Finite State Transducer Calculus for Whole Word Morphology

Maciej Janicki
NLP Group, University of Leipzig
Augustusplatz 10, 04109 Leipzig, Germany
macjan@o2.pl

Abstract

The research on machine learning of morphology often involves formulating morphological descriptions directly on surface forms of words. As the established two-level morphology paradigm requires the knowledge of the underlying structure, it is not widely used in such settings. In this paper, we propose a formalism describing structural relationships between words based on theories of morphology that reject the notions of internal word structure and morpheme. The formalism covers a wide variety of morphological phenomena (including non-concatenative ones like stem vowel alternation) without the need of workarounds and extensions. Furthermore, we show that morphological rules formulated in such way can be easily translated to FSTs, which enables us to derive performant approaches to morphological analysis, generation and automatic rule discovery.

1 Introduction

In computational linguistics, morphological analysis is usually understood as segmenting words into smaller meaningful units, called *morphs*. There exists a well-established computational model for such analysis, called two-level morphology (Koskenniemi, 1983; Beesley and Karttunen, 2003). It models the mapping between the surface forms of words and the morph sequences using handwritten rules, which are compiled to Finite State Transducers. This allows for a composition of lexicon and rules to an efficient morphological analyzer. Examples of such analyzers include Omorfi for Finnish (Pirinen, 2015), Morphisto for German (Zielinski and Simon, 2008) and TRMorph for Turkish (Çöltekin, 2010).

However, the research coming from the machine learning side often requires models that describe string transformations between surface forms directly, without referring to any underlying structures which cannot be observed in the data and are difficult to infer by a learning algorithm. Such transformations can also be described and implemented as finite-state transducers. Despite that, a standardized model of this kind of morphological description seems to be lacking. Instead, many authors develop their own models and implementations for the purpose of a concrete learning algorithm. With some exceptions, the design, implementation and performance of the string processing algorithms is usually not described in detail and the approaches used for that are sometimes suboptimal.

In this paper, we present a finite-state computational model of string transformations on surface forms based on a linguistic theory called Whole Word Morphology. We first review research on machine learning of morphology which motivates the need for such a model (Sec. 2). In Sec. 3, we describe the formalism and its linguistic foundations, and in Sec. 4, we present the implementation of the formalism within the FST calculus. Sec. 5 contains a procedure for automatic rule discovery from unannotated data, while in Sec. 6, we measure the performance of our implementation of the model.

2 Motivation and Related Work

The recent research on machine learning of morphology tends more and more often towards models describing transformations on whole words, instead of representing words as concatenations of morphs. Arguably the most important reason for this is that morph boundaries are often not clearly visible in surface forms due to morphophonology and orthography.[1] In the following, we review

[1]This was also the reason for the emergence of two-level morphology. However, two-level morphology was de-

Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing, pages 37–45
Dresden, Germany, September 23-25, 2019.

some of the papers utilizing such transformational models. Our focus here is not the learning algorithm (which is usually the main focus of the respective paper), but the assumed model of morphology, together with its linguistic and computational foundations.

(Neuvel and Fulop, 2002) present a computational model of morphology based on Whole Word Morphology (Ford et al., 1997). Morphology is described in terms of patterns which summarize structural similarities and differences between pairs of words. The patterns consist of constant elements and wildcards: for example, the relationship within the pair (*receive*, *reception*) would be expressed as $/X\text{ceive}/ \leftrightarrow /X\text{ception}/$. In order to discover such rules automatically, the authors use rather simple string processing algorithms: they try matching every word to every other and check whether the beginnings or the ends of the words match. They subsequently compute an alignment by anchoring the words either at their beginning or end.

(Wicentowski, 2002) proposes a transformational model designed for learning mappings between inflected forms and lemmas. It is based on splitting words into seven parts and describing the changes in each part separately. In addition to prefixation and suffixation, it aims to cover phenomena such as internal vowel changes or changes at the boundary between stem and prefix/suffix (e.g. *hop* $\sim$ *hop**p**ing*), which are attributed to a separate segment. (Lindén, 2008, 2009) likewise attempts to model the transformation between base and inflected form part by part, but adopts a simpler, three-way split into prefix, stem and suffix. (Lindén, 2009) mentions that the model was implemented as a cascade of Finite State Transducers.

