

A paper presented at:

North Texas Natural Language Processing Workshop

May 23, 1994

University of Texas at Arlington

Morphological Parsing with a Unification-based Word Grammar

Evan L. Antworth

Academic Computing Department

Summer Institute of Linguistics

7500 W. Camp Wisdom Road

Dallas, TX 75236

Evan.Antworth@sil.org

Abstract

This paper describes PC-KIMMO, a morphological parser based on Kimmo Koskenniemi's model of two-level morphology ([Koskenniemi 1983](#)). While PC-KIMMO was adequate to decompose a word into morphemes, it was not able directly to compute the part of speech of a derivationally complex word or return a word's inflectional features--precisely the information required for syntactic parsing. These deficiencies have now been remedied by adding a unification-based word grammar component to version 2 of PC-KIMMO which can provide parse trees and feature structures. A substantial analysis of English for use with PC-KIMMO is also described.

Why do morphological parsing?

Parsing is a standard technique used in the field of natural language processing. When you think of parsing, you likely think of syntactic parsing. But before a syntactic parser can parse a sentence, it must be supplied with information about each word in the sentence. For instance, to parse the sentence *The cat chased the rat*, a parser must know that *cat* is a singular noun, *chased* is a past tense verb, and so on. In English, such information may be supplied by a lexicon that simply lists all wordforms with their part of speech and inflectional information such as number and tense. Because English has a relatively simple inflectional system, the number of forms that must be listed in such a lexicon is manageable. Note that countable nouns such as *cat* have only have two inflected forms, singular and plural, and regular verbs such as *chase* have only four inflected forms: the base form, the *-s* form, the *-ed* form, and the *-ing* form. But an exhaustive lexical listing is simply not feasible for many other languages, such as Finnish, Turkish, and Quechua, which may have hundreds of inflected forms for each noun or

verb. For such languages, one must build a word parser that will use the morphological system of the language to compute the part of speech and inflectional categories of any word.

Even for English a morphological parser may be necessary. Although English has a limited inflectional system, it has very complex and productive derivational morphology. For example, from the root *compute* come derived forms such as *computer*, *computerize*, *computerization*, *recomputerize*, *noncomputerized*, and so on. It is impossible to list exhaustively in a lexicon all the derived forms (including coined terms or inventive uses of language) that might occur in natural text.

The two-level model of morphology

A major breakthrough in the field of morphological parsing came in 1983 when Kimmo Koskenniemi, a Finnish computer scientist, produced his dissertation *Two-level morphology: A general computational model for word-form recognition and generation* ([Koskenniemi 1983](#)). Koskenniemi's model of two-level morphology was based on the traditional distinction that linguists make between *morphotactics*, which enumerates the inventory of morphemes and specifies in what order they can occur, and *morphophonemics*, which accounts for alternate forms or "spellings" of morphemes according to the phonological context in which they occur. For example, the word *chased* is analyzed morphotactically as the stem *chase* followed by the suffix *-ed*. However, the addition of the suffix *-ed* apparently causes the loss of the final *e* of *chase*; thus *chase* and *chas* are allomorphs or alternate forms of the same morpheme. Koskenniemi's model is "two-level" in the sense that a word is represented as a direct, letter-for-letter correspondence between its lexical or underlying form and its surface form. For example, the word *chased* is given this two-level representation (where + is a morpheme boundary symbol and 0 is a null character):

Lexical form:	c	h	a	s	e	+	e	d
Surface form:	c	h	a	s	0	0	e	d

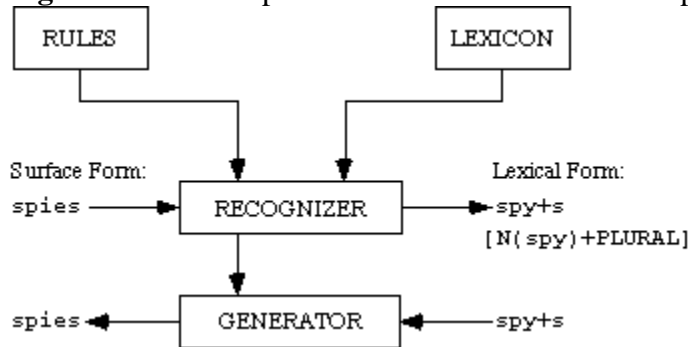
For more on the phonological properties of the two-level model, see [Antworth 1991](#).

