

Chapter 3

Englex: A Computational Morphology of English

Last modified November 22, 1995

3.1 What is Englex?

3.2 Design philosophy and goals

3.3 General design features

- 3.3.1 Inflection and derivation
- 3.3.2 Multiple senses and homonyms
- 3.3.3 Lexical conversion
- 3.3.4 Compounds

3.4 The rules

- 3.4.1 Alphabet
- 3.4.2 Accented characters (diacritics)
- 3.4.3 Stress marks
- 3.4.3 Capitalization

3.5 The lexicon

3.6 The word grammar

- 3.6.1 Rules
- 3.6.2 Features and unification
- 3.6.3 Feature constraints
- 3.6.4 Interface between the lexicon and the word grammar

Figure 3.1 Positional analysis of English morphology

Figure 3.2 Finite state machine for English morphotactics

Figure 3.3 ALTERNATION declarations in main lexicon file

Figure 3.4 Key to sublexicon names

Figure 3.5 The INITIAL sublexicon

Figure 3.6 Sample prefix entries

Figure 3.7 Sample root entries

Figure 3.8 Sample suffix entries

Figure 3.9 Sample inflection entries

Figure 3.10 Sample clitic entries

Figure 3.11 The End sublexicon

Figure 3.12 Context-free word grammar rules

Figure 3.13 Positional analysis of *enlargement*

Figure 3.14 Parse tree for *enlargement*

Figure 3.15 Word grammar rules with feature constraints

Figure 3.16 Parse tree and full feature display for *enlargement*

Figure 3.17 List of features and possible values

This chapter is an exposition of a PC-KIMMO description of English called Englex. It is intended to serve as an extended example of how to develop a morphological description of a language with PC-KIMMO. Because the word grammar component is new to PC-KIMMO version 2, the section in this chapter on Englex's word grammar serves as a tutorial on writing word grammars.

3.1 What is Englex?

Englex is a description of English morphology and lexicon using the two-level model as it is implemented by PC-KIMMO. Since it is intended to be used to do practical natural language processing, it uses the standard orthography for English rather than phonological transcription. With Englex and PC-KIMMO (or programs using the PC-KIMMO parser), you can morphologically parse English words and text. Practical applications include morphologically preprocessing text for a syntactic parser and producing morphologically tagged text. Englex can also be used to explore English morphological structure for purposes of linguistic analysis

Englex consists of the three basic components that make up any PC-KIMMO description: a set of phonological (or orthographic) rules, a lexicon, and a word grammar. These components are described in detail in this chapter.

Englex is a development of the sample English description described in appendix A of Antworth 1990. The rules file (which had its origins in Karttunen and Wittenburg 1983) is largely the same, but some differences are noted in section 3.4. The lexicon described in appendix A is only a small toy lexicon and is superseded by Englex. Also, the morphotactic structure described in appendix A of Antworth 1990 has been completely revised in Englex. The word grammar is of course new with version 2 of PC-KIMMO.

An earlier version of Englex, intended for use with version 1 of PC-KIMMO, was released in 1991. It is superseded by this version of Englex which accompanies PC-KIMMO version 2.

3.2 Design philosophy and goals

Englex represents a convergence of two disciplines: natural language processing (NLP) and linguistics. Since the presuppositions, interests, and goals of linguists and NLP researchers do not necessarily coincide, Englex is by necessity a bundle of compromises.

Englex is natural language processing (NLP) tool based on generally-accepted linguistic principles and analyses of English morphology. The basic strategy in building an NLP system like Englex is two-pronged: first, ensure that all well-formed input is analyzed correctly, and second, incrementally refine the system so that it rejects ill-formed input. Both the linguist and NLP researcher would insist that the first goal be met (though even here the NLP researcher might be more forgiving). But with regard to the second goal, only the linguist would require that it be fully met in order for the description to be adequate. For the NLP researcher,

as long as well-formed input is assured, it does not necessarily matter if the system "overrecognizes" (but see below).

For example, Englex will correctly recognize the comparative and superlative forms of adjectives such as *big*, *bigger*, and *biggest*. But it will also recognize the dubious form *aliver* as the comparative form of *alive*. In other words, Englex underspecifies the morphotactic constraints related to adjective inflection; it assumes that all adjectives can have a comparative form, which of course is not true. In practice, we assume that forms such as *aliver* do not occur in well-formed text; thus overrecognition does little harm.

However, overrecognition is by no means innocuous; it can result in spurious parses that seriously degrade the performance of an NLP system. For instance, consider what would happen if we relax the constraint that the comparative *-er* suffix only attaches to adjectives and permit it after any word. A word such as *bigger* would still be correctly parsed as a comparative adjective; but a word such as *writer* would get two parses: one where *-er* is correctly recognized as an agentive suffix that attaches to a verb, and another where *-er* is incorrectly posited as the comparative suffix. By simply encoding the constraint that the comparative suffix can only attach to adjectives, we capture the obvious and important linguistic fact that only adjective have comparative forms and at the same time reduce the number of spurious parses the system produces.

The degree to which we refine a system like Englex depends on our purpose in using the system: to characterize precisely English morphological structure (the linguist's goal) or to process natural language texts to some acceptable degree of accuracy (the NLP researcher's goal). Englex steers a middle course between these purposes, but ultimately it is up to the user to determine the behavior of the system.

In terms of its coverage of English, Englex has these goals:

- To account for all major spelling rules of English.
- To account for all productive morphological structure (affixes, morphotactic constraints, word class conversion, and so on). While a 20,000-entry lexicon sounds small, Englex can actually recognize many times that number of words because it analyzes productive derivational morphology. For example, the lexicon contains entries for *re-*, *compute*, *-er*, *-ize*, and *-ation* and can thus recognize any complex word formed from those parts.
- To establish a critical mass of lexical entries that would handle a large percentage of non-technical, non-specialized English text.
- To provide an interface to syntactic parsing. For each input word, Englex should return its part-of-speech and all syntactically relevant inflectional categories (such as number and tense).

3.3 General design features

3.3.1 Inflection and derivation

Morphological processes are traditionally divided into two types: inflection and derivation. While this distinction remains controversial among linguists (see for example Anderson 1992:73ff.), we will adopt the widely held view that derivation produces new words (in the sense of lexemes) while inflection produces forms of the same word. Thus the words *compute*, *computer*, *computerize*, *recompute*, *recomputerize*, and so on are related by derivation since they are different words (lexemes), though based on the same root *compute*. Derivational affixes often change the part-of-speech of a word (*compute* is a verb, *computer* is a noun). They also may add semantic content (the prefix *re-* in *recompute* means "again"). In contrast, the words *compute*, *computes*, *computing*, and *computed* are related by inflection since they are all forms of the same word (lexeme) *compute*. Inflection typically encodes categories such as number, tense, gender, and case which are relevant to syntax. This is the conclusion drawn by Anderson: "'Inflection' thus seems to be just the morphology that is accessible to and/or manipulated by rules of the syntax" (Anderson 1992:83).