(Botha and Blunsom, 2013) propose a model of morphology aimed at capturing especially the templatic morphology found in Semitic languages. The model is based on Simple Range Concatenating Grammars (SRCGs), which are a mildly context-sensitive class of formal grammars. It is thought as an extension of the purely concatenative model, which can be represented by a context-free (or perhaps even regular) grammar.

signed with the goal of efficient implementation of handwritten grammars and, despite some research in this direction (Theron and Cloete, 1997; Koskenniemi, 2013), is rather not suitable for the machine learning scenario.

(Durrett and DeNero, 2013) and (Ahlberg et al., 2014) present two different approaches to learning inflection from complete paradigm tables. The input data in such setting are lists of tuples (b, w, t), where b is the base word (lemma), w the inflected word and t a tag, i.e. a bundle of inflectional features. An important point of learning algorithms for this task is an appropriate model of string transformations from b to w. (Durrett and DeNero, 2013) use a semi-Markov log-linear model to model the probability of application of individual transformations (like prefix, stem or suffix change) independently, while (Ahlberg et al., 2014) model string transformations on whole words in form of patterns with wildcards. We note that the string transformation model of (Durrett and DeNero, 2013) is tightly coupled to the machine learning method applied by the authors, while the model of (Ahlberg et al., 2014) is more general and independent of the classification method (in this case, memory-based classification).

With works like (Soricut and Och, 2015; Narasimhan et al., 2015; Luo et al., 2017), we can observe a shift from segmentation to word-based string transformations also in the area of unsupervised learning of morphology. Currently, they appear to adopt very simple transformation models that only involve affixation. On the other hand, (Janicki, 2015) and (Sumalvico, 2017) present a probabilistic model suitable for unsupervised learning, which is based on Whole Word Morphology and describes morphology in terms of whole-word transformation patterns.

As a conclusion from the above literature review, we recognize a need for a standardized model of morphological relationship between surface forms of words. As most of the models presented above are motivated by the need to cover non-concatenative phenomena, especially internal vowel changes and Semitic templatic morphology, the model we aim at should be able to handle those phenomena in a natural and general fashion. Following (Neuvel and Fulop, 2002), we see Whole Word Morphology as the right linguistic foundation for such formalism, and following (Lindén, 2008, 2009), we consider FSTs to be the right tool for implementing string transformations efficiently. Thus, the contribution of the present paper is twofold:

1. A formal definition of a transformational model of morphology, similar to the ones em-

ployed by (Neuvel and Fulop, 2002; Ahlberg et al., 2014; Janicki, 2015),

2. An implementation of the model based on the FST calculus.

3 The Formalism

3.1 Definitions

We base our formalism on the linguistic theory of Whole Word Morphology (henceforth, WWM) introduced by (Ford et al., 1997; Neuvel and Singh, 2002). It models structural similarities in form and meaning between words in form of rules, which are expressed as patterns containing wildcards. For example, the relationship within the French pair (*chanteur*, *chanteuse*) can be expressed by the following rule:[2]

$$/X\text{œr}/_{\text{N.MASC}} \leftrightarrow /X\text{øz}/_{\text{N.FEM}} \quad (1)$$

In the above rule, X denotes a variable which can be instantiated with any string of phonemes and represents the common part of both words. The units inside slashes refer to whole words in their surface forms.

In general, we express a *morphological rule with n variables* as follows:

$$/a_0 X_1 a_1 \dots X_n a_n/ \mapsto /b_0 X_1 b_1 \dots X_n b_n/ \quad (2)$$

The elements a_i and b_i are constants (literal strings), which usually represent the differing parts of words on the left-hand and right-hand side of the rule.[3] The elements X_i are variables (wildcards), which represent the part that is preserved by the rule, but varies from pair to pair. Additionally, the following conditions must be satisfied:

1. The variables must be retained in the same order on both sides of the rule.

2. For $0 < i < n$, either a_i or b_i has to be non-empty.

[2]The example comes from (Ford et al., 1997), which is a linguistic monography, thus it represents words in form of phonemic transcriptions. All further examples use written representations.

[3]However, the constants a_i, b_i for a given position i do not have to differ. By being equal or containing a common part, they might also represent the context necessary for the rule to apply. For example, in the rule $/X\text{ate}/ \mapsto /X\text{ation}/$, both constants contain the common prefix 'at'. Formulating this rule as $/X\text{e}/ \mapsto /X\text{ion}/$ would correspond to the same string transformation, but would extend its coverage to a few further cases, like (*deplete*, *depletion*).

