D-RECOGNIZE fails if the string it is pointing includes as a proper substring a legal string for the FSA. That is, D-RECOGNIZE fails if there is an extra character at the end of the string.

2.10 Give an algorithm for negating a deterministic FSA. The negation of an FSA accepts exactly the set of strings that the original FSA rejects (over the same alphabet), and rejects all the strings that the original FSA accepts.

2.11 Why doesn’t your previous algorithm work with NFSA? Now extend your algorithm to negate an NFSA.

3 MORPHOLOGY AND FINITE-STATE TRANSDUCERS

A writer is someone who writes, and a stinger is something that stings. But fingers don’t fing, grocers don’t groce, haberdashers don’t haberdash, hammers don’t ham, and humdingers don’t humding.

Richard Lederer, Crazy English

Chapter 2 introduced the regular expression, showing for example how a single search string could help a web search engine find both woodchuck and woodchucks. Hunting for singular or plural woodchucks was easy; the plural just tacks an s on to the end. But suppose we were looking for another fascinating woodland creatures; let’s say a fox, and a fish, that surly peccary and perhaps a Canadian wild goose. Hunting for the plurals of these animals takes more than just tacking on an s. The plural of fox is foxes; of peccary, peccaries; and of goose, geese. To confuse matters further, fish don’t usually change their form when they are plural (as Dr. Seuss points out: one fish two fish, red fish, blue fish).

It takes two kinds of knowledge to correctly search for singulars and plurals of these forms. Spelling rules tell us that English words ending in -y are pluralized by changing the -y to -i- and adding an -es. Morphological rules tell us that fish has a null plural, and that the plural of goose is formed by changing the vowel.

The problem of recognizing that foxes breaks down into the two morphemes fox and -es is called morphological parsing.

Key Concept #2. Parsing means taking an input and producing some sort of structure for it.

We will use the term parsing very broadly throughout this book, including many kinds of structures that might be produced; morphological, syntactic,
Section 3.1. Survey of (Mostly) English Morphology

Morphology is the study of the way words are built up from smaller meaning-bearing units, morphemes. A morpheme is often defined as the minimal meaning-bearing unit in a language. So for example the word *fox* consists of a single morpheme (the morpheme *fox*) while the word *cats* consists of two: the morpheme *cat* and the morpheme *-s*.

As this example suggests, it is often useful to distinguish between two broad classes of morphemes: stems and affixes. The exact details of the distinction vary from language to language, but intuitively, the stem is the "main" morpheme of the word, supplying the main meaning, while the affixes add "additional" meanings of various kinds.

Affixes are further divided into prefixes, suffixes, infixes, and circumfixes. Prefixes precede the stem, suffixes follow the stem, circumfixes do
both, and infixes are inserted inside the stem. For example, the word eats is composed of a stem eat and the suffix -s. The word unbuckle is composed of a stem buckle and the prefix un-. English doesn’t have any good examples of circumfixes, but many other languages do. In German, for example, the past participle of some verbs formed by adding ge- to the beginning of the stem and -t to the end; so the past participle of the verb sagen (to say) is gesagt (said). Infixed, in which a morpheme is inserted in the middle of a word, occur very commonly for example in the Philippine language Tagalog. For example the affix um, which marks the agent of an action, is infixed to the Tagalog stem hingi “borrow” to produce humingi. There is one infix that occurs in some dialects of English in which taboo morpheme like **f****s**king or **bl****dy** or others like it are inserted in the middle of other words (**Man-f****s**king-hattan, “abs-o-bl**dy-lutely”) (McCawley, 1978).

Prefixes and suffixes are often called concatenated morphology since a word is composed of a number of morphemes concatenated together. A number of languages have extensive non-concatenative morphology, in which morphemes are combined in more complex ways. The Tagalog inflexion example above is one example of non-concatenative morphology, since two morphemes (hingi and um) are intermingled. Another kind of non-concatenative morphology is called templatic morphology or root-and-pattern morphology. This is very common in Arabic, Hebrew, and other Semitic languages. In Hebrew, for example, a verb is constructed using two components: a root, consisting usually of three consonants (CCC) and carrying the basic meaning, and a template, which gives the ordering of consonants and vowels and specifies more semantic information about the resulting verb, such as the semantic voice (e.g., active, passive, middle). For example the Hebrew tri-consonantal root lmd, meaning ‘learn’ or ‘study’, can be combined with the active voice CaCaC template to produce the word lamed, ‘he studied’, or the intensive CiCeC template to produce the word limed, ‘he taught’, or the intensive passive template CuCaC to produce the word lamed, ‘he was taught’.

A word can have more than one affix. For example, the word writes has the prefix re-, the stem write, and the suffix -s. The word unbelievably has a stem (believe) plus three affixes (un-, -able, and -ly). While English doesn’t tend to stack more than four or five affixes, languages like Turkish can have words with nine or ten affixes, as we saw above. Languages that tend to string affixes together like Turkish does are called agglutinative languages.

There are two broad (and partially overlapping) classes of ways to form words from morphemes: inflection and derivation. Inflection is the combination of a word stem with a grammatical morpheme, usually resulting in a word of the same class as the original stem, and usually filling some syntactic function like agreement. For example, English has the inflectional morphe *s* for marking the plural on nouns, and the inflectional morpheme *ed* for marking the past tense on verbs. Derivation is the combination of a word stem with a grammatical morpheme, usually resulting in a word of a different class, often with a meaning hard to predict exactly. For example the verb computerize can take the derivational suffix *-ation* to produce the noun computerization.

### Inflectional Morphology

English has a relatively simple inflectional system; only nouns, verbs, and sometimes adjectives can be inflected, and the number of possible inflectional affixes is quite small.

English nouns have only two kinds of inflection: an affix that marks plural and an affix that marks possessive. For example, many (but not all) English nouns can either appear in the bare stem or singular form, or take a plural suffix. Here are examples of the regular plural suffix *s*, the alternative spelling *-es*, and irregular plurals:

<table>
<thead>
<tr>
<th>Regular Nouns</th>
<th>Irregular Nouns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singular</td>
<td>Plural</td>
</tr>
<tr>
<td>cat</td>
<td>cats</td>
</tr>
<tr>
<td>thrush</td>
<td>thrashes</td>
</tr>
<tr>
<td>mouse</td>
<td>mice</td>
</tr>
<tr>
<td>ox</td>
<td>oxen</td>
</tr>
</tbody>
</table>

While the regular plural is spelled *-s* after most nouns, it is spelled *-es* after words ending in *-s* (ibis/bises) , -z, (waltz/waltzes) -sh, (thrush/thrasher) -ch, (fitch/fitches) and sometimes -x (box/boxes). Nouns ending in *-y* preceded by a consonant change the *-y* to *-i* (butterfly/flies). The possessive suffix is realized by apostrophe *-*s* for regular singular nouns (llama’s) and plural nouns not ending in *-s* (children’s) and often by a lone apostrophe after regular plural nouns (llamas’) and some names ending in *-s* or *-z* (Euripides’ comedies).

