12 MORPHOLOGICAL ANALYSIS

Kemal Oflazer

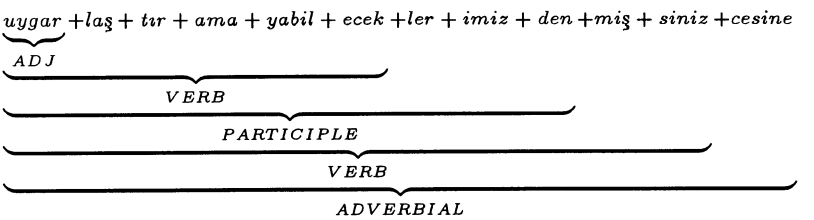
12.1 INTRODUCTION

In the previous chapters, we have seen that a lot of information about the potential tags of tokens in a text is found by lexicon lookup. Another, often complementary source of information is *morphological analysis*, i.e. the process of decomposing words into their constituents. The information about the individual constituents can be used to determine the necessary information about the word as a whole. Such information may range from basic wordclass information assigned from a fixed inventory of tags to structural information consisting of the relationships between components of the word further annotated with various features and their values (cf. Chapter 10). The English word "redness" could thus either be analysed as having the tag NN (singular noun) hiding its internal details, or be analysed by a suitable word grammar to have the structure `Adj(red) + N(+ness)` where the internal structure of the word has been made explicit.

This chapter will present issues in implementing morphological analysers to be used in wordclass tagging or other natural language processing activities, such as syntactic parsing, speech recognition, text–to–speech, spelling checking and correction, document indexing and retrieval. The purpose of this chapter, however, is not to provide a detailed coverage of various aspects of computational morphology; the reader is referred to several recent books covering this topic (see e.g. Sproat (1992) for a quite comprehensive treatment of computational morphology and Ritchie *et al.* (1992) for a description of a morphological analyser and lexicon for English). Instead, after

H. van Halteren (ed.), Syntactic Wordclass Tagging, 175-205.

a short overview of the relevant concepts involved, highlighted with some examples from different languages, this chapter will present issues involved in implementing an industrial strength high-coverage morphological analyser using the *two-level morphology* approach using tools that are either publicly or commercially available.[1] The presentation will not only focus on the usual topics of implementing *morphophonemic/morphographemic* phenomena and *morphotactics* (word grammar) but also more mundane issues such as foreign words, acronyms and abbreviations, numerical tokens, etc., which turn out to be quite important when one has to process real text. This part of the presentation will be based on Turkish, a Ural-Altaic language with agglutinative wordforms. Apart from being the native tongue of the author, and a language that has not until recently been computationally investigated, Turkish is quite interesting for an exposition of this nature for a number of reasons: Turkish (along with languages like Finnish and Hungarian) exhibits phenomena such as vowel harmony which do not show up in Western European languages. Turkish also has very productive inflectional and derivational morphological phenomena. The latter may pose challenging issues in developing a tagset, as the number of forms one can derive from a root form may be in the thousands (some researchers actually give a much higher figure in the millions; cf. Hankamer 1989). Owing to this productivity, Turkish exhibits a quite complex morphotactics, an issue typically not found or not addressed in morphological analysers for many European languages. To illustrate this, we can provide the following rather exaggerated example of a Turkish word: "uygarlaştıramayabilecek-lerimizdenmişsinizcesine"[2] which has the structure:



Despite this complexity, the rules governing Turkish morphology are for most part quite regular and hopefully easily understandable. Understanding issues in developing a morphological analyser for Turkish may actually be quite helpful in dealing with

[1] We will not touch upon quite number of issues such as rule compilation, rule conflicts and their resolution, non-concatenative morphological combinations or the details of specific systems, such as PC-KIMMO or the Xerox Tools, and refer the interested reader to more technical sources with ample coverage of these topics, such as Antworth (1990), Karttunen and Beesley (1992) and Karttunen (1993).

[2] Meaning "behaving as if s/he was one of those whom we could not civilize." Obviously this is not a word that one would use everyday. Turkish words found in typical text average about 10 letters.

many languages that have, for a number of reasons, received less attention from a computational viewpoint.

The chapter starts with a brief overview of morphology and computational morphology and then presents an overview of two-level morphology as a mature state-of-the-art paradigm to implement wide-coverage morphological analysers. It then discusses two general systems for implementing two-level morphological analysers, PC-KIMMO and the Xerox Finite State Tools, covering and contrasting issues such as ease of development, rule compilation, tracing and debugging facilities, speed, memory requirements, etc. This section will mainly look at Turkish as a source of quite interesting problems in implementing an analyser, some of which were alluded to above.

12.2 MORPHOLOGY

Morphology is the study of the structure of the words and how words are formed by combining smaller units of linguistic information called *morphemes*. We will briefly summarize some preliminary notions on morphology, taken from the book by Sproat (1992).

Morphemes can be classified into two groups depending on how they can occur: *free morphemes* can occur by themselves as a word while *bound morphemes* are not words in their own right but have to be attached in some way to a free morpheme. The way in which morphemes are combined and the information conveyed by the morphemes and by their combination differs from language to language. Languages can be loosely classified with the following characterizations: *Isolating languages* are languages which do not allow any bound morphemes to attach to a word. Mandarin Chinese with some minor exceptions is a close example of such a language. *Agglutinative languages* are languages in which bound morphemes are attached to a free morpheme like beads on a string. Turkish, Finnish, Hungarian and Swahili are examples of such languages. In Turkish, e.g., each morpheme usually conveys one piece of morphological information such as tense, agreement, case, etc. *Inflectional languages* are languages where a single bound morpheme (or closely united free and bound forms) simultaneously conveys multiple pieces of information. Latin is a classical example. In the Latin word "amō" (I love), the suffix +ō expresses 1^{st} person singular agreement, present tense, active voice and indicative mood. *Polysynthetic languages* are languages which use morphology to express certain elements (such as verbs and their complements) that often appear as separate words in other languages. Sproat (1992) cites certain Eskimo languages as examples of this kind of a language.

12.2.1 Types of morphology

There are three main types of morphological processes involving morphemes.

Derivational morphology produces a new word usually of a different part-of-speech category by combining morphemes. The new word is said to be *derived* from the

original word. For example, the noun "happiness" is a word derived from the adjective "happy". A derivational process is never demanded by the syntactic context the word is to be used in.

Inflectional morphology introduces relevant information to a word so that it can be used in the syntactic context properly. Such processes do not change the part-of-speech, but add information like person and number agreement, case, definiteness, tense, aspect, etc. For instance in order to use a verb with a third person singular subject in present tense, English syntax demands that the agreement morpheme +s be added, e.g. "comes". Turkish will indicate possible functions for a noun phrase, but requiring that a case morpheme be attached to the head of the phrase, e.g. "ev+i" (the accusative form of "ev" ("house") which can only serve the function of a direct object).

Compounding (cf. 4.3.1) is the concatenation of two or more free morphemes (usually nouns) to form a new word (usually with no or very minor changes in the words involved). Compounding may occur in different ways in different languages. The boundary between compound words and normal words is not very clear in languages like English where such forms can be written separately though conceptually they are considered as one unit, e.g. "firefighter" or "fire-fighter" is a compound word in English while the noun phrase "coffee pot" is an example where components are written separately. German is the prime example of productive use of compounding to create new words on the fly, a textbook example being "Lebensversicherungsgesellschaftsangesteller" consisting of the words "Leben" ("life"), "Versicherung" ("insurance"), "Gesellschaft" ("company") and "Angesteller" ("employee") with some glue in between.

12.2.2 Types of morphological combination

Morphemes can be combined together in a number of ways. In *purely concatenative* combination, the free and bound morphemes are just concatenated. *Prefixation* refers to a concatenative combination where the bound morpheme is affixed to the beginning of the free morpheme or a stem, while *suffixation* refers to a concatenative combination where the bound morpheme is affixed to the end of the free morpheme or a stem. Turkish uses purely concatenative morphological combination with only suffixes attaching to a free morpheme.

In *infixation*, the bound morpheme is inserted to the stem it is attached to. An example is the derivation of "fumikas" ("to be strong") from "fikas" ("strong") in the Bontoc language (Sproat 1992). In *circumfixation*, part of the attached morpheme comes before the stem while another part goes after the stem. In German, e.g., the past participle of a verb such as "tauschen" ("to deceive") is indicated by "getauscht".