This distinction between inflection and derivation has several ramifications for building a morphological parsing system such as Englex. First, for Englex to interface with a syntactic parser, it must provide all the inflectional categories of a word as well as its part-of-speech. This is accomplished mainly in the word grammar component which returns the inflectional categories and syntactic category of a word as *features*. For

example, here are the Recognizer results returned for the words *fox*, (a singular noun), *foxes* (a regular plural), *mice* (an irregular plural), and *computer* (a noun derived from a verb):

```
`fox      `fox
[ head:   [ number:SG
           pos:    N]]

`fox+s    `fox+PL
[ head:   [ number:PL
           pos:    N]]

`mice     `mice
[ head:   [ number:SG
           pos:    N]]

com`pute+er      com`pute+NR
[ head:   [ number:SG
           pos:    N]]
```

Each result consists of the lexical form and gloss as they are returned by the lexicon followed by the (partial) feature structure returned by the word grammar. These points should be noted.

- Each word has a *pos* feature standing for part-of-speech. While the part-of-speech of a simple stem such as *fox* is identical to its gloss, the part-of-speech of the word *computer* cannot be directly extracted from its gloss but must be inferred by the word grammar.
- Since each word is a noun, it has an obligatory *number* feature. While the number of a regular plural noun such as *foxes* is overtly marked by a suffix, the number of the other nouns is not. The word grammar is able to provide a default value for the feature *number*: nouns are singular unless marked otherwise. In the case of the irregular plural *mice*, it's lexical entry includes a feature that marks it as plural.

Second, Englex must be able to recognize words formed by derivation. There are two strategies for accomplishing this:

- List all derived words (such as *computer*, *computerize*, *computerization*, and so on) in the lexicon.
- List only roots and affixes (such as *computer*, *+er*, *+ize*, and *+ation*) in the lexicon and decompose derived words into their constituent parts.

The second approach has several obvious advantages:

- The parser will be able to recognize any word regularly derived from a root without having to list it in the lexicon; thus by simply adding a single root entry to the lexicon, all complex words based on it will automatically be recognized.
- The lexicon will be much smaller and use much less storage space than if it listed all derived words.
- It is a better model of morphological structure.

However, the decomposition approach is not without its disadvantages, the main one being that it increases the number of spurious parses. For example, if you posit the suffix *+age* in order to derive *baggage*, *acreage*, *voltage*, and so on from noun roots, then the system will happily tell you that *cabbage* is derived from *cab*. The problem is that *+age* is not a very productive suffix and the number of attested words that use it are far fewer than the number of potential but unattested words that use it. In such cases, it is usually better to compromise and simply list all the attested words with the unproductive suffix in the lexicon. However, if you desire, the word grammar does give you the power to limit the collocational behavior of even a single affix like *+age* by simply requiring that it can only attach to roots that bear a feature that means "takes the suffix *+age*." The decomposition approach to derivational morphology engenders several other problems.

- Many morphologically derived forms cannot be decomposed due to phonological or morphological irregularity. For example, the word *reception* is clearly derived from the verb root *receive* plus the nominalizing suffix *-tion*; other words exemplifying this pattern include *deceive/deception*, *conceive/conception*, *perceive/perception*, and so on. While in principle it is possible to write two-level phonological rules to relate the forms *ceive* and *cep*, in practice it is not feasible for two reasons: the phonological distance between the forms is so great that the rules would be very complex, and the rules apply only to a small number of words. The better solution is simply to list derived forms such as *reception* in the lexicon, even though the morphological relationship between *receive* and *reception* is no longer captured by the lexicon. However, the derivational source of a word such as *reception* can be recorded in the lexicon by giving it a gloss such as *V(re`ceive)+NR*.
- Many regularly derived words in English have acquired specialized meanings. For example, the word *business* is a regular nominal derivation of the verb *busy*, but no longer retains its transparent meaning of "state of being busy." In such cases, it may be desirable to list such words in the lexicon. Thus *business* would return two parses: *`business N* and *`busy+ness AJ+NR*. The problem then becomes deciding which derived words to list and which not to list. A great many nominalized verbs in English have both a transparent meaning and an extended meaning; for instance, *government* can mean "the act of governing" or it can refer to a particular political institution. Englex has taken a conservative course and lists very few derived forms.
- It is not easy to draw a sharp line between productive, synchronic formations and static, diachronic formations. For example, the adjective *resilient* is actually derived from the verb *resile*. Even though the semantic relation is perfectly transparent, the fact that *resile* is no longer in wide currency may put this analysis more in the arena of etymology.

3.3.2 Multiple senses and homonyms

Englex's lexicon is a parsing lexicon, not a full dictionary. In general, multiple senses of words are not distinguished. For example, there is only one entry for the adjective *fair*, ignoring the fact that it has several senses (including "not stormy", "impartial" and "light-colored"). However the noun *fair* meaning "a festival" is considered a homonym and because it is a different part-of-speech it is given its own entry in the noun sublexicon. There are also instances of homonyms of the same part-of-speech; for instance, *bat* in the sense "instrument for hitting" and *bat* in the sense "flying mammal." Note that these two words have different derivational possibilities: the first can be converted to a verb while the second cannot. Nevertheless, *bat* is given only one lexical entry.

The larger issue is how to incorporate semantic information in a PC-KIMMO lexicon. While PC-KIMMO doesn't support a semantics field in lexical entries, there are other ways you can include semantic information in entries.

- You can place semantic information in the features field; for example, features for semantic categories such as animate, human, and so on.
- You can use glosses that encode semantic information.
- You can include a user-defined semantics field in lexical entries, while it would be ignored by PC-KIMMO, it would be accessible to other software.