English verbal inflection is more complicated than nominal inflection. First, English has three kinds of verbs; main verbs, (eat, sleep, impeach), modal verbs (can, will, should), and primary verbs (be, have, do) (using...
the terms of Quirk et al., 1985). In this chapter we will mostly be concerned
with the main and primary verbs, because it is these that have inflectional
endings. Of these verbs a large class are regular, that is to say all verbs of
this class have the same endings marking the same functions. These regular
verbs (e.g. walk, or inspect), have four morphological forms, as follow:

<table>
<thead>
<tr>
<th>Morphological Form Classes</th>
<th>Regularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem</td>
<td>walk</td>
</tr>
<tr>
<td>-s form</td>
<td>merge</td>
</tr>
<tr>
<td>-ing participle</td>
<td>try</td>
</tr>
<tr>
<td>Past form or -ed participle</td>
<td>map</td>
</tr>
</tbody>
</table>

These verbs are called regular because just by knowing the stem we
can predict the other forms, by adding one of three predictable endings, and
making some regular spelling changes (and as we will see in Chapter 4, reg-
ular pronunciation changes). These regular verbs and forms are significant
in the morphology of English first because they cover a majority of the verbs,
and second because the regular class is productive. As discussed earlier, a
productive class is one that automatically includes any new words that enter
the language. For example the recently-created verb fax (My mom faxed me
the note from cousin Everett), takes the regular endings -ed, -ing, -es (Note
that the -s form is spelled faxes rather than fass; we will discuss spelling
rules below).

The irregular verbs are those that have some more or less idiosyn-
cratic forms of inflection. Irregular verbs in English often have five different
forms, but can have as many as eight (e.g. the verb be) or as few as three (e.g.
cut or hit). While constituting a much smaller class of verbs (Quirk et al.
(1985) estimate there are only about 250 irregular verbs, not counting auxil-
iaries), this class includes most of the very frequent verbs of the language.2
The table below shows some sample irregular forms. Note that an irregular
verb can inflect in the past form (also called the preterite) by changing its
vowel (e.g. ate), or its vowel and some consonants (caught), or with no
ending at all (cut/cut).

---

2 In general, the more frequent a word form, the more likely it is to have idiosyncratic
properties; this is due to a fact about language change; very frequent words preserve their
form even if other words around them are changing so as to become more regular.

Section 3.1. Survey of (Mostly) English Morphology

<table>
<thead>
<tr>
<th>Morphological Form Classes</th>
<th>Irregularly Inflected Verbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>stem</td>
<td>eat</td>
</tr>
<tr>
<td>-s form</td>
<td>catch</td>
</tr>
<tr>
<td>-ing participle</td>
<td>cut</td>
</tr>
<tr>
<td>Past form</td>
<td>eats</td>
</tr>
<tr>
<td>-ed participle</td>
<td>catches</td>
</tr>
<tr>
<td></td>
<td>eating</td>
</tr>
<tr>
<td></td>
<td>catching</td>
</tr>
<tr>
<td></td>
<td>cutting</td>
</tr>
<tr>
<td></td>
<td>ate</td>
</tr>
<tr>
<td></td>
<td>caught</td>
</tr>
<tr>
<td></td>
<td>eaten</td>
</tr>
<tr>
<td></td>
<td>caught</td>
</tr>
</tbody>
</table>

The way these forms are used in a sentence will be discussed in Chap-
ters 8–12 but is worth a brief mention here. The -s form is used in the “habit-
ual present” form to distinguish the third-person singular ending (She jogs
every Tuesday) from the other choices of person and number (I/you/we/they
jog every Tuesday). The stem form is used in the infinitive form, and also
after other certain verbs (I'd rather walk home, I want to walk home). The
-ing participle is used when the verb is treated as a noun; this particular
kind of nominal use of a verb is called a gerund: Fishing is fine if you
live near water. The -ed participle is used in the perfect construction (He's
eaten lunch already) or the passive construction (The verdict was overturned
yesterday).

In addition to noting which suffixes can be attached to which stems,
we need to capture the fact that a number of regular spelling changes occur
at these morpheme boundaries. For example, a single consonant sound is
doubled before adding the -ing and -ed suffixes (beg/begging/begged). If the
final letter is "e", the doubling is spelled "ke" (picnic/picnicking/picknicked).
If the base ends in a silent -e, it is deleted before adding -ing and -ed (merge/
merging/merged). Just as for nouns, the -s ending is spelled -es after verb
stems ending in -s (osses), -z (waltzes) -sh, (wash/washes) -ch, (catch/catches) and sometimes -x (tax/taxes). Also like nouns, verbs ending
in -y preceded by a consonant change the -y to -i (try/tries).

The English verbal system is much simpler than for example the Eu-
ropean Spanish system, which has as many as fifty distinct verb forms for
each regular verb. Figure 3.1 shows just a few of the examples for the verb
*amar, 'to love'. Other languages can have even more forms than this Spanish
example.

Derivational Morphology

While English inflection is relatively simple compared to other languages,
derivation in English is quite complex. Recall that derivation is the combi-
nation of a word stem with a grammatical morpheme, usually resulting in a word of a different class, often with a meaning hard to predict exactly.

A very common kind of derivation in English is the formation of new nouns, often from verbs or adjectives. This process is called nominalization. For example, the suffix -ation produces nouns from verbs ending often in the suffix -ize (computerize → computerization). Here are examples of some particularly productive English nominalizing suffixes.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Base Verb/Adjective</th>
<th>Derived Noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ation</td>
<td>computerize (V)</td>
<td>computerization</td>
</tr>
<tr>
<td>-ee</td>
<td>appoint (V)</td>
<td>appointee</td>
</tr>
<tr>
<td>-er</td>
<td>kill (V)</td>
<td>killer</td>
</tr>
<tr>
<td>-ness</td>
<td>fuzzy (A)</td>
<td>fuzziness</td>
</tr>
</tbody>
</table>

Adjectives can also be derived from nouns and verbs. Here are examples of a few suffixes deriving adjectives from nouns or verbs.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Base Noun/Verb</th>
<th>Derived Adjective</th>
</tr>
</thead>
<tbody>
<tr>
<td>-al</td>
<td>computation (N)</td>
<td>computational</td>
</tr>
<tr>
<td>-able</td>
<td>embrace (V)</td>
<td>embracing</td>
</tr>
<tr>
<td>-less</td>
<td>clue (N)</td>
<td>clueless</td>
</tr>
</tbody>
</table>

Derivation in English is more complex than inflection for a number of reasons. One is that it is generally less productive; even a nominalizing suffix like -ation, which can be added to almost any verb ending in -ize, cannot be added to absolutely every verb. Thus we can’t say *eating or *spelling (we use an asterisk (*) to mark “non-examples” of English). Another is that there are subtle and complex meaning differences among nominaliz-

### Section 3.2. Finite-State Morphological Parsing

Let’s now proceed to the problem of parsing English morphology. Consider a simple example: parsing just the productive nominal plural (-s) and the verbal progressive (-ing). Our goal will be to take input forms like those in the first column below and produce output forms like those in the second column.