Arabic, a language that has long been of interest and challenge to computational morphology uses *templatic* combination where a root word consisting of just consonants is modulated with a template of consonant and vowel alternations. For instance the

root "ktb" (meaning the general concept of writing) can be combined with the template CVCCVC to derive new words such as "kattab" ("to cause to write") or "kuttib" ("to be caused to write").

Reduplication refers to duplicating (some part of) a word to convey morphological information. In Indonesian, e.g., total reduplication is used to mark plurals: "orang" ("man"), "orangorang" ("men") (Sproat 1992). Turkish uses partial reduplication for a limited number of adjectives to derive some emphatic adjectives: "sarı" ("yellow"), "sapsarı" ("very yellow").

In *zero morphology*, derivation/inflection takes place without any additional morpheme. In English the verb "to second (a motion)" is derived from the ordinal "second".

In *subtractive morphology*, part of the wordform is removed to indicate a morphological feature. Sproat (1992) gives the Muskogean language Koasati as an example of such a language, where a part of the form is removed to mark plural agreement.

12.2.3 Computational morphology

Computational morphology studies the computational analysis and synthesis of wordforms for eventual use in natural language processing applications. Almost all applications of computational analysis of wordforms have been on written or orthographic forms of words where tokens are neatly delineated. Since the main theme in this book is the processing of written language, we will from now on assume that we are dealing with written forms of words.

Morphological analysis breaks down a given wordform into its morphological constituents, assigning suitable labels or tags to these constituents. Morphological analysis has analogous problems to all those in full-blown parsing albeit usually at a smaller scale. Words may be ambiguous in their wordclass, e.g. in French, a form such as "danse" has six interpretations:[3]

1)	danse	V(danse)+MOOD/Subj+AGR/3SG	lest s/he dance
2)	danse	V(danse)+MOOD/Subj+AGR/1SG	lest I dance
3)	danse	V(danse)+MOOD/Imp+AGR/2SG	(you) dance!
4)	danse	V(danse)+MOOD/Ind+AGR/3SG	(s/he) dances
5)	danse	V(danse)+MOOD/Ind+AGR/1SG	(I) dance
6)	danse	N(danse)+GEN/Fem+AGR/3SG	dance

In a language like Turkish, whose morphology is more extensive, words may be divided up in a number of ways, e.g. a simple word like "oyun" may be decomposed into constituents in four ways:

1)	oyun	N(oyun)+AGR/3SG+POSS/none+CASE/nom	game
2)	oy+un	N(oy)+AGR/3SG+POSS/2SG+CASE/nom	your vote
3)	oy+[n]un	N(oy)+AGR/3SG+POSS/none+CASE/gen	of the vote
4)	oy+un	V(oy)+MOOD/imp+AGR/2SG	carve !

[3] Unless the output of a specific system is being presented, we will display morphological parses by a sequence of feature/value pairs. [..] indicates elided material.

A number of systems have been developed for computational morphology. These have been mainly intended for a specific language (e.g. DECOMP for English, cf. Allen *et al.* 1987; keçi, for Turkish, cf. Hankamer 1986) though the underlying ideas can be extended to certain other languages. Computational morphology has gained a substantial boost after Koskenniemmi's work which introduced the *two-level morphology* approach (Koskenniemmi 1983). This work was immediately followed by substantial activity to apply the approach to many different languages (Alam 1983; Lun 1983; Karttunen 1983; Karttunen and Wittenburg 1983; Khan 1983) and eventually lead to language independent software tools such as PC-KIMMO (Antworth 1990) and the Xerox Finite State Tools (Karttunen 1993; Karttunen and Beesley 1992).

12.3 TWO-LEVEL MORPHOLOGY

Two level morphology posits two distinct levels of representations for a wordform: the *lexical level* refers to the abstract internal structure of the word consisting of the morphemes making up the word and the *surface level* refers to the orthographic representation of a wordform as it appears in text. The morphemes in the lexical level representation are combined together according to language-specific combination rules possibly undergoing changes along the way, resulting in the surface level representation. The changes that take place during this combination process are defined or constrained by language-specific rules. Such rules can be considered to define the correspondence between the string of symbols making up the lexical level representation and the string of symbols making up the surface level representation. For instance, in English, the lexical form of the word "blemishes" can be represented as `blemish+s` indicating that the root word is `blemish` and the plural marker is the bound morpheme `+s` combined by concatenation indicated by the `+`. The English spelling rule of *epenthesis* requires that an `e` has to be inserted after a root ending with `sh` and before the morpheme `s`, resulting in `blemishes`. We textually represent this correspondence by aligning the lexical and surface characters that map to each other as shown below. In this example and in the examples to follow later the symbol `0` stands for the *null symbol* of zero length which never appears in any surface form when printed.

Lexical: `blemish+0s`
Surface: `blemish0es  blemishes`

A two-level description for a language requires that two components be specified. The *morphographemic component* describes the orthographic changes between lexical and surface levels. The *morphotactics component* describes how the morphemes from the inventory of root words and affixes in the language make up the words. Current implementations of two-level morphology usually assume that morphemes are combined by concatenation which seems to be sufficient at least for languages on which NLP applications are developed. The two components are then used by a *morphological analysis engine* (either directly at run time or after a compilation process) to perform

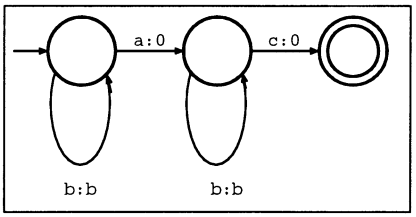


Figure 12.1 The finite-state recognizer for $(b : b)^*(a : 0)(b : b)^*(c : 0)$.

morphological analysis. They could also be used by a *morphological generation engine* which can be used in applications like language generation.

12.3.1 The morphographemic component

The morphographemic component describes the spelling changes that take place between the lexical and surface levels when morphemes are combined to make new wordforms. The changes are expressed by a set of *two-level rules* each of which describes one specific phenomenon (such as epenthesis above), along with the contexts the phenomenon occurs in and whether it is obligatory or optional.

Before we proceed further, some automata-theoretic background would be helpful. Let us consider a finite alphabet whose symbols are actually pairs of atomic symbols `l:s`, where `l` is a lexical symbol and `s` is a surface symbol. One can define regular languages over such pairs of symbols using regular expressions. For instance given the alphabet $A = \{\texttt{a} : \texttt{0}, \texttt{a} : \texttt{a}, \texttt{b} : \texttt{b}, \texttt{c} : \texttt{0}, \texttt{c} : \texttt{c}\}$, the regular expression

$$R = (b : b)^*(a : 0)(b : b)^*(c : 0)$$

describes a regular language containing examples like `b:b b:b b:b a:0 b:b b:b c:0`, where the first three `b:b` pairs match $(b : b)^*$ in the regular expression, `a:0` pair matches the $(a : 0)$, the next two `b:b` pairs match the $(b : b)^*$ and finally the `c:0` pair matches $(c : 0)$. We can also view this string of pairs of lexical–surface symbols as a correspondence, showing the sequence of lexical and surface symbols separately:

Lexical: `bbbabbc`
Surface: `bbb0bb0 bbbbb`

Such a regular expression can be converted into a finite-state recognizer over the same alphabet using standard techniques, as shown in figure 12.1 (cf. e.g. Hopcroft and Ullman 1979).

Another way to view this recognizer is as a *transducer* that maps between strings consisting of the lexical symbols and strings consisting of the surface symbols.[4] Thus, for the example above, the lexical string `bbbabbc` would be transduced to the surface string `bbbbb`, if the lexical level is treated as the input string and the surface level is treated as the output string. The transduction would be in the reverse direction if the roles of the levels are interchanged. On the other hand, the lexical string `bbabbbb` cannot be transduced because it is missing a `c` at the end and hence cannot lead the transducer to its final state.

In general, regular expressions are too low a notation to describe morphographemic changes or correspondences. Two-level morphology provides higher-level notational mechanisms for describing constraints on strings over an alphabet, called the *set of feasible pairs* in two-level terminology. The set of feasible pairs is the set of all possible lexical–surface pairs. Morphographemic changes are expressed by four kinds of rules that specify in which context and how morphographemic changes take place. The contexts are expressed by regular expressions (over the set of feasible pairs) and describe what comes on the left (`LC`, for left context) and on the right (`RC`, for right context), of a morphographemic change.

The *context restriction rule* `a:b => LC _ RC` states that a lexical `a` may be paired with a surface `b` only in the given context, i.e. `a:b` may only occur in this context (if it ever occurs in a string). In this case *the correspondence implies the context*. For instance in English, the `y:i` correspondence (in a word like *happiness* is only allowed between a consonant (possibly followed by an optional morpheme boundary) and a morpheme boundary. This is expressed by a rule like `y:i => C (+:0) _ +:0` where `C` denotes a consonant.

