Computational morphology

Kyle Gorman

Graduate Center, City University of New York

Abstract

This is a draft of a chapter for the third edition of Aronoff and Fudeman's textbook *What is Morphology?*

1 Introduction

Computers are used in virtually all stages of linguistic research, to record and collate field notes, to gathering word frequency statistics, and to measure reaction times in a psycholinguistic experiment. In this chapter, we focus on a narrower sense of **computational morphology**: the design of software that analyzes or generates words not as atomic, indivisible units, but as the intricately structured objects linguists have long recognized them to be.

2 Early work

The linguistic potential of digital computers was recognized not long after their debut in 1945. In the first two decades of the Cold War, the United States Defense Department and its research and development arm, the Advanced Research Projects Agency (ARPA, later known as DARPA) funded massive projects to enable automated translation from Russian to English. Governmental enthusiasm cooled after a 1966 report, commissioned by the government, concluded that little progress towards acceptable-quality translations had been made.

Electronic dictionaries—initially, little more than a digitization of older print dictionaries—appeared in the 1960s and became ubiquitous in the 1980s during the personal computing boom. Since word processing was one of the applications for which early personal computers were best suited, such dictionaries were exploited for spell-checking and typesetting. In scripts like Chinese and Japanese, written using thousands of unique characters, these electronic dictionaries enabled word processing for the first time. Mechanical typewriters had been made for these scripts, but they were so complex and unwieldly that they never saw widespread adoption (Sproat 2010: 169f.). Modern-day Japanese word processing systems allow the user to type either in **rōmaji**, a romanization system, or in **hiragana**, in which each glyph is roughly a syllable, and a digital dictionary is used to suggest alternative spellings in **kanji**, roughly word-sized glyphs used for most roots. Several different approaches exist for Chinese text entry; in one popular method, users type in **pinyin**, a romanization system, and the dictionary maps pinyin onto roughly word-sized **hanzi** glyphs. The user chooses the correct glyph from among a list of candidates the computer presents.

Yet even small dictionaries overwhelmed the limited memory of early personal computers. For instance, a list of English headwords from a large English dictionary easily exceeds the capacity of the 3.5-inch, 1.44-megabyte floppy disks first sold in 1986. The problem only becomes more acute if one must store all inflected forms. For English, for example, one needs only to list two forms of most nouns (the singular and plural) and at most six forms of any given verb. In Russian, nouns may have as many as twelve inflectional forms, and verbs have more than a hundred. And Archi, a language spoken in Dagestan, is believed to have more than 1.5 million possible forms of each verb (Kibrik 1998).

Unsurprisingly, one highly-effective method method for compressing lexicons is to generate complex words using lists of stems and word formation rules. An interesting computational treatments of English morphology was undertaken during the development of MITalk, an early **text-to-speech synthesis** (TTS) system. (A commercial version of MITalk is best known as the "voice" of the late physicist Stephen Hawking.) TTS systems consist of a "frontend" responsible for performing linguistic analysis on the input text, and a “backend” tasked with synthesizing the audio. During frontend processing, words must be mapped from orthographic form to phonemic transcriptions, a task sometimes known as **grapheme-to-phoneme conversion**. Rather than listing hundreds of thousands of words, however, MITalk “generates” many complex words by concatenating stems and affixes. For instance, consider the English word *scarcity*. MITalk analyzes this entry much as a linguist would: as *scarce* plus the suffix *-ity*, the latter triggering an orthographic rule deleting a word-final *e* before certain vowel-initial suffixes. By concatenating the corresponding pronunciation entries for these two pieces, one obtains the correct phonemic transcription, /ˈskɛɚ.sɪti/. The system also considers, but rejects, various alternative analyses, such as treating this word as a compound of *scar* and *city*, which would lead to an incorrect pronunciation. With the help of these procedures, MITalk can generate the pronunciations of over 100,000 words from roughly 12,000 stems and affixes (Klatt 1987: 773).

Beyond the issue of compression, data sparsity was also a major issue for early computing efforts. In any sample of text, we are likely to see only a small fraction of the possible complex words (Lignos and Yang 2016). Morphological analysis for **out of vocabulary** (or OOV) words—words not previously seen—remains a major challenge for computational morphology. Yet if you search the internet for *fishing* you may be interested in documents that mention *fish* or *fishers*, even if those documents do not mention *fishing* itself. Therefore, we may wish to conflate related words when indexing (i.e., ingesting) web pages, and when searching for good matches for a query. One simple approach, **stemming**, processes the text by applying a language-specific rules to each word, remove affixes like *-ing*, *-er*, *-s*. Unlike the MITalk decompositions, which depend on lists of stems and affixes, a stemmer does not require a lexicon, merely a sequence of rules. Various sources claim that Google integrated stemming into its search engine sometime around 2003.

MITalk, and most systems of similar vintage, take an ad hoc, “whatever works” approach to morphology. For instance, early spelling correction systems save memory by allowing fanciful analyses, for instance, deriving *forest* by affixing the superlative suffix *-est* to *fore*. Similarly, a stemmer need not produce real words, or even “stems” that are linguistically well-defined; it simply needs to form semantically coherent equivalence classes of words. For instance, the Porter (1980) stemmer, when applied to the previous sentence, produces non-words like `produc`, `semant`, and `equival`; however, these stems are shared by closely-related words like *producer*, *semantics*, and *equivalent*. Furthermore, many of these early techniques for morphological processing take advantage of peculiarities of English's rather-impoverished inflectional system, and it is unclear how they might be generalized to languages with richer morphologies.

Much of this has changed for the better in the intervening decades. One key step away from ad hoc inelegances and towards rigor and multilingualism was the comprehensive Finnish morphological analyzer developed by Koskenniemi (1983). Koskenniemi's model became—and to some degree, still is—a standard for documenting morphologically rich languages, and it ultimately led to the discovery of new methods for computationally encoding morphological knowledge reviewed below. Furthermore, the introduction of machine learning techniques to computational morphology has added further rigor and reduced the need for perform painstaking grammar engineering work.

3 Problem specification

Early work in computational morphology provides solutions to specific applications—be it pronunciation generation, spell checking, or information retrieval—but does not provide a general-purpose solution to the problem of making a computer perform morphological analysis. There are several ways we can define this task.

In the simplest setting, we simply wish to obtain a detailed morphological summary of a given word. For the English word *puppies*, for example, might mention that the word is a plural noun. Such a task is sometimes called **morphological tagging**. While this information may not seem terribly useful on its own, it is extremely valuable for "downstream" tasks, such as **parsing**, recovering the syntactic structure of a sentence. This morphological summary might also include the word's **segmentation** (or **decomposition**) which might be written `puppy+s`, where `+` is used to indicate word-internal morpheme boundaries. There is an orthographic rule which requiring that word-final *-y*, when preceded by a consonant, be spelled as *-ie* in the plural, and segmentation reverses this rule, allowing us to obtain the citation form or headword `puppy` and the `s` suffix. This suggests a more-sophisticated alternative to stemming: **lemmatization**, or replacing inflected words with their **lemmas**, as we sometimes call citation forms. Both tagging and lemmatization can be performed in isolation—providing all possible analyses in the presence of ambiguity—or they may process entire sentences or documents and attempt to resolve grammatical ambiguities using the broader linguistic context. For instance, the word *cooler* might receive different analyses in the contexts *the __ was full of beer and bait...* and *the English daisy prefers __ weather....*

The inverse problem, **morphological generation** (or **inflection**), is a key part of many **natural language generation** systems, such as digital assistants like Cortana, Siri, and the Google Assistant. For example, imagine an application that "reads" the current weather conditions out loud using text-to-speech synthesis. For this application, we might use simple **templates** with specific slots to be filled. For instance, for English one might imagine a template reading `It is currently $TEMP degrees` where `$TEMP` indicates a "slot" to be filled with the current temperature. However this is not quite right: on particularly cold days, it would produce the ungrammatical *`It is one degrees` instead of `It is one degree`. This problem recurs in many other languages. For instance, in Russian, which has a much richer inflectional system than English; one needs three different words for 'degree'—*grádus*, *grádusa*, or *grádusov*—depending on what the temperature is. Similar problems would reoccur—though with different units—were one to expand this template to include additional measurements like as wind speed, barometric pressure, and so forth. What is really needed is a mapping between the singular and plural forms of English nouns, or between as the many as twelve unique case-and-number variants of Russian nouns. The template designer may either choose to manually specify the contexts in which one uses the singular or plural (for instance), or the generation system may be expected to provide the appropriate inflectional form of *degree* (or *grádus*) using nearby words.