<table>
<thead>
<tr>
<th>Input</th>
<th>Morphological Parsed Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>cats</td>
<td>cat +N +PL</td>
</tr>
<tr>
<td>cat</td>
<td>cat +N +SG</td>
</tr>
<tr>
<td>cities</td>
<td>city +N +PL</td>
</tr>
<tr>
<td>geese</td>
<td>goose +N +PL or (goose +V)</td>
</tr>
<tr>
<td>geoses</td>
<td>goose +V +3SG</td>
</tr>
<tr>
<td>merging</td>
<td>merge +V +PRES-PART</td>
</tr>
<tr>
<td>caught</td>
<td>(catch +V +PAST-PART) or (catch +V +PAST)</td>
</tr>
</tbody>
</table>

The second column contains the stem of each word as well as assorted morphological features. These features specify additional information about the stem. For example the feature +N means that the word is a noun; +SG means it is singular, +PL that it is plural. We will discuss features in Chapter 11; for now, consider +SG to be a primitive unit that means “singular”. Note that some of the input forms (like caught or goose) will be ambiguous between different morphological parses.

In order to build a morphological parser, we’ll need at least the following:

1. **lexicon**: the list of stems and affixes, together with basic information about them (whether a stem is a Noun stem or a Verb stem, etc.).
2. **morphotactics**: the model of morpheme ordering that explains which classes of morphemes can follow other classes of morphemes inside a word. For example, the rule that the English plural morpheme follows the noun rather than preceding it.
3. **orthographic rules**: these spelling rules are used to model the changes that occur in a word, usually when two morphemes combine (e.g., the
The Lexicon and Morphotactics

A lexicon is a repository for words. The simplest possible lexicon would consist of an explicit list of every word of the language (every word, i.e., including abbreviations (“AAA”) and proper names (“Jane” or “Beijing”) as follows:

a
AAA
AA
Aachen
aardvark
aardwolf
aba
abaca
aback

Since it will often be inconvenient or impossible, for the various reasons we discussed above, to list every word in the language, computational lexicons are usually structured with a list of each of the stems and affixes of the language together with a representation of the morphotactics that tells us how they can fit together. There are many ways to model morphotactics; one of the most common is the finite-state automaton. A very simple finite-state model for English nominal inflection might look like Figure 3.2.

The FSA in Figure 3.2 assumes that the lexicon includes regular nouns (\textit{reg-noun}) that take the regular -\textit{s} plural (e.g., \textit{cat}, \textit{dog}, \textit{fox}, \textit{aardvark}). These are the vast majority of English nouns since for now we will ignore the fact that the plural of words like \textit{fox} have an inserted \textit{e}: \textit{foxes}. The lexicon also includes irregular noun forms that don’t take -\textit{s}, both singular \textit{irreg-sg-noun} (goose, mouse) and plural \textit{irreg-pl-noun} (geese, mice).

A similar model for English verbal inflection might look like Figure 3.3.

This lexicon has three stem classes (\textit{reg-verb-stem}, \textit{irreg-verb-stem}, and \textit{irreg-past-verb-form}), plus four more affix classes (-\textit{ed} past, -\textit{ed} participle, -\textit{ing} participle, and third singular -\textit{s}).
### English derivational morphology

English derivational morphology is significantly more complex than English inflectional morphology, and so automata for modeling English derivation tend to be quite complex. Some models of English derivation, in fact, are based on the more complex context-free grammars of Chapter 9 (Sproat, 1993; Orgun, 1995).

As a preliminary example, though, of the kind of analysis it would require, we present a small part of the morphotactics of English adjectives, taken from Antworth (1990). Antworth offers the following data on English adjectives:

- big, bigger, biggest
- cool, cooler, coolest, coolly
- red, redder, reddest
- clear, clearer, clearest, clearly, unclear, unclosely
- happy, happier, happiest, happily
- unhappy, unhappier, unhappiest, unhappily
- real, unreal, really

An initial hypothesis might be that adjectives can have an optional prefix (un-), an obligatory root (big, cool, etc) and an optional suffix (-er, -est, or -ly). This might suggest the the FSA in Figure 3.4.

<table>
<thead>
<tr>
<th>reg-verb-stem</th>
<th>irreg-verb-stem</th>
<th>irreg-past-verb</th>
<th>past</th>
<th>past-part</th>
<th>pres-part</th>
<th>3sg</th>
</tr>
</thead>
<tbody>
<tr>
<td>walk</td>
<td>cut</td>
<td>caught</td>
<td>-ed</td>
<td>-ed</td>
<td>-ing</td>
<td>-s</td>
</tr>
<tr>
<td>fry</td>
<td>speak</td>
<td>eaten</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>talk</td>
<td>sing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>impeach</td>
<td>sang</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Figure 3.5](image)

This gives an idea of the complexity to be expected from English derivation. For a further example, we give in Figure 3.6 another fragment of an FSA for English nominal and verbal derivational morphology, based on Sproat (1993), Bauer (1983), and Porter (1980). This FSA models a number of derivational facts, such as the well known generalization that any verb ending in -ice can be followed by the nominalizing suffix -ation (Bauer, 1983; Sproat, 1993). Thus since there is a word fossilize, we can predict the word fossilization by following states $q_0$, $q_1$, and $q_2$. Similarly, adjectives ending in -al or -able at $q_3$ (equal, formal, realizable) can take the suffix -ity, or sometimes the suffix -ness to state $q_4$ (naturalness, casualness). We leave it as an exercise for the reader (Exercise 3.2) to discover some of the individual exceptions to many of these constraints, and also to give examples of some of the various noun and verb classes.

We can now use these FSAs to solve the problem of morphological recognition: that is, of determining whether an input string of letters makes up a legitimate English word or not. We do this by taking the morphotactic FSAs, and plugging in each “sub-lexicon” into the FSA. That is, we expand each arc (e.g., the reg-noun-stem arc) with all the morphemes that make up the set of reg-noun-stem. The resulting FSA can then be defined at the level of the individual letter.
by letter with each word on each outgoing arc, and so on, just as we saw in Chapter 2.