The *surface coercion rule* `a:b <= LC _ RC` states that a lexical `a` *must be* paired with a surface `b` in the given context, i.e. no other pairs with `a` as its lexical symbol can appear in this context. In this case *the context implies the correspondence*. Note that `a:b` is not prohibited from occurring in other contexts. For instance in English, the `s` in a genitive suffix has to be deleted on the surface if the previous consonant is an `s` that belongs to the plural morpheme. One would express this by a rule of the sort `s:0 <= +:0 (0:e) s +:0 ' __`. Note that there are other contexts where an `s` may be dropped, but no obligatorily.

The *composite rule* `a:b <=> LC _ RC` states that a lexical `a` must be paired with a surface `b` in the given context and this correspondence is valid only in the given context. This rule is the combination of the previous two rules. For instance, in English the `i:y` correspondence (as in *tie+ing* being *tying*), is valid only before an `e:0` correspondence followed by a morpheme boundary followed by an `i`, and furthermore, in this context,

[4] Such transducers are slightly different from the classical finite state transducers in that (i) they have final states just like finite state recognizers and (ii) a transduction is valid only when the input leads the transducer into one of the final states.

a lexical `i` *has* to be paired with a surface `y`. This is expressed by the composite rule `i:y <=> __ e:0 +:0 i`.

The *exclusion rule* `a:b /<= LC _ RC` states that lexical symbol `a` may not be paired with a surface symbol `b`, i.e. `a:b` cannot occur in this context. For instance, the `y:i` correspondence in the context restriction rule above cannot occur if the morpheme on the right hand side starts with an `i` or a `'` (the genitive marker). Thus a rule like `y:i /<= C (+:0) _ +:0 [ i | ' ]` prevents the context restriction rule from applying in situations like *try+ing* or *spy+'s*.

The constraints expressed by these rules are compiled into finite-state recognizers which operate in parallel on the lexical and surface symbol pairs. A given string of lexical–surface pairs is accepted by a collection of such recognizers if none of the individual recognizers ends up in a rejecting state.

We will illustrate the possibilities of this system with some examples for two-level rules and the corresponding recognizers.

Turkish Vowel Harmony. Turkish has a phenomenon called *vowel harmony* where with some exceptions, the vowels in suffix morphemes have to agree in certain phonetic features with the most recent vowel in the stem the morpheme is attached to. For instance, in its (considerably) simplified form, if the surface representation of the last vowel in the stem is a front vowel (one of "e", "i", "ö" or "ü" in Turkish), then an unrounded back vowel (which will be represented in the lexical representation by the symbol as `A`) in a morpheme is resolved as "e" on the surface. Otherwise, if the last vowel is a back vowel (one of "a", "ı", "o" or "u"), then it is resolved as "a". The following data exemplifies this phenomenon:

Lexical:	`masa+lAr`	`N(table)+AGR/3pl`
Surface:	`masa0lar`	`masalar`
Lexical:	`okul+lAr`	`N(school)+AGR/3pl`
Surface:	`okul0lar`	`okullar`
Lexical:	`ev+lAr`	`N(house)+AGR/3pl`
Surface:	`ev0ler`	`evler`
Lexical:	`gül+lAr`	`N(rose)+AGR/3pl`
Surface:	`gül0ler`	`güller`

Thus, we have two feasible pairs in our set of feasible pairs with `A` as its lexical symbol: `A:a` and `A:e`. Let us also assume we have the additional feasible pairs `a:a, b:b , ..., z:z`, (called the default pairs), and `+:0` for morpheme boundaries. The rule

```
A:a  <=>  [ A:a | a:a | ı:ı | u:u | o:o ]
          [ b:b | c:c | ...  | z:z ]* +:0
          [ b:b | c:c | ...  | z:z ]* __
```

indicates the A should be paired only with an a in the left context comprising 1) a surface back vowel (indicated by the first set of alternatives following <=>), 2) followed by any number of feasible pairs of consonants pairing with themselves (indicated by the second set of alternatives), 3) followed by a morpheme boundary (+:0) and 4) followed by again any number of consonant pairs. The right context is irrelevant, which is shown by there being is nothing after the __.

As such, the rule looks quite verbose and clumsy, but a little bit of additional notational convention leads to quite succinct rule descriptions. We define the following shortcuts:

- @ acts as a wildcard, matching any symbol
- Vback indicates any surface back vowel
- C is the set of all surface consonants

So @:Vback would denote the set of feasible pairs whose surface symbol is a back vowel. With these conventions, we can write the rule above in a much shorter form:

```
A:a <=> @:Vback @:C* +:0 @:C* __
```

Although the compilation of two-level rules to finite state transducers is beyond the scope of this chapter,[5] let us investigate what the transducer for this rule would look like. We want the transducer for this rule to reject a string of feasible pairs either if A is paired with some surface symbol other than an a in the specified context or if A is paired with a without the requisite context. Otherwise it can accept the string. The transducer in this case is quite simple, as depicted in figure 12.2.[6]
The transducer will go into the *sink state* (state 0) if it encounters an A:a before it encounters the requisite left context or if encounters an A:e (the other feasible pair with A as its lexical symbol) in the context specified.

Two-level transducers are traditionally described using state tables such as:

	@	@	+	A	A	@
	Vback	C	0	a	e	@
1:	2	1	1	0	1	1
2:	2	2	2	2	0	1

The accepted notation in two-level terminology is that states with a . in their label are non-final or rejecting, while states with : are final accepting states. State 1 is the start state, while state 0 is a rejecting sink state which has no transitions to any other state.

[5] See Antworth (1990) on how to manually compile two-level rules.

[6] @:@ is a special case of wild-card use, which does not stand for all possible feasible pairs, but only those that are not covered by the other transition labels. Note also that the tranducer in figure 12.2 has been simplified for expository purposes as it allows any number of occurrences of +:0. However, this is unproblematic as the input never contains such a repetition.

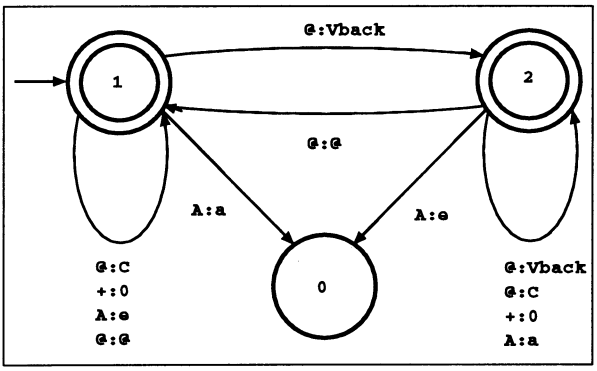


Figure 12.2 Transducer for the `A:a` vowel harmony rule.

For example this recognizer would accept a lexical–surface pair of strings

Lexical: `masa+lAr N(table)+AGR/3pl`
Surface: `masa0lar masalar`

by going through the following sequence of pairs:[7]

Step	State	Input (Matches)	Next State
1	1	m:m (@:C)	1
2	1	a:a (@:Vback)	2
3	2	s:s (@:C)	2
4	2	a:a (@:Vback)	2
5	2	+:0 (+:0)	2
6	2	l:l (@:C)	2
7	2	A:a (A:a)	2
8	2	r:r (@:C)	2
9	2	#:# (@:@)	2

After the input is consumed, the machine is in state 2 which is an accepting state, hence this sequence is accepted. However, for the pair of strings

Lexical: `masa+lAr`
Surface: `masa0ler`

[7] There are many morphographemic phenomena which are sensitive to recognition of the beginning or the end of a word. The symbol # denotes the special word boundary symbol which marks the beginning or the end of a word. In our tracing examples we will just show this symbol to mark the end of a word.

the recognizer would go to the sink state 0 at step 7 as the vowel harmony condition is violated.

This recognizer, then, handles the back vowel cases. To cover the complementary instance of this kind of vowel harmony, we have to combine it with its companion recognizer,

	@	@	+	A	A	@
	Vfr	C	0	e	a	@
1:	2	1	1	0	1	1
2:	2	2	2	2	0	1

which corresponds to the front vowel (Vfr) rule

```
A:e => @:Vfr @:C* +:0 @:C*_
```

Epenthesis in English. For another example we look to the phenomenon of epenthesis in English, where an e is inserted on the surface. The phenomenon can be exemplified by the following data:

Lexical: fox+s kiss+s church+s spy+s
Surface: foxes kisses churches spies

The two-level rule describing epenthesis could be written as:

```
+:e <=> [ Csib | s h | c h | y:i | o ] _ s [+:@ | #]
```

where Csib stands for the sibilant consonants {s, x, z}. Note that, in this example, instead of using a + : 0 pair and a 0 : e pair, a single pair + : e has been used. The right context is such that either a further morpheme boundary or the end of a word may follow the s.