The KIMMO parser

Shortly after Koskenniemi's dissertation appeared, Lauri Karttunen and others produced a LISP implementation of Koskenniemi's two-level model and dubbed it KIMMO ([Karttunen 1983](#)). The main components of the KIMMO parser are shown in [Figure 1](#). It had two analytical components: the rules component and the lexical component, or lexicon. First, the rules component consisted of two-level rules that accounted for regular phonological or orthographic alternations, such as *chase* versus *chas*. Second, the lexicon listed all morphemes (stems and affixes) in their lexical form and specified morphotactic constraints. For example, the lexicon would have included lexical entries for the verb stem *chase* and the suffix *-ed*, and would have specified their relative order. Using these data components were two processing functions, the Generator and the Recognizer. The

Generator would accept as input a lexical form such as *spy+s* and return the surface form *spies*. The Recognizer would accept as input a surface form such as *spies* and return an underlying form divided into morphemes, namely *spy+s*, plus a gloss string such as N+PLURAL.

Figure 1 Main components of Karttunen's KIMMO parser



The PC-KIMMO parser

In 1990, the Summer Institute of Linguistics produced PC-KIMMO version 1, an implementation of the two-level model that closely followed Karttunen's KIMMO (see [Antworth 1990](#)). Written in C, it ran on personal computers such as IBM PC compatibles and the Macintosh as well as UNIX. PC-KIMMO was quite good at what it was designed to do--tokenize a word into a sequence of tagged morphemes. But it had a serious deficiency: it could not directly determine the part of speech of a word or its inflectional categories. For example, given the word *enlargements*, PC-KIMMO could tokenize it into the sequence of morphemes *en+large+ment+s* and gloss each morpheme, but it could not determine that the entire word was a plural noun. This meant that PC-KIMMO was not adequate to act as a morphological front end to a syntactic parser--its most desirable application.

Unification-based word grammar

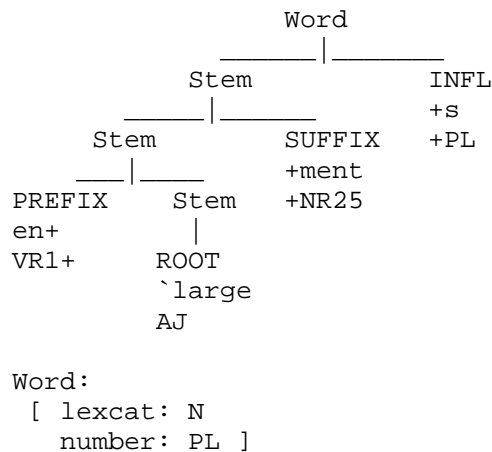
In 1993, version 2 of PC-KIMMO was developed specifically to correct this deficiency. It does so by adding a third analytical component, a word grammar. The word grammar is a unification-based chart parser (based on the PATR-II formalism described in [Shieber 1986](#)) that provides parse trees and feature structures. The chart parser was originally designed for syntactic parsing. Just as a sentence parser produces a parse tree with words as its leaf nodes, a word parser produces a parse tree with morphemes as its leaf nodes. When you parse a sentence, it is normally already tokenized into words (since we put white space between words); but when you parse a word, you must first tokenize it into morphemes. This tokenizing is done by the rules and lexicon. When a surface word is submitted to PC-KIMMO's Recognizer, the rules and lexicon analyze the word into a sequence of morpheme structures (or possibly more than one sequence if more than one analysis is found). A morpheme structure consists of a lexical form, its gloss, its category,

and its features. For example, the word *enlargements* is tokenized into this sequence of morpheme structures:

Form:	en+	large	+ment	+s
Gloss:	VR1+	AJ	+NR25	+PL
Cat:	PREFIX	ROOT	SUFFIX	INFL
Feat:	[fromcat: AJ tocat: V finite: !-]	[lexcat: AJ aform: !POS]	[fromcat: V tocat: N number: !SG]	[fromcat: N tocat: N number: SG reg: +]

This analysis is then passed to the word grammar, which returns the parse tree and feature structure shown in [Figure 2](#).