3.3.3 Lexical conversion

Many words in English belong to more than one part-of-speech; for instance, the word *ride* can be either verb or noun. Since the verb *ride* and the noun *ride* appear to have the same sense, they are derivationally related. The relation between them is often described as zero derivation or conversion. In contrast, the verb *shed* and the noun *shed* have unrelated senses and thus are not derivationally related, by merely homonyms. Clearly homonyms such as verb *shed* and noun *shed* should receive separate lexical entries since they have no lexical relation to each other. But if you posit separate lexical entries for words related by lexical conversion, you would not only lose the linguistic generalization that such words are lexically related but you would also greatly increase the size of the lexicon, since English has a very large number of such words. Englex handles lexical conversion by positing special sublexicons such as N-V for words that occur as both noun and verb. A

word belonging to the N-V sublexicon is expanded into both a noun and a verb by the word grammar. For example, here is the Recognizer results for the input word *hope* (trees are not shown):

```
`hope      `hope
[ head:    [ finite:-
            pos:    V ]]

[ clitic:-
  head:    [ number:SG
            pos:    N ]]
```

The lexicon returns the single lexical form *`hope*, but the word grammar returns two results, one where *hope* is a verb and another where it is a noun. The direction of lexical conversion can be distinctive. Examples of verb to noun conversion include *love*, *laugh*, *answer*, *cover*, and *walk*, while examples of noun to verb conversion include *bottle*, *grease*, *peel*, and *father*. However, it is often difficult or arbitrary to determine the direction of conversion of related words. Englex does not require the analyst to decide the direction of conversion. In the results above for the word *hope*, there is nothing that indicates whether *hope* is basically a verb or a noun. However, the lexical entry for *hope* is found in the file containing verbs, reflecting the analyst's decision that it is basically a verb.

When adding new lexical entries, you should take the possibility of conversion into consideration. For example, say you find that Englex fails to recognize the inflected verb *partied*. Before adding *party* to the file of verb entries, first check to see if *party* already exists in the file of noun entries. If it does, then you need only to change its sublexicon from N to N-V.

For a discussion of lexical conversion in English, see Quirk and others 1972:1009ff.

3.3.4 Compounds

There are three types of orthographic compounds in english (see Quirk and others 1972:1019):

- solid, e.g. *bedroom*
- hyphenated, e.g. *moth-eaten*
- open, e.g. *rose bush*

Open compounds are not handled by Englex at all. If you want to treat open compounds as single lexical items, you must preprocess the text to join them as either hyphenated or solid compounds (for instance, replace *rose bush* with *rose-bush* or *rosebush* and put these forms in the lexicon).

Englex can handle hyphenated compounds. If it recognizes a whole word and then encounters a hyphen, it will recurse and attempt to recognize the part after the hyphen as another word. It will even handle phrasal compounds this way, such as *his come-what-may attitude*. If you do not want to decompose hyphenated compounds, find the End sublexicon near the bottom of the file ENGLISH.LEX and comment out the hyphen entry.

Englex treats solid hyphens as if they were indivisible stems; they are simply listed in the lexicon. It should be possible to cause Englex to decompose solid compounds by using a null lexical entry in the End sublexicon. However, a large number of spurious parses would likely result.

3.4 The rules

Englex's rules component covers most of the spelling alternations that were described in appendix A of Antworth 1990. Rather than repeat that discussion here, you are referred to that work. However, a few changes in the rules file should be noted here.

The environment of the Geminaton rule has been relaxed. Formerly, Geminaton was not permitted in an unstressed syllable; for instance, *traveling* was permitted but not *travelling*. To accomodate common British

usage, either form is now permitted.

The *s*-deletion and *i:y*-spelling rules described in appendix A of Antworth 1990 are not used in Englex. This was done to achieve better processing performance. Because deletions are computationally expensive for the recognizer function, removing the *s*-deletion rule resulted in nearly a 20% speed increase. Removing the *i:y*-spelling rule resulted in a 10% speed increase. The trade-off is that there is some loss in linguistic felicity. The *s*-deletion rule deletes a possessive suffix *s* when it follows an *s*, for example lexical *boy+s+'s* to surface *boys'*. In order to do away with this rule, it is necessary to add the allomorph *+'* to the GENITIVE sublexicon (in the file english.lex). The result is that a word such as *boys'* returns the lexical form *boy+s+'* rather than *boy+s+'s*; however, the gloss string is unaffected and remains N+PL+GEN. If you prefer to use the *s*-deletion rule, it is located in the file ENGLISH.RUL after the END keyword. Simply move it into the main body of rules and comment out the *+'* lexical entry in the file AFFIX.LEX.

The *i:y*-spelling rule accounts for alternations such as *tie* and *tying*. However, there is such a small number of words that exhibit this alternation that it is more economical to list them in the lexicon. If you want to restore the *i:y*-spelling rule, it is located in the file ENGLISH.RUL after the END keyword.

Hyphens are handled differently now. The problem is in handling prefixes such as *re* which can optionally be followed by a hyphen, as in *retry* or *re-try*. Formerly, this prefix was given two lexical entries: *re+* and *re-*. This required that *-:0* be admitted as a feasible pair (an unrestricted deletion of a lexical hyphen). There are two problems with this solution. First, it requires that a great many prefixes have two lexical entries. And second, a deletion correspondence such as *-:0* is costly, since the Recognizer will spend a lot of time trying it everywhere. The present solution is to posit a single lexical entry for prefixes without a hyphen, for instance *re+*, and to replace the *-:0* pair with *+:-*. Since a *+:-* pair is independently necessary for certain hyphenated words, the rules will recognize either *retry* or *re-try* and return the lexical form *re+try*. The only drawback to this solution is that it causes the Generator to produce spurious forms; for instance, the lexical form *`fox+s* will produce both the correct surface form *foxes* and the spurious form *fox-s*. Thus in its present form, Englex is optimized for recognition and is not very useful for generation. However, if you want to use the generation function with Englex's rules, simply open a copy of the file ENGLISH.RUL, find the *+:-* pair in the first table of default pairs, and change it to *-:0*.

3.4.1 Alphabet

The alphabet of word-forming characters is declared in the file ENGLISH.RUL. It consists of these characters:

```
b c d f g h j k l m n p q r s t v w x y z a e i o u ' - ` + .  
B C D F G H J K L M N P Q R S T V W X Y Z A E I O U
```

Only these characters can be used in the lexical form part of a lexical entry. The gloss part of a lexical entry is not restricted to these characters. Accented characters and punctuation in input data must be handled by preprocessing software.

3.4.2 Accented characters (diacritics)

Englex's alphabet does not include accented characters (characters with diacritics). English words usually spelled with diacritics are given lexical forms without them. For instance, the word *naïveté* is usually spelled with a diaeresis over the *i* and an acute accent over the final *e*; but the lexical entry for *naïveté* is spelled *naivete* with no diacritics. This is done for reasons of economy and portability. While PC-KIMMO can handle eight-bit accented characters such as *ï* and *é*, every character added to the alphabet slightly increases processing time. Also, the ASCII code for the character *é* on an IBM-compatible PC is not the same code on the Macintosh; thus you would have to maintain several versions of the rules file for different computing platforms.