There are two ways one might go about building a morphological analyzer or generator. In the **knowledge-based** approach, the linguist-developer simply constructs a digital lexicon and encodes all morphological rules of interest. The lexicon and rules are often expressed using abstract computational devices known as **finite-state automata**, which support efficient analysis and generation procedures. Alternatively, one might take a **data-driven** approach, using machine learning to learn morphological generalizations from linguistic data. As you will see, these two approaches have different strengths and weaknesses, and can even be combined in various fashions to form hybrid models.

In most cases, we assume that inputs and outputs for analysis and generation are represented in orthographic (i.e., written) rather than phonemic form, largely as a matter of convenience. For some languages, for instance Spanish or German, this decision has little impact, because these languages use "shallow", highly-consistent orthographies which is quite close to phonemic representation. Other languages, such as English or Korean, use a "deep" orthography in which the relationship between spelling and pronunciation is more abstract. In practice, this reduces the need to model any pronunciation variation not indicated in written form. For example, English spelling does not generally indicate changes in vowel quality triggered by the addition of derivational suffixes like *-ity*. Thus *sane* [seɪn] and and *sanity* [sæ.nɪ.ti], for instance, are spelled more similarly than they are pronounced (Chomsky and Halle 1968: 44f.), and the vowel change does not need to be modeled if English orthography is used.

We rarely have the luxury to focus on transcribed words in isolation. Often, we are provided with entire documents (e.g., newspaper articles or web pages) and must split them into sentences and words before performing further analyses. These procedures, **sentence boundary detection** (or **sentence splitting**) and **tokenization** are not described here, but neither step is trivial and may require some degree of linguistic sophistication (e.g., Gillick 2009). For instance, in an English document, the fragment "…U.S. Treasury…" could either be a noun phrase or it could straddle the boundary between two sentences.

> "…suggesting declining consumer confidence in the U.S. Treasury officials declined to comment for this story."

	singular	plural
nominative	grádus	grádusy
genitive	grádusa	grádusov
dative	grádusy	grádusam
accusative	grádus	grádusy
instrumental	grádusom	grádusami
prepositional	gráduse	grádusax

Table 1: Inflectional paradigm of the Russian noun *grádus* ‘degree’.

Even the notion of how to divide a sentence into words is more complex than it may seem at first. In languages like Chinese, Japanese, and Thai, there are no explicit cues to word boundaries in most texts. Even in English, it is not immediately obvious whether compounds like *podcast* or possessives like *Queen's* should be treated as one or two words for morphological analysis. However, we set these issues aside and assume text has been segmented into sentences and tokenized into word-like units.

4 Knowledge-based methods

In knowledge-based approaches, we encode our knowledge of the target language (possibly combined with printed or digital dictionaries) as simple computer programs which perform morphological analyses and generation.

Recall from chap. 2 that a paradigm is simply all the forms of a given lexeme. In English, for instance, the paradigm of a noun contains the singular and plural forms, whereas in Russian, it has twelve cells defined by two numbers (singular and plural) and six cases (nominative, genitive, dative, accusative, instrumental, and prepositional). Each of the twelve cells in the Russian nominal paradigm are defined by a combination of one of the six cases and one of the two numbers. However, some forms may appear in multiple cells in the paradigm due to **syncretism** (§6.4). For instance, *grádus*, like all inanimate Russian nouns, exhibits a syncretism between the nominative and accusative in the singular and plural. The full paradigm is shown in Table 1.

We can form the genitive plural (henceforth, gen.pl.) form of *grádus* simply by appending the gen.pl. suffix *-ov* to the lemma, and we can generate the rest of the paradigm by appending the appropriate case/number suffixes, assuming the existence of a "zero" suffix for the nominative and accusative singular, both of which are identical to the lemma in inanimate nouns.

Few dictionaries contain—or even could contain—all paradigm cells for all lemmas. Rather, lexicographers provide the lemma itself followed by additional information needed to complete the paradigm. In Zaliznyak's celebrated Russian dictionary, each entry gives a lemma and its **inflectional class**. For instance, the entry for *grádus* is marked "M 1a". This seemingly-cryptic code tells us this noun follows the "hard stem masculine, stress pattern A" inflectional class.[1] Students of Russian—at least those familiar with Zaliznyak's dictionary—can use these codes to generate all twelve cells of a noun's paradigm.

A slightly different approach is taken in the Oxford Latin Dictionary (OLD). There, each verb is given with its four **principal parts**. For instance, the entry for the verb *adaerō* reads "adaerō ~āre ~āuī ~ātum". This entry contains enough information to allow students of Latin to generate the entire verb paradigm. The lemma *adaerō*, along with the rest of the entry, gives us four inflectional verb forms, and from these one can generate the several dozen other verb forms, the remainder of the paradigm. As is common practice in Latin pedagogy, the lemma *adaerō*, the first principal part, is the first person singular present active indicative 'I calculate'. To form the second principal part, for instance, one deletes the final *-ō*—the first person singular present active indicative suffix—and appends the second element of the entry, *-āre*, producing the present infinitive *adaerāre* 'to calculate'. The third and fourth principal parts, the first person singular perfect active indicative *adaerāuī* 'I calculated' and the supine *adaerātum* 'for to be calculated', are obtained by appending *-āuī* and *-ātum*, respectively, to the first principal part minus *-ō*. Of course, these four principal parts are only a tiny fraction of the dozens of other inflectional variants of any given Latin verb, but any student of Latin can generate the remaining forms of the paradigm using just these four principal parts. As it happens, these inflectional patterns are quite common and are shared with hundreds of other Latin verbs, but there are dozens of other Latin conjugation classes. These classes vary widely in the number of lemmas they cover. Note that the class *adaerō* belongs to is by far the largest conjugation class in Latin, and it likely would apply to most OOV lemmas; a well-designed morphological analyzer will encode facts of this sort.

Thus, entries in Zaliznyak's dictionary and the OLD are not simply a list of of inflectional forms and their analyses. Rather, they contain just enough information to allow a careful student of Russian or Latin to generate full paradigms. As computational morphologists, we too could simply create a list of inflected words and their analyses, but this would be an extremely taxing project in a richly inflected language like Russian.

Instead, Koskenniemi (1983) uses abstract computational devices known as finite-state transducers (or FSTs), to automatically, implicitly, and efficiently generate this list for a large portion of Finnish, another richly inflected language. You may not be intimately familiar with FSTs, but if you own a mobile phone, you likely carry around FSTs in your pocket, since they're used to power mobile text entry systems (**input method engines**) and **virtual assistant** systems like Alexa, Cortana, Google Assistant, and Siri. We will gradually introduce FSTs and demonstrate how they can be used to encode morphological knowledge for both analysis and generation, beginning with a more-general concept, that of state machines.

4.1 State machines

A **state machine** is a machine—either hardware or software—that can be described in terms of **states**, corresponding loosely to the device's "memory"—and **transitions** between those states—corresponding to the operations performed by the device.

Let us imagine an enchanted old-fashioned gumball machine, one which never breaks down, never becomes too full of coins, and never runs out of gumballs. At any point in time, the machine is in one of two states. In one state, arbitrarily denoted 0, there is no coin present. It is possible to insert a coin at state 0, but not at the other state, henceforth 1, since in that state, a coin is already present in the slot. In both states, it is possible to turn the knob. However, in state 0, this has no effect on the machine at all, whereas in state 1, turning the knob causes the machine to yield a gumball and clears the coin slot. The gumball machine is schematicized in Figure 1, with

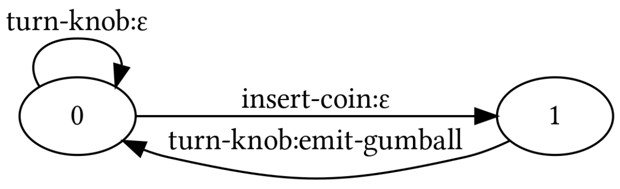

Figure 1: State diagram for an enchanted gumball machine.

the transitions between states, the **arcs**, indicated by arrows. Each arc is labeled with an input and an output, separated by a colon, and the Greek letter ε ("epsilon") is used to the absence of an input and/or output. For example, because it is possible—but pointless—to turn the knob at state 0, there is a "self-arc" at that state labeled 'turn-knob:ε'.