Figure 2 Parse tree and feature structure for enlargements



While each node of the tree has a feature structure associated with it, the feature structure for the top node is the most important, since these are the features attributable to the entire word. The feature structure for the word *enlargements* specifies two features. First, the feature *lexcat* has the value *N*, meaning that the lexical category (part-of-speech) of the word is Noun. Second, the feature *number* has the value *PL* for plural. If PC-KIMMO were being called from a syntactic parser, then it would return to the syntactic parser the word *enlargements* with these features. In its overall architecture, version 2 of PC-KIMMO now resembles the morphological parser described in [Ritchie and others 1992](#). That parser also first tokenizes a word into morphemes and then parses the morpheme sequence with a unification-based parser. However, our unification parser differs considerably from theirs in its implementation.

The word grammar component uses a grammar file written by the user. A grammar consists of context-free rules and feature constraints. The format of the grammar closely

Figure 3 Fragment of a word grammar of English

;FEATURE ABBREVIATIONS:

Let pl be <number> = PL
Let irreg be <reg> = -
Let v/n be <fromcat> = V
<toocat> = N
<number> = !SG
Let v\aj be <fromcat> = AJ
<toocat> = V
<finite> = !-

;CATEGORY TEMPLATES:

Let N be <cat> = ROOT
<lexcat> = N
<number> = !SG
<reg> = !+
<proper> = !-
Let V be <cat> = ROOT
<lexcat> = V
<finite> = !-
<reg> = !+
Let AJ be <cat> = ROOT
<lexcat> = AJ
<aform> = !POS
<reg> = !+

;Rule 1

Word = Stem INFL
<Stem lexcat> = <INFL fromcat>
<Word lexcat> = <INFL toocat>
<Word number> = <INFL number>
<Word tense> = <INFL tense>
<Word aform> = <INFL aform>

;Rule 2

Stem = PREFIX Stem_1
<PREFIX fromcat> = <Stem_1 lexcat>
<Stem lexcat> = <PREFIX toocat>
<Stem number> = <PREFIX number>
<Stem tense> = <PREFIX tense>
<Stem aform> = <PREFIX aform>
<Stem reg> = <Stem_1 reg>

;Rule 3

Stem = Stem_1 SUFFIX
<Stem_1 lexcat> = <SUFFIX fromcat>
<Stem lexcat> = <SUFFIX toocat>
<Stem number> = <SUFFIX number>
<Stem tense> = <SUFFIX tense>
<Stem aform> = <SUFFIX aform>
<Stem reg> = <SUFFIX reg>

;Rule 4

Stem = ROOT
<Stem lexcat> = <ROOT lexcat>
<Stem number> = <ROOT number>

```
<Stem tense> = <ROOT tense>
<Stem aform> = <ROOT aform>
<Stem reg> = <ROOT reg>
```

follows Shieber's PATR-II formalism ([Shieber 1986](#)). [Figure 3](#) shows a fragment of a word grammar of English.

The first section of the grammar file contains feature abbreviations. Feature abbreviations can be used either in lexical entries or in grammar rules and are expanded by "Let" statements. For example, the feature abbreviation *pl* is expanded into the feature structure [number: PL].

The second section of the grammar file contains category templates. These are feature specifications that are attached to lexical categories such as Noun and Adjective. This greatly reduces the amount of information that must be stored in the lexicon. For example, the statement Let N be <number> = SG means that all nouns are assigned singular number. The grammar in [Figure 3](#) actually contains the statement Let N be <number> = !SG. The exclamation point in !SG means that this is a default value which can be overridden. For example, the lexical entry for *fox* does not need to specify that it is singular; that information is supplied by the category definition of Noun. However, the lexical entry for *mice* (an irregular plural) explicitly sets the feature number feature to PL (plural), thus overriding the default value.

The third section of the grammar file contains the word grammar rules. Associated with each rule are feature constraints. A feature constraint consists of two feature structures which must unify with each other. Feature constraints have two functions: they constrain the operation of a rule and they pass features from one node to another up the parse tree. For example, in rule 1 of [Figure 3](#) the feature constraint <Stem lexcat> = <INFL fromcat> requires that the *lexcat* feature of the Stem node must have the same value as the *fromcat* feature of the INFL node in order for the rule to succeed, while the feature constraint <Word lexcat> = <INFL tocat> passes the value of the *tocat* feature from the INFL node to the *lexcat* feature of the Word node.