Clearly English has many more phenomena and the reader is referred to more detailed sources for these such as Ritchie *et al.* (1992) or Karttunen and Wittenburg (1983). In addition, there are quite a number of other sources for information on writing two-level rules and one can refer to those for more comprehensive treatments of both general and language-specific phenomena (e.g. Antworth 1990).

12.3.2 The morphotactics component

So far we have assumed that both the lexical and surface sides were somehow available when we checked whether all the morphographemic constraints were satisfied or not. This by itself is not very interesting, since the whole point in morphological analysis is to find out if a given surface form is valid, and what the (possible) underlying representation(s) is (are).

In order to check if a given surface form corresponds to a properly constructed word in a language, one needs a model of the word structure. This model includes the root words for all parts-of-speech in the language (nouns, adjectives, verbs, adverbs,

connectives, pre/postpositions, exclamations, etc.), the affixes and the paradigms of how root words and affixes combine to create words. Once one has a such a model, it is possible to use a test-and-generate approach to generate a possible lexical representation for a word and then check if this lexical form actually corresponds to the given surface form, subject to all the morphographemic constraints. This generate-and-test approach is constrained by the surface form so that the search for possible lexical representations need not be grossly inefficient in practice (see Barton (1986) for the computational complexity of two-level recognition and Koskenniemmi and Church (1988) for an empirical performance evaluation of a two-level analyser).

Most two-level systems provide simple finite state mechanisms for describing lexicons of root words and affixes and how they are combined. This approach makes the assumption that all affixations are essentially concatenative or can be 'faked' with concatenation. For instance, PC-KIMMO represents the root words and affixes and their sequencing with interlinked lexicons. A simple fragment of a lexicon for English is shown in figure 12.3.[8] Here, the ALTERNATION keyword defines possible continuation lexicons that should be tried after the lexicon entry naming that alternation as its successor. Each lexicon entry has the *lexical* representation of a root word or an affix as its first part. The second part names the successor alternation with End indicating no further successors. The third part is the *gloss* which corresponds to what the morphological analyser prints out when a match occurs.

In PC-KIMMO, the recognition engine operates in the following fashion when attempting to recognize a given surface form:

1. Starting with the initial lexicon, the whole lexicon network is traversed in a *depth first* manner.
2. During this traversal, the linked lexicon structure is used to incrementally create partial lexical forms, by concatenating the lexical symbols from the entries in the lexicon.
3. Whenever the candidate lexical string is extended by a symbol all two-level rule constraints are concurrently checked by all corresponding finite state recognizers (note that the surface form is already available).
4. If none of the recognizers raise any objections to the partial string so far, then the string is extended further.
5. If at least one of the recognizers rejects the partial string by going to a sink state, then the search backs up: the last lexical string that was concatenated is stripped off and a new candidate lexical symbol is appended and checked. If no such candidates are found, then search backs up to a prior point where alternative lexical symbol choices are available.

[8] Such a set of lexicons is sometimes also represented with a finite state machine where arcs are labelled with pairs of lexical forms and corresponding glosses.

```
ALTERNATION VERBS                    AUX IRREGULAR_VERBS  REGULAR VERBS
ALTERNATION REGULAR_VERB-SUFFIXES    V_SUFFIX1 V_SUFFIX2  V_INFL
ALTERNATION NOUN_SUFFIXES                 .....
ALTERNATION ADJ_SUFFIXES                  .....
ALTERNATION ADV_SUFFIXES                  .....
...

LEXICON REGULAR_VERBS
abandon          REGULAR_VERB-SUFFIXES       "V(abandon)"
...
zip              REGULAR_VERB-SUFFIXES       "V(zip)"
END

LEXICON V_SUFFIX1
+ant             NOUN_SUFFIXES               "+N(ant)"
...
+or              NOUN_SUFFIXES               "+N(or)"
...
+ability         NOUN_SUFFIXES               "+ADJ(able)+N(ity)"
END

LEXICON V_SUFFIX2
+able            ADJ_SUFFIXES                "+ADJ(able)"
...
+ibly            ADV_SUFFIXES                "+ADJ(able)+ADV(ly)"
END

LEXICON V_INFL
+ed              End                         "+PAST"
+ing             End                         "+PROG"
+s               End                         "+3SG"
0                End                         ""
END
```

Figure 12.3 Lexicons in PC-KIMMO.

6. If, when the string is completed, none of the recognizers reject the string, then the constructed lexical form corresponds to the given surface form.

Although the linked lexicon mechanism is quite useful for implementing certain paradigms, it limits the options of morpheme combinations to just concatenation. This is generally not a real problem: prefixing, suffixing and compounding phenomena can be implemented in an obvious way. Phenomena such as infixation or circumfixation can also be handled, though not as cleanly (Antworth 1990). On the other hand, when developing a wide-coverage and accurate morphological analyser, one needs to be concerned with a number of issues such as handling *exceptions* to general paradigms, handling *lexicalized derivations* and preventing *overgeneration* by *restricting affixation.*

All languages have wordforms which are *exceptions* to the general inflectional or derivational paradigm. For instance, a large number of English verbs do not follow the general paradigm for inflections. The past tense form of `eat` is not formed by the suffix `+ed`, but rather is `ate`. The way to handle this is quite obvious when the number of wordforms are small: one can manually enter all regular and exceptional forms directly into the lexicon. Thus, in the simplified lexicon above, `eat` would be associated with an alternation which only points to the derivational suffixes and the inflected forms would be entered into the same lexicon as:

```
...
eat       VB_DER_SUFFIXES   "V(eat)"
eats      End               "V(eat)+3SG"
ate       End               "V(eat)+PAST"
eaten     End               "V(eat)+PART"
eating    End               "V(eat)+PROG"
...
```

If there are a number of exceptional cases that behave similarly, it is certainly possible to create affix lexicons for those cases and have their successor point there, rather than explicitly listing all the forms. For instance the wide coverage morphological analyser developed (using PC-KIMMO) at University of Pennsylvania (Karp *et al.* 1992) has eight separate continuation classes for verbs. Verbal roots are partitioned into eight classes, depending on which subset of the three verb suffixes apply regularly to the verbs in that class. For instance, the verb `admire` is in a class which can take all the suffixes regularly, while `teach` is in a class which only takes the `+ing` and `+s` suffixes regularly. The exceptional and idiosyncratic cases are entered explicitly as described above.

Another problem area is the treatment of *lexicalized derivations*. In English, e.g., the word "application" has a lexicalized interpretation when it is used to mean "a computer program". On the other hand, in a sentence like "The application of this theory to this case is useful.", the same word is a noun derived from the verb "apply". Note that in both cases the final part-of-speech of the word is noun so the difference does not matter in the case of part-of-speech tagging. But if the output of the morphological analyser is to be used in parsing, the parser may want to know if the word is derived from a verb to check any relationships with verbal complements. An obvious solution is to have an entry for the lexicalized form in the appropriate lexicon and have the derivation paradigm generate it from the original form as well.

Finally, there is the matter of *overgeneration*. There are many instances of productive derivational affixes which apply only to a restricted subset of the roots. For instance, in English the suffix `+ity`, which derives a noun from an adjective, is not applicable to all adjectives. If the analyser is to be used with the assumption that only valid input will be processed, then overgeneration is not a real problem. But usually, the same two-level morphological description is used for generation or spelling correction where

the generation capabilities may be used to construct words and then one certainly would like to generate only valid words. Hence, if the aim of the morphological analyser is to accept and analyse all valid words in a language and reject all words that are not in the language, then one has to build the morphotactics component very accurately. This can be done by partitioning the root lexicons into classes depending on the subsets of suffixes that can be applied. Since the underlying mechanism, based on linked lexicons, is quite low level, this may become a non-trivial and perhaps unmanageable task. One may have to duplicate certain affix lexicons with very minor changes to account for minor variations in morphotactics. And worse, when the dependencies governing morphotactics are long distance, one may have to duplicate quite a substantial portion of the lexicons. This may easily become a maintenance and debugging nightmare. This is generally a much more serious problem for languages with rich inflectional and derivational morphology, such as Turkish and Finnish, as compared to a language like English. Other solutions are possible as well. Some systems provide special lexical symbols with 0 as their surface realization (e.g. *diacritics* in the Xerox TWOLC rule compiler), which can be used in the lexical representation of appropriate words. These symbols can be referred to by the rules to block or enable any affixation, but are irrelevant to any rule that does not refer to them.