If your input data contains accented characters, they must be converted to corresponding unaccented characters before processing by the rules. Otherwise, if you prefer to use accented characters directly, then add them to the alphabet in ENGLISH.RUL and use them in the lexical entries.

3.4.3 Stress marks

Word stress in full words is indicated with the back quote (grave accent) ` . Be careful not to confuse it with apostrophe; for instance, the lexical form of the word *woman's* is written `*woman*+'s. The stress marks were placed according to intuition and the authority of Webster's Ninth New Collegiate Dictionary. Notice that even monosyllabic words require a stress mark because the Gemination rule crucially refers to it (see the file ENGLISH.RUL and appendix A of Antworth 1990).

3.4.4 Capitalization

Englex now handles words with lexical capitalization, e.g. proper nouns. Lexical capitalization is distinct from orthographic capitalization. For words such as *September* and *France*, capitalization is part of their lexical form. Thus *May*, the month, can be distinguished from *may*, the modal. Also, many acronyms have lexical capitalization, such as IBM and NATO. Orthographic capitalization refers to sentence-initial capitalization or use of all-caps for emphasis. Englex does not handle these cases; they must be handled with preprocessing.

3.5 The lexicon

Englex's lexicon contains approximately 20,000 lexical entries. These entries are affixes, roots, indivisible stems and solid compounds. Of these, there are approximately 11,000 nouns, 4,000 verbs, and 3,400 adjectives. Since Englex analyzes productive morphology, it will recognize several times this number of English words. The lexicon is contained in the following files:

english.lex	main lexicon file (loads other files)
affix.lex	affixes
noun.lex	nouns
verb.lex	verbs
adjectiv.lex	adjectives
adverb.lex	adverbs
minor.lex	prepositions, determiners, conjunctions, quantifiers, demonstratives, interjections, ordinals, cardinals

The following files may be optionally loaded

proper.lex	proper nouns
abbrev.lex	acronyms and abbreviations
technica.lex	technical terms
foreign.lex	foreign words and phrases
natural.lex	fauna, flora, etc.

At the beginning of each file is a table of contents. In the noun, verb, and adjective files, irregular forms are listed in the first part of the file followed by regular forms. Morphotactic constraints are found in two places in Englex: the lexicon and the word grammar. The general strategy is to balance the morphotactic description between the two components without making either one overly complex. The lexicon has a strictly "beads on a string" view of morphotactics. Its main job is to decompose a word into a sequence of morphemes using a simple positional analysis (see pp. 106ff. of Antworth 1990). The positional analysis need only go far enough to ensure that all correct parses are produced but not too many incorrect parses (what counts as "too many" is left to the judgment of the analyst and the practical behavior of the system). Cooccurrence restrictions between morpheme positions are best handled in the word grammar, not the lexicon.

The positional analysis used in Englex's lexicon is very simple. First, some words are particles that never bear affixes; these include pronouns, prepositions, and conjunctions. Second, words that may bear affixes have this structure shown in figure 3.1.

Figure 3.1 Positional analysis of English morphology

Prefix* Root Suffix* (Infl) (Clitic)

In this notation, parentheses indicate an optional element, and an asterisk (Kleene star) indicates zero or more occurrences of an element. Thus a word consists of an obligatory root (or indivisible stem) preceded by zero or

more prefixes and followed by zero or more suffixes. This accounts for all derivational structure. Following a derivational stem is an optional slot for an inflectional suffix and an optional slot for a clitic. Since PC-KIMMO's lexicon implements morphotactic constraints as finite state machines, this positional analysis can be expressed as the finite state machine digrammed in figure 3.2. Note that the Prefix and Suffix loops permit any number of prefixes or suffixes.

Figure 3.2 Finite state machine for English morphotactics
[missing]

This obviously is a rather coarse analysis of morphotactic structure, and as such greatly overrecognizes. While it enforces the relative order of prefixes, roots, and suffixes, it does not enforce any order among prefixes or suffixes. For example, it will analyze the non-word **computizer* into the parts *com`pute+ize+er* (reversing the order of the suffixes produces the real word *computerize*). However, this incorrect parse would be filtered out by the word grammar, which knows that the suffix *+ize* can only attach to a noun stem. Of course, in practical use the system should never encounter a form such as **computizer* since input is assumed to be valid English words.

The important point to note in this discussion is that the morphotactic constraints in the lexicon do not enforce any cooccurrence restrictions among the positional morpheme slots; rather, this is done by the word grammar. For example, the lexicon will return two analyses for the word *foxes*: the root *fox* followed either by the plural suffix *+s* or the verbal suffix *+s*. Assuming that *fox* only occurs as a noun, the word grammar will discard the second analysis and keep the first. In this simple case, it seems like it would be more efficient to expand the positional analysis of the lexicon so that it would distinguish noun and verb roots, which could be followed only by the inflectional suffixes appropriate to them. The problem comes with a word such as *enlarges*, which consists of the verbalizing prefix *en+*, the adjective root *large*, and the verb suffix *+s*. If the verb suffix *+s* is constrained to follow only a verb root, then the word *enlarges* would be rejected, since *large* is an adjective. This result is due to the finite state basis of PC-KIMMO's morphotactics: for a given morpheme, you can only state what can follow it. You cannot know that *large* has been preceded by a verbalizing prefix and thus is eligible to take a verbal suffix. The only solution would be to specify two paths through the adjective sublexicon: one which first takes the verbalizing prefix, and one which doesn't. Unfortunately, this would entail physically duplicating the entire adjective sublexicon--not a very practical solution. The better solution is to let the lexicon recognize a verbal suffix wherever it can and let the word grammar filter out the incorrect results.

The morphotactic analysis shown in figures 3.1 and 3.2 is implemented in Englex using ALTERNATION declarations in the main lexicon file (ENGLISH.LEX). Figure 3.3 shows these alternations, while figure 3.4 is a key to the abbreviations for the sublexicon names used in the alternations. The alternation name stands for a positional slot (as in figure 3.1), while the sublexicon names stand for the classes of lexical items that can fill that slot. The Particle alternation lists all the sublexicons of words that do not accept affixes, such as auxiliaries, prepositions, and determiners. The Prefix alternation stands for the Prefix slot in figure 3.1, the first possible slot in a complex word. Recall that a PC-KIMMO lexicon must start with an INITIAL sublexicon. Figure 3.5 shows that the INITIAL sublexicon in Englex contains just two null entries, one for the Particle alternation and one for the Prefix alternation. The purpose of these entries is to provide the initial two entry points into the lexicon system. Note that the gloss field of a null entries must be empty.