Morphological Parsing with Finite-State Transducers

Now that we've seen how to use FSAs to represent the lexicon and incidentally do morphological recognition, let's move on to morphological parsing. For example, given the input cats, we'd like to output \texttt{cat +N +PL}, telling us that cat is a plural noun. We will do this via a version of two-level morphology, first proposed by Koskenniemi (1983). Two-level morphology represents a word as a correspondence between a lexical level, which represents a simple concatenation of morphemes making up a word, and the surface level, which represents the actual spelling of the final word. Morphological parsing is implemented by building mapping rules that map letter sequences like cats on the surface level into morpheme and features sequences like \texttt{cat +N +PL} on the lexical level. Figure 3.8 shows these two levels for the word cats. Note that the lexical level has the stem for a word, followed by the morphological information +N +PL which tells us that cats is a plural noun.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example_of_lexical_and_surface_tapes}
\caption{Example of the lexical and surface tapes.}
\end{figure}

The automaton that we use for performing the mapping between these two levels is the finite-state transducer or FST. A transducer maps between one set of symbols and another; a finite-state transducer does this via a finite automaton. Thus we usually visualize an FST as a two-tape automaton which recognizes or generates pairs of strings. The FST thus has a more general function than an FSA; where an FSA defines a formal language by defining a set of strings, an FST defines a relation between sets of strings. This relates to another view of an FST; as a machine that reads one string and generates another. Here's a summary of this four-fold way of thinking about transducers:
• **FST as recognizer**: a transducer that takes a pair of strings as input and outputs accept if the string-pair is in the string-pair language, and a reject if it is not.

• **FST as generator**: a machine that outputs pairs of strings of the language. Thus the output is a yes or no, and a pair of output strings.

• **FST as translator**: a machine that reads a string and outputs another string.

• **FST as set relation**: a machine that computes relations between sets.

An FST can be formally defined in a number of ways; we will rely on the following definition, based on what is called the Mealy machine extension to a simple FSA:

- **Q**: a finite set of N states \( q_0, q_1, \ldots, q_N \)
- **\( \Sigma \)**: a finite alphabet of complex symbols. Each complex symbol is composed of an input-output pair \( i : o \); one symbol \( i \) from an input alphabet \( I \), and one symbol \( o \) from an output alphabet \( O \), thus \( \Sigma \subseteq I \times O \). \( I \) and \( O \) may each also include the epsilon symbol \( \varepsilon \).
- **\( q_0 \)**: the start state
- **\( F \)**: the set of final states, \( F \subseteq Q \)
- **\( \delta(q, i : o) \)**: the transition function or transition matrix between states. Given a state \( q \in Q \) and complex symbol \( i : o \in \Sigma \), \( \delta(q, i : o) \) returns a new state \( q' \in Q \). \( \delta \) is thus a relation from \( Q \times \Sigma \to Q \).

Where an FSA accepts a language stated over a finite alphabet of simple symbols, such as the alphabet of our sheep language:

\[ \Sigma = \{b, a, !\} \]  

an FST accepts a language stated over **pairs** of symbols, as in:

\[ \Sigma = \{a : a, b : b, ! : !, a : !, a : \varepsilon, \varepsilon : !\} \]  

In two-level morphology, the pairs of symbols in \( \Sigma \) are also called **feasible pairs**.

Where FSAs are isomorphic to regular languages, FSTs are isomorphic to **regular relations**. Regular relations are an extension of pairs of strings, a natural extension of the regular languages, which are sets of strings. Like FSAs and regular languages, FSTs and regular relations are closed under union, although in general they are not closed under difference, complementation and intersection (although some useful subclasses of FSTs are closed under these operations; in general FSTs that are not augmented with the \( \varepsilon \) are more likely to have such closure properties). Besides union, FSTs have two additional closure properties that turn out to be extremely useful:

- **inversion**: The inversion of a transducer \( T \) (\( T^{-1} \)) simply switches the input and output labels. Thus if \( T \) maps from the input alphabet \( I \) to the output alphabet \( O \), \( T^{-1} \) maps from \( O \) to \( I \).

- **composition**: If \( T_1 \) is a transducer from \( I_1 \) to \( O_1 \) and \( T_2 \) a transducer from \( I_2 \) to \( O_2 \), then \( T_1 \circ T_2 \) maps from \( I_1 \) to \( O_2 \).

Inversion is useful because it makes it easy to convert a FST-as-parser into an FST-as-generator. Composition is useful because it allows us to take two transducers that run in series and replace them with one more complex transducer. Composition works as in algebra; applying \( T_1 \circ T_2 \) to an input sequence \( S \) is identical to applying \( T_1 \) to \( S \) and then \( T_2 \) to the result; thus \( T_1 \circ T_2(S) = T_2(T_1(S)) \). We will see examples of composition below.

We mentioned that for two-level morphology it’s convenient to view an FST as having two tapes. The **upper or lexical tape**, is composed from characters from the left side of the \( a : b \) pairs; the **lower or surface tape**, is composed of characters from the right side of the \( a : b \) pairs. Thus each symbol \( a : b \) in the transducer alphabet \( \Sigma \) expresses how the symbol \( a \) from one tape is mapped to the symbol \( b \) on the other tape. For example \( a : \varepsilon \) means that an \( a \) on the upper tape will correspond to nothing on the lower tape. Just as for an FSA, we can write regular expressions in the complex alphabet \( \Sigma \). Since it’s most common for symbols to map to themselves, in two-level morphology we call pairs like \( a : a \) **default pairs**, and just refer to them by the single letter \( a \).

We are now ready to build an FST morphological parser out of our earlier morphotactic FSAs and lexica by adding an extra "lexical" tape and the appropriate morphological features. Figure 3.9 shows an augmentation of Figure 3.2 with the nominal morphological features (+SG and +PL) that correspond to each morpheme. Note that these features map to the empty string \( \varepsilon \) or the word/morpheme boundary symbol # since there is no segment corresponding to them on the output tape.