4.2 Notation

It is now necessary to define a few mathematical notions.[2] The first is the **set**, a collection of distinct objects of some type. The elements in a set are said to be its **members**. Sets are denoted by curly braces, except for the **empty set**, the set with no elements, for which we use the special symbol $\emptyset$. A set is said to be a **subset** of another set if every element in the first set is also a member of the second set. In principle, sets can contain any type of object, but one particularly useful type of set contains **strings**, ordered sequences of characters. Sets of strings are known as **languages**, with the caveat that this is merely a term of art and is not intended to supplant any other sense of this word. For instance, the language $\{\texttt{OK}, \texttt{NM}, \texttt{AZ}, \texttt{AK}, \texttt{HI}\}$ contains the two-character postal abbreviations of the five American states admitted to the Union during the 20th century. Finally, we use ε to denote the empty string, since it is not obvious how one might write it otherwise.

4.3 Finite-state acceptors and regular languages

We are now ready to formally introduce a type of state machine known as the **finite-state acceptor** (FSA). It is defined by three sets. The first is a set of states denoted by Q. How we label these states—with strings, numbers, or something else—does not matter so long as this set is fi-

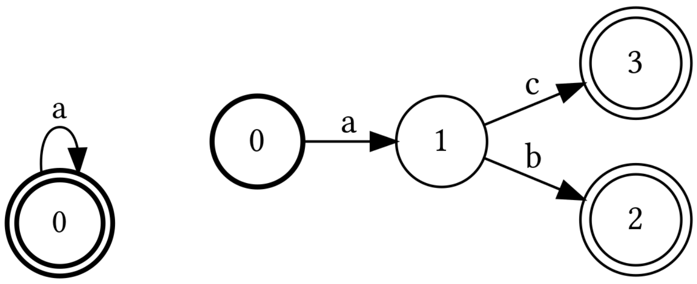

Figure 2: State diagrams for two finite-state acceptors; the initial state is indicated by a bold circle, and final states are indicated by double circles. Left: an acceptor corresponding to the regular language $\{\mathsf{a}\}^* = \{\varepsilon, \mathsf{a}, \mathsf{aa}, \mathsf{aaa}, \mathsf{aaaa}, \ldots\}$. Right: an acceptor corresponding to the regular language $\{\mathsf{a}\}\{\mathsf{b}, \mathsf{c}\} = \{\mathsf{ab}, \mathsf{ac}\}$.

nite. By convention, one of the states, s, is designated the **initial state**, and a subset of the states F, are designated as **final states**.[3] Another set is known as the **alphabet**, and is indicated by the Greek letter Σ ("sigma"). The alphabet is set of symbols we'll be using in our modeling. In computational morphology, the alphabet usually consists of orthographic characters, though for other applications, it might consist of morphemes, morphosyntactic features, words, and so on. The third set, the **transition relation**, is denoted by the Greek letter δ ("delta"). Each element of δ is a triple consisting of a source state, a symbol in Σ (or the empty string ε), and a destination state. If q, z, r is a member of δ, there is an arc from q to r labeled z. Figure 2 gives state diagrams for two FSAs.

An FSA is said to **accept** (or **match**, or **recognize**) a string if there is a path from the initial state to a final state, and the sequence of symbols of the arcs traversed by that path spell out that string. For instance, the FSA in the right panel of Figure 2 accepts the string ab because a path that begins in the initial state 0, then traverses to state 1, then to state 2—a final state—spells out ab. Similarly, we can thus easily determine whether a string is accepted by an FSA simply by traversing the machine, starting at the start state, and determining whether there is some appropriately labeled path to a final state. The set of strings an FSA accepts is again known as its language. FSAs accept a class of languages known as the **regular languages** (Kleene 1956); once again, this is merely a term of art; the relevant sense of the word **regular** is unrelated to others. We now introduce three operations over languages—union, concatenation, and closure—which together define the regular languages.

The **union** of two languages is a new language in which all strings are elements of the first language, the second language, or possibly both. For instance, let $A = \{\mathtt{a}, \mathtt{b}\}$ and $B = \{\mathtt{b}, \mathtt{c}, \mathtt{d}\}$. Then, their union, written $A \cup B$, is the language $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$; note that we don't repeat c when writing out the union, even though c is a member of both the A and B languages. This can easily be generalized to the unions of more than two languages. For instance, if $C = \{\mathtt{d}, \mathtt{e}, \mathtt{f}\}$, then $A \cup B \cup C$ is $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}, \mathtt{f}\}$.

The **concatenation** of a language is slightly more complex. The concatenation of two strings is simply a new string formed by joining the two strings end-to-end. For instance, the concatenation of OK and AY is OKAY. Naturally, concatenation of any string s with ε gives s. Then, the concatenation of two languages (rather than two strings) is a language in which each string is formed by joining, end-to-end, a string from the first language with a string from the second language. For instance, the concatenation of A and B, written as AB, is the language $\{\mathtt{ac}, \mathtt{ad}, \mathtt{bc}, \mathtt{bd}, \mathtt{cc}, \mathtt{cd}\}$.

The **closure** of a language consists of the union of zero or more "self-concatenations" of a language with itself. Closure is notated by a superscript asterisk, the **Kleene star**. For instance, $A^* = \{\varepsilon\} \cup A \cup AA \cup AAA \cup \ldots$. If $A = \{\mathtt{a}, \mathtt{b}\}$, then this denotes the infinite language $\{\varepsilon, \mathtt{a}, \mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{bb}, \mathtt{ba}, \mathtt{aaa}, \mathtt{aab}, \ldots\}$.

We now are prepared to give a formal definition of the regular languages. Assuming a finite alphabet of symbols Σ:

- $\emptyset$, the empty set, is a regular language.
- The language $\{\varepsilon\}$, consisting of just the empty string, is a regular language.
- If s is some symbol in Σ, then $\{s\}$ is a regular language.
- If X is a regular language, then its closure X^* is also a regular language.
- If X and Y are regular languages, then:
  - their union $X \cup Y$ is also a regular language, and

– their concatenations, XY and YX, are also regular languages.

Any languages which cannot be so derived are not regular languages.[4] By definition, the regular languages are **closed** under union, concatenation, and closure, meaning that the the closure of any regular language, and the concatenation of any two regular languages, and the union of any two regular languages, is itself a regular language.

The regular languages, and thus finite-state acceptors, are a natural tool for representing some lexical information. For instance, a set of stems can be represented by a union of strings. However, more powerful models are required to perform general-purpose morphological analysis.

4.4 Finite-state transducers and rational relations

The **finite-state transducer** is a generalization of finite-state acceptors. Like acceptors, transducers have a finite set of states Q, one of which (s) is designated as the initial state, and a subset of which (F) are designated as final states. However, transducers have two alphabets, an **input alphabet** denoted by Σ and an **output alphabet** denoted by Φ. The definition of the transition relation is slightly modified: each element of δ is a four-tuple consisting of a source state, a input symbol in Σ (or the empty string ε), a output symbol in Φ (or ε), and a destination state. Then, if q, x, y, r is a member of δ, there is an arc from q to r with input symbol x and output symbol y. Figure 3 gives a state diagram for an FST.

The key difference between FSAs and FSTs is that FSAs encode sets of strings, whereas FSTs map between sets of strings. In knowledge-based morphology, FSAs are commonly used to represent lists of stems, lemmas, affixes, and so forth, whereas FSTs are used to represent various types of phonological and morphological rules. Note FSAs can also be thought of as a special case of FSTs; they are simply FSTs which have the same input and output alphabets, and in which each transition has the same input and output label. Finite-state software often takes advantage of this fact by using FSTs to implement FSA.