Figure 3.3 ALTERNATION declarations in main lexicon file

ALTERNATION Particle	AUX AUX-V PP CJ PP-CJ DT PR DT-PR IJ
ALTERNATION Prefix	PREFIX
ALTERNATION Root	N AJ V AV N-V N-AJ AJ-V AJ-AV CD OD
ALTERNATION Suffix	SUFFIX
ALTERNATION Infl	INFL
ALTERNATION PN_Suffix	PN_SUFF ;proper nouns
ALTERNATION Y_Suffix	Y_SUFF
ALTERNATION IC_Suffix	IC_SUFF
ALTERNATION PT_Suffix	PT_SUFF ;participles
ALTERNATION Clitic	GEN CNTR End
ALTERNATION Contraction	CNTR End
ALTERNATION CD	CD OD ORDR ;cardinals and ordinals
ALTERNATION Compound	INITIAL
ALTERNATION End	End

Figure 3.4 Key to sublexicon names

INITIAL	initial sublexicon
End	final sublexicon
AUX	auxiliary
PP	preposition
DT	determiner
PR	pronoun
CJ	conjunction
IJ	interjection
N	noun
AJ	adjective
V	verb
CD	cardinal
OD	ordinal
AV	adverb
N-V	noun-verb
N-AJ	noun-adjective
AJ-V	adjective-verb
AJ-AV	adjective-adverb
DT-PR	determiner-pronoun
PREFIX	prefix
SUFFIX	suffix
INFL	inflection
GEN	genitive
ORDR	ordinalizer
PN_SUFF	proper noun suffix
Y_SUFF	suffixes on final-y words
IC_SUFF	suffixes on final-ic words
PT_SUFF	participle suffixes
CNTR	contractions

Figure 3.5 The INITIAL sublexicon

```
\lf 0
\lx INITIAL
\alt Particle
\gl

\lf 0
\lx INITIAL
\alt Prefix
\gl
```

The Prefix slot can be filled by zero or more prefixes from the PREFIX sublexicon. Figure 3.6 shows three sample prefix entries. The first is a null entry whose alternation field names the Root alternation; this permits the prefix slot to be optional. The other two entries specify the Prefix alternation; thus if a prefix is recognized, the system keeps looping through the PREFIX sublexicon.

Figure 3.6 Sample prefix entries

```
\lf 0
\lx PREFIX
\alt Root
\gl

\lf non+
\lx PREFIX
\alt Prefix
\gl NEG3+
```

```

\lf pseudo+
\lx PREFIX
\alt Prefix
\gl PEJ3+

```

The Root alternation stands for the Root slot in figure 3.1 which can be filled by a root sublexicon such as noun, verb, or adjective. Figure 3.7 shows several sample lexical entries for roots. The Root slot is obligatory, so there are no null entries in any of the root sublexicons. Three of these entries, *fox*, *carry*, and *happy*, are morphologically regular roots. Their alternation fields name Suffix, indicating that they may take derivational suffixes. The other three entries, *mice*, *began*, and *worse*, are irregular inflected forms. Their alternation fields name Clitic, the final positional slot shown in figure 3.1; this reflects the fact that, as inflected forms, they cannot be further affixed. The features field of each of these irregular forms contains feature abbreviations that convey inflectional information to the word grammar. For example, the features field for *mice* includes the abbreviation *pl*, which the word grammar will expand into the feature structure [number: PL]. The gloss field of each irregular form specifies the root on which the inflected form is based (thus *mice* is an inflected form of *mouse*).

Figure 3.7 Sample root entries

```

\lf `fox
\lx N
\alt Suffix
\gl

\lf `mice
\lx N
\alt Clitic
\fea pl irreg
\gl `mouse

\lf `carry
\lx V
\alt Suffix
\gl

\lf be`gan
\lx V
\alt Clitic
\fea ed irreg
\gl be`gin

\lf `happy
\lx AJ
\alt Suffix
\gl

\lf `worse
\lx AJ-AV
\alt Suffix
\fea comp
\gl `bad

```

The Suffix slot can be filled by zero or more suffixes from the SUFFIX sublexicon. Figure 3.8 shows three sample suffix entries. The first is a null entry whose alternation field names the Infl alternation; this permits the suffix slot to be optional. The other three entries specify the Suffix alternation; thus if a suffix is recognized, the system keeps looping through the SUFFIX sublexicon. The features field of these three entries contains feature abbreviations that will convey morphotactic information about the suffixes to the word grammar. For example, the abbreviation *aj/n* in the features field for *+ness* indicates that it attaches to an adjective and produces a noun, such as *happiness*.

Figure 3.8 Sample suffix entries

```

\lf 0
\lx SUFFIX
\alt Infl
\gl

\lf +ism
\lx SUFFIX
\alt Suffix
\fea n/n
\gl +NR8

\lf +ness
\lx SUFFIX
\alt Suffix
\fea aj/n
\gl +NR27

\lf +ize
\lx SUFFIX
\alt Suffix
\fea n/v
\gl +VR6

```

TheInfl slot is filled by a suffix from the INFL sublexicon. Figure 3.9 shows three sample INFL entries. The first is a null entry whose alternation field names the Clitic alternation; this permits the Infl slot to be optional. The other three entries also specify the Clitic alternation; thus in contrast with the Prefix and Suffix slots, only one inflectional suffix is allowed. The features field of these three entries contains feature abbreviations that will convey morphotactic information about the inflectional suffixes to the word grammar.

Figure 3.9 Sample inflection entries

```

\lf 0
\lx INFL
\alt Clitic
\gl

;noun plural
\lf +s
\lx INFL
\alt Clitic
\fea n/n pl reg
\gl +PL

;adjective comparative
\lf +er
\lx INFL
\alt Clitic
\fea aj/aj comp reg
\gl +CMP

;verb past tense
\lf +ed
\lx INFL
\alt Clitic
\fea v/v ed reg
\gl +ED

```

The Clitic slot is filled by a form from the CLITIC sublexicon. Figure 3.10 shows three sample clitic entries. The first is a null entry whose alternation field names the End alternation; this permits the Clitic slot to be optional. The other three entries also specify the End alternation; thus only one clitic is allowed. The features field of these three entries contains feature abbreviations that will convey morphotactic information about the inflectional suffixes to the word grammar.

Clitics are distinguished from affixes. Affixes are constrained in what word classes they can attach to; for instance, the plural suffix *+s* can only attach to a noun. Clitics, however, are syntactically bound to phrases but phonologically bound to the last word of the phrase; thus they are not constrained by the words they attach to. For instance, the possessive clitic *+'s* normally attaches to nouns as in *the man's hat*, but can attach to other word classes such as adjectives in a phrase such as *the president elect's hat*. Also handled as clitics in Englex are contracted forms such as *+'ll* for *will*, *+'d* for *would*, and *+'ve* for *have*.

