguish between the direction of the consonant gradation, nor does it supply the type of vowel, front or back, endings should take[3].

Joukahainen is a freely editable online dictionary started to serve as a source of data for Voikko, a Finnish language spell checker predating OMorFi (Pitkänen 2006). It has a similar system to NSSL, except information about consonant gradation is included in the inflection class and inflection classes are referred to by example words. *Wikisanakirja* is the Finnish language version of *Wiktionary*, a freely editable online dictionary which also uses Kotus inflection classes.

FinnWordNet is a Finnish translation of the Wordnet database, which contains semantic relations between words (Lindén & Carlson 2011). It doesn't contain inflection class information, but most of the words in OMorFi from FinnWordNet are either compound words, for which we can make a reasonable guess by using the inflection class of the final subword, or the result of morphological derivation in which case the particular type of derivation might be sufficient to guess the inflection class. *Omegawiki* is a user editable dictionary, which unlike Wikisanakirja, tries to group by meaning across languages. As with FinnWordNet there is no inflection class information and so it must be obtained by automatic guessing informed by linguistic knowledge of Finnish. *Unihu* is a project of the University of Helsinki which hasn't yet received a publication. Finally, there is the *Direct* source, which is for lexemes which have been entered into OMorFi manually.

OMorFi has 2482 word paradigms for all parts of speech. Since there are only 79 Kotus inflection classes, it follows that each word paradigm encodes more information than

[2]Kotus inflection classes are labelled 1-78 and 99.

[3]The agreement between vowels in a stem and an ending is known as vowel harmony.

an inflection class. As well as inflection class, each OMorFi paradigm encodes: the type of consonant gradation, similar to Joukahainen; whether the vowel harmony is front or back, or whether either is possible, as with some loan words; and the part of speech. Ultimately the word paradigm must determine which characters are deleted from the end of the lemma to form the *stub*. This is the initial part of the word which remains unchanged by inflection. The *morphophonology* table maps each paradigm to a more sparse representation, explicitly stating a value for each paradigm and phenomenon, for example, whether vowel harmony is front or back. The *stub-deletions* table contains the part of the lemma that is not a part of the stub, that is, the ending of the lemma. The notion of the stub here is distinct from the stem as referred to in descriptions of the Finnish grammar such as Karlsson (1999). The stem is the part of the word which remains unchanged in some subset of inflectional endings whereas the stub doesn't change between the lemma form and any inflected form.[4]

All upstream data must be mapped onto one of OMorFi's classes by its classifiers. The classifiers begin by guessing a number of features, such as categorising the lemma's vowel harmony as front or back based on its final syllable. Then, based on a combination of these features, Kotus inflection class, Kotus gradation type and the ending of the lemma, a paradigm is chosen for the lexeme. On top of the data encoded in paradigms, there is data stored on a per word basis in the *attributes* tables. An important example is the *boundaries* table which contains internal word boundaries for compound words.

All of the tables in figure 3 are joined to obtain the *master table*. It should be noted at this point that OMorFi's database tables are stored as tab separated values text files, and operations, such as join, are implemented manually in Python, rather than using a relational database management system as might be inferred from the terminology. Due to Codd (1970), the relational database model can be defined in terms of sets (or multisets), independently from any implemention details. A key advantage of

[4]For example, for nominals, there is a common stem for genitive, plural and most case endings. For the word "politiikka" this is "politiika-". We can then directly attach, for example, the genitive ending "-n" to the stem. In this case the stub is "politii-". To form the genitive, we must first attach the missing part of the genitive stem "-ka-" and then the genitive ending "-n".

OMorFi's approach is the interoperability it affords with Unix-style tools, for example git, diff, awk and grep, which are designed primarily to operate on text files.