Because of the first condition, we can represent such rule as a vector of $2n + 2$ strings: $\langle a_0, a_1, \dots, a_n, b_0, b_1, \dots, b_n \rangle$.

Contrary to (1), which is a relational description and thus uses a bidirectional arrow, we formulate our rules as having a privileged direction. Although most rules can be applied in both directions, the productivity of back-formation is mostly much lower, so that specifying a direction seems linguistically plausible. Modeling rules which are similarly productive in both directions can be achieved by including the reverse rule separately in the grammar.

As an illustration of (2), the rule expressing the relationship between the German pairs (*singen*, *gesungen*), (*klingen*, *geklungen*), (*trinken*, *getrunken*) could have the following form:

$$/X_1\text{i}X_2/ \mapsto /\text{ge}X_1\text{u}X_2/ \quad (3)$$

The rule could also contain more constant elements to express the necessary conditions for its application:

$$/X_1\text{in}X_2\text{en}/ \mapsto /\text{ge}X_1\text{un}X_2\text{en}/ \quad (4)$$

With each rule, we can associate a function r, which transforms a word fitting to the left-hand side of the rule into a set of corresponding words fitting to the right-hand side:

$$r(v) = \{b_0 x_1 b_1 \dots x_n b_n : x_1, \dots, x_n \in \Sigma^+ \wedge v = a_0 x_1 a_1 \dots x_n a_n\} \quad (5)$$

Note that the outcome of the rule application is a *set* of words, rather than a single word. In general, the rule application might result in multiple different words, because there might be different ways of splitting the word into the sequence $a_0 X_1 a_1 \dots X_n a_n$. For example, the application of the rule $/X_1\text{a}X_2/ \rightarrow /X_1\text{ä}X_2\text{e}/$ to the German word *Kanal* results in the set: $\{$*Känale*, *Kanäle*$\}$. In case the word does not fit to the left-hand side of the rule, the rightmost condition is never fulfilled and the result is an empty set. Thus, the function r is defined on the whole of Σ^+.

3.2 Coverage of Morphological Phenomena

In addition to covering affixation, circumfixation and stem vowel alternations, as shown already in the previous section, the following further morphological phenomena can be handled by the formalism:

Templatic morphology. A relationship between pairs like the Arabic (*kataba*, *kutiba*) can be generalized as the following rule:

$$/X_1\text{a}X_2\text{a}X_3\text{a}/ \mapsto /X_1\text{u}X_2\text{i}X_3\text{a}/ \quad (6)$$

Although the formalism does not provide a way to restrict the instantiations of variables to a single consonant, it could be easily extended to express such restrictions on variables in a form similar to regular expressions.

Compounding. The proponents of Whole Word Morphology and similar theories explicitly reject the analysis of compounds as 'words composed of multiple words' (Singh and Dasgupta, 2003; Starosta, 2003). In consequence, compounds are also analyzed as related to a *single* word, while the other part is considered to be a morphological constant. For example, the English word *blackberry* would be related to *black* via the following rule:

$$/X/_{\text{N/ADJ}} \mapsto /X\text{berry}/_{\text{N}} \quad (7)$$

According to (Singh and Dasgupta, 2003; Starosta, 2003), the relationship between the a rule like (7) and the word 'berry' is purely etymological and thus not a part of a synchronic description of morphology. This claim is supported by the fact that newly coined compounds (in languages that exhibit compounding) virtually always involve at least one part that is already known as 'compound-forming', rather than combining two arbitrary words. Indeed, in morphological analyzers based on two-level morphology, the cyclicity used to model compouding often causes massive overgeneration.

4 WWM Rules as FSTs

A rule defined as in (2) can be easily converted to an FST. The general scheme for that is given in Fig. 1. The arrows represent concatenation and each rectangular block represents a transducer. There are two kinds of blocks: transducers mapping corresponding constants, like $a_0 : b_0$, and transducers representing the variables. The latter are simply identity transducers accepting Σ^+. Figure 2 shows a concrete FST corresponding to the rule (4).

4.1 Analysis

There is no concept of a 'morphological analysis' in WWM. Each word is treated as an independent unit of language. However, given a word, we might be interested in its structural relationships to other words.