12.3.3 Development tools

The two-level methodology is supported by several development tools. One of these is the PC-KIMMO system, to which we already have made quite a few references. The system exists in two different versions.

PC-KIMMO Version 1 is a publicly available system for developing two-level morphological analysers. It is quite a robust system and has been used to develop a number of quite substantial analysers (e.g. Oflazer 1993, 1994; Karp *et al.* 1992).

The main advantages of PC-KIMMO Version 1 are:

- It is publicly available at no cost (see `http://www.sil.org`).
- It runs on UNIX, DOS and MacOS platforms and the data developed is portable between these platforms (except perhaps for character set encoding).
- It provides quite comprehensive facilities for debugging and tracing analyser operations.
- In addition to being accessible as a stand-alone program the morphological analysis engine can be accessed from user programs.
- It has facilities for processing large batches of words for either analysis or generation, or for comparing with previous results.

- A quite comprehensive user's manual and a book specifically intended to be used with this system are available.

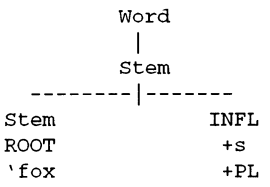
However, PC-KIMMO also has a number of drawbacks that need to be taken into account when developing a large system:

- The rules have to be given as finite state transducers as the software system proper does not provide a rule compiler. There is a preliminary version of a compiler called KGEN that is expressly built for PC-KIMMO rule files, but this compiler is limited in a number of respects. The compiler is a very crucial component in the development of two-level rules, especially for languages with a large number of morphographemic constraints sometimes interacting in unforeseen ways. Hand-compiling is feasible only for the simplest of rules and is very prone to errors.

- PC-KIMMO Version 1 provides linked lexicons as the only mechanism for constructing lexicons. These lexicons are stored as a trie data structure in the underlying analysis engine. There is some removal of redundancy with the trie compression, but nevertheless the necessary storage may be quite substantial. As mentioned earlier, the linked lexicon mechanism is too low a mechanism for implementing complex morphotactic paradigms.

- PC-KIMMO is quite slow in analysis especially with large lexicons. On an Ultra-SPARC 1 system, it can process about 10 words per second when loaded with Turkish description of about 30,000 root words and 35,000 proper nouns.

Version 2 of PC-KIMMO is an enriched version of the original system. The rule component now supports the use of multiple character symbols (multigraphs) and can accept recognizer/transducer descriptions in the format generated by the Xerox TWOLC compiler (see below). Another very important improvement in this version is the inclusion of a unification-based word grammar component, along the lines of the Ritchie *et al.* (1992) and based on the PATR formalism (Shieber 1986). The word parser produces a tree structure with morphemes at the leaves and nodes in the tree being annotated with any relevant features. For instance, for the word "foxes", the word grammar of the English description would produce



```
[cat:  Word
 clitic: -
 head: [pos: N
        number: PL]
 root_pos: N
 root:'fox]
```

Since this output makes the internal structure of the word explicit, it is very easy to integrate the output of this analyser with a syntactic parser. The word grammar is specified using a context-free backbone augmented with constraint equations on the set of morphosyntactic features. In addition to providing a richly annotated word parse, the main advantage of the word grammar is in providing a more powerful model of morphotactics. This lets the developer express morphotactic variations and long-distance dependencies directly in the grammar. The practical effect is that a more accurate and less overgenerating morphotactic component can be constructed.

There is also another rich set of software tools (developed by Xerox and known as the *Xerox Finite State Tools*), based on the theory of finite state calculus developed by Kaplan and Kay (1994), for building and manipulating finite state recognizers and transducers. Although there is support for building classical two-component two-level morphological analyser systems, their current approach is now more and more based on the notion of regular relations applied to NLP problems at different levels starting on tokenization all the way up to finite state parsing.

For developing morphological analysis systems Xerox provides TWOLC (Karttunen and Beesley 1992) for developing the morphographemic rule component and LEXC (Karttunen 1993) for developing the lexicon component. TWOLC is a very sophisticated two-level rule compiler, accepting rule definitions with a very general syntax and powerful operators, and is able to produce finite state transducers with very compact representations. It contains a facility for intersecting the automata generated for a rule set to obtain a single transducer that combines the constraints of all rules. There is also extensive support for testing the rules. LEXC is a lexicon compiler which builds a finite state transducer from a lexicon specification. The main mechanism for lexicon specification is again a linked lexicon model very much like PC-KIMMO Version 1, at least as far as the developer is concerned. There is however a very important provision for describing regular expressions as lexicon entries. This comes in very handy when building number or date parsers right into the lexicon.

The finite state transducers built by both LEXC and TWOLC are determinized and minimized as far as possible (it may not be possible to completely determinize transducers!) and then combined by composition to obtain a final transducer (cf. figure 12.5 below) which contains the complete morphological model of a language. There is no rule interpretation at run time. A generic transduction engine fed with the morphological transducer will map any surface form to a lexical form and vice versa. The system

is very fast. For instance for Turkish with the lexicon sizes given earlier, it can process about 2000 words per second on an UltraSPARC 1 system. Commercial grade versions of the Xerox system have been quoted as providing a much higher performance and much better transducer compression.

The development of high-level and efficient tools for algebraic operations on finite state machines exemplified by the Xerox Finite State Tools has fostered development of morphological analysers which use the full power of finite state calculus. A recent interesting example of this is Beesley's morphological analyser for Arabic (Beesley 1996) in which the lexicon entries for roots are actually regular expressions and finite state calculus operations are used to restrict the overgenerating lexicon entries by the templates. Another example is Chanod's system for French (Chanod 1994), where linguistic generalizations shared across various inflectional paradigms are expressed as separate transducers and then composed to produce the final transducer.

12.3.4 Developing a Morphological Analyser

Developing a morphological analyser for a language involves a considerable amount of planning and preparation. The choice of the tool and/or the formalism depends on a number of issues. As discussed earlier there are public domain tools available which are quite usable but these have considerable drawbacks (such as speed or lack of a rule compiler) if one wants to develop an industrial strength system. Commercial systems such as the Xerox Finite State Tools on the other hand provide speed, good compilation and debugging facilities, and additional machinery for handling problems outside the classical two-level paradigm.

Before any attempt to code the required information in the formalism of the selected tool, the developer should compile:

1. A list of root words annotated with parts-of-speech and other information that may conceivably be needed in the analysis (e.g. any semantic information that may be required for morphotactics). It may also be necessary to include the citation forms (e.g. infinitive forms of verbs) of the roots, if these are different from the root forms.

2. A list of morphemes along with the associated morphological information they encode. Obviously this information has to be represented in a form that is in line with the tagset that will be used or that can be converted to the tagset encoding.

3. A model of the morphotactics described as paradigms or as a high-level description of the sequence of morphemes.

4. A comprehensive listing of the morphographemic phenomena, along with as many examples as possible for each.

5. A large corpus of tokens collected from various sources on which the resulting analyser will be used.

Unless the language has complex morphotactics (such as Turkish or Finnish), the most crucial and time consuming component in the process is the coding and debugging of the rules that describe the morphographemic processes. In the case of two-level morphology, this involves abstracting phenomena and coding them using the rule formalism. This is usually not a trivial process, but Anthworth (1990) provides some very good guidelines and examples for this process (see Karttunen and Beesley (1992) for further examples.) The following table from Antworth (1990) aids one in choosing the rules describing a given morphographemic phenomenon, such as the realization (inhibition) of a lexical symbol `l` as the surface symbol `s` in given left and right contexts `LC` and `RC`.

Rule to use	Is l:s allowed in context LC _ RC ?	Is l:s only allowed in context LC _ RC ?	Must l always correspond to S in context LC _ RC ?
`l:s => LC __ RC`	Yes	Yes	No
`l:s <= LC __ RC`	Yes	No	Yes
`l:s <=> LC __ RC`	Yes	Yes	Yes
`l:s /<= LC __ RC`	No	NA	NA

Rule ordering is not really an issue in two-level morphology, but sometimes rules may interact or conflict in unforeseen ways and these will need to be resolved by careful re-crafting and/or refining of context descriptions (see Karttunen and Beesley (1992) for details on this). Debugging the rule component involves repeated testing of the rules on the examples compiled in (4) above until all are satisfactorily handled.