Figure 3.10 Sample clitic entries

```
\lf +'s
\lx GEN
\alt End
\fea gen
\gl +GEN

\lf +'d've
\lx CNTR
\alt End
\fea modal -3sg
\gl +would+have
```

The End alternation points to the End sublexicon which contains the two entries shown in figure 3.11 (note that spelling the sublexicon name END would conflict with the keyword END that indicates the end of the main lexicon file). The first entry is a null entry that simply points to the word boundary symbol #, thus ending the word. The second entry, however, handles hyphenated compounds such as *rose-bush*, *well-formed*, *fast-acting*, *moth-eaten*, *water-repellent*, and *machine-readable* (see also section 3.3.4 on compounds). It works like this. Given an input form such as *machine-readable*, the Recognizer will first recognize *machine*; since that is a whole word, it will enter the End sublexicon, where it matches the hyphen and takes the Compound alternation. The Compound alternation (shown in figure 3.3) causes the Recognizer to go back to the INITIAL sublexicon and start all over again. After successfully recognizing *readable*, it again comes to the End sublexicon, where it terminates. If you do not want Englex to attempt to decompose hyphenated compounds, then simply comment out the hyphen entry in the End sublexicon.

Figure 3.11 The End sublexicon

```
\lf -
\lx End
\alt Compound
\fea compound
\gl -

\lf 0
\lx End
\alt #
\gl
```

3.6 The word grammar

Englex's word grammar is contained in the file ENGLISH.GRM (see section 4.7.3 for a reference guide to the word grammar file). The grammar file has three main parts: feature templates and rules.

3.6.1 Rules

The heart of the grammar is of course the rules. Figure 3.12 shows the major rules used in Englex's word grammar.

Figure 3.12 Context-free word grammar rules

```
Word = Word CLITIC
```

```

Word = PARTICLE

Word = Stem (INFL)

Stem = PREFIX Stem

Stem = Stem SUFFIX

Stem = ROOT

```

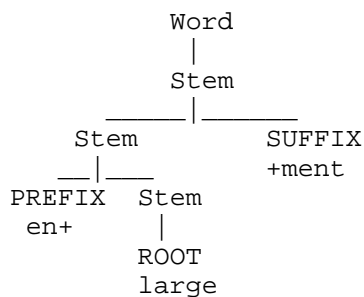
The grammar uses context-free rules consisting of a nonterminal symbol on the left side of the rule which is expanded into one or more symbols on the right side. The symbols on the right can be either terminal or nonterminal symbols. In figure 3.12, Word and Stem are nonterminal symbols, while CLITIC, PARTICLE, PREFIX, SUFFIX, and ROOT (writin in all caps) are terminal symbols. The grammar must have a single "start" symbol; in this case it is Word. The rules in figure 3.12 define a simple and familiar word structure. A Word may be (1) a Word plus a CLITIC; (2) a PARTICLE (an unaffixable word); or (3) a Stem plus an optional INFL (inflection) element. Stem is a nonterminal symbol and thus requires further expansion. A Stem may be (1) a PREFIX plus a Stem; or (3) a Stem plus a SUFFIX; or (3) a ROOT. Notice the congruence between this analysis and the positional analysis used in the lexicon (see figures 3.1 and 3.2 above). Both use the categories CLITIC, PARTICLE, PREFIX, SUFFIX, and ROOT; the difference is that the positional analysis is strictly linear, while the word grammar analysis imposes a branching tree structure on words. For example, this is the positional analysis of the word *enlargement*:

Figure 3.13 Positional analysis of *enlargement*

PREFIX	ROOT	SUFFIX
en+	large	+ment

and this is the structure produced by the word grammar:

Figure 3.14 Parse tree for *enlargement*



The tree structure shows that the word *enlargement* is composed of the stem *enlarge* plus the suffix *+ment*, and the stem *enlarge* in turn is composed of the prefix *en+* plus the root *large*. The context-free grammar in figure 3.12, while an improvement over the simple positional analysis, still has several deficiencies. First, it does not tell us the part-of-speech of a word; note that the tree in figure 3.14 does not reveal that the word *enlargement* is a noun. Second, it still does not adequately characterize important morphotactic constraints. For example, it would also return an analysis of the word *enlargement* with it bracketed as [en+ [large +ment]]. This of course is an incorrect analysis since it posits the nonexistent stem *largement*. The analysis still does not capture the fact that a suffix such as *+ment* can attach to verb stems only.

One way to remedy this problem is to introduce more category symbols into the rules. For example, these rules would recognize the words *large*, *enlarge*, and *enlargement* while rejecting the nonword *largement*:

```

Word = Noun

Word = Adjective

```

Word = Verb

Verb = VerbPrefix Adjective

Noun = Verb NounSuffix

However, this approach will quickly result in a grammar with a great many confusing rules which still may not adequately handle the data. PC-KIMMO remedies this problem by augmenting the context-free rules with *feature structures*. In this approach, the categories used in rules are not just simple labels, but represent complex feature structures. Associated with each rule are feature constraints which are satisfied by using an operation called *unification*. Thus PC-KIMMO's word grammar component is a type of grammar formalism called a *unification grammar*. Its implementation closely follows the PATR-II grammar described in Shieber 1986. Before demonstrating how to use feature constraints, we must first review the basics of feature structures and unification.

3.6.2 Features and unification

A feature structure consists of a feature name and a value. The notation used for feature structures looks like this:

```
[number: singular]
```

where *number* is the feature name and *singular* is the value, separated by a colon. A structure containing more than one feature uses square brackets around the entire structure:

```
[number: singular  
 case:      nominative]
```

Feature structures can have either simple values, such as the example above, or complex values, such as this:

```
[agreement: [number: singular]]
```

where the value of the *agreement* feature is another feature structure. Feature structures can be infinitely nested in this manner. Portions of a feature structure can be referred to using the "path" notation. A path is a sequence of feature names (minimally one) enclosed in angled brackets (<>). For example, consider this feature structure:

```
[agreement: [number: singular  
              case: nominative]]
```

These are feature paths based on this structure:

```
<number>  
<case>  
<agreement number>  
<agreement case>
```

Paths are used in feature templates and feature constraints, described below. Feature structures are manipulated using an operation called *unification*. Two feature structures can unify if none of their constituent features have conflicting values; the resulting structure is a union of all their features. For example, feature structures (a) and (b) unify as (c):

```
(a) [a: P  
     b: Q]
```

```
(b) [b: Q]
```

```

c: R]

(c) [a: P
     b: Q
     c: R]

```

But feature structures (d) and (e) fail to unify because they have conflicting values for the *q* feature:

```

(d) [a: P
     b: Q]

(e) [b: S
     c: R]

```

3.6.3 Feature constraints

Figure 3.15 shows the rules from figure 3.12 with some feature constraints attached to each rule. A feature constraint consists of two feature paths separated by an equals sign. The feature paths refer to features of rule categories; thus *<PREFIX from_pos>* refers to the *from_pos* feature of the *PREFIX* category and *<Stem_1 pos>* refers to the *pos* feature of the *Stem_1* category. (Note that subscripts are used to distinguish multiple instances of a symbol in a rule (for instance, *Word_1* and *Word_2* in the first rule). In order for a feature constraint to succeed, the values referred to by the feature paths must unify with each other. Note that, in spite of the equals sign, unification is not the same thing as saying that the values must be the same. For example, a feature constraint that referred to the feature structures in (a) and (b) in the section above would succeed, since they unify—even though they are not identical.