Let R be the set of rules found in the morphology of a language of interest and let T_r be a transducer corresponding to rule r. The disjunction of all rules, T_R, yields a transducer accepting morphologically related pairs:

$$T_R = \bigcup_{r \in R} T_r \quad (8)$$

Further, let V denote a vocabulary and T_V the identity transducer corresponding to V. With the following composition, we obtain a transducer capable of mapping all words from V to all their possible derivations:

$$T_A = T_V \circ T_R \quad (9)$$

T_A can be called a 'WWM analyzer'. A lookup of an unknown word v in T_A yields all words from the known vocabulary from which v can be derived. Furthermore, a three-way composition $T_V \circ T_R \circ T_V$ gives us all pairs of related words from V.

4.2 Generation

Another common question of morphology is: Given a vocabulary V and a set of rules R, what further words can be postulated? The identity transducer for such new words, T_N, is obtained from the following formula:

$$T_N = T_A \downarrow \setminus T_V \quad (10)$$

where $T_A \downarrow$ denotes the output projection of T_A and $\setminus$ denotes subtraction.

5 Automatic Rule Discovery

As shown in Sec. 3, our definition of rule is general enough to capture many morphological phenomena, including some important non-concatenative ones. On the other hand, the resulting computational model is simple enough to allow for completely unsupervised rule discovery without prior linguistic knowledge. In this section, we show how to achieve this in two stages: first, we identify pairs of string-similar words in the vocabulary. Then, we extract candidate rules from each such pair. Frequent patterns are good candidates for rules, which can be passed to a further statistical model, like the one of (Janicki, 2015; Sumalvico, 2017).

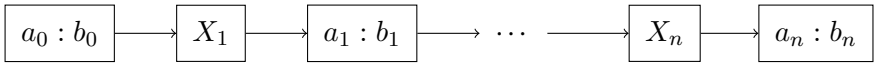

Figure 1: A scheme for converting morphological rules into FSTs.

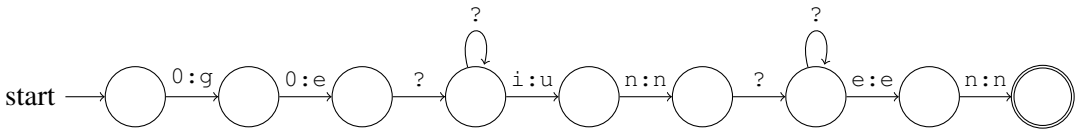

Figure 2: The transducer corresponding to rule (4).

5.1 Finding Pairs of Similar Words

A plausible and widely used string similarity measure is edit distance (Levenshtein, 1966). Using the Fast Similarity Search algorithm (Bocek et al., 2007), we are able to identify pairs of words with edit distance at most k without comparing each word to every other. The algorithm works by generating a *deletion neighborhood* of each word, consisting of strings that can be obtained from that word by deleting up to k characters. The resulting list of pairs (word, substring) is sorted according to the substring. Observe that words with edit distance $\leq k$ are guaranteed to share a common substring, although words sharing a common substring might also have edit distance $> k$. Thus, we treat pairs of words sharing a common substring as candidates, for which edit distance has to be computed with usual means.

For the purpose of discovering potential morphological rules, it is reasonable to modify the notion of edit distance. Firstly, morphological rules usually operate on groups of consecutive letters, rather than single letters independently, so deletion or substitution of a segment of consecutive letters should yield higher similarity than deletion or substitution of the same number of non-consecutive letters. Secondly, although we are going to permit word-internal alternations, more change should be permitted at the beginning and at the end of words, since that is where most morphological rules operate. Bearing in mind the representation (2), let l_{affix} denote the maximum length of a morphological constant at the beginning or the end of a word (a_0, b_0, a_n, b_n in (2)), l_{infix} the maximum length of a morphological constant inside the word (a_i, b_i for $0 < i < n$ in (2)) and $k_{\max}$ the maximum number of variables. In order to generate pairs which are related by a rule satisfying this constraint, we obtain the following constraints on a deletion environment: deleting up to l_{affix} consecutive letters at the beginning and end of the word, and up to l_{infix} consecutive letters in at most $k_{\max} - 1$ slots inside the word. The usual setting for those parameters, which covers a vast majority of morphological rules encountered in practice, is $l_{\text{affix}} = 5, l_{\text{infix}} = 3, k_{\max} = 2$.