In order to use Figure 3.9 as a morphological noun parser, it needs to be augmented with all the individual regular and irregular noun stems, replacing the labels **regular-noun-stem** etc. In order to do this we need to update the lexicon for this transducer, so that irregular plurals like *goose* will parse into the correct stem *goose* +N +PL. We do this by allowing the lexicon to also have two levels. Since surface *goose* maps to underlying *goose*, the new lexical entry will be "g: g: o:e o:e s:s e:e". Regular forms are
cascade of transducers with many different levels of inputs and outputs and
converting them into a single "two-level" transducer with one input tape
and one output tape. The algorithm for composition bears some resemblance to
the algorithm for determination of FSAs from page 48; given two automata
T₁ and T₂ with state sets Q₁ and Q₂ and transition functions δ₁ and δ₂, we
create a new possible state (x,y) for every pair of states x ∈ Q₁ and y ∈ Q₂.
Then the new automaton has the transition function:

\[ \delta_3((x_a, y_a), i : c) = (x_b, y_b) \] if

\[ 3 \leq s.t. \delta_1(x_a, i : c) = x_b \]
and \[ \delta_2(y_a, c : i) = y_b \] (3.4)

The resulting composed automaton, T_{lex} = T_{num} \circ T_{items}, is shown
in Figure 3.11 (compare this with the FSA lexicon in Figure 3.7 on page 70).³
Note that the final automaton still has two levels separated by the ;. Because
the colon was reserved for these levels, we had to use the \(|\) symbol in T_{items}
in Figure 3.10 to separate the upper and lower tapes.

![Diagram](image)

Figure 3.10 The transducer T_{items}, which maps roots to their root-class.

This transducer will map plural nouns into the stem plus the morpho-
logical marker +PL, and singular nouns into the stem plus the morpheme
+SG. Thus a surface cats will map to \( \text{cat} +N :i \text{PL}\) as follows:

\[ \text{c} : a :a \text{ t:t} +N :i \text{E} +\text{PL} :i \text{S}\]

That is, \text{c} maps to itself, as do \text{a} and \text{t}, while the morphological feature
+\text{N} (recall that this means "noun") maps to nothing (\text{E}), and the feature +\text{PL}

³ Note that for the purposes of clear exposition, Figure 3.11 has not been minimized in the
way that Figure 3.7 has.

simpler; the two-level entry for fox will now be "f : f o : o x : x", but by
relying on the orthographic convention that \( f \) stands for \( f : f \) and so on,
we can simply refer to it as fox and the form for geese as "g o : e o : e s e".
Thus the lexicon will look only slightly more complex:

<table>
<thead>
<tr>
<th>reg-noun</th>
<th>irreg-pl-noun</th>
<th>irreg-sg-noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>go e o e s e</td>
<td>goose</td>
</tr>
<tr>
<td>cat</td>
<td>sheep</td>
<td>sheep</td>
</tr>
<tr>
<td>dog</td>
<td>m o i u e s c e</td>
<td>mouse</td>
</tr>
<tr>
<td>aardvark</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Our proposed morphological parser needs to map from surface forms
like geese to lexical forms like goose +N +SG. We could do this by cascading
the lexicon above with the singular/plural automaton of Figure 3.9.
Cascading two automata means running them in series with the output of
the first feeding the input to the second. We would first represent the lexicon
of stems in the above table as the FST T_{items} of Figure 3.10. This FST
maps e.g. dog to reg-noun-stem. In order to allow possible suffixes, T_{items}
in Figure 3.10 allows the forms to be followed by the wildcard @ symbol;
\( @ : @ \) stands for "any feasible pair". A pair of the form \( @ : x \), for example will
mean "any feasible pair which has \( x \) on the surface level", and correspondingly
for the form \( x : @ \). The output of this FST would then feed the numertaot
T_{num}.

Instead of cascading the two transducers, we can compose them using
the composition operator defined above. Composing is a way of taking a

![Diagram](image)

Figure 3.9 A transducer for English nominal number inflection T_{num}.
Since both Q₁ and Q₂ are accepting states, regular nouns can have the plural
suffix or not. The morpheme-boundary symbol " and word-boundary marker
# will be discussed below.
Orthographic Rules and Finite-State Transducers

The method described in the previous section will successfully recognize words like aardvarks and mice. But just concatenating the morphemes won’t work for cases where there is a spelling change; it would incorrectly reject an input like foxes and accept an input like fox. We need to deal with the fact that English often requires spelling changes at morpheme boundaries by introducing spelling rules (or orthographic rules). This section introduces a number of notations for writing such rules and shows how to implement the rules as transducers. Some of these spelling rules:

(meaning “plural”) maps to ‘s. The symbol ‘ indicates a morpheme boundary, while the symbol # indicates a word boundary. Figure 3.12 refers to tapes with these morpheme boundary markers as intermediate tapes; the next section will show how the boundary marker is removed.

We can think of these spelling changes as taking as input a simple concatenation of morphemes (the “intermediate output” of the lexical transducer in Figure 3.11) and producing as output a slightly-modified, (correctly-spelled) concatenation of morphemes. Figure 3.13 shows the three levels we are talking about: lexical, intermediate, and surface. So for example we could write an E-insertion rule that performs the mapping from the intermediate to surface levels shown in Figure 3.13. Such a rule might say some-
Since the symbol $\epsilon$ means an empty transition, replacing it means inserting something. The symbol '*' indicates a morpheme boundary. These boundaries are deleted by including the symbol *$\epsilon$* in the default pairs for the transducer; thus morpheme boundary markers are deleted on the surface level by default. (Recall that the colon is used to separate symbols on the intermediate and surface forms). The # symbol is a special symbol that marks a word boundary. Thus (3.5) means "insert an $\epsilon$ after a morpheme-final $z$, $s$, or $x$, and before the morpheme $s'$." Figure 3.14 shows an automaton that corresponds to this rule.

![Figure 3.14](image)

**Figure 3.14** The transducer for the E-insertion rule of (3.5), extended from a similar transducer in Antworth (1990).

The idea in building a transducer for a particular rule is to express only the constraints necessary for that rule, allowing any other string of symbols to pass through unchanged. This rule is used to insure that we can only see the $\epsilon$-$\epsilon$ pair if we are in the proper context. So state $q_0$, which models having seen only default pairs unrelated to the rule, is an accepting state, as is $q_1$, which models having seen a $z$, $s$, or $x$. $q_2$ models having seen the morpheme boundary after the $z$, $s$, or $x$, and again is an accepting state. State $q_3$ models having just seen the E-insertion; it is not an accepting state, since the insertion is only allowed if it is followed by the s morpheme and then the end-of-word symbol #.

The other symbol is used in Figure 3.14 to safely pass through any parts of words that don't play a role in the E-insertion rule. other means "any feasible pair that is not in this transducer"; it is thus a version of @::@ which is context-dependent in a transducer-by-transducer way. So for example when leaving state $q_0$, we go to $q_1$ on the $z$, $s$, or $x$ symbols, rather than following the other arc and staying in $q_0$. The semantics of other depends on what symbols are on other arcs; since # is mentioned on some arcs, it is (by definition) not included in other, and thus, for example, is explicitly mentioned on the arc from $q_2$ to $q_0$.