An FST is said to **transduce** (or **map**) from an input string to an output string if there is a path from the initial state to a final state, and the sequences of input and output symbols along

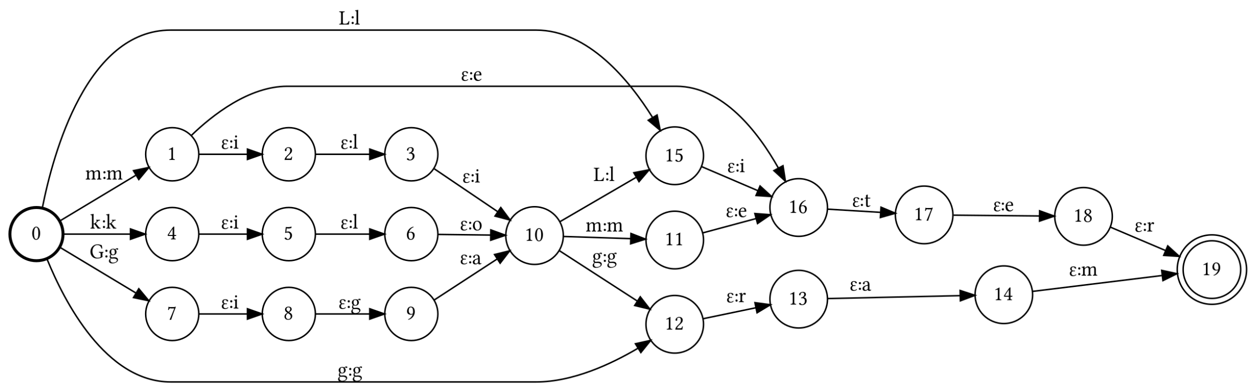

Figure 3: A state diagram for a finite-state transducer. The depicted transducer is a fragment of a larger machine mapping between SI unit abbreviations (e.g., `mL`) and their full form in English (e.g., `mililiter`).

that path spell out the input and output strings, respectively. The set of transductions that can be performed by FSTs are known as the **rational relations**. Rational relations can be defined using two regular languages and an operation known as the **cross-product**. The cross-product of two sets X and Y, written $X \times Y$, is the set that contains all x, y pairs where x is a member of X and y is a member of Y. All rational relations, then, are subset of the cross-product of two regular languages. That is, they map from some subset of strings in X to some subset of strings in Y.

Two other properties of FSTs will be relevant shortly. First, FSTs are closed under an operation known as **composition**. For instance, if α is a transducer over $X \times Y$ and β is a transducer over $Y \times Z$, their composition $\alpha \circ \beta$ is an FST over $X \times Z$. Secondly, FSTs are closed under **inversion**. If we are given an FST α over $X \times Y$, one can obtain the inverse relation $Y \times X$ simply by swapping the input and output label of each of the transducers' arcs.

4.5 Finite-state analyzers and generators

While there are several ways we might use finite-state transducers for morphological analysis and generation, we now briefly describe an updated variant of Koskenniemi's model proposed by Roark and Sproat (2007: ch. 2) and implemented by Gorman and Sproat (2021: ch. 6).[5]

In this model, each paradigm is a set of lexemes all following the same inflectional pattern, defined by a set of stems, specifications for the cells in the paradigm—each consisting of a **fea-**

ture vector, i.e., a list of morphosyntactic features, and a transducer that introduces the affixes associated with that cell—and an optional list of orthographic rules. Once defined, a paradigm can be used for both analysis and generation.

Stems As mentioned above, a set of stems, like the "M 1a" class $\{\texttt{grádus}, \texttt{žurnál}, \ldots\}$, can be expressed as an FSA constructed from the union of those strings. If we later encounter an OOV stem, all we have to do to ensure that it is covered by the model is to add it to the stem set of the appropriate paradigm.

Cells We assume that each cell in the paradigm corresponds to a vector of morphosyntactic features. The cells are defined by a set of feature vectors that make up its cells. Furthermore, it is necessary to know which feature vector corresponds to the lemma for this paradigm. In Zaliznyak's dictionary, for example, the lemma of a Russian noun is assumed to be the nominative singular form, whereas in the OLD, the lemma of a Latin verb is assumed to be the first person singular present active indicative. Each paradigm cell must also be linked to the affixes it is associated with. To accomplish this, each cell is associated with a transducer which introduces affixes characterizing that cell. For instance, we can implement the insertion of a prefix p to a set of stems S by building a transducer representing the rational relation $(\emptyset \times \{p\})S$.[6]

Rules Finally, we may want to specify a series of orthographic rules to be applied, in order, to the strings generated by the cells' affixation transducers. For instance, the English spelling convention which converts a stem-final *-y* to *-ie* after a consonant and before plural *-s*, as in *puppies*, is expressed by the rule $\texttt{y} \rightarrow \texttt{ie} \;/\; \{\texttt{b}, \texttt{c}, \texttt{d}, \ldots\}\_\_\texttt{+s}$. Rules of this general type, known as **context-dependent rewrite rules**, all correspond to some FST (Johnson 1972), and there are efficient algorithms to compile them into FSTs (e.g., Kaplan and Kay 1994).[7]

Analysis and generation It is relatively straightforward to construct analyzers, taggers, lemmatizers, and generators given the FSAs and FSTs just described. To perform analysis, we first compose an FSA representing the set of stems stems with the transducers introducing affixes for each slot, and then apply the sequence of context-dependent rewrite rules. After performing some minor book-keeping, we obtain a transducer which maps the inflected form `grádusov` to its analysis string `grádus+ov[num=pl][case=gen]`. Tagging is performed by composing the analyzer with an FST which deletes the + boundary characters from the analyzer's output. Lemmatization is performed by mapping the analyzed form first to its stem, then composing the affixation transducer associated with the lemma cell—and any context-dependent rewrite rules. Finally, generation is performed simply by inverting the lemmatizer FST.

4.6 Limitations

It is easy to imagine that finite-state transducers and rational relations are little more than minor mathematical curiosities, but linguists have argued that nearly all morphological and phonological processes of the world's languages correspond to well-defined subsets of the rational relations (Heinz 2018). However, a few challenging phenomena remain, such as **total reduplication**, the copy of a full stem. Total reduplication is used, for example, in Warlpiri, a language of Australia, to form certain noun plurals (e.g., *kurdu* 'child', *kurdukurdu* 'children'; Nash 1986: 130). However, rational relations cannot copy arbitrarily long sequences. Nor can rational relations move segments or sequences over long distances, as seems to be required for the formation of the Kujamaat Jóola definite article described in chapter 6. Furthermore, rational relations cannot reverse arbitrary strings, as in the French language game Verlan described in chapter 3.[8]

5 Data-driven methods

Knowledge-based morphological analyzer-generators are unsung heroes of computational morphology, and free finite-state analyzers can be obtained for dozens of languages. However, the development of knowledge-based resources can be arduous, particularly in languages with large paradigms or complex series of rewrite rules. **Machine learning** is the study of algorithms that "learn" to perform tasks directly from data without explicit instruction. Instead of encoding lists of stems, affixes, and rules, we can treat morphology as a machine learning problem, performing morphological analysis and generation in a data-driven fashion.

5.1 Tagging

A **tagger** is presented with a sentence or utterance and assigns each token with a label from some finite set. The **tag set**, the set of possible labels, varies from language to language, corpus to corpus, and task to task. In **part-of-speech** (POS) **tagging**, tokens are labeled as nouns, verbs, adjectives, and so on. Some POS tag sets, however, make finer distinctions. For instance, the Penn Treebank tagset, used for English, has 36 distinct tags and including six different types of verbs. plus a special tag for the modals *might*, *should*, etc. In morphologically rich languages, the tagset can in fact be expanded to cover all possible combinations of number and case, tense and aspect, etc., found in the language. for instance, the Szeged corpus of Hungarian, another richly inflected language, has 744 tags in all. Tagging performed with large, morphologically-aware tag sets is sometimes described as **morphological tagging**, though the distinction between morphological and part-of-speech tagging is not well-defined. Finally, we can map these large, language-specific tag sets to a smaller "universal tagset" for cross-linguistic work. For instance, whereas the Penn Treebank tagset distinguishes between several different types of verb— the simple past (`VBD`) as in *sang*, the gerund/present participle (`VBG`) as in *singing*, the past participle (`VBN`) as in *sung*, and so on—all of these, along with modals (`MD`) like *might* are all labeled `VERB` in the Petrov et al. (2012) universal tagset, since not all languages draw these morphological distinctions.

Lemmatization can also be framed as a tagging task. Here, the tags are **edit scripts** (Chrupała et al. 2008), instructions for converting the inflected word to its lemma. For instance, the edit script used to lemmatize the word *puppies* might correspond to the rational relation $\Sigma^*(\{\texttt{ies}\} \times \{\texttt{y}\})$. A tagger is used to predict the edit script for each token in the sentence, and the edits are then applied to produce a lemmatized sentence. Lemmatization is usually performed after morphological tagging since the predicted morphological tags are very help for predicting which

wordform	lemma	POS	morphology
Pies	pies	NOUN	Animacy=Nhum|Case=Nom|Gender=Masc|Number=Sing
płynie	płynąć	VERB	Aspect=Imp|Mood=Ind|Number=Sing|Person=3|Tense=Pres|...
z	z	ADP	AdpType=Prep|Variant=Short
małą	mały	ADJ	Case=Ins|Degree=Pos|Gender=Fem|Number=Sing
,	,	PUNCT	PunctType=Comm
żółtą	żółty	ADJ	Case=Ins|Degree=Pos|Gender=Fem|Number=Sing
piłką	piłka	NOUN	Case=Ins|Gender=Fem|Number=Sing
w	w	ADP	AdpType=Prep|Variant=Short
pysku	pysk	NOUN	Animacy=Inan|Case=Loc|Gender=Masc|Number=Sing
.	.	PUNCT	PunctType=Peri

Table 2: A Polish sentence ('The dog is swimming with a small yellow ball in its mouth') from the Polish Dependency Treebank, with morphological annotations; one entry has been truncated (indicated by ellipses) for reasons of space. From this we can reason, for instance, that *pysku* is here the locative singular form of the masculine noun *pysk* 'mouth (of an animal)'.

edit script goes with which token. An example sentence with wordforms, lemmas, part-of-speech tags, and morphological tags is shown in Table 2.