Such settings allow for deletion of up to 13 letters in total, so that even for middle-length words it would consider all pairs to be similar. In order to prevent this, we introduce an additional constraint: the total amount of deleted characters must be smaller than half of the word's length. In this way, we can consider long affixes, but only if enough of the word is still left to form a recognizable stem.

With all those constraints, computing a deletion neighborhood of a word becomes a complex operation. It is therefore helpful to visualize and implement it using transducers. We will construct the transducer S mapping words to their deletion neighborhoods as a composition of two simpler transducers: $S = S_1 \circ S_2$. The transducer S_1 (Fig. 3) performs the deletions, substituting a special symbol δ for each deleted character. The transducer consists of segments, corresponding to the deleted sequences: states 0-5 represent the prefix, 10-15 the suffix and 7-9 the infix. Between each pair of segments, an arbitrary number of identity mappings is performed (state sequences 5-6 and 9-10). The epsilon transitions, for example from states 0-4 to 5, correspond to a less-than-maximum number of deletions in a given slot. It can easily be seen that changing e.g. the parameter l_{affix} simply corresponds to altering the length of the top and bottom chains, just as l_{infix} correspond to the length of the middle chain and $k_{\max} - 1$ to the number of such middle chains.

The transducer S_2 (Fig. 4) takes the output of S_1 and checks whether the number of deletions is smaller than the number of remaining characters. As the general formulation of this problem cannot be solved by a finite-state machine, it requires

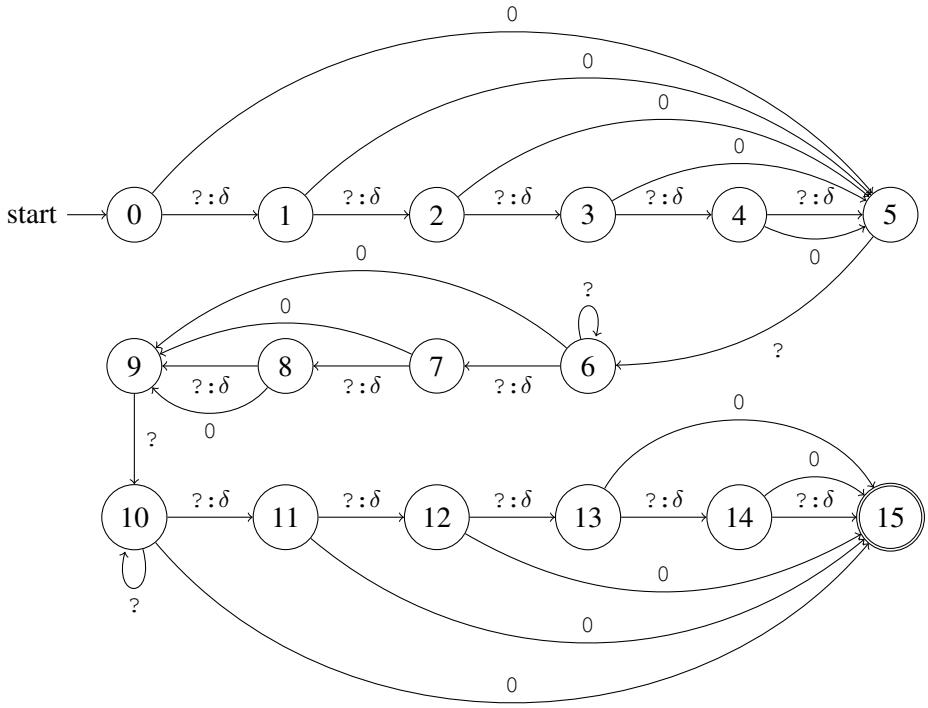

Figure 3: The transducer S_1 for generating a deletion neighborhood.

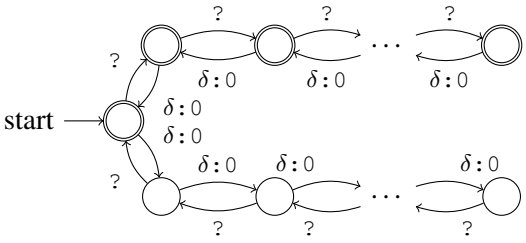

Figure 4: The filter S_2 ensuring that no more than the half of a word is deleted.

a bound on word length. In my implementation, I restrict the maximum word length to 20 characters, but it is easy to change this parameter. The states of S_2 correspond to the difference between the number of letters and the number of deletions seen so far. The states above the initial state correspond to positive, and the ones below to negative values. Furthermore, S_2 removes the deletion symbols and returns the substring consisting of the remaining letters.