A transducer needs to correctly reject a string that applies the rule when it shouldn't. One possible bad string would have the correct environment for the E-insertion, but have no insertion. State $q_3$ is used to insure that the $\epsilon$ is always inserted whenever the environment is appropriate; the transducer reaches $q_5$ only when it has seen an $s$ after an appropriate morpheme boundary. If the machine is in state $q_5$ and the next symbol is #, the machine rejects the string (because there is no legal transition on # from $q_5$). Figure 3.15 shows the transition table for the rule which makes the illegal transitions explicit with the "---" symbol.

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>$\epsilon$</th>
<th>$x$</th>
<th>$z$</th>
<th>$\epsilon$</th>
<th>$\epsilon$</th>
<th>#</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$q_1$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>$q_5$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.15** The state-transition table for E-insertion rule of Figure 3.14, extended from a similar transducer in Antworth (1990).

The next section will show a trace of this E-insertion transducer running on a sample input string.

### 3.3 COMBINING FST LEXICON AND RULES

We are now ready to combine our lexicon and rule transducers for parsing and generating. Figure 3.16 shows the architecture of a two-level morphology system, whether used for parsing or generating. The lexicon transducer maps between the lexical level, with its stems and morphological features, and an intermediate level that represents a simple concatenation of morphemes. Then a host of transducers, each representing a single spelling rule constraint, all run in parallel so as to map between this intermediate level and the surface level. Putting all the spelling rules in parallel is a design choice;
we could also have chosen to run all the spelling rules in series (as a long cascade), if we slightly changed each rule.

![Diagram](image)

**Figure 3.16** Generating or parsing with FST lexicon and rules

The architecture in Figure 3.16 is a two-level cascade of transducers. Recall that a cascade is a set of transducers in series, in which the output from one transducer acts as the input to another transducer; cascades can be of arbitrary depth, and each level might be built out of many individual transducers. The cascade in Figure 3.16 has two transducers in series: the transducer mapping from the lexical to the intermediate levels, and the collection of parallel transducers mapping from the intermediate to the surface level. The cascade can be run top-down to generate a string, or bottom-up to parse it; Figure 3.17 shows a trace of the system accepting the mapping from fox's to foxes.

The power of finite-state transducers is that the exact same cascade with the same state sequences is used when the machine is generating the surface tape from the lexical tape, or when it is parsing the lexical tape from the surface tape. For example, for generation, imagine leaving the Intermediate and Surface tapes blank. Now if we run the lexicon transducer, given \( f_{ox} +N +PL \), it will produce fox's# on the Intermediate tape via the same states that it accepted the Lexical and Intermediate tapes in our earlier example. If we then allow all possible orthographic transducers to run in parallel, we will produce the same surface tape.

**Figure 3.17** Accepting foxes: The lexicon transducer \( T_{lex} \) from Figure 3.11 cascaded with the E-insertion transducer in Figure 3.14.

Parsing can be slightly more complicated than generation, because of the problem of ambiguity. For example, foxes can also be a verb (albeit a rare one, meaning "to baffle or confuse"), and hence the lexical parse for foxes could be \( f_{ox} +v +3sg \) as well as \( f_{ox} +N +PL \). How are we to know which one is the proper parse? In fact, for ambiguous cases of this sort, the transducer is not capable of deciding. Disambiguating will require some external evidence such as the surrounding words. Thus foxes is likely to be a noun in the sequence I saw two foxes yesterday, but a verb in the sequence That trickster foxes me every time! We will discuss such disambiguation algorithms in Chapters 8 and 17. Barring such external evidence, the best our transducer can do is just enumerate the possible choices; so we can transduce fox's# into both fox +v +3sg and fox +N +PL.

There is a kind of ambiguity that we need to handle: local ambiguity that occurs during the process of parsing. For example, imagine parsing the input verb \( as\). After seeing ass, our E-insertion transducer may propose that the e that follows is inserted by the spelling rule (for example, as far as the transducer is concerned, we might have been parsing the word \( as\).). It is not until we don't see the # after \( as\), but rather run into another s, that we realize we have gone down an incorrect path.

Because of this non-determinism, FST-parsing algorithms need to incorporate some sort of search algorithm. Exercise 3.8 asks the reader to modify the algorithm for non-deterministic FSA recognition in Figure 2.21 in Chapter 2 to do FST parsing.
Running a cascade, particularly one with many levels, can be unwieldy. Luckily, we’ve already seen how to compose a cascade of transducers in series into a single more complex transducer. Transducers in parallel can be combined by \textit{automaton intersection}. The automaton intersection algorithm just takes the Cartesian product of the states, i.e., for each state $q_i$ in machine 1 and state $q_j$ in machine 2, we create a new state $q_i \times q_j$. Then for any input symbol $a$, if machine 1 would transition to state $q_i$ and machine 2 would transition to state $q_j$, we transition to state $q_i \times q_j$.

Figure 3.18 sketches how this intersection ($\wedge$) and composition ($\circ$) process might be carried out.

![Intersection and composition of transducers.](image)

Since there are a number of rule→FST compilers, it is almost never necessary in practice to write an FST by hand. Kaplan and Kay (1994) give the mathematics that define the mapping from rules to two-level relations, and Antworth (1990) gives details of the algorithms for rule compilation. Mohri (1997) gives algorithms for transducer minimization and determinization.

### 3.4 LEXICON-FREE FSTs: THE PORTER STEMMER

While building a transducer from a lexicon plus rules is the standard algorithm for morphological parsing, there are simpler algorithms that don’t require the large on-line lexicon demanded by this method. These are used especially in Information Retrieval (IR) tasks (Chapter 17) in which a user needs some information, and is looking for relevant documents (perhaps on the web, perhaps in a digital library database). She gives the system a query with some important characteristics of documents she desires, and the IR system retrieves what it thinks are the relevant documents. One common type of query is Boolean combinations of relevant \textit{keywords} or phrases, e.g. (marsupial OR kangaroo OR koala). The system then returns documents that have these words in them. Since a document with the word marsupials might not match the keyword marsupial, some IR systems first run a stemmer on the keywords and on the words in the document. Since morphological parsing in IR is only used to help form equivalence classes, the details of the suffixes are irrelevant; what matters is determining that two words have the same stem.

One of the most widely used such \textit{stemming} algorithms is the simple and efficient Porter (1980) algorithm, which is based on a series of simple cascaded rewrite rules. Since cascaded rewrite rules are just the sort of thing that could be easily implemented as an FST, we think of the Porter algorithm as a lexicon-free FST stemmer (this idea will be developed further in the exercises (Exercise 3.7)). The algorithm contains rules like:

- (3.6) \textit{ATIONAL} $\rightarrow$ \textit{ATE} (e.g., relational $\rightarrow$ relate)
- (3.7) \textit{ING} $\rightarrow$ $\varepsilon$ if stem contains vowel (e.g., motoring $\rightarrow$ motor)

The algorithm is presented in detail in Appendix B.