Taggers are important building blocks for many other natural language processing tasks. Part-of-speech and morphological tags are used for various "downstream" processing tasks, such as **named entity recognition**, the extraction of proper noun phrases, or **parsing**, which recovers the syntactic structure of a sentence. For instance, in German, which has a rich inflectional system and a relatively free word order, it is difficult for computers to accurately parse sentences without morphological tags (e.g., Fraser et al. 2013). Morphological tagging is in some sense more challenging than knowledge-based morphological analysis because it requires the system to reason not just about the morphological properties of words, but also to resolve grammatical ambiguities. For instance, many English words participate in **zero derivation** (or **conversion**) from nouns to verbs (e.g., *run*) or vice versa (e.g., *drink*); the tagger must decide which analysis is more appropriate in context, using nearby tokens and tags. There is something of a chicken-and-egg problem here. There is a strong dependence between the tag being predicted and nearby tags. For example, if the previous word is a determiner like *a* or *the*, the next word is much more likely to be a noun than a verb. Yet, we do not yet know what tag the previous word may have.

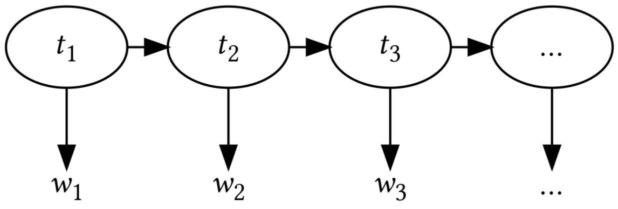

Figure 4: Illustration of the hidden Markov model generative story; t: tag, w: word.

Thus, tagging is a **structured prediction** problem, one that requires us to simultaneously make a series of interdependent decisions to obtain the best overall prediction.

One method that allows us to make structured predictions for tagging problems is the **hidden Markov model**, or HMM for short. While HMMs are often outperformed by newer machine learning techniques, they illustrate the challenge of structured prediction. HMMs are abstract computational devices closely related to finite-state machines. An HMM tells a simple "story" about how data—here, tagged sentences—are produced; it imagines that each tag is generated by the previous tag, and each word is then generated by its tag.

1. Generate a morphological tag sequence $\mathbf{T} = t_1, t_2, t_3, \ldots, t_n$.

2. Then, for each morphological tag t emit a word w which can be tagged with t, producing $\mathbf{W} = w_1, w_2, w_3, \ldots, w_n$.

This story is illustrated in Figure 4. To turn it into a full-fledged computational model, we operationalize it using probability theory.

The first step, the generation of the tag sequence $\mathbf{T}$, is naturally expressed as a probability distribution $P(\mathbf{T})$ which assigns probabilities to all possible sequences to tags. But how should we go about estimating the probabilities of a tag sequence, given that there is no upper bound on how long a sentence can be? The solution is to approximate $P(\mathbf{T})$ by making a **Markov assumption**: we assume that the next tag in the sequence depends on no more than the previous k tags, where k is a small natural number. For instance, under a first-order Markov assumption (i.e., $k = 1$), we assume that each tag depends on just the previous tag. We write this dependence as $P(t_i \mid t_{i-1})$—the probability of t_i given that we have observed t_{i-1}—and refer to these terms as **transition**

probabilities, since they give the probability of a transition from tag t_{i-1} to tag t_i. The product of all such transition probabilities is the first-order Markov approximation of $P(\mathbf{T})$:

$$P(\mathbf{T}) \approx P(t_1)P(t_2 \mid t_1)P(t_3 \mid t_2)\ldots P(t_n \mid t_{n-1}).$$

The second step, the mapping from tags to words, can be expressed as a probability distribution $P(\mathbf{W} \mid \mathbf{T})$. Much like we did with the tag sequence distribution, one can factor this distribution into the product of several simpler probability distributions. We define a distribution of **emission probabilities**, $P(w \mid t)$, which gives the probability that the tag t "emits" w. For instance, $P($`grádusa` $\mid$ `N;GEN;SG`$)$ would give the probability that a gen.sg. noun is *grádusa*. The product of all these emission probabilities gives us $P(\mathbf{W} \mid \mathbf{T})$:

$$P(\mathbf{W} \mid \mathbf{T}) = P(w_1 \mid t_1)P(w_2 \mid t_2)P(w_3 \mid t_3)\ldots P(w_n \mid t_n).$$

At first glance the emission probability seems to go in the wrong direction: we want to know the probabilities that a word gets a certain tag, not the other way around. However, there is a good reason for this seemingly backwards approach. When we tag a sentence, we observe $\mathbf{W}$ and want to compute the highest probability tag sequence. In other words, we want to find a tag sequence $\hat{\mathbf{T}}$ which maximizes—i.e.., gives us the highest-possible value of—$P(\mathbf{T} \mid \mathbf{W})$. We do not have an estimate of $P(\mathbf{T} \mid \mathbf{W})$, but thanks to the mathematical principle known as **Bayes' rule** (or **Bayes' theorem**), we know that if $\hat{\mathbf{T}}$ maximizes $P(\mathbf{T} \mid \mathbf{W})$, it also maximizes the product of the transition probabilities and the emission probabilities too.[9]

$$\begin{aligned}\hat{\mathbf{T}} &= \arg\max_{\mathbf{T}} P(\mathbf{T} \mid \mathbf{W}) \\ &= \arg\max_{\mathbf{T}} P(\mathbf{T})P(\mathbf{W} \mid \mathbf{T})\end{aligned}$$

How do we find the best $\hat{\mathbf{T}}$? Naïvely, we could simply enumerate all possible taggings $\hat{\mathbf{T}}$ and pick whichever one gives us the highest value for $P(\mathbf{T})P(\mathbf{W} \mid \mathbf{T})$. However, with even short sentences and small tag inventories, the number of possible tag sequences is astronomically large, making this method infeasible. Instead, we find the best tag sequence using a specialized technique, the **Viterbi algorithm**, which is guaranteed to find the best tag sequence according to our story.[10]

Virtually every machine learning technique developed in the last three decades—not just hidden Markov models but also **decision trees** and **random forests**, **linear-** and **log-linear models**, **conditional random fields**, and many types of **artificial neural networks**—has been applied to tagging. While there is an enormous amount of variation from corpus to corpus, language to language, and tagger to tagger, it is quite often possible to tag nine out of ten, or even 97 out of 100, tokens correctly.

What accounts for the remaining errors? Taggers have difficulty tagging rare or OOV tokens. However, the tagger can sometimes make reasonable inferences about the morphological properties of a new word using a finite-state morphological analyzer, or in the case of English, simply by looking at the last few characters of the word. But even with such tricks, taggers have much higher error rates on out-of-vocabulary tokens than in-vocabulary tokens. Another major source of tagging error is genuine linguistic ambiguity introduced by phenomena such as syncretism and zero derivation.

Most taggers are trained in a **supervised** fashion, meaning that the system "learns" from a collection of sentences that have already been tagged by a team of linguist-annotators who are familiar with the language. It is also possible to train certain types of taggers, including hidden Markov models, in an **unsupervised** fashion, that is, without any tagged data provided. However, the performance of unsupervised taggers is generally poor, and only a small amount of tagged data is needed for a supervised model to outperform the best unsupervised models.

	wordform	lemma	features
UD	schlössest	schließen	`Mood=Sub|Number=Sing|Person=2|Tense=Imp`
UM	schlössest	schließen	`V;SBJV;PST;2;SG`

Table 3: The German word *schlössest* 'you would have closed' as encoded in Universal Dependencies (UD) and UniMorph (UM) formats; the two analyses are largely equivalent.