Getting the morphotactics right involves careful linking of root and affix lexicons so that all valid sequences are allowed while invalid sequences are disallowed. Machinery provided in the two-level development tools is usually sufficient to implement this sequencing, but sometimes one may need to enforce long distance constraints (across multiple morphemes) and this may cause substantial complexity in the lexicon component. Certain tools such as Xerox's provide sophisticated mechanisms for automatically enforcing such constraints.

Once the lexicon and rule components are coded, real testing should be done on actual corpora to gauge and improve coverage. This is quite a time-consuming process as one continually encounters new words, which have to be retrofitted to the lexicon, and as yet uncovered phenomena, for which rules have to be updated. This process continues until a stable and satisfactory point is reached. From this point on, maintenance is a low-intensity effort where updates become progressively infrequent.

12.4 A MORPHOLOGICAL ANALYSER FOR TURKISH

We conclude this chapter with a description of the implementation of a morphological analyser for Turkish that has been built using the Xerox Finite State Tools. The analyser

is two-level in spirit but uses additional levels for a variety of reasons. It uses about 30,000 Turkish root words and about 35,000 proper name roots and implements all derivational and inflectional paradigms of Turkish quite accurately. It has its roots in an earlier PC-KIMMO implementation (Oflazer 1993, 1994).

Turkish is an agglutinative language with word structures formed by productive affixations of derivational and inflectional suffixes to root words. Turkish has finite-state but nevertheless rather complex morphotactics. Morphemes added to a root word or a stem can convert the word from nominal to verbal or vice versa. The morphotactics is highly productive, leading to such adverbial constructs such as the one on page 176, and circular, at least from a competence point of view.[9]

The surface realizations of morphological constructions are constrained and modified by a number of phonetic rules. Vowels in an affixed morpheme have to agree with the preceding vowel in certain aspects to achieve *vowel harmony*, although there are a small number of exceptions. Under certain circumstances vowels in the roots and morphemes are elided. Similarly, consonants in the root words or in the affixed morphemes undergo certain modifications and may again sometimes be deleted. Nevertheless, things are quite regular, especially compared to many European languages. However, the assimilation of a large number of words into the language from various foreign languages – notably French, Arabic and Persian – have resulted in word formations which behave as exceptions to many rules. Our intention in this section is not to cover a two-level description of Turkish morphology in gruesome detail (for a detailed exposition, see Oflazer 1994) but rather to highlight aspects that are potentially of more general interest to researchers who intend to build two-level descriptions for various languages.

12.4.1 Requirements

Underlying the design of the morphological analyser, we find a number of requirements.

1. It was to be used in a number of applications: (i) in parsing, as a component of the lexicon (Güngördü and Oflazer 1995), (ii) in morphological disambiguation (Oflazer and Tür 1996), (iii) in generation (Hakkani and Oflazer 1998) and (iv) as the basis for spelling correction (Oflazer 1996).

2. The morphosyntactic representation produced had to cater to at least all these applications, abstracting from all details of the surface representation. Furthermore, full derivational history of the word form had to be represented.

[9] Turkish allows, among other examples, words of the sort *ev+(ler+de+ki)**, (those things at those things at ... at the houses). Here the morphotactics has to loop back.

Table 12.1 Inflectional morphosyntactic features for verbs.

FEATURE	VALUES	INTERPRETATION
CAT	VERB	Major Category
TYPE	TRANS, INTRANS	Minor Category
VOICE	REFLEX, RECIP CAUS, PASS	Verbal Voice
POLARITY	POS, NEG, NEGC	Sense polarity
COMP	POSSIBILITY, etc.,	Modal Compounding
TAM1	PAST, NARR, PRES, IMP, PROG OPT, NECES, FUT, DESR	Tense-Aspect-Mood Marker 1
TAM2	PAST, NARR, COND	Tense-Aspect-Mood Marker 2
AGR	1SG–3PL	Person Agreement

```
REFLEX: Reflexive voice, RECIP: Reciprocal/Collective voice
CAUS:Causative voice, PASS: Passive voice
COMP corresponds to a set of 10 modal compounding suffixes, the most
important of which is the equivalent of the possibility auxiliary
(``can''/``may'') in English.
Most of the others do not have any close equivalents in English.

PAST: Past Tense, NARR: Narrative Past Tense, PRES: Present Tense
FUT:  Future tense, PROG: Progressive
IMP: Imperative, OPT: Optative Mood, NECES: Necessitative Mood
DESR: Desire mood, COND: Conditional
```

3. For (i) and (ii) above, all analyses and interpretations of a word had to be produced even if the underlying morpheme structure is the same.[10] Furthermore, to alleviate any problems in (iii) and (iv) the morphotactics has to be very accurate with minimal overgeneration.

4. We had to be able to process free-running text such as one finds in newspapers: news items, editorials, etc. Thus it had to be wide-coverage with root lexicons augmented with words from economic, social, legal and technical jargon, in addition to the standard Turkish words one finds in published word lists. Such texts contain lots of proper names,[11] foreign and/or unknown words, abbreviations and acronyms, and numerical tokens, which, though written with numerals in orthography, nevertheless go through the same suffixations demanded by the syntactic context. All these put additional demands on the morphographemic component.

[10] Note that this does not involve any analyses due to polysemy of the root. We are only concerned with morphosyntactic ambiguity.

[11] What makes this a problem in Turkish especially for morphological disambiguation is that any Turkish word with its inflectional and derivational markers is a potential last name.

5. In addition to standard morphographemic phenomena in Turkish, one has to be concerned with the intermingling of such phenomena with words imported into the language from foreign languages. These cause many exceptions and have to be cared for in the morphographemic component. For instance, the author's name "Kemal" will violate vowel harmony when something is affixed (e.g. "Kemal'i"and not "Kemal'ı" because of the interaction between "a" and the palatal "l". A similar phenomenon is found in the French import "alkol" which has an accusative form "alkolü" as opposed to "alkolu".[12]

6. The system had to run with sufficient speed so as not to be the bottleneck in performance.

Let us briefly touch upon some of the issues above:

Morphosyntactic Features. The morphological representation that we have opted to produce is a linear *feature–value* representation of the morphological information encoded by the morphemes, accounting for the fact that certain morphemes signal multiple features, while certain features are signalled by a zero morpheme. The inflectional features for finite verbs, e.g., are presented in table 12.1.

There are also many very productive derivational morphemes – too numerous to list here – that transform words between all major parts-of-speech. The final representation that we would like to get should also represent any derivational history. The linear feature–value sequence represents derivations by the target part-of-speech category and the suffix (if any), marking the derivation, e.g. the word "geldiğimdeki" ("at the time I came"), would have the feature sequence:

```
[[CAT VERB] [ROOT gel] [SENSE POS]
                       [NOUN DIK] [AGR 3SG] [POSS 1SG] [CASE LOC]
                       [ADJ KI]]
```

The linear representation can, when necessary (e.g. for use in parsing), be transformed into a hierarchical representation representing the inflectional features of the last derived category and the derivational history (figure 12.4).

Tokens with numerical components. Tokens involving digits or numbers are frequently found in real text. This again may not be a problem in languages like English where the only tokens that may follow a number may be an "s", "'s" or an ordinal abbreviation. But in Turkish, one encounters tokens like "122.ye" ("to the 122nd") or "33'ünün" ("of 33 of them") where the suffixation proceeds depending on how the numerical component is actually pronounced. Thus the morphotactic engine has to

[12] Some sources still make a distinction in the orthography of such words by putting a little cap on the vowel, but the ISO Latin-5 standard for Turkish character set does not make such distinctions.

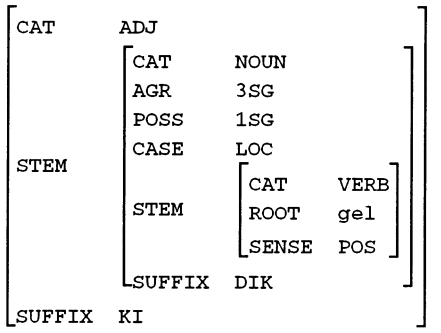


Figure 12.4 Hierarchical representation of a morphological derivation.

have quite a sophisticated finite state parser in order to be able to handle numeric constructions such as these and additional ones like "16:15'te" (time designator), "3:4" (ratio), "%23'ü" (percentage), "2/3'si" and "2/3'ü" (ratios, but suffixation again proceeds depending on the pronunciation of the ratio – numerator or denominator last). The morphographemic rules for vowel harmony and vowel and consonant ellipsis also must take into account the phenomena in such numerical constructions.