We can now generate all pairs of similar words from a lexicon automaton L by performing the following composition:

$$P = (L \circ S) \circ (L \circ S)^{-1} \tag{11}$$

There are various ways to implement this in practice. Computing the composition directly is usually not feasible because of high memory complexity. One possibility is to use S for substring generation, but otherwise proceed as in the original FastSS algorithm: store the words and substrings in an index structure, either on disk or in memory, then retrieve words for each substring. Another possibility is to use S to generate substrings for a given word and then look the substrings up in the transducer $(L \circ S)^{-1}$ to obtain similar words. The latter composition can be computed statically. We additionally convert the resulting transducer to HFST optimized lookup format (Silfverberg and Lindén, 2009). While the lookup approach is still significantly slower, it has an advantage in providing a way to retrieve *all* words w' similar to a given word w at once. It is thus better suited for parallelization, especially in case the pairs (w, w') are subject to further processing.

5.2 Extraction of Rule Candidates

Given a pair (w, w') of string-similar words, we want to extract morphological rules modeling the difference between those words. For this purpose, we first align the words on character-to-character basis using the well-known dynamic programming

algorithm for computing edit distance (Wagner and Fischer, 1974). Then, we attribute each character mapping either to a morphological constant or a variable, in a way that fulfills the constraints on l_{affix}, l_{infix} and k. The candidate rules are constructed incrementally while iterating over the alignment and unfinished rules are stored in a priority queue. In case an aligned character pair can be attributed either to a constant or to a variable, both possibilities are stored in a queue, so that at the end we obtain multiple rules with varying degrees of generality. For example, the rules extracted from the German pair (*trifft*, *getroffen*) include $/X_1\text{i}X_2\text{t}/ \rightarrow /\text{ge}X_1\text{o}X_2\text{en}/$ (the most general rule), as well as e.g. $/X\text{ifft}/ \rightarrow /\text{ge}X\text{offen}/$.

Table 1 shows example rules extracted from a word list coming from German Wikipedia. While the top of the list consists entirely of morphological patterns, the bottom of the table shows that patterns resulting from accidental word similarities can also become frequent enough to be confused with morphological rules. Thus, this approach identifies rule *candidates*, which have to be further filtered based on other criteria than mere frequency.

6 Experiments

We have implemented the algorithms described in the previous section using the HFST library (Lindén et al., 2011). Furthermore, we conducted experiments realizing the algebraic operations described in Sec. 4 and the rule discovery procedure described in Sec. 5. The results demonstrate that our model is suitable for building analyzers based on the Whole Word Morphology paradigm and the required computational resources are easily achievable.

First, we run the rule discovery procedure on word lists extracted from German Wikipedia.[4] The generation of pairs of similar words and the subsequent rule extraction is implemented in a parallelized fashion. Table 2 shows the computation times for various sizes of input vocabulary and numbers of processes. The results demonstrate that this step is feasible for input data of as much as 150,000 words (and probably even somewhat larger). In our view, this is enough to discover the vast majority of productive morphological rules.

We disjunct several thousand most frequent rules to construct a rule transducer T_R, which is used in algebraic operations shown in Table 3. Most operations are realized within at most several minutes, the longest one being the construction of the largest generator in slightly above 11 minutes.

Note that the computation times reported in Table 3 are much shorter than the ones in Table 2. Moreover, the former appear to increase linearly in both $|V|$ and $|R|$. Thus, although the limits on the vocabulary size in the rule discovery procedure are quite tight, once we have discovered the rules (or obtained them in another way, e.g. manually written), we can apply the transducer to find pairs of related words in much larger lexica. Using 3-way composition (Allauzen and Mohri, 2008) for computing $T_A \circ T_V$ could probably further improve the analysis of a new lexicon.

7 Conclusion

We have presented a formalism allowing for the description of morphological regularities as transformational patterns on whole words in their surface forms. The formalism is grounded in linguistic theories rejecting the notion of internal structure of words and can be especially useful in the context of machine learning, where descriptions of such underlying structures are not available. It captures non-concatenative phenomena naturally and allows for representing rules as FSTs, so that performant algorithms for morphological analysis and generation are readily available as algebraic operations on transducers. We suggest that such standardized formalism can present an alternative to models of morphology and string processing algorithms developed for a specific machine learning method, which are common in the literature.