5.2 Generation

Morphological generation was one of the earliest uses of machine learning in linguistics. For example, Rumelhart and McClelland (1986) describe an early neural network used to predict the past tense forms of English verbs.[11] In one common format, the generation system is presented with a triple consisting of the input word, its morphological specification, and a desired output morphological specification.[12] For instance, the input triple might be of the German word *schlössest* 'you would have closed', its morphological specification—it is a second person singular past tense subjunctive verb— and the desired morphological specification, such as the infinitive; this is illustrated in Table 3. The model would then be expected to generate the infinitive form, here *schließen* 'to close'.

Modern **sequence-to-sequence** neural networks are generally quite adept at morphological generation tasks. These models were originally developed for **machine translation** (Bahdanau et al. 2015) but can be used for morphological generation with minor modifications. We now briefly describe MED, a relatively simple and highly effective morphological generation model proposed by Kann and Schütze (2016).[13]

The first component of the MED is the **encoder**; it converts the input triple to a numerical representation that the remainder of the network can use to predict the output sequence. During training and prediction, each symbol in the input triple—including the input characters and the input and output morphological tags—is first **embedded**, or mapped onto a vector of real numbers. Let e be the dimensionality of these vectors, chosen by the experimenter. Then, after embedding, an input triple of length n can be represented by a single $e \times n$ matrix. We then pass this matrix through a **recurrent neural network** (or RNN), which produces an $f \times n$ matrix of **hidden**

state activations, where the f, the dimensionality of the hidden state activations, is once again chosen by the experimenter. This second matrix is a sequence of f-length real number vectors—one for each symbol in the input triple—in which each column is a numeric representation of what the network knows about that symbol. This representation of the symbols is conditioned in part by the context in which the symbol occurs: the same symbol may have different numeric representations if they occur in different positions in the input sequence.

MED's **decoder** component is responsible for taking the hidden state activations and predicting the output sequence. It generates the output sequence iteratively (i.e., character-by-character). For each output symbol predicted, we use a component known as an **attention mechanism** to compute a probability distribution over the n columns of the hidden state activations. The attention mechanism can be thought of a model which generates "soft", probabilistic alignments between the input and output sequence. By multiplying each hidden state column by its attention probability and summing across the rows, we obtain the **context matrix**, a f-length vector of real numbers. The prediction of the next symbol is conditioned on the context matrix and on the previously generated outputs. By convention, we add an end-of-string symbol `<eos>` to the end of the output sequence and continue generating output symbols until this symbol is generated. The decoding process is illustrated in Figure 5.

6 Hybrid models

Knowledge-based and data-driven approaches to computational morphology represent two extremes, and various attempts have been made to hybridize these approaches so as to exploit their particular strengths.

Neural network morphological generators, while powerful, tend to make occasional grievous errors (e.g., Corkery et al. 2019, Gorman et al. 2019). Therefore, one might combine a finite-state generator and a neural generator. Under one strategy, the finite-state generator produces a set of hypothesis forms, and a neural network is trained to pick the most likely form given

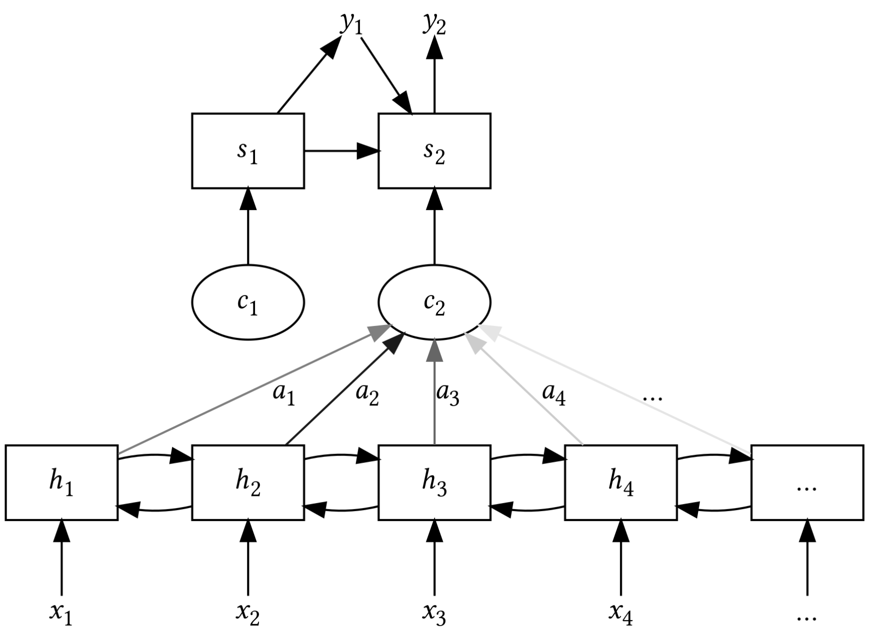

Figure 5: Illustration of decoding in the MED neural network morphological generation system; x_i: input; y_i: output; h_i: hidden state; a_i: attention probability; c_i: context; s_i: decoder state. The system is shown in the process of generating the second output symbol, y_2.

the local linguistic context. Similar techniques could also be used to combine the strengths of finite-state and neural morphological taggers. In either case, the hybrid system is far less likely to make severe errors than the neural network by itself thanks to the filtering effect of the finite-state analyzer/generator component, whereas the neural network component is well-suited for disambiguation using local linguistic context.

Neural networks are considered to be "data-hungry" models in the sense that they often require large amounts of training data to achieve acceptable performance on linguistic tasks like morphological analysis. **Data augmentation** refers to techniques used to gather additional data. One simple data augmentation technique for neural network analyzers-generators is to sample training data—complex words and their analyses—that are generated by, and thus consistent with, a finite-state morphological analyzer (e.g., Schwartz et al. 2019).

7 Resources for computational morphology

Nearly all the resources needed to study computational morphology are free to anyone with a library card and a reliable internet connection. We briefly list some of the data and software that might be used by the modern computational morphologist.

7.1 Data

Finite-state morphological analyzers, of varying coverage and quality, are available for many languages. Apertium, an open-source machine translation platform, provides free finite-state analyzer-generators for several dozen languages at time of writing (Tyers et al. 2012). Similarly, Giellatekno also provides finite-state analyzer-generators, with a particular focus on Uralic languages (Moshagen et al. 2013).

There are several free databases of multilingual morphological annotations. These consist of either inflectional paradigms, or entire sentences, with annotations indicating each word's lemma and morphological features. There are two distinct—but largely compatible—standards for morphological feature specifications. UniMorph (McCarthy et al. 2020) is a free multilingual database of inflectional paradigms for over a hundred languages. Most of these paradigms have been extracted from Wiktionary, a free online dictionary. Each entry in UniMorph consists of a triple: a wordform, that word's lemma, and a morphological feature specification. UDLexicons (Sagot 2018) provides morphological triples for 38 languages using a slightly different set, but largely compatible, set of morphological features.

Universal Dependencies (Nivre et al. 2020) is a free database of 150 treebanks, collections of sentences with syntactic annotations. In most of these treebanks, each word has been annotated with its lemma, part of speech, and a morphological feature bundle. This database been used to study morphological analysis and generation in context.

None of the above resources include item-and-process-style morphological segmentations, but such data has been made for a handful of languages—Arabic, English, Finnish, German, and

Turkish—from the Morpho Challenge shared tasks (Kurimo et al. 2010).

Resources for derivational morphology are more limited. The proprietary CELEX database (Baayen et al. 1996) contains derivational analyses for Dutch, English, and German words. MorphoLex (Sánchez-Gutiérrez et al. 2018) provides morphological decompositions (including derivational affixes) for roughly 70,000 English words. DErivBase (Zeller et al. 2013) is a free database of German derivational relationships. Kyjánek (2018) reviews computational resources for the study of computational derivational morphology.

7.2 Software

The Natural Language Processing Toolkit (NLTK; Bird et al. 2009), an open-source Python library, provides a number of stemmers and lemmatizers. While most of these tools are specific to a single language, NLTK's implementation of the Snowball stemmer includes grammars for fifteen languages at time of writing. NLTK also provides implementations of several part-of-speech models, and provides a Python interface for several other models. There are many open-source tools for finite-state grammar development, including Foma (Hulden 2009) which uses a specialized, domain-specific programming language, and Pynini (Gorman 2016), a Python library. Open-source tools for unsupervised morphological analysis include Morfessor (Virpioja et al. 2013) and Linguistica (Lee and Goldsmith 2016). In contrast, there are few "off-the-shelf" options for neural network-based morphology. Such systems are often built using open-source neural network frameworks like PyTorch or TensorFlow, but such systems may require custom hardware such as graphics processing units (GPUs) for efficient processing.