OMorFi has two of its own tag sets. The *omor* tag set, for which a lexical transducer is produced, is intended for public consumption. The *stuffs* tag set is internal and is used to generate tags for every tag set OMorFi supports from its continuation database. The continuation database consists of the *stems* and *inflections* tables. The stems table maps the 3-tuple of a paradigm, an analysis in terms of stuffs, and a *stem part*, to a set of *continuation classes*. The stem part, when affixed to the stub, will produce the stem as defined in Karlsson (1999). The inflections table has a similar structure to the

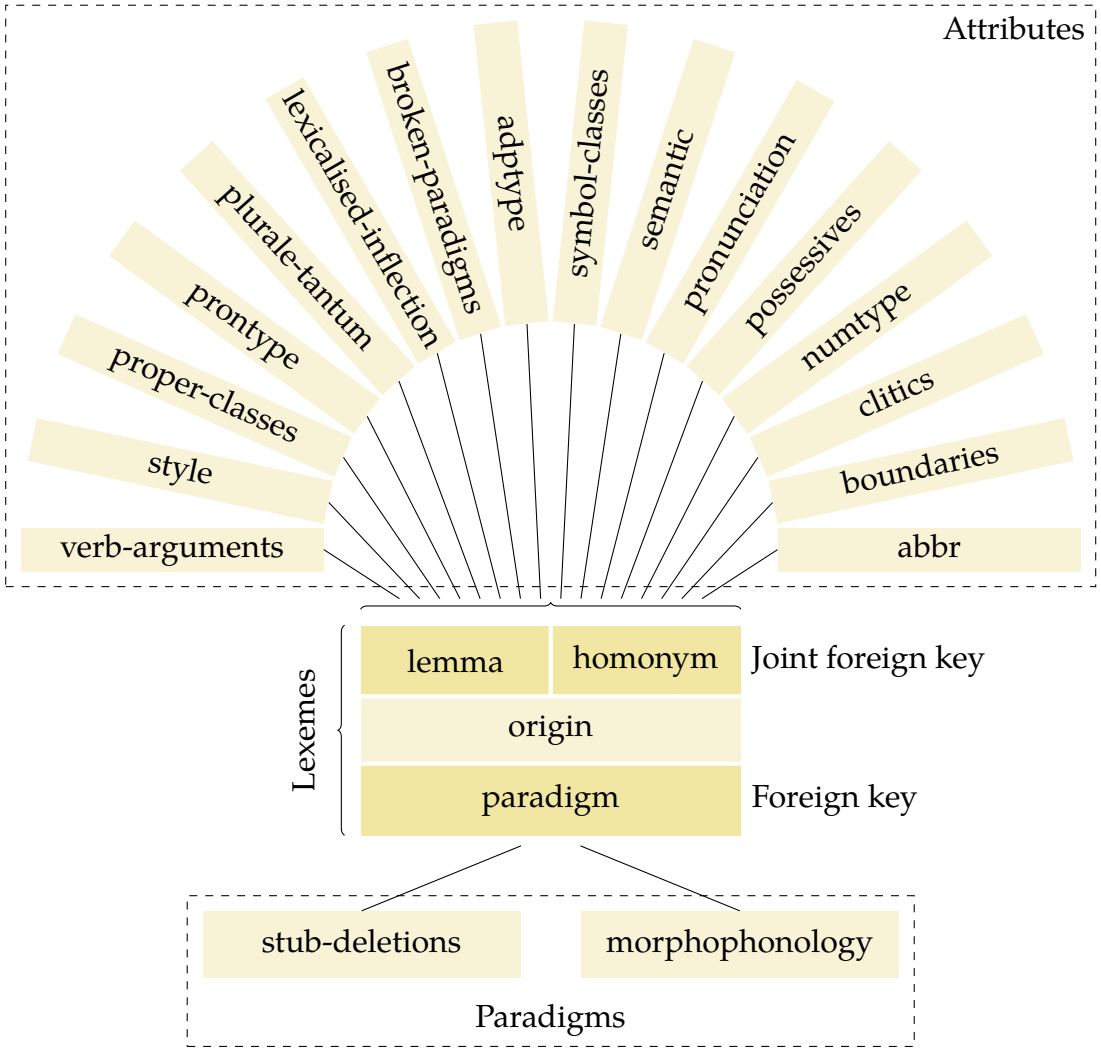


Figure 3. The tables which make up OMorFi's lexical database

stems table. Instead of mapping from paradigms, it maps from continuation classes. To deal with compound words, the inflection table can also map to parts of speech. The stem part column in these two tables contains markers delimiting morpheme boundaries and word boundaries in compound words.