[4] Note that unsupervised learning of morphology *per se* is not our focus in this paper. The rule discovery procedure would constitute only a preprocessing step to proper learning. However, we use the resulting rule transducer T_R in further compositions to demonstrate their computational feasibility.

References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2014. Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the EACL*, pages 569–578.

Cyril Allauzen and Mehryar Mohri. 2008. 3-way composition of weighted finite-state transducers. In *International Conference on Implementation and Application of Automata (CIAA 2008)*, pages 262–273, San Francisco, CA, USA.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Fi-*

rank	rule	frequency	example
1	$/X\text{n}/ \rightarrow /X/$	8555	Epoche → Epochen
2	$/X/ \rightarrow /X\text{n}/$	8555	Epochen → Epoche
3	$/X\text{en}/ \rightarrow /X\text{e}/$	7465	aufgehenden → aufgehende
4	$/X\text{e}/ \rightarrow /X\text{en}/$	7465	aufgehende → aufgehenden
5	$/X\text{en}/ \rightarrow /X/$	6030	Abkürzungen → Abkürzung
6	$/X/ \rightarrow /X\text{en}/$	6030	Abkürzung → Abkürzungen
7	$/X\text{e}/ \rightarrow /X/$	5640	niedrige → niedrig
8	$/X/ \rightarrow /X\text{e}/$	5640	niedrig → niedrige
9	$/X\text{s}/ \rightarrow /X/$	4917	Erdbebens → Erdbeben
10	$/X/ \rightarrow /X\text{s}/$	4917	Erdbeben → Erdbebens
	...		
53	$/X\text{en}/ \rightarrow /X\text{t}/$	1194	nutzen → nutzt
54	$/X\text{en}/ \rightarrow /X\text{es}/$	1194	einfachen → einfaches
	...		
746	$/X_1\text{a}X_2/ \rightarrow /X_1\text{o}X_2/$	196	unterbrachen → unterbrochen
747	$/X_1\text{t}X_2/ \rightarrow /X_1\text{m}X_2/$	195	warten → warmen
748	$/X_1\text{n}X_2/ \rightarrow /X_1\text{l}X_2/$	195	Zähnen → Zählen
749	$/X_1\text{m}X_2/ \rightarrow /X_1\text{t}X_2/$	195	warmen → warten
750	$/X_1\text{l}X_2/ \rightarrow /X_1\text{n}X_2/$	195	Zählen → Zähnen
751	$/X_1\text{ge}X_2\text{t}/ \rightarrow /X_1X_2\text{en}/$	195	zugefügt → zufügen
752	$/\{\text{CAP}\}X_1X_2/ \rightarrow /X_1\text{sch}X_2/$	195	Allergie → allergische
753	$/X_1\text{sch}X_2/ \rightarrow /\{\text{CAP}\}X_1X_2/$	195	allergische → Allergie
754	$/X_1\text{ä}X_2\text{er}/ \rightarrow /X_1\text{a}X_2/$	195	Häuser → Haus
755	$/X_1\text{a}X_2/ \rightarrow /X_1\text{ä}X_2\text{er}/$	195	Haus → Häuser
756	$/\text{g}X/ \rightarrow /\text{ausg}X/$	194	gegraben → ausgegraben
757	$/\text{ausg}X/ \rightarrow /\text{g}X/$	194	ausgegraben → gegraben

Table 1: Example rules extracted from the German Wikipedia.

$\lvert V\rvert$	num. processes			
	1	**2**	**4**	**6**
10k	426	238	146	134
50k	8757	5273	4128	3657
100k	33629	22981	18624	15297
150k	82287	55268	41827	37623

Table 2: Computation times (in seconds) for the rule discovery procedure.

Computation	$\lvert V\rvert$	$\lvert R\rvert$			
		1k	**2k**	**5k**	**10k**
$T_V \circ T_R$	10k	2.86	5.72	13.6	22.8
	50k	13.4	28.7	67.2	115
	100k	26.9	56.3	137	237
	150k	41.3	86.6	208	355
$T_A \circ T_V$	10k	0.68	1.17	2.36	3.77
	50k	3.58	5.82	11.4	18.0
	100k	7.21	11.6	22.9	35.3
	150k	10.9	16.8	33.7	51.7
$T_A \downarrow \backslash T_V$	10k	5.39	10.0	23.1	40.4
	50k	28.6	51.9	125	219
	100k	57.9	107	266	452
	150k	88.6	159	397	682

Table 3: Computation times (in seconds) for various operations related to the WWM analyzer. All algebraic operations include the minimization of the resulting transducer.

nite State Morphology. Center for the Study of Language and Information.