The word *enlargements* is an especially good example of the power of the word grammar because of its complex derivational structure. Its root is the adjective *large*; the prefix *en-* forms the verb *enlarge*; the suffix *-ment* forms the noun *enlargement*; and finally the inflectional suffix *s* marks it plural. To accomplish this, each root or stem has a lexical category such as Noun or Adjective and each affix has a *fromcat* feature and a *tocat* feature. The *fromcat* feature specifies the lexical category of the stems to which it can attach. The *tocat* feature specifies the lexical category of the resulting stem. For instance, the prefix *en-* has a *fromcat* of Adjective and a *tocat* of Verb, since it attaches to an adjective such as *large* and produces a verb, *enlarge*. Rule 2 in [Figure 3](#) says that a Stem is composed of a PREFIX plus a Stem, for instance *enlarge* = *en*+*large*. The first feature constraint, <PREFIX fromcat> = <Stem_1 lexcat>, requires that the *fromcat* feature of the PREFIX node must have the same value as the *lexcat* feature of the Stem_1 node. Since the *fromcat* feature of the prefix *en-* and the *lexcat* feature of the stem *large* are both AJ, the rule succeeds. The second feature constraint, <Stem lexcat> = <PREFIX

`tocat`>, passes the value of the *tocat* feature from the PREFIX node to the *lexcat* feature of the Stem node; that is, since the *tocat* of the prefix *en-* is *v*, the *lexcat* of the resulting stem *enlarge* is also *v*.

Rule 1 in [Figure 3](#) accounts for the plural suffix *-s*. The rule simply says that a word is composed of a stem plus an inflectional element. The first two feature constraints use the *lexcat* feature to ensure that the plural suffix attaches only to a noun stem and produces a noun word (that is, it does not change the part of speech of the stem as do the derivational affixes). The third feature constraint, `<Word number> = <INFL number>`, passes the value of the *number* feature from the inflectional suffix to the Word node.

Englex: a two-level description of English morphology

PC-KIMMO version 2 can be used with Englex, a two-level description of English morphology. Englex consists of a set of orthographic rules, a 20,000-entry lexicon of roots and affixes, and a word grammar. With Englex and PC-KIMMO, you can morphologically parse English words and text. Practical applications include morphologically preprocessing text for a syntactic parser and producing morphologically tagged text (see [Antworth 1993](#)). Englex can also be used to explore English morphological structure for purposes of linguistic analysis.

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 lexical category (part-of-speech) and all syntactically relevant inflectional categories (such as number and tense).

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 lexical category it is given its own entry in the noun sublexicon. 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.

Lexical category conversion

Many words in English belong to more than one lexical category; for instance, the word *ride* can be either verb or noun. Since the verb *ride* and the noun *ride* appear to share the same basic 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, but 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 category 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 noun and verb instances by the following "Let" statement in the word grammar:

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

The curly braces indicate a disjunction of the elements [N] and [V], which in turn are expanded by category templates for N and V (that is, Noun and Verb).

Conclusion

Koskenniemi's original model of a two-level morphological parser, instantiated as PC-KIMMO version 1, made it possible to decompose complex words into their constituent morphemes, but it lacked the ability directly to compute the part of speech of a complex word or to return inflectional features in a form that could easily be used by a syntactic parser. By feeding the output of the original parser into a unification-based word grammar, version 2 of PC-KIMMO can now provide full parse trees and feature structures for handling part of speech and inflectional categories. A substantial description of English including rules, lexicon, and word grammar is also available for use with PC-KIMMO version 2.

References

Antworth, Evan L. 1990. *PC-KIMMO: a two-level processor for morphological analysis*. Occasional Publications in Academic Computing No. 16. Dallas, TX: Summer Institute of Linguistics.

_____. 1991. Introduction to two-level phonology. *Notes on Linguistics* 53:4-18. Dallas, TX: Summer Institute of Linguistics.

_____. 1993. Glossing text with the PC-KIMMO morphological parser. *Computers and the Humanities* 26:475-484.

Karttunen, Lauri. 1983. KIMMO: a general morphological processor. *Texas Linguistic Forum* 22:163-186.

Koskenniemi, Kimmo. 1983. *Two-level morphology: a general computational model for word-form recognition and production*. Publication No. 11. Helsinki: University of Helsinki Department of General Linguistics.

Ritchie, Graeme D., Graham J. Russell, Alan W. Black, and Stephen G. Pulman. 1992. *Computational morphology: practical mechanisms for the English lexicon*. Cambridge, MA: The MIT Press.

Shieber, Stuart M. 1986. *An introduction to unification-based approaches to grammar*. CSLI Lecture Notes No. 4. Stanford, CA: Center for the Study of Language and Information.