Foreign words and abbreviations. Most foreign words used in a text follow their standard orthography in the original language. On the other hand, they do receive the Turkish morphosyntactic markers dictated by the context, the form of the suffixation again being based on the original pronunciation. This may make some constraints like vowel harmony inapplicable on the graphemic representation, though harmony is in effect in the pronunciation. For instance, one sees the form "Carter'a" where the last vowel in "Carter" is pronounced so that it harmonizes with `a` in Turkish, while the `e` in the written surface form does not harmonize with `a`.

A similar problem occurs with abbreviations such as "PTT" or "TV", whose orthographic form cannot supply sufficient information for morphographemic constraints. So a form like "PTT'ye", ("to the PTT") cannot be processed, since it does not have any vowels in its orthographic representation to serve as context for various morphographemic constraints.

The simplest solution to this problem is to collect as many such root forms as possible, place them into the appropriate part-of-speech lexicons and add *hidden* lexical symbols to their lexical representation which indicate how the word is really pronounced. The hidden symbols are considered when evaluating the morphographemic rules but disappear at the surface level (i.e. they are paired with `0` as their surface symbol). For instance, the lexical form for "Carter" would actually be `CarteHidAr`,

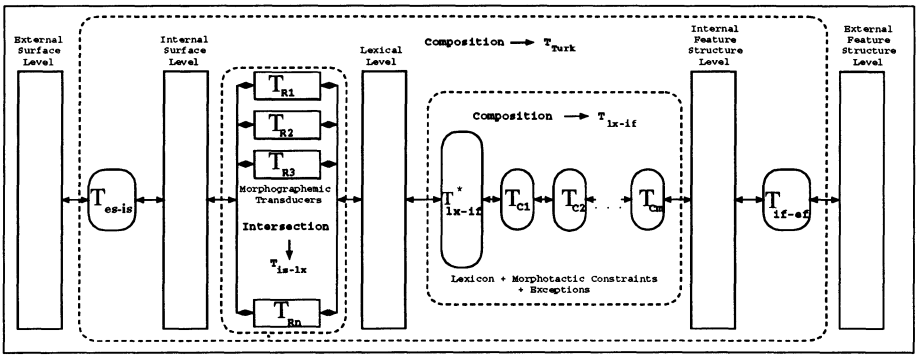


Figure 12.5 The architecture of the Turkish morphological analyser.

with `HidA:0` in the list of feasible pairs. A more principled solution should involve phonetic representations of the words.

12.4.2 System architecture

The morphological analyser has been based on the general architecture proposed by Karttunen (1993), with some additional components. The system has the five level architecture shown in figure 12.5.

The *external surface level* corresponds to a platform/OS/standard specific coding of the surface representations of Turkish words.

The *internal surface level* corresponds to the traditional surface level in two-level morphology. It uses a platform independent ASCII encoding of Turkish surface characters.

The *lexical level* corresponds to the traditional lexical level in two-level morphology, where the internal form of the word is represented as a sequence of morphemes.

The *internal feature structure level* corresponds to the information that is essentially provided by the lexicon glosses in PC-KIMMO. However, in PC-KIMMO itself these glosses are not accessible except for printing. In our system this level serves the very important level of controlling and fine-tuning morphotactics.

The *external feature structure level* corresponds to the actual output of the morphological analysis.

When given a Turkish word at the external surface level, such as: "yediriyordum" ("I was causing someone to eat (it)") the analyser produces the output at the external feature structure level:

```
[[CAT=VERB][ROOT=ye][VOICE=CAUS]
  [SENSE=POS][TAM1=PROG][TAM2=PAST][AGR=1SG]]
```

which indicates the word is a verb with root `ye`, in causative voice, with past tense, progressive aspect and first person singular agreement. The representations of the internal levels are not available directly: in fact, they do not exist as they are factored out during the construction of the analyser.

The analyser as a whole (T_{Turk}) is a large finite state transducer of about 250,000 states and 840,000 transitions, that maps between the external surface level and the external feature structure level. All morphographemic and morphotactic variations are handled at compile time, so there is no rule interpretation at run time: transduction is done directly. This machine is constructed from four finite state transducers, each of which maps between two levels.

The first transducer, T_{es-is}, maps between the external surface level and the internal surface level. On UNIX systems, e.g., there is not much of a support for special Turkish characters so we encode them by prefixing them with a "\". On PC or MacOS platforms one would choose an appropriate character encoding supported by the underlying system and use that as the external surface level encoding. T_{es-is} also deals with upper-case character folding and translates everything to lower case in the internal representation. On a Unix system this would map `\uz\ulm\u\st\um` ("I had felt sorry") to `UzUlmUStUm` while on a PC or a MacOS system with Turkish character support, it will map üzülmüştüm to the same internal surface string.

The second one, T_{is-lx}, maps between the internal surface level and the lexical level. It corresponds to the morphographemic mapping defined by the two-level rules in traditional two-level morphology. It is constructed by intersecting the transducers for each of the two-level morphographemic rules. For instance, it will map the internal surface string `UzUlmUStUm` into its lexical form `Uz+Hl+mHS+DH+Hm`.[13]

The third one, T_{lx-if}, maps between the lexical level and the internal feature structure level. This is the transducer that comprises the root word and suffix lexicons and the morphotactic paradigms. It is the most complicated and crucial transducer and is elaborated further below. As an example, consider the word `yaptırttığında`. This has two lexical level representations. The first, `yap+DHr+t+DHk+sH+nDA` ("at the time you caused someone to have someone else do (it)"), is mapped to the internal feature structure representation

```
[[CAT=VERB][ROOT=yap][VOICE=CAUS-DIR][VOICE=CAUS-T]
  [SENSE=POS][NOUN=DIK][AGR=3SG][POSS=3SG][CASE=LOCn]]
```

and the second, `yap+DHr+t+DHk+Hn+DA` ("at the time s/he caused someone cause someone else do (it)"), to the representation

[13] Here, `H` is a lexical symbol denoting a high-vowel whose surface realization is one of `ı`, `i`, `u`, `ü` and `D` is a lexical symbol whose surface realization is one of `d` or `t`.

```
[[CAT=VERB][ROOT=yap][VOICE=CAUS-DIR][VOICE=CAUS-T]
  [SENSE=POS][NOUN=DIK][AGR=3SG][POSS=2SG][CASE=LOCy]]
```

Here, the root word is represented using internal surface symbols. The voice markers and case markers bear signs of the underlying morpheme (e.g. CAUS-DIR indicates that the causative is marked by the +DHr morpheme). Although such details are not relevant at the external feature structure level, they are crucial in controlling the morphotactics as detailed below.

The fourth transducer, T_{if-ef}, cleans up the internal feature structure representation, mainly by replacing things like CAUS-DIR or LOCn by more meaningful tokens like CAUS or LOC and by changing the lexical form of the root word so that it corresponds to the external surface lemma form. So, for instance, for the first representation above one would get

```
[[CAT=VERB][ROOT=yap][VOICE=CAUS][VOICE=CAUS]
  [SENSE=POS][NOUN=DIK][AGR=3SG][POSS=3SG][CASE=LOC]]
```

at the external feature structure level.

The complete transducer for Turkish, T_{Turk}, is constructed by composing the transducers above:

$$T_{Turk} = T_{es-is} \circ T_{is-lx} \circ T_{lx-if} \circ T_{if-ef}$$

The composition operation eliminates all intermediate levels of representation and provides a direct mapping between the external surface level and the external feature structure level. Thus, e.g., when yaptırttığında is fed to T_{Turk} on the lower side, both

```
[[CAT=VERB][ROOT=yap][VOICE=CAUS][VOICE=CAUS]
  [SENSE=POS][NOUN=DIK][AGR=3SG][POSS=3SG][CASE=LOC]]
```

and

```
[[CAT=VERB][ROOT=yap][VOICE=CAUS][VOICE=CAUS]
  [SENSE=POS][NOUN=DIK][AGR=3SG][POSS=2SG][CASE=LOC]]
```

are produced at the upper side (the right side in the figure). And when one of these are provided at the upper side, the original Turkish word and its all capitalized and initial capitalized variants are produced at the lower side.

The transducers T_{es-is} and T_{if-ef} are quite simple and do not really warrant any further attention but it is instructive to look at T_{is-lx}, which is responsible for the morphographemic phenomena, and especially at T_{lx-if}, which is responsible for very accurate morphotactics.