Do stemmers really improve the performance of information retrieval engines? One problem is that stemmers are not perfect. For example Krovetz (1993) summarizes the following kinds of errors of omission and of commission in the Porter algorithm:

\begin{align*}
\text{Errors of Commission} & \quad \text{Errors of Omission} \\
\text{organization} & \quad \text{European} \\
\text{doing} & \quad \text{analyzes} \\
\text{generalization} & \quad \text{matrices} \\
\text{numerical} & \quad \text{matrix} \\
\text{policy} & \quad \text{noise} \\
\text{university} & \quad \text{spare} \\
\text{negligible} & \quad \text{explain} \\
\text{negligent} & \quad \text{urgency}
\end{align*}

Krovetz also gives the results of a number of experiments testing whether the Porter stemmer actually improved IR performance. Overall he found some improvement, especially with smaller documents (the larger the document, the higher the chance the keyword will occur in the exact form used in the query). Since any improvement is quite small, IR engines often don’t use stemming.
3.5 Human Morphological Processing

In this section we look at psychological studies to learn how multi-morphemic words are represented in the minds of speakers of English. For example, consider the word walk and its inflected forms walks, and walked. Are all three in the human lexicon? Or merely walk plus as well as -ed and -s? How about the word happy and its derived forms happily and happiness? We can imagine two ends of a theoretical spectrum of representations. The full listing hypothesis proposes that all words of a language are listed in the mental lexicon without any internal morphological structure. On this view, morphological structure is simply an epiphenomenon, and walk, walks, walked, happy, and happily are all separately listed in the lexicon. This hypothesis is certainly untenable for morphologically complex languages like Turkish (Hankamer (1989) estimates Turkish as 200 billion possible words). The minimum redundancy hypothesis suggests that only the constituent morphemes are represented in the lexicon, and when processing walks, (whether for reading, listening, or talking) we must access both morphemes (walk and -s) and combine them.

Most modern experimental evidence suggests that neither of these is completely true. Rather, some kinds of morphological relationships are mentally represented (particularly inflection and certain kinds of derivation), but others are not, with those words being fully listed. Stanners et al. (1979), for example, found that derived forms (happiness, happily) are stored separately from their stem (happy), but that regularly inflected forms (pouring) are not distinct in the lexicon from their stems (pour). They did this by using a repetition priming experiment. In short, repetition priming takes advantage of the fact that a word is recognized faster if it has been seen before (if it is primed). They found that lifting primed lift, and burned primed burn, but for example selective didn’t prime select. Figure 3.19 sketches one possible representation of their finding.

![Figure 3.19 Stanners et al. (1979) result: Different representations of inflection and derivation.](image)

In a more recent study, Marslen-Wilson et al. (1994) found that spoken derived words can prime their stems, but only if the meaning of the derived form is closely related to the stem. For example government primes govern, but department does not prime depart. Grainger (1991) found similar results with prefixed words (but not with suffixed words). Marslen-Wilson et al. (1994) represent a model compatible with their own findings as follows:

![Figure 3.20 Marslen-Wilson et al. (1994) result: Derived words are linked to their stems only if semantically related](image)

Other evidence that the human lexicon represents some morphological structure comes from speech errors, also called slips of the tongue. In normal conversation, speakers often mix up the order of the words or initial sounds:

if you break it it'll drop
I don't have time to work to watch television because I have to work

But inflectional and derivational affixes can also appear separately from their stems, as these examples from Fromkin and Rainer (1998) and Garrett (1975) show:

it's not only us who have screw loose (for "screws loose")
words of rule formation (for "rules of word formation")

easy enough (for "easily enough")

which by itself is the most unimplausible sentence you can imagine

The ability of these affixes to be produced separately from their stem suggests that the mental lexicon must contain some representation of the morphological structure of these words.

In summary, these results suggest that morphology does play a role in the human lexicon, especially productive morphology like inflection. They also emphasize the important of semantic generalizations across words, and suggest that the human auditory lexicon (representing words in terms of their sounds) and the orthographic lexicon (representing words in terms of letters)
may have similar structures. Finally, it seems that many properties of language processing, like morphology, may apply equally (or at least similarly) to language comprehension and language production.

3.6 Summary

This chapter introduced morphology, the arena of language processing dealing with the subparts of words, and the finite-state transducer, the computational device that is commonly used to model morphology. Here’s a summary of the main points we covered about these ideas:

- **Morphological parsing** is the process of finding the constituent morphemes in a word (e.g., *cat* → *N* + *PL* for cats).
- English mainly uses *prefixes* and *suffixes* to express *inflectional* and *derivational* morphology.
- English *inflectional* morphology is relatively simple and includes person and number agreement (-s) and tense markings (-ed and -ing).
- English *derivational* morphology is more complex and includes suffixes like *-ation, -ness, -able* as well as prefixes like *co- and re-.*
- Many constraints on the English *morphotactics* (allowable morpheme sequences) can be represented by finite automata.

**Finite-state transducers** are an extension of finite-state automata that can generate output symbols.

**Two-level morphology** is the application of finite-state transducers to morphological representation and parsing.

**Spelling rules** can be implemented as transducers.

- There are automatic transducer-compilers that can produce a transducer for any simple rewrite rule.
- The lexicon and spelling rules can be combined by composing and intersecting various transducers.

- The Porter algorithm is a simple and efficient way to do stemming, stripping off affixes. It is not as accurate as a transducer model that includes a lexicon, but may be preferable for applications like *information retrieval* in which exact morphological structure is not needed.

### Bibliographical and Historical Notes

Despite the close mathematical similarity of finite-state transducers to finite-state automata, the two models grew out of somewhat different traditions. Chapter 2 described how the finite automaton grew out of Turing’s (1936) model of algorithmic computation, and McCulloch and Pitts finite-state-like models of the neuron. The influence of the Turing machine on the transducer was somewhat more indirect. Huffman (1954) proposed what was essentially a state-transition table to model the behavior of sequential circuits, based on the work of Shannon (1938) on an algebraic model of relay circuits. Based on Turing and Shannon’s work, and unaware of Huffman’s work, Moore (1956) introduced the term *finite automaton* for a machine with a finite number of states with an alphabet of input symbols and an alphabet of output symbols. Mealy (1955) extended and synthesized the work of Moore and Huffman.

The finite automata in Moore’s original paper, and the extension by Mealy differed in an important way. In a Mealy machine, the input/output symbols are associated with the transitions between states. The finite-state transducers in this chapter are Mealy machines. In a Moore machine, the input/output symbols are associated with the state; we will see examples of Moore machines in Chapter 5 and Chapter 7. The two types of transducers are equivalent; any Moore machine can be converted into an equivalent Mealy machine and vice versa.