Acknowledgements

Thanks to Amal Aissaoui for help with Arabic, Magdalena Markowska for help with Polish, and M. Elizabeth Garza for help with Spanish. Andrew Kirby, Arya McCarthy, Brian Roark, Allison Sliter, and Richard Sproat provided helpful comments.

Further Reading

Eistenstein, Jacob. 2019. *An Introduction to Natural Language Processing*. MIT Press.

Goldberg, Yoav. 2017. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool.

Gorman, Kyle, and Richard Sproat. 2021. *Finite-State Text Processing*. Morgan & Claypool.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2008. *Introduction to Automata Theory, Languages, and Computation*. Pearson.

Roark, Brian, and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press.

Notes

[1] In traditional grammar, these are sometimes referred to as **declensional classes** for the inflectional patterns of nouns and adjectives, and **conjugation classes** for the inflectional patterns of verbs.

[2] Definitions given here are discussed in greater detail in Gorman and Sproat 2021, ch. 1, and in a different form in Hopcroft et al. 2008, ch. 3.

[3] In the formulation here, there is only one initial state but may be multiple final states, and that the initial state may also be final. However, we can conceive of equivalent alternative formulations with multiple initial states, with singleton final states, or both.

[4] You may have noticed the similarity between regular languages and **regular expressions**, a computing technique for string matching. The relationship between regular languages and regular expressions are discussed by Hopcroft et al. (2008: ch. 3) and Eistenstein (2019: ch. 9), among others.

[5] The description of this model given here largely follows item-and-process theories of morphology (§2.5). However, Roark and Sproat (2007: ch. 3) argue that item-and-process and item-and-arrangement theories are computationally equivalent in the sense that both can be implemented by rational relations and finite-state transducers, so this seems to be a purely notational distinction.

[6] One can also introduce other types of affixes such as infixes (p. 3, 121f.) using context-dependent rewrite rules. For a worked example, see Gorman and Sproat 2021, p. 86f.

[7] For more information about context-dependent rewrite rules and their compilation into FSTs, see Gorman and Sproat 2021, ch. 5.

[8] However, it is possible to model total reduplication patterns with **two-way finite-state transducers** (Dolatian and Heinz 2018), an extension to ("one-way") finite-state transducers which allows the machine to move backwards (and not just forwards) along the input tape.

[9] The arg max operator denotes the "argument of the maximum", or the argument—here the tag sequence $\mathbf{T}$—that gives the highest possible value of the following expression.

[10] For more information about HMMs and the Viterbi algorithm in particular, see Rabiner 1989.

[11] Rumelhart and McClelland's model required them to make many outmoded simplifying assumptions—in part due to severe limitations of the underpowered digital computers and the primitive state of neural networks of the era—and the authors arguably made overly rosy claims about the performance of their system (Pinker and Prince 1988). See Kirov and Cotterell 2018, and Corkery et al. 2019 for some recent discussion.

[12] For simplicity of exposition, we focus on the general case of **reinflection**, which involves mapping between arbitrary cells of morphological paradigms. When input forms are lemmas, this task is instead known as **morphological generation**; when output forms are lemmas, this task is instead known as **lemmatization**. A related task

is **paradigm completion**, filling in the missing cells of a partial morphological paradigm.

[13] For simplicity of exposition, we omit details of training since the parameters of this model are learned using well-known techniques. Neural network training is discussed in detail by Goldberg (2017: ch. 2) and Eistenstein (2019: ch. 2), among others.

Exercises

Regular languages (after Hopcroft et al. 2008: 114) Give a regular expression that describes all the possible ways one might write a phone number, including area and/or country codes. Then, draw it as a finite-state automaton.

Finite automata The superscript plus sign denotes a variant of the Kleene star closure that excludes the empty string language. That is, if A is a language, $A^{+} = A \cup AA \cup AAA \cup \ldots$. The superscript question mark indicates the union of a language with the empty string language. That is, if A is a language, $A^{?} = \{\varepsilon\} \cup A$. Using these new definitions, draw a finite-state acceptor that accepts the regular language $\{\texttt{ab}\}\{\texttt{cde}\}^{*}\{\texttt{fg}\}^{+}\{\texttt{hi}\}^{?}$.

Esperanto (after Beesley and Karttunen 2003: 219f.) The constructed language Esperanto uses suffixation to form nouns. Nouns minimally consist of a stem such as *hund-* 'dog' and a noun-forming suffix *-o*. Three suffixes may occur between the stem and the *-o*: the feminine *-in*, the diminutive *-et*, and the augmentative *-eg*. These latter three suffixes may co-occur, in any order, without any appreciable difference in meaning. Finally, a word-final *-j* marks a noun plural. Some examples are shown below.

hundo	'dog'
hundino	'female dog'
hundinoj	'female dogs'
hundinego	'big female dog'
hundegino hundinego	'big female dog'
hundegetino hundinetego	'big small female dog'

Write a regular language that accepts all possible words derived from the stem *hund-*. Then, compile this language into an automaton using a finite-state toolkit like Foma or Pynini, and show that it accepts all of the Esperanto words listed above.

Arabic (after McCarthy and Prince 1990) Broadly speaking, Arabic has two types of plurals. The "sound plurals" are formed by adding a suffix to the singular, whereas the "broken plurals", are somewhat more complex. Some broken plurals are shown below.

	sg.	pl.	
a.	ɣurfah	ɣuraf	'room'
	rukbah	rukab	'knee'
	luʕbah	luʕab	'toy'
	ʔusrah	ʔusar	'family'
	nusxah	nusax	'copy'
b.	ħikmah	ħikam	'wisdom'
	qitˤtˤah	qitˤatˤ	'female cat'
	fitnah	fitan	'temptation'
	miħnah	miħan	'ordeal'
	sikkah	sikak	'rail'
c.	qalb	qulu:b	'heart'
	baħθ	buħu:θ	'research'
	taqs	tuqu:s	'weather'
	qasˤr	qusˤu:r	'castle'
	ʕilm	ʕulu:m	'science'
d.	fa:kihah	fawa:kih	'fruit'
	ba:rid͡ʒah	bawa:rid͡ʒ	'battleship'
	ra:ʔiħah	rawa:ʔiħ	'smell'
	ʕa:tˤifah	ʕawa:tˤif	'emotion'
	na:fiðah	nawa:fið	'window'

For each of the four groups, state, in prose, the relationship between the singular and plural forms. Then, write rational relations for each of the four groups that maps from the singular to the plural. You may want to think of each relation as passing through consonants (and certain vowels), mapping some vowels onto other vowels, and deleting or inserting certain sequences of segments. Then, if possible, compile these relations into transducers using a finite-state toolkit like Foma or Pynini, and show that the transdcuers produce the correct mappings.

Spanish (after Gorman et al. 2019) For the 2017 CoNLL-SIGMORPHON shared task, participants trained neural networks to perform an inflection generation task in 52 languages. The systems take lemmas and UniMorph feature vectors as inputs, and attempt to predict the corresponding inflected form. Some of the task's Spanish data—consisting of indicative, present-tense verbs (`V;IND;PRS`)—is shown below.

lemma	features		inflection	
acosar	`V;IND;PRS;3;SG`	→	acosa	'pursue'
aforar	`V;IND;PRS;2;SG`	→	afueras	'lease'
apostar	`V;IND;PRS;3;PL`	→	apuestan	'bet'
conmover	`V;IND;PRS;3;SG`	→	conmueve	'affect'
consolar	`V;IND;PRS;2;SG`	→	consuelas	'console'
dentar	`V;IND;PRS;1;SG`	→	diento	'teethe'
disolver	`V;IND;PRS;2;SG`	→	disuelves	'dissolve'
encerrar	`V;IND;PRS;2;SG`	→	encierras	'lock up'
fusilar	`V;IND;PRS;2;SG`	→	fusilas	'shoot'
implicar	`V;IND;PRS;1;SG`	→	implico	'imply'
infringir	`V;IND;PRS;3;SG`	→	infringe	'infringe'
moler	`V;IND;PRS;3;PL`	→	muelen	'grind'
presagiar	`V;IND;PRS;1;SG`	→	presagio	'presage'
prescribir	`V;IND;PRS;3;PL`	→	prescriben	'prescribe'
recontar	`V;IND;PRS;2;SG`	→	recuentas	'recount'
recusar	`V;IND;PRS;3;SG`	→	recusa	'recuse'
rondar	`V;IND;PRS;2;SG`	→	rondas	'patrol'
sublevar	`V;IND;PRS;1;SG`	→	sublevo	'revolt'
subvertir	`V;IND;PRS;3;PL`	→	subvierten	'subvert'
surcar	`V;IND;PRS;3;SG`	→	surca	'plow'
trasegar	`V;IND;PRS;3;SG`	→	trasiega	'move around'
trovar	`V;IND;PRS;2;SG`	→	trovas	'write poetry'

Gorman et al. (2019) analyze the errors made by the top-performing models in this task, and find errors like **encom**e**nda* for *encom**ie**nda* 's/he entrust' and **rec**o**lan* for *rec**ue**lan* 'they re-strain'. What examples in the training data given above might be responsible for these errors? Are lemmas and features sufficient to predict the inflectional forms of unseen nouns, or is additional information needed?