The first part of generating the final set of lexical transducers is to generate a lexc file for each output tag set. Each lexc file is a combination of the of the master, stems and inflections tables. In each lexc file, the root lexicon points to all the part of speech lexicons. The table below shows how each row of of each of these database tables maps to records in the generated lexc file:

Table	Lexicon	Input	Output	Continuations
Master	Part of speech	Stub	The lemma & a part of speech	The lexeme's paradigm
Stems	Paradigm	Stem part	Converted stuffs	Continuation classes
Inflections	Continuation class	Stem part	Converted stuffs	Continuation classes

Table 1. A mapping from rows in OMorFi's tables to records in generated lexc files

This transducer produced from this lexc file still needs markers delimiting morpheme boundaries in its input. To obtain a transducer which can deal with word forms as they appear in text we must make another transducer which attempts to insert these boundaries between every character in the input word and compose it with the transducer produced by lexc. This is realised in OMorFi using a twolc file which disjunctions a series of replacement rules from ε to each of the boundary markers. There are other post processing steps implemented with replacement rules. For example allowing "š" to be spelt as "sh'[5]. Some output transducers, such as the transducer for the FinnTreeBank 1 tag set, are implemented by replacing the tags output by another transducer. None of these replacement rules are context-sensitive.

[5]As it is in many English words, and as is now common in Finnish in loan words such as "shakki".

5.2 Analysis of design choices

OMorFi is a rule-based, data driven tool for tasks involving Finnish morphology, including morphological analysis and generation. As in section 5.1, the current version of OMorFi is based almost entirely on lexc, with twolc and hfst-regex being used only for simple search and replace tasks, rather than for encoding morphophonemic rules. OMorFi has a thin layer of abstraction on top of lexc. By construction, descriptions written in the twolc or xfst languages can be compiled to FSTs. As in chapter 4, the structure of a lexc file has a one to one correspondence to an FST. Thus, anything describable with either twolc or xfst is describable with lexc alone.

Why doesn't OMorFi use twolc or xfst? It's certainly surprising since Koskenniemi's two level system was designed with Finnish in mind (Koskenniemi 1984, pp. 42-88). Older versions of OMorFi did use twolc[6] to deal with phonological phenomena such as vowel harmony and consonant gradation[7]. One reason might be that while fairly good coverage was possible with the combination of lexc and twolc, encoding a set of interacting rules which dealt with the general case became prohibitively difficult. For example, vowel harmony is reasonably easy to encode for Finnish words, but loan words which contain both front and back vowels can take endings with both. These types of edge cases may have meant that the twolc-based approach was no longer tenable. This would be an instance of the 80/20 rule of engineering, where the last 20 % of a project takes 80 % of the effort. Another related reason might be that since the structure of a lexc file has a close correspondence to the structure of the final transducer, relying mostly on lexc makes it easy to track the size of the final transducer. This is analogous to choosing to program in a lower level language at the cost of certain types of abstraction because of predictable performance characteristics.

OMorFi's inflections table contains around 3000 entries and its stems table contains around 28 000 entries for its roughly 2500 paradigms. Each paradigm must determine

[6]Such as a version from 2009, which is available here: `https://github.com/flammie/purplemonkeydishwasher/tree/master/2009-sfcm`.