Figure 3.15 Word grammar rules with feature constraints

```

Word_1 = Word_2 CLITIC
  <Word_1 pos> = <Word_2 pos>

Word = PARTICLE
  <Word pos> = <PARTICLE pos>

Word = Stem
  <Word pos> = <Stem pos>

Word = Stem INFL
  <Stem pos> = <INFL from_pos>
  <Word pos> = <INFL pos>

Stem_1 = PREFIX Stem_2
  <PREFIX from_pos> = <Stem_2 pos>
  <Stem_1 pos> = <PREFIX pos>

Stem_1 = Stem_2 SUFFIX
  <Stem_2 pos> = <SUFFIX from_pos>
  <Stem_1 pos> = <SUFFIX pos>

Stem = ROOT
  <Stem pos> = <ROOT pos>

```

The rules and constraints in figure 3.15 demonstrate the basic strategy used in Englex to accomplish two important tasks:

1. To constrain the input and output categories of affixes; for example, to ensure that the suffix +ment attaches to a verb and produces a noun.
2. To determine the part-of-speech of a word; for example, that a complex word such as enlargement is a noun.

The grammar uses two features for these purposes: *pos* and *from_pos*. Stems and roots have a *pos* feature standing for part-of-speech, such as N (noun), V (verb), AJ (adjective) and so on. Affixes have a *from_pos* feature and a *pos* feature. The *from_pos* feature is the part-of-speech of the stem to which the affix can be attached; the *pos* feature is the part-of-speech of the resulting derived stem. For example, here are the feature structures for the the root *large*, the prefix *en+*, and the suffix *+ment*:

```
[cat: ROOT
 pos: AJ
 lex: `large
 gloss: `large]

[cat: PREFIX
 from_pos: AJ
 pos: V
 lex: en+
 gloss: VR1]

[cat: SUFFIX
 from_pos: V
 pos: N
 lex: +ment
 gloss: NR25]
```

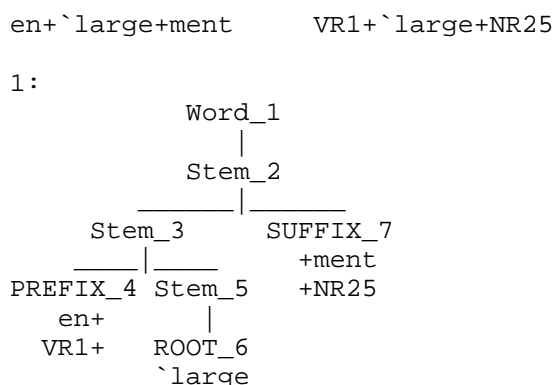
In figure 3.15 the feature constraint $\langle \text{PREFIX } from_pos \rangle = \langle \text{Stem_1 } pos \rangle$ under the fifth rule says that the *from_pos* of the PREFIX must unify with the *pos* of the stem to which it is attached. Thus the prefix *en+*, whose *from_pos* is AJ, can attach to the root *large*, whose *pos* is also AJ, but not with other categories of roots such as nouns or verbs. In this way, the grammar constrains prefixes such as *en+* to occur only on verbs.

The second feature constraint under the fifth rule in figure 3.15 $\langle \text{Stem } pos \rangle = \langle \text{PREFIX } pos \rangle$, says that the *pos* of the Stem must unify with the *pos* of the PREFIX. Now the *pos* of the prefix *en+* is V, but prior to the application of this rule, the Stem category has no *pos* feature. Thus the effect of the constraint is to add [pos: V] to the feature structure for the Stem category; in other words, the part-of-speech of the derived stem *enlarge* is V (verb).

These two constraints demonstrate the two main uses of feature constraints. The first constraint refers to categories which are both on the right side of the rule (PREFIX and Stem_1); its effect is to either permit or block the application of the rule. The second constraint, however, refers to the category on the left side of the rule (Stem) and a category on the right side (PREFIX); its effect is to pass a feature value from a lower node in the tree (PREFIX) to a higher node (Stem). The distinction between these two types of constraints is subtle but extremely important.

Figure 3.16 shows how the grammar from figure 3.15 analyzes the word *enlargement* into a parse tree and the feature structures for each node of the tree.

Figure 3.16 Parse tree and full feature display for *enlargement*



```

        `large

Word_1:
[ cat:   Word
  number:SG
  pos:   N ]

Stem_2:
[ cat:   Stem
  number:SG
  pos:   N ]

Stem_3:
[ cat:   Stem
  pos:   V ]

PREFIX_4:
[ cat:   PREFIX
  from_pos:AJ
  gloss: VR1+
  pos:   V
  lex:   en+ ]

Stem_5:
[ cat:   Stem
  pos:   AJ ]

ROOT_6:
[ cat:   ROOT
  gloss: `large
  pos:   AJ
  lex:   `large]

SUFFIX_7:
[ cat:   SUFFIX
  from_pos:V
  gloss: +NR25
  number:SG
  pos:   N
  lex:   +ment]

```

3.6.4 Interface between the lexicon and the word grammar

Now that we have developed a minimal set of grammar rules with feature constraints (figure3.15), we can look at the interface between the lexicon and the word grammar. For example, here is the lexical entry for *mice*:

```

\lf `mice
\lx N
\alt Clitic
\fea pl irreg
\gl `mouse

```

The word grammar represents this lexical item as a feature structure:

```

[cat: N
 lex: `mice
 gloss: `mouse
 number: PL
 reg: -]

```

In this structure, the features *cat*, *lex*, and *gloss* are reserved features automatically constructed by the word

grammar:

- The value of the *cat* feature is taken from the sublexicon field (*\lx*).
- The value of the *lex* feature is taken from the lexical form field (*\lf*).
- The value of the *gloss* feature is taken from the gloss field (*\gl*).