Thomas Bocek, Ela Hunt, and Burkhard Stiller. 2007. Fast similarity search in large dictionaries. Technical report, University of Zurich.

Jan A. Botha and Phil Blunsom. 2013. Adaptor grammars for learning non-concatenative morphology. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 345–356, Seattle, Washington.

Çağrı Çöltekin. 2010. A freely available morphological analyzer for Turkish. In *LREC 2010, Seventh International Conference on Language Resources and Evaluation*.

Greg Durrett and John DeNero. 2013. Supervised learning of complete morphological paradigms. In *Proceedings of NAACL-HLT*, pages 1185–1195.

Alan Ford, Rajendra Singh, and Gita Martohardjono. 1997. *Pace Pāṇini: Towards a word-based theory of morphology*. American University Studies. Series XIII, Linguistics, Vol. 34. Peter Lang Publishing, Incorporated.

Maciej Janicki. 2015. A multi-purpose bayesian model for word-based morphology. In *Systems and Frameworks for Computational Morphology – Fourth International Workshop, SFCM 2015*. Springer.

Kimmo Koskenniemi. 1983. *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production*. Ph.D. thesis, University of Helsinki.

Kimmo Koskenniemi. 2013. An informal discovery procedure for two-level morphological rules.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710.

Krister Lindén. 2008. A probabilistic model for guessing base forms of new words by analogy. In *CICling-2008, 9th International Conference on Intelligent Text Processing and Computational Linguistics*, Haifa, Israel.

Krister Lindén. 2009. Entry generation by analogy – encoding new words for morphological lexicons. *Northern European Journal of Language Technology*, 1:1–25.

Krister Lindén, Erik Axelson, Sam Hardwick, Tommi A. Pirinen, and Miikka Silfverberg. 2011. HFST – framework for compiling and applying morphologies. In *Systems and Frameworks for Computational Morphology – Second International Workshop, SFCM 2011*. Springer.

Jiaming Luo, Karthik Narasimhan, and Regina Barzilay. 2017. Unsupervised learning of morphological forests. *TACL*.

Karthik Narasimhan, Regina Barzilay, and Tommi Jaakkola. 2015. An unsupervised method for uncovering morphological chains. *TACL*.

Sylvain Neuvel and Sean A. Fulop. 2002. Unsupervised learning of morphology without morphemes. In *Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 31–40.

Sylvain Neuvel and Rajendra Singh. 2002. Vive la différence! What morphology is about. *Folia Linguistica*, 35(3-4):313–320.

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.

Miikka Silfverberg and Krister Lindén. 2009. Hfst runtime format - a compacted transducer format allowing for fast lookup. In *Finite-State Methods and Natural Language Processing - FSMNLP 2009 Eight International Workshop*.

Rajendra Singh and Probal Dasgupta. 2003. On so-called compounds. In (Singh and Starosta, 2003), pages 77–89.

Rajendra Singh and Stanley Starosta, editors. 2003. *Explorations in Seamless Morphology*. SAGE Publications, New Delhi.

Radu Soricut and Franz Josef Och. 2015. Unsupervised morphology induction using word embeddings. In *NAACL 2015*, pages 1626–1636.

Stanley Starosta. 2003. Do compounds have internal structure? A seamless analysis. In (Singh and Starosta, 2003), pages 116–147.

Maciej Sumalvico. 2017. Unsupervised learning of morphology with graph sampling. In *Proceedings to RANLP 2017*, Varna, Bulgaria.

Pieter Theron and Ian Cloete. 1997. Automatic acquisition of two-level morphological rules. In *Fifth Conference on Applied Natural Language Processing*, Washington, DC, USA.

Robert A. Wagner and Michael J. Fischer. 1974. The string-to-string correction problem. *Journal of the ACM*, 21(I):168–173.

Robert Wicentowski. 2002. *Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework*. Ph.D. thesis, Johns Hopkins University.

Andrea Zielinski and Christian Simon. 2008. Morphisto – an open source morphological analyzer for German. In *Finite State Methods and Natural Language Processing, 7th International Workshop, FSMNLP 2008*, Ispra, Italy.