[7]Since consonant gradation can't be determined from the word form on its own, strings coming from the lexicon transducer into the rules transducer are tagged with their gradation class.

the set of inflections a lexeme can take and the particular form they must take. Many paradigms are identical apart from one small difference in the form of a particular suffix[8]. Based on these facts and the lack of usage of context-sensitive replacement rules, OMorFi can be characterised as more data-driven versus a possible rule-driven approach, which would make heavier usage of context-sensitive replacement rules. One question then is: Are there disadvantages of this representation, and if so, can they be remedied?

As a Finnish learner, I am quite interested in the possibility of certain types of human-computer interactive language learning. This application area is known as Computer Aided Language Learning (CALL)[9]. One potential application here is a question/answer system, such as Oahpa! (Antonsen, et al. 2009), which includes automatic question generation and answer checking for the Sámi languages. Koskenniemi (2006, pp. 428-429) gives an example of a system which generates questions asking a language learner to give a particular inflection of a lexeme, incorporating morphophonemic information from two level rules by using the pair-test utility. In his example, the user gives an answer which is almost correct, apart from not incorporating the result of a single phonological rule. The CALL software can then make decisions based on this information, such as give positive feedback ("almost right") and give information about the rule the learner failed to incorporate along with examples, as well as asking more questions where the rule needs to be taken into account in the future.

Is Koskenniemi's technique possible with OMorFi's current paradigms, stubs and stems approach? Certainly it doesn't seem to be as easy as in his example, but perhaps at least some of this functionality of modelling the mistakes of language learners could be recovered by making a series of error transducers[10]. These could be constructed by changing OMorFi's continuation database in ways which model

[8]For example, gray (as in the SI unit of radiation) can be put in the illative by attaching "-hyn" or "-hin" or "-ihin" but reggae can be put in the illative by attaching "-en", "-hen", "-hin" or "-ihin". Even though the paradigms inflect identically apart from this, the fact that they differ for a single inflection means they necessarily belong to different paradigms.

[9]For background see, for example, (Levy 1997).

[10]As in spell checking.

different types of errors and then compiling new, incorrect versions of OMorFi based on these. Another CALL application would be an information retrieval system on OMorFi's lexicographic database. A language leaner might like to be able to find out the inflection class of a new word, but OMorFi has too many paradigms for a language learner. For example, it has 237 verb paradigms. We might be able to remedy this and retrieve the 5 verb types taught to Finnish learners by coalescing paradigms into larger classes by, for example, merging otherwise identical front and back paradigms and ignoring certain rare inflections.

Currently OMorFi supports analysing derived forms which are not in its lemmas database for a number of deverbal suffixes and the "-sti" deadjectival suffix, but this is a subset of possible types of morphological derivation in Finnish[11]. Analyses produced this way include the lemma and part of speech as they are before undergoing derivation, along with a tag for the derivational morpheme. OMorFi also contains a mechanism to include *lexicalised* derivation. This is where individual entries are tagged as having undergone a derivational process. In this case, OMorFi will output the word form and part of speech as it is after derivation.

[11]Older versions of OMorFi have supported a slightly different set of types of morphological derivation, including "-ja'.

6 Conclusions

Dealing with the internal structure of words can present problems in natural language processing. These problems are most pronounced for synthetic languages like Finnish, which have a high morpheme to word ratio. For these languages, a full form lexicon, where all possible forms of every lexeme are listed, is not tenable. OMorFi's approach emphasises lexical data and employs classical data management and processing techniques to control complexity. It builds on freely available tools based on decades of incremental advances in the theory of finite-state language processing, resulting in an efficient solution with good coverage of the Finnish language. In this chapter, I will first point to some related pieces of work and briefly characterise them. I will close by outlining some potential opportunities for future work in this area.