References

Baayen, R. Harald, Richard Piepenbrock, and Léon Gulikers. 1996. CELEX2. LDC96L14.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations*.

Beesley, Kenneth R., and Lari Karttunen. 2003. *Finite State Morphology*. CSLI.

Bird, Steven, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly.

Chomsky, Noam, and Morris Halle. 1968. *Sound Pattern of English*. Harper & Row.

Chrupała, Grzegorz, Georgiana Dinu, and Josef van Genabith. 2008. Learning morphology with Morfette. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation*, 2362–2367.

Corkery, Maria, Yevgen Matusevych, and Sharon Goldwater. 2019. Are we there yet? Encoder-decoder neural networks as cognitive models of English past tense inflection. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3868–3877.

Dolatian, Hossep, and Jeffrey Heinz. 2018. Modeling reduplication with 2-way finite-state transducers. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, 66–77.

Eistenstein, Jacob. 2019. *An Introduction to Natural Language Processing*. MIT Press.

Fraser, Alexander, Helmut Schmid, Richárd Farkas, Renjing Wang, and Hinrich Schütze. 2013. Knowledge sources for constituent parsing of German, a morphologically rich and less-configurational language. *Computational Linguistics* 39:57–85.

Gillick, Dan. 2009. Sentence boundary detection and the problem with the U.S. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, 241–244.

Goldberg, Yoav. 2017. *Neural Network Methods for Natural Language Processing*. Morgan & Claypool.

Gorman, Kyle. 2016. Pynini: a Python library for weighted finite-state grammar compilation. In *Proceedings of the ACL Workshop on Statistical NLP and Weighted Automata*, 75–80.

Gorman, Kyle, Arya D. McCarthy, Ryan Cotterell, Ekaterina Vylomova, Miikka Silfverberg, and Magdalena Markowska. 2019. Weird inflects but OK: making sense of morphological generation errors. In *Proceedings of the 23rd Conference on Computational Natural Language Learning*, 140–151.

Gorman, Kyle, and Richard Sproat. 2021. *Finite-State Text Processing*. Morgan & Claypool.

Heinz, Jeffrey. 2018. The computational nature of phonological generalizations. In *Phonological Typology*, ed. Larry M. Hyman and Frans Plank, 126–195. Mouton de Gruyter.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. 2008. *Introduction to Automata Theory, Languages, and Computation*. Pearson.

Hulden, Mans. 2009. Foma: a finite-state compiler and library. In *Proceedings of the Demonstrations Session at EACL 2009*, 29–32.

Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description*. Mouton.

Kann, Katharina, and Hinrich Schütze. 2016. Single-model encoder-decoder with explicit morphological representation for reinflection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 555–560.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kibrik, Aleksandr E. 1998. Archi. In *The Handbook of Morphology*, ed. Andrew Spencer and Arnold M. Zwicky, 455–476. Blackwell.

Kirov, Christo, and Ryan Cotterell. 2018. Recurrent neural networks in linguistic theory: revisiting Pinker and Prince (1988) and the past tense debate. *Transactions of the Association for Computational Linguistics* 6:651–665.

Klatt, Dennis. 1987. Review of text-to-speech conversion for English. *Journal of the Acoustical Society of America* 82:737–793.

Kleene, Stephen C. 1956. Representation of events in nerve nets and finite automata. In *Automata Studies*, ed. Claude E. Shannon and J. McCarthy, 3–42. Princeton University Press.

Koskenniemi, Kimmo. 1983. Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Doctoral dissertation, University of Helsinki.

Kurimo, Mikko, Sami Virpioja, Ville Turunen, and Krista Lagus. 2010. Morpho Challenge 2005-2010: evaluations and results. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, 87–95.

Kyjánek, Lukáš. 2018. Morphological resources of derivational word-formation relations. Technical Report 2018-61, Institute of Formal and Applied Linguistics, Charles University.

Lee, G. M., ed. 2012. *Oxford Latin Dictionary*. Claredon Press, 2nd edition.

Lee, Jackson, and John Goldsmith. 2016. Linguistica 5: unsupervised learning of linguistic structure. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 22–26.

Lignos, Constantine, and Charles Yang. 2016. Morphology and language acquisition. In *The Cambridge Handbook of Morphology*, ed. Andrew R. Hippisley and Gregory Stump, 765–791. Cambridge.

McCarthy, Arya D., Christo Kirov, Matteo Grella, Amrit Midhi, Patrick Xia, Kyle Gorman, Ekaterina Vylomova, Sebastian Mielke, Garrett Nicolai, Miikka Silfverberg, Stefanie Reed, Yuval Pinter, Cassandra L. Jacobs, Ryan Cotterell, Mans Hulden, and David Yarowsky. 2020. UniMorph 3.0: universal morphology. In *Proceedings of the 12th Language Resources and Evaluation Conference*, 3922–3931.

McCarthy, John J., and Alan S. Prince. 1990. Foot and word in prosodic morphology: the Arabic broken plural. *Natural Language & Linguistic Theory* 8:209–283.

Moshagen, Sjur N., Tommi Pirinen, and Trond Trosterud. 2013. Building an open-source development infrastructure for language technology projects. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013)*, 343–352.

Nash, D. 1986. *Topics in Warlpiri Grammar*. Garland.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. 2020. Universal Dependencies v2: an evergrowing multilingual treebank collection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, 4034–4043.

Petrov, Slav, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation*, 2089–2096.

Pinker, Steven, and Alan Prince. 1988. On language and connectionism: analysis of a parallel distributed processing model of language acquisition. In *Connections and Symbols*, ed. Steven Pinker and Jacques Mehler. MIT Press.

Porter, Martin F. 1980. An algorithm for suffix stripping. *Program* 14:130–137.

Rabiner, Lawrence R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77:257–286.

Roark, Brian, and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press.

Rumelhart, David, and Jay McClelland. 1986. On learning the past tenses of English verbs. In *Parallel Distributed Processing: Explorations into the Microstructure of Cognition. Vol. 2: Psychological and Biological Models*, ed. Jay McClelland, David Rumelhart, and the PDP Research Group, 216–271. Bradford Books/MIT Press.

Sagot, Benoît. 2018. A multilingual collection of CoNLL-U-compatible morphological lexicons. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, 1861–1867.

Schwartz, Lane, Emily Chen, Benjamin Hunt, and Sylvia L.R. Schreiner. 2019. Bootstrapping a neural morphological analyzer for St. Lawrence Island Yupik from a finite-state transducer. In *Proceedings of the 3rd Workshop on the Use of Computational Methods in the Study of Endangered Languages Volume 1 (Papers)*, 87–96.

Sproat, Richard. 2010. *Language, Technology, and Society*. Oxford University Press.

Sánchez-Gutiérrez, Claudia H., Hugo Mailhot, S. Hélène Deacon, and Maximiliano A. Wilson. 2018. MorphoLex: A derivational morphological database for 70,000 English words. *Behavior Research Methods* 50:1568–1580.

Tyers, Francis M., Felipe Sánchez-Martínez, and Mikel L. Forcada. 2012. Flexible finite-state lexical selection for rule-based machine translation. In *Proceedings of the 16th Annual Conference of the European Association for Machine Translation*, 213–220.

Virpioja, Sami, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. 2013. Morfessor 2.0: Python implementation and extensions for Morfessor baseline. Technical Report 25/2013, Aalto University.

Zaliznyak, Andrey A., ed. 1977. *Grammatičeskij slovar' russkogo jazyka.* Russkij Jazyk.

Zeller, Britta, Jan Šnajder, and Sebastian Padó. 2013. DErivBase: inducing and evaluating a derivational morphology resource for German. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1201–1211.