The features *number* and *reg*, however, are user-defined features taken from the abbreviations *pl* and *irreg* in the features field (*\fea*). They indicate that *mice* has plural number and is marked as an irregular form. The abbreviations are defined by *feature templates* in the word grammar file (see also section 4.7.3):

```
Let pl be <number> = PL
Let irreg be <reg> = -
```

This kind of feature template is called a feature definition. As another example, here is the lexical entry for the regular plural suffix +s:

```
\lf +s
\lx INFL
\alt Clitic
\fea n/n pl reg
\gl +PL
```

The abbreviations in the features field are defined by these templates:

```
Let n/n be <from_pos> = N
           <pos> = N
Let pl be <number> = PL
Let reg be <reg> = +
```

The abbreviation *n/n* expands into two features: *[from_pos: N]* and *[pos: N]*. This is the same mechanism as described above for controlling the input and output categories of affixes. Note the scheme used by the abbreviations: prefix abbreviations have the form *pos\from_pos* while suffix abbreviations have the form *from_pos/pos*. Every affix entry in Englex must have a feature abbreviation of this type in order to satisfy the feature constraints shown in figure 3.15. The plural suffix +s, then, is represented in the word grammar as this feature structure:

```
[cat: INFL
 lex: +s
 gloss: +PL
 from_pos: N
 pos: N
 number: PL
 reg: +]
```

The lexical entry above for *mice* explicitly contains the information that it is plural. But how do we indicate that nouns such as *mouse* or *fox* are singular? We could of course include the feature abbreviation *sg* in their entries which is defined by this template:

```
Let sg be <number> = SG
```

However, this would require us to include the abbreviation *sg* in the entries for all singular nouns in the lexicon. To capture the simple generalization that noun stems are singular by default, we instead place this template in the word grammar file:

```
Let N be <number> = SG
```

In this kind of template, the symbol being defined must be a sublexicon name. Thus this template will add the feature structure *[number: SG]* to all lexical items belonging to the N sublexicon. Similarly, to capture the fact that by default nouns form regular plurals (for instance, *fox*), we add another feature specification to the N template:

```
Let N be <number> = SG
      <reg> = +
```

Thus the lexical entry for *fox*:

```
\lf `fox
\lx N
\alt Suffix
\gl
```

is given this feature structure:

```
[cat: N
 lex: `fox
 gloss: `fox
 number: SG
 reg: +]
```

However, this analysis has a problem. The N template will be applied to all lexical items in the N sublexicon, including items such as *mice*. However, the entry for *mice* explicitly specifies values for the features *number* and *reg* that conflict with the values specified by the N template. What we really want the N template to say is this: noun stems are singular and regular by default unless they are explicitly marked as plural or irregular. This is accomplished by placing an exclamation sign before the feature values in the template:

```
Let N be <number> = !SG
      <reg> = !+
```

Now the template will assign the values SG and + only if the lexical item does not already have values for the number and reg features; thus the word *mice* will retain its own values for these features. Some nouns have identical singular and plural forms, such as *sheep* and *deer*. Englex uses the feature abbreviation *sg-pl* to mark such nouns. Here is the lexical entry for *deer*:

```
\lf `deer
\lx N
\alt Suffix
\fea sg-pl irreg
\gl
```

The abbreviation *sg-pl* is defined in the word grammar by this template:

```
Let sg-pl be <number> = {SG PL}
```

This template has a disjunctive definition--the *number* feature can have either the value SG or the value PL. The effect of this template is that when the grammar uses the word *deer*, it creates two instances of it, one with the feature specification *[number: SG]* and another with the feature specification *[number: PL]*. Template have another important use: they are used to map sublexicon names to categories. In the default case, sublexicon names constitute the terminal categories used in the word grammar rules. As was shown above, the value of the *cat* feature of a lexical item is its sublexicon name; thus *mice* belongs to the category N. The category N can then be used as a terminal category in the grammar rules. For instance, you could write a

grammar with rules such as these:

```
Word = Stem
Stem = N
Stem = V
Stem = AJ
```

which say that a **Word** is composed of a **Stem** which in turn is composed of one of the terminal categories **N**, **V**, or **AJ**. Notice, however, that the grammar in figure 3.15 above collapses these three rules into one:

```
Stem = ROOT
```

N, **V**, and **AJ** are declared as instances of the **ROOT** category with these templates:

```
Let N be <cat> = ROOT
Let V be <cat> = ROOT
Let AJ be <cat> = ROOT
```

Now the *cat* feature of a noun such as *fox* will be **ROOT**, not **N**. To encode the part-of-speech of **N** entries, Englex uses the *pos* feature. Thus the full templates for **N** and **V** look like this:

```
Let N be <cat> = ROOT
    <pos> = N
    <number> = !SG
    <reg> = !+
Let V be <cat> = ROOT
    <pos> = V
    <reg> = !+
    <finite> = !-
    <vform> = !BASE
```

Category templates provide a good solution to the problem of handling roots that belong to more than one part-of-speech. First, such roots are placed in special sublexicons in the lexicon. For example, the **N-V** sublexicon contains roots that can be either a noun or a verb, such as *hope* and *spy*. Then, the word grammar this template for **N-V**:

```
Let N-V be {[N] [V]}
```

This template says that **N-V** is expanded into instances of both the **N** and **V** categories. Thus when a root from the **N-V** sublexicon such as *spy* is used by the word grammar, two instances of it are created--one with the features specified by the **N** template above and another with the features specified by the **V** template.

Figure 3.17 List of features and possible values

cat	ROOT, PARTICLE, PREFIX, SUFFIX, CLITIC, COMPOUND
lex	
gloss	
from_pos	N, V, AJ, AV
base_pos	N, V, AJ
root_pos	N, V, AJ, AV, AUX
drvstem	+, -
change_pos	+, -
clitic	+, -
participle	+, -
stress_shift	+, -
boundstem	+, -
root	

```

prefix_cooccur: rev1    +, -
                  deg2    +, -
suffix_cooccur: ajr8     +, -
                  ajr13    +, -
                  ...

```

head:

```

pos          N, V, AJ, AV, AUX, PP, PR, CJ, DT, IJ, INF
person       1, 2, 3
number       SG, PL
agr: 3sg     +, -
proper       +, -
tense        PRES, PAST
vform        S, ED, EN, ING
finite       +, -
aform        ABS, COMP, SUPER
verbal       +, -
case         NOM, ACC, GEN, IND
reflex       +, -
wh           +, -
reg          +, -
modal        +, -
neg          +, -
clitic: cform

```