6.1 Related work

Hunspell is an open source spell checker, written with Hungarian, a distant linguistic cousin of Finnish in the Uralic language family[1], in mind. Its dictionaries use an interpreted language which has evolved to accommodate different languages. This lacks the clean mathematical basis and generality of the finite-state approach. Voikko is another open source spell checker, but written with Finnish in mind. Voikko began life as a Hunspell dictionary, but quickly ran up against the limitations of the system. Later versions were based on Suomi-malaga, which implements morphological parsing in terms of parsing a context-free grammar. This approach is general enough to deal with the same morphological phenomena as FSTs, but context-free grammars are a needlessly powerful formalism. Current versions of Voikko are based on FSTs.

OMorFi and Voikko are separate projects, but have had a mutually beneficial relationship. The Joukahainen database is a product of Voikko's development, and the improvements in finite-state tooling via the concurrent development of HFST with OMorFi (Koskenniemi 2008, pp. 92-93) enabled the current finite-state based Voikko

[1]For background on the Uralic languages see, for example, Abondolo (2015).

backend. Since Hunspell dictionaries can be converted to Voikko (Pirinen, T., & Lindén 2010) and Voikko integrates with many pieces of software such as OpenOffice and Firefox, it's well positioned as a replacement for Hunspell.

There are a number of open source morphological analysers for other languages in the Uralic family. For Hungarian there is the Hunmorph analyser (Trón, et al. 2005) and the Morphdb.hu database (Trón, et al. 2006). These project share a similar relationship to OMorFi and Joukahainen. Hunmorph uses a similar technique to Hunspell, based on affix stripping, but as the authors note in Trón, et al. (2005), the linguistic data could be reused in a finite-state morphological analyser. For Estonian, Pruulmann-Vengerfeldt (2010) has implemented a finite-state description as part of his masters thesis[2]. The previously mentioned Oahpa! is part of a larger project at the Giellatekno Center for Sámi Language Technology at the University of Tromsø (Antonsen, et al. 2009), which has produced linguistic applications and resources for the Sámi languages, including morphological analysers based on HFST.

A recent piece of work is FinnPos (Silfverberg, et al. 2015). While the software is designed to be modular enough to apply to other tasks, it is currently distributed with data and scripts that allow it be to used as an extension of OMorFi. One direction in which OMorFi is extended is the addition of morphological disambiguation. This implementation is at its core an averaged perceptron, as introduced in Freund & Schapire (1999). The other direction in which OMorFi is extended is the addition of probabilistic lemmatisation of unknown words.

6.2 Potential future work

As described in section 5.1, OMorFi's lexical database is created by scraping and classifying upstream linguistic data. However, this description is of an idealised version of OMorFi. The data in OMorFi's database as of writing has first been scraped and then undergone conversion from an old database format. Additionally, only a subset of the upstream sources of linguistic data have publicly available scrapers.

[2]The implementation is available from `https://github.com/jjpp/plamk`.

OMorFi could include a direct route from upstream linguistic data into its lexical database. This would enable the addition of new lexemes and the modification of old lexemes upstream to enter OMorFi's lexical database. This work might need to be accompanied by a conflict resolution strategy.[3]

As mentioned in section 5.2, a CALL system based on OMorFi is a potential future application. Another application would be as part of a machine translation pipeline, as outlined in chapter 3. OMorFi could be improved by addressing the shortfalls of its derivation system given at the end of section 5.2. The different formats for analyses of lexicalised and non-lexicalised derivation could be made more consistent with a post processing step. As mentioned in chapter 2, encoding restrictions on derivation can be tricky, but without sufficient restrictions on new types of derivation, the system will *overgenerate*, that is generate word forms which are ungrammatical. To account for new types of restrictions on derivation, it may be necessary to introduce new criteria beyond the per-paradigm and per-lexeme ones already in OMorFi.