Many early programs for morphological parsing used an affix-stripping approach to parsing. For example Packard’s (1973) parser for ancient Greek iteratively stripped prefixes and suffixes off the input word, making note of them, and then looked up the remainder in a lexicon. It returned any root that was compatible with the stripped-off affixes. This approach is equivalent to the bottom-up method of parsing that we will discuss in Chapter 10.

AMPLE (A Morphological Parser for Linguistic Exploration) (Weber and Mann, 1981; Weber et al., 1988; Hankamer and Black, 1991) is another early bottom-up morphological parser. It contains a lexicon with all possible surface variants of each morpheme (these are called *allomorphs*), together with constraints on their occurrence (for example in English the *-es allomorph of the plural morpheme can only occur after s,* *x,* *z,* *sh,* or *ch).* The system finds every possible sequence of morphemes which match the input and then filters out all the sequences which have failing constraints.
An alternative approach to morphological parsing is called *generate-and-test* or *analysis-by-synthesis* approach. Hankamer's (1986) keCI is a morphological parser for Turkish which is guided by a finite-state representation of Turkish morphemes. The program begins with a morpheme that might match the left edge of the word, and applies every possible phonological rule to it, checking each result against the input. If one of the outputs succeeds, the program then follows the finite-state morphotactics to the next morpheme and tries to continue matching the input.

The idea of modeling spelling rules as finite-state transducers is really based on Johnson's (1972) early idea that phonological rules (to be discussed in Chapter 4) have finite-state properties. Johnson's insight unfortunately did not attract the attention of the community, and was independently discovered by Roland Kaplan and Martin Kay, first in an unpublished talk (Kaplan and Kay, 1981) and then finally in print (Kaplan and Kay, 1994) (see page 15 for a discussion of multiple independent discoveries). Kaplan and Kay's work was followed up and most fully worked out by Koskenniemi (1983), who described finite-state morphological rules for Finnish. Karttunen (1983) built a program called KIMMO based on Koskenniemi's models. Antworth (1990) gives many details of two-level morphology and its application to English. Besides Koskenniemi's work on Finnish and that of Antworth (1990) on English, two-level or other finite-state models of morphology have been worked out for many languages, such as Turkish (Oflazer, 1993) and Arabic (Beesley, 1996). Antworth (1990) summarizes a number of issues in finite-state analysis of languages with morphologically complex processes like inflexion and reduplication (e.g., Tagalog) and gemination (e.g., Hebrew). Karttunen (1993) is a good summary of the application of two-level morphology specifically to phonological rules of the sort we will discuss in Chapter 4. Barton et al. (1987) bring up some computational complexity problems with two-level models, which are responded to by Koskenniemi and Church (1988).

Students interested in further details of the fundamental mathematics of automata theory should see Hopcroft and Ullman (1979) or Lewis and Papadimitriou (1981). Mohri (1997) and Roche and Schabes (1997b) give additional algorithms and mathematical foundations for language applications, including, for example, the details of the algorithm for transducer minimization. Sproat (1993) gives a broad general introduction to computational morphology.

### Exercises

3.1 Add some adjectives to the adjective FSA in Figure 3.5.

3.2 Give examples of each of the noun and verb classes in Figure 3.6, and find some exceptions to the rules.

3.3 Extend the transducer in Figure 3.14 to deal with /sh/ and /ch/.

3.4 Write a transducer(s) for the K insertion spelling rule in English.

3.5 Write a transducer(s) for the consonant doubling spelling rule in English.

3.6 The Soundex algorithm (Odell and Russell, 1922; Knuth, 1973) is a method commonly used in libraries and older Census records for representing people's names. It has the advantage that versions of the names that are slightly misspelled or otherwise modified (common, for example, in handwritten census records) will still have the same representation as correctly-spelled names. (e.g., Jurafsky, Jarofsky, Jarovsky, and Jarovski all map to J612).

   a. Keep the first letter of the name, and drop all occurrences of non-initial
      a, e, i, o, u, w, y

   b. Replace the remaining letters with the following numbers:
      \[ \begin{align*}
      b, f, p, v & \rightarrow 1 \\
      c, g, j, k, q, s, x, z & \rightarrow 2 \\
      d, t & \rightarrow 3 \\
      l & \rightarrow 4 \\
      m, n & \rightarrow 5 \\
      r & \rightarrow 6
      \end{align*} \]

   c. Replace any sequences of identical numbers with a single number (i.e.,
      666 → 6)

   d. Convert to the form Letter Digit Digit Digit by dropping
digits past the third (if necessary) or padding with trailing zeros (if necessary).

   The exercise: write a FST to implement the Soundex algorithm.

3.7 Implement one of the steps of the Porter Stemmer as a transducer.
3.8 Write the algorithm for parsing a finite-state transducer, using the pseudo-code introduced in Chapter 2. You should do this by modifying the algorithm ND-RECOGNIZE in Figure 2.21 in Chapter 2.

3.9 Write a program that takes a word and, using an on-line dictionary, computes possible anagrams of the word, each of which is a legal word.

3.10 In Figure 3.14, why is there a $s, x$ arc from $q_5$ to $q_i$?

## COMPUTATIONAL PHONOLOGY AND TEXT-TO-SPEECH

You like po-tay-to and I like po-tah-to.
You like to-may-to and I like to-mah-to.
Po-tay-to, po-tah-to,
To-may-to, to-mah-to,
Let’s call the whole thing off!

George and Ira Gershwin, *Let’s Call the Whole Thing Off* from *Shall We Dance*, 1937

The debate between the “whole language” and “phonics” methods of teaching reading to children seems at very glance like a purely modern educational debate. Like many modern debates, however, this one recapitulates an important historical dialectic, in this case in writing systems. The earliest independently-invented writing systems (Sumerian, Chinese, Mayan) were mainly logographic: one symbol represented a whole word. But from the earliest stages we can find, most such systems contain elements of syllabic or phonemic writing systems, in which symbols are used to represent the sounds that make up the words. Thus the Sumerian symbol pronounced *ba* and meaning “ration” could also function purely as the sound /ba/. Even modern Chinese, which remains primarily logographic, uses sound-based characters to spell out foreign words and especially geographical names. Purely sound-based writing systems, whether syllabic (like Japanese *hiragana* or *katakana*), alphabetic (like the Roman alphabet used in this book), or consonantal (like Semitic writing systems), can generally be traced back to these early logo-syllabic systems, often as two cultures came together. Thus the Arabic, Aramaic, Hebrew, Greek, and Roman systems all derive from a West Semitic script that is presumed to have been modified by Western Semitic mercenaries from a cursive form of Egyptian hieroglyphs. The