12.4.3 The morphographemic transducer: T_{is-lx}

T_{is-lx} implements all the morphographemic phenomena in Turkish. It is actually the *intersection* of the transducers which are produced by the Xerox rule compiler TWOLC from rules using the Xerox syntax as specified in Karttunen and Beesley (1992). There are a total of 31 two-level rules covering phenomena such as: vowel harmony, vowel and consonant ellipsis, gemination in words borrowed from Arabic and Persian, devoicing, phenomena involving proper noun suffix separators and productive phenomena in words borrowed from Western languages. The resulting transducer for all these rules, obtained after intersection, has about 3600 states and about 170,000 transitions.

The two-level rules implemented at this level are essentially the ones in the original PC-KIMMO implementation (Oflazer 1994). There are, however, quite a number of additional cases covered, related to phenomena such as exceptions, numerical tokens, foreign words, abbreviations, etc.

It is quite instructive to examine the real version of the `A:e` vowel harmony rule used as an example earlier in 12.3.1:

```
A:e <=> [ VOWEL:FRONTV |
          HidE:0 |
          â:a | û:u | ô:o |
          NumECons:0 | NumEConsT:0 | NumE:0 |
          NumÜCons:0 | NumÜConsT:0 | NumÜ:0 ]
          [ ':'  | CONS | =:0 | +:0 | 0:CONS ]+ __
```

The first part of the left context consists of a number of alternatives regarding the vowel that will enforce the harmony:

`VOWEL:FRONTV` is the basic case covered in the earlier example where one of the front vowels is present as the last vowel on the surface, either in the root or in a preceding morpheme.

`HidE:0` handles all those cases of acronyms or abbreviations whose orthography does not contain any vowel, but whose pronunciation has a front vowel (actually just `e`) as the last vowel (e.g. "PTT"). Such entries in the lexicon have a lexical symbol `HidE` in their lexical representation. This symbol disappears on the surface while enforcing the vowel harmony constraint in words like `PTTdeki`.

The `â:a | û:u | ô:o` alternatives deal with the cases of certain vowels which surface as back vowels but harmonize as front vowels due to a number of reasons. The lexical entries for such words contain a lexically different vowel representation.

The next set of options deal with a variety of cases of the pronunciation of numerical tokens. The parser for such constructs inserts into the lexical representation a symbol that represents how the last vowel and consonant of that number (not necessarily of the

last digit) are pronounced. The vowel harmony rules then take this into account and act appropriately.

Once the vowel context is set, it can be followed by one or more of the proper noun suffix separator (`':'`), any number of consonants, morpheme boundary, lexical symbols that disappear and consonants that are geminated (`0:CONS`). This is intentionally kept quite over-generating as the sequence is actually enforced by the lexicon.

The proper noun suffix separator (represented by an apostrophe) has to be dealt with carefully in *all* the morphographemic rules. The reason is that all morphographemic changes on the left side of a morpheme boundary (in the stem) are blocked if the proper noun suffix separator is present, even though such a distinction is not made in pronunciation. The effect becomes clear when we compare the proper noun "Işık" and the common noun "ışık" ("light")

Lexical:	`Işık+'+nHn`	`ışık+nHn`
Surface:	`Işık0'00ın`	`ışığ00ın`
	`Işık'ın`	`ışığın`

For the proper noun, the apostrophe blocks the `k:ğ` pair in the orthography (but not in pronunciation).

12.4.4 The morphotactics transducer: T_{lx-if}

T_{lx-if} is compiled from Turkish root word and suffix lexicons using the Xerox lexicon compiler LEXC. The basic implementation of the paradigms is intentionally overgenerating. With very minor exceptions, the lexicons and their linking are specified so that they very closely follow the paradigms.

Although it would be possible to implement the paradigms with lexicons linked through continuation classes, exceptions and idiosyncrasies start causing maintenance and debugging problems after a while. Consider the following examples of morphotactic constraints:

1. Personal pronouns have the same inflectional paradigm as nouns, except that possessive suffixes (which happen to be in the middle of the paradigm) are not applicable.

2. 1SG and 2SG pronouns have irregular forms in dative case.

3. The choice of the suffixes to mark a certain morphological feature depends on the morphographemic and morphotactic context, e.g.

    i) The causative voice marker morpheme is `+t` following a previous causative marker (except `+t`) or a verbal root ending in a vowel.

ii) The case morphemes for certain cases are different depending on whether they follow nominals with 3^{rd} person possessive suffixes or monosyllabic 3^{rd} person personal or demonstrative pronouns.

iii) Which verbal agreement suffixes are used depends on the combination of the previous tense-aspect-mood markers.

4. Derivations should be carefully controlled to prevent duplicate applications of the same derivation or conflicting derivations. For instance, in Turkish the `+ci` suffix and its allomorphs productively derive nouns from nouns, but should not be applied repeatedly. The `+li` suffix ("with") should not be applied after the suffix `+siz` ("without") as that would produce a semantically anomalous form.

Although all these (and many more) constraints could be handled within the finite state lexicon mechanism, by duplicating portions of lexicons here and there, manually handling them is cumbersome and verification is quite a nightmare. Instead, we can use the full power of finite state constraints to incorporate such exceptions with minimal effort. We again use the notion of transducer composition. We start with T^*_{lx-if}, a (significantly overgenerating) transducer that is constructed by just implementing the regular derivational and inflectional paradigms, i.e. a given inflectional or derivational suffix is assumed to be applicable productively as long as it is in the correct morphotactic order. This results in a rather clean lexicon structure.

The morphotactic constraints beyond the basic paradigm ordering are expressed as constraints on regular relations using finite state operators over the alphabet of internal feature structure representations. Each such constraint C_i is then compiled into a finite state transducer T_{C_i} (cf. figure 12.5).

As an example, assume that constraint 1 above is expressed in C_1:

```
%[POSS=1SG%] /<= %[CAT=PRON%] %[ROOT= ?+ %] %[TYPE=PERSONAL%] ? __
```

This and five other similar rules indicate that possessive suffixes can *not* appear in a context following a personal pronoun, with any root (matching `?+`), and an agreement marker (matching the last `?`).[14] C_1 (and hence its transducer T_{C_1}) defines a regular relation that maps every string *not containing* the sequence matching the left context above to itself but rejects any string where a possessive marker follows a personal pronoun root and agreement marker. Composing $T^*_{lx-if} \circ T_{C_1}$ produces a new transducer T^1_{lx-if} which is exactly like T^*_{lx-if} except that invalid cases with regard to C_1 are weeded out during the intersection of the upper language of T^*_{lx-if} and the lower language of T_{C_1}.[15]

[14] Note that tokens such as `[POSS=1SG]` or `[CAT=PRON]` are single symbols in the internal feature structure representations. The `%` is the escape character in the Xerox syntax and has to be used here because `[` and `]` are also used as brackets in the rule syntax.

[15] This is slightly different than the way it is implemented in Xerox LEXC because of the direction of composition.

We then add further rules for each of the other constraints. For example 2 above,

```
%[CASE=DAT%] /<= [ %[AGR=1SG%] | %[AGR=2SG%] ] %[POSS=NONE%] __
```

disables the dative suffix for 1SG and 2SG personal pronouns while the exceptional dative entries are entered in to the lexicon directly. And for point i) of example 3,

```
%[VOICE=CAUS-T%] => [ %[VOICE=CAUS-DIR%] | %[VOICE=CAUS-AR%] |
                      %[VOICE=CAUS-HR%]  | %[ROOT= ?* VOWEL %] ] __
```

constrains the selection of the causative voice marker suffix: the first three options denote other causative markers that can precede while the last one constrains the verbal root to end with a vowel. Each such constraint rule $C_1 \ldots C_n$ leads to a new transducer $T_{C_1} \ldots T_{C_n}$ and, after composition, to new transducers T^2_{lx-if}, T^3_{lx-if}, . . ., each a bit more accurate than the previous one. Eventually, we get

$$T_{lx-if} = T^*_{lx-if} \circ T_{C_1} \circ T_{C_2} \circ \cdots \circ T_{C_n}$$

which incorporates all the finite state constraints for morphotactics expressed in our rules.

There is a crucial point to pay attention to in this process. The composition operation is order dependent. However, if the rules are coded so that they specify their contexts very accurately and pass through all strings that they are not sensitive to, the final result of all these compositions is a transducer that maps the lexical level representations to an order independent intersection of the upper languages of individual transducers and that of T^*_{lx-if}.

Composition may be a time-consuming operation, especially when the transducers involved are large. Thus for the situation above it is best to perform the composition of the constraint transducers first in batches and then compose the result of this with the lexicon transducer. The number of constraints for our Turkish implementation is about 200 and all the compilations and composition operations take about 3 minutes of CPU time on an Ultra Sparc 1 system.