A general line of work in open linguistic resources is the effort to try and share resources and enable cooperation between different research groups and across languages by harmonising the results of linguistic analyses and data exchange formats, and increasing interoperability of tools[4]. This is one of the achievements of the HFST project, which is implemented on top of and interoperates with a number of existing tools. Cross language resource sharing is a major motivation for the Universal Dependencies project (Nivre 2015). This work has already begun to be integrated into OMorFi in the form of changes to the omor tag set to include the Universal Part Of Speech (UPOS). The aim of the UPOS work is to make sure people use the same set of a parts of speech across different languages consistently. As we've seen, there are a large number of different tools and analyses representing a number of opposing approaches, and the rules and exceptions of natural language make converging on universal truths extremely difficult. Since this topic involves overcoming both these challenges, it seems there will be work to be done in this area for quite some time.

[3] An example of a scenario in which a conflict would occur is the addition of a Wiktionary entry with a Kotus class which disagrees with the paradigm of a word already in OMorFi's lexical database.

[4] An overview of this topic is given in Witt, et al. (2009)

Bibliography

Abondolo, D. (2015). Introduction. In *The Uralic Languages* (pp. 1-42). Abingdon, Oxon, England: Routledge.

Adams, J., Khan, H. T., Raeside, R., & White, D. I. (2007). *Research methods for graduate business and social science students.* SAGE Publications India.

Aitchison, J. (2012). *Words in the mind: An introduction to the mental lexicon* (4th ed.). New York, NY: John Wiley & Sons.

Antonsen, L., Huhmarniemi, S., & Trosterud, T. (2009). Interactive pedagogical programs based on constraint grammar. *Proceedings of the 17th Nordic Conference of Computational Linguistics* Nealt Proceedings Series Volume 4 (pp. 10-17). Oslo, Norway: North European Association for Language Technology.

Antonsen, L., Gerstenberger, C., Trosterud, T. & Wiechetek, L. *Sámi Language Technology at the University of Tromsø.* Poster presented as part of the Made for Arctic Languages: Syntax, Morphology, Lexicon course at the University of Tromsø. Retrieved from `http://giellatekno.uit.no/background/giellatekno3.pdf`.

Beesley, K., & Karttunen, L. (2003). *Finite state morphology.* Stanford, CA: Center for the Study of Language and Information.

Booij, G. (1996). Inherent versus contextual inflection and the split morphology hypothesis. *Yearbook of morphology 1995* (pp. 1-16). Dordrecht, Netherlands: Springer Netherlands.

Brants, T. (2000). TnT: a statistical part-of-speech tagger. *Proceedings of the sixth conference on Applied natural language processing* (pp. 224-231). Stroudsburg, PA: Association for Computational Linguistics.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory* Volume 2, Issue 3 (pp. 113-124). Piscataway, NJ: IEEE.

Chomsky, N., & Halle, M. (1968). *The sound pattern of English.* Harper & Row.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM* Volume 13, Issue 6 (pp. 377-387.). New York, NY: ACM.

Johnson, C. D. (1970). *Formal Aspects of Phonological Description.* (Doctoral dissertation). CA: University of California.

Droste, M. & Kuich, W. Semirings and Formal Power Series. In *Handbook of weighted automata* (pp. 1-28). Berlin, Germany: Springer Science & Business Media.

Ésik, Z. & Kuich, W. Finite Automata. In *Handbook of weighted automata* (pp. 69-104). Berlin, Germany: Springer Science & Business Media.

Forcada, M.L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J.A., Sánchez-Martínez, F., Ramírez-Sánchez, G. & Tyers, F.M. (2011). Apertium: a free/open-source platform for rule-based machine translation. *Machine translation* Volume 25, Issue 2 (pp. 127-144). Berlin, Germany: Springer Science & Business Media.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning* Volume 37, Issue 3 (pp. 277-296). Berlin, Germany: Springer Science & Business Media.

Hobbs, J. R., Appelt, D., Bear, J., Israel, D., Kameyama, M., Stickel, M., & Tyson, M. (1997). FASTUS: A Cascaded Finite-State Transducer for Extracting Information from Natural-Language Text. In *Finite-state language processing* (pp. 383-406). Cambridge, MA: MIT Press.

Hulden, M., & Jerid F. Boosting statistical tagger accuracy with simple rule-based grammars. *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12).* Istanbul, Turkey: European Language Resources Association (ELRA).

Regular Expressions. In *The Open Group Base Specifications Issue 7* Chapter 9. The IEEE and The Open Group. Retrieved from `http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html`.

Kaplan, R. M., & Kay, M. (1981). Phonological rules and finite-state transducers. *Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting* (pp. 27-30).

Kaplan, R. M., & Kay, M. (1994). Regular models of phonological rule systems. *Computational linguistics - Special issue on computational phonology* Volume 20, Issue 3 (pp. 331-378). Cambridge, MA: MIT Press.

Karlsson, F. (1990). Constraint grammar as a framework for parsing running text. *Proceedings of the 13th conference on Computational linguistics* Volume 3 (pp. 168-173).

Stroudsburg, PA: Association for Computational Linguistics.

Karlsson, F. (1999). *Finnish: An Essential Grammar.* Abingdon, Oxon, England: Routledge.

Karttunen, L., Kaplan, R. M., & Zaenen, A. (1992). Two-level morphology with composition. *Proceedings of the 14th conference on Computational linguistics.* Volume 1 (pp. 141-148). Stroudsburg, PA: Association for Computational Linguistics.

Kokkinakis, D., Niemi, J., Hardwick, S., Lindén, K., & Borin, L. (2014). HFST-SweNER–A New NER Resource for Swedish. *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14).* Reykjavik, Iceland: European Language Resources Association (ELRA).

Kotimaisten Kielten Keskus (2007). *Nykysuomen Sanalista*. Retrieved from `http://kaino.kotus.fi/sanat/nykysuomi/`.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., et al. (2007). Moses: Open source toolkit for statistical machine translation. *Proceedings of the 45th annual meeting of the ACL, interactive poster and demonstration sessions* (pp. 177-180). Prague, Czech Republic: Association for Computational Linguistics.

Koskenniemi, K. (1984). A general computational model for word-form recognition and production. *Proceedings of the 10th international conference on Computational Linguistics* (pp. 178-181). Stroudsburg, PA: Association for Computational Linguistics.

Koskenniemi, K. (2006). Notes on the Two-Level Morphology. *A man of measure: Festschrift in Honour of Fred Karlsson on His 60th Birthday – A special supplement to SKY Journal of Linguistics* Volume 19 (pp. 422-431).

Koskenniemi, K. (2008). How to build an open source morphological parser now. *Resourceful Language Technology: Festschrift in Honor of Anna Sågvall Hein* (pp. 86-95). Uppsala, Sweden.

Laporte, É. (1997). Rational Transducers for Phonetic Conversion and Phonology. In *Finite-state language processing* (pp. 407-430). Cambridge, MA: MIT Press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady* Volume 10, Issue 8 (pp. 707-710). Russia: Springer Science & Business Media.

Levy, M. (1997). *Computer-assisted language learning: Context and conceptualization.* Oxford, England: Oxford University Press.

Liang, F. M. (1983). *Word Hy-phen-a-tion by Com-put-er.* (Doctoral dissertation). Stanford, CA: Department of Computer Science, Stanford University.

Lindén, K., & Carlson, L. (2010). Finn WordNet-WordNet på finska via översättning. *LexicoNordica* Volume 17 (pp. 119-140).

Lovins, J. B. (1968). Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics* Volume 11, Numbers 1 and 2. Cambridge, MA: MIT Information Processing Group, Electronic Systems Laboratory.

Lyons, J. (1968). *Introduction to Theoretical Linguistics.* Cambridge, England: Cambridge University Press.

Matthews, P. H. (1991). *Morphology* (Cambridge Textbooks in Linguistics). Cambridge, England: Cambridge University Press.

Matthews, P. H. (Ed.). (2007). *The concise Oxford dictionary of linguistics.* Oxford, England: Oxford University Press.

Mills, A. J., Durepos, G., & Wiebe, E. (Eds.). (2010). *Encyclopedia of case study research* Volume 1. Thousand Oaks, CA: SAGE Publications, Inc.

Nivre, J. (2015). Towards a universal grammar for natural language processing. *Computational Linguistics and Intelligent Text Processing* (pp. 3-16). Springer International Publishing.

Partee, B. H., Meulen, A. T., & Wall, R. E. (2012). *Mathematical methods in linguistics.* Berlin, Germany: Springer Science & Business Media.

Pereira, F. C. N., & Riley, M. (1997). Speech recognition by composition of weighted finite automata. In *Finite-state language processing* (pp. 431-453). Cambridge, MA: MIT Press.

Pirinen, T., & Lindén, K. (2010). Creating and weighting hunspell dictionaries as finite-state automata. *Investigationes Linguisticae* Volume 21, (pp. 1-16). Poland: Poznań.

Pirinen, T. (2014).' *Weighted Finite-State Methods for Spell-Checking and Correction.* (Doctoral dissertation). University of Helsinki, Finland.

Pirinen, T. A. (2015). Omorfi – Free and open source morphological lexical database for Finnish. *Proceedings of NoDaLiDa 2015: 20th Nordic Conference on Computational*

Linguistics. Linköping, Sweden: Linköping University Electronic Press.

Pitkänen, H. (2006). *Hunspell-fi in Kesäkoodi 2006: Final report.* Retrieved from `http://www.puimula.org/htp/archive/kesakoodi2006-report.pdf`.

Pruulmann-Vengerfeldt, J. (2010). *Praktiline lõplikel automaatidel põhinev eesti keele morfoloogiakirjeldus.* (Master's thesis). Tartu, Estonia: University of Tartu.

Raymond, E. S. (Ed.). (1991). AI-complete. In *The Jargon File, Version 2.8.2.* Retrieved from `http://catb.org/esr/jargon/oldversions/jarg282.txt`.

Roche, E. (1997). Parsing With Finite-State Transducers. In *Finite-state language processing* (pp. 241-281). Cambridge, MA: MIT Press.

Roche, E., & Schabes, Y. (1997). Introduction. In *Finite-state language processing* (pp. 1-66). Cambridge, MA: MIT Press.

Schramm, W. (1971). *Notes on Case Studies of Instructional Media Projects.* Stanford, CA: California Institute for Communication Research.

Silfverberg, M., Ruokolainen, T., Lindén, K., & Kurimo, M. (2015). FinnPos: an open-source morphological tagging and lemmatization toolkit for Finnish. *Language Resources and Evaluation*. Advance online publication. doi:10.1007/s10579-015-9326-3

Trón, V., Kornai, A., Gyepesi, G., Németh, L., Halácsy, P., & Varga, D. (2005). Hunmorph: open source word analysis. *Proceedings of the ACL-05 Workshop on Software* (pp. 77-85). Stroudsburg, PA: Association for Computational Linguistics.

Trón, V., Halácsy, P., Rebrus, P., Rung, A., Vajda, P., & Simon, E. (2006). Morphdb.hu: Hungarian lexical database and morphological grammar. *Proceedings of 5th International Conference on Language Resources and Evaluation* (pp. 1670-1673). Genoa, Italy.

Voutilainen, A., Purtonen T. & Muhonen, K. (2012). *FinnTreeBank2: Manual*. Retrieved from `http://www.ling.helsinki.fi/kieliteknologia/tutkimus/treebank/sources/FinnTreeBankManual.pdf`.

Witt, A., Heid, U., Sasaki, F., & Sérasset, G. (2009). Multilingual language resources and interoperability. *Language Resources and Evaluation* Volume 43, Issue 1 (pp. 1-14).