

# Chapter 2

## Overview of PC-KIMMO Version 2

*Last modified November 22, 1995*

---

### 2.1 What's new in version 2

### 2.2 A sample session using the word grammar component

Using the *Synthesizer* function

### 2.3 Changes to the rules component

### 2.4 Changes to the lexical component

### 2.5 Changes to the user interface

**Figure 2.1** A morphological parse tree

**Figure 2.2** Feature structure

**Figure 2.3** A word grammar rule

---

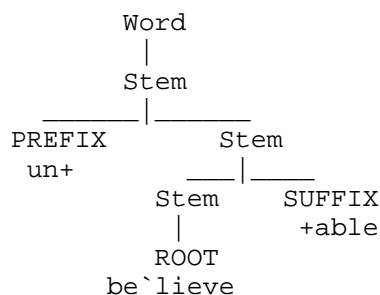
## 2.1 What's new in version 2

This chapter gives an overview of the new features in PC-KIMMO version 2. These include the following:

- The rules component supports multigraphs.
- The rules component is compatible with Xerox's *twolc* rule compiler.
- Lexical entries have a new encoding format.
- Lexical entries permit a *features* field.
- A word grammar component has been added.
- A *Synthesizer* function has been added.
- The internal data structures returned by the Recognizer have been enriched.

Of these new features, the word grammar component is the most significant. The word grammar component uses a unification-based parser based on the PATR-II formalism described in Shieber 1986. Although parsers of this type have typically been used for syntactic analysis, they can also be used for morphological analysis with equal success. Just as a sentence parser produces a tree structure with words as its leaf nodes, a word parser produces a tree structure with morphemes as its leaf nodes. For example, figure 2.1 shows a parse tree of the word *unbelievable* as produced by PC-KIMMO's word grammar component:

**Figure 2.1** A morphological parse tree



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 top node of the tree in figure 2.1 is shown in figure 2.2. It gives three features for the word *unbelievable*. First, the feature *cat* has the value *Word*, which is simply the name the node. Second the feature *pos* has the value *AJ*, meaning that the lexical category (part-of-speech) of the word is Adjective. And third, the feature *aform* has the value *POS*, meaning that it is the Positive form of the adjective (as opposed to the Comparative *-er* or Superlative *-est* forms). If PC-KIMMO were being called from a syntactic parser, then it would return to the syntactic parser the word *unbelievable* with its features.

**Figure 2.2** Feature structure

```
[ cat:   Word
  head: [pos:   AJ
        aform: POS ]]
```

The word grammar component uses a file containing a grammar written by the user. A grammar consists of context-free rules and feature constraints. An example of a rule with constraints is shown in figure 2.3.

**Figure 2.3** A word grammar rule

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

One obvious difference between parsing a sentence and parsing a word is that a sentence is typically already tokenized into words while a word is not tokenized into morphemes. In other words, we put white space between words but not between morphemes. In PC-KIMMO, the Recognizer uses the rules and lexicon to tokenize a word into a sequence of morphemes which in turn is passed to the word grammar component for parsing. 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.

There are several reasons why we added a word grammar component to PC-KIMMO.

*The word grammar component offers a more powerful model of morphotactics.*

PC-KIMMO version 1 used only the continuation class model of morphotactics which was used in Koskenniemi's original model (1983). In the continuation class model, the morphotactic properties of a morpheme can be stated only in terms of the classes of morphemes that can directly follow it in a word. This meant that it was very difficult or at least practically unfeasible to enforce certain discontinuous dependencies between morphemes. The word grammar, however, has the entire power of a context-free grammar at its disposal and can model word structure as arbitrarily complex branching trees (both left- and right-branching). The practical result is that with PC-KIMMO version 2 you can eliminate most of the bad parses that were so difficult to prevent with version 1.

*The word grammar component can deduce the lexical category (part-of-speech) of a word.*

PC-KIMMO version 1 could break a word into its morphemes and gloss each morpheme, but it could not tell you the category of the whole word. For example, given the word *computerization*, version 1 would return this analysis:

```
com`pute+er+ize+ation      com`pute+NR19+VR6+NR23
```

The original word has been broken into four morphemes with glosses, but there is no indication that the whole word is a noun. This deficiency made PC-KIMMO less useable as a front-end to a syntactic parser, since a syntactic parser must know the category of each word. In version 2, the feature-passing mechanism can be used to determine the lexical category of a word.

*The word grammar component can provide a full feature specification for a word.*

Besides lexical category, a word grammar can also determine all features of a word that are relevant to syntactic parsing, such as tense, number, gender, and case.

## 2.2 A sample session using the word grammar component

For instructions on installing and starting up PC-KIMMO, see section 5. For this sample session, we will use the English description called Englex, which accompanies the version 2 release. This description includes a rules file, a lexicon file, and a grammar file. To load these files, start up PC-KIMMO and type *take englex*. Your screen should look like this:

```
-----
PC-KIMMO TWO-LEVEL PROCESSOR
Version 2.0 (December 15, 1994), Copyright 1994 SIL
Type ? for help
PC-KIMMO>take englex
PC-KIMMO>load rules english
Loading rules from english.rul
PC-KIMMO>load lexicon english
Loading lexicon from english.lex
PC-KIMMO>
-----
```

The rules file and the lexicon file have now been loaded, but not the grammar file. This demonstrates that use of the word grammar component is optional. If you do not use it, then PC-KIMMO will behave just as it did in version 1. Thus you can use version 2 with your existing descriptions without having to write grammar files (however, you must convert your existing lexicon files to the new format required by version 2). To try the Recognizer without the word grammar, just type some *recognize* commands:

```
-----
PC-KIMMO>recognize foxes
`fox+s      `fox+PL
`fox+s      `fox+3SG

PC-KIMMO>
-----
```

Two results were returned for the word *foxes*. The two results are due to two analyses of the suffix *+s*, one the plural suffix for nouns, the other the third, singular suffix for verbs. Obviously our knowledge of English tells us that the first result is correct and the second is incorrect. The point to note here is that the lexicon constructed for this example does not have sufficient morphotactic constraints to disallow the incorrect analysis. There is a new display option that displays the results of the Recognizer in an interlinear format. Type *set alignment on* and then *rec foxes* again:

```
-----
PC-KIMMO>set alignment on
PC-KIMMO>rec foxes
`fox +s
-----
```

```
`fox +PL
N INFL

`fox +s
`fox +3SG
N INFL
```

---

This display vertically aligns each morpheme of the lexical form with its gloss on the second line and its sublexicon name on the third line. Thus we can visually see that each Recognizer result is a sequence of morpheme structures. Not shown in this display, though present internally, are the features associated with each morpheme. Now load the English word grammar and try recognizing the same word again. Type *load grammar english* and then *rec foxes* (features with empty values are not displayed):

---

```
PC-KIMMO>load grammar english
Loading grammar from english.grm
```

```
PC-KIMMO>rec foxes
`fox +s
`fox +PL
N INFL
```

```
1:
  Word
  ____|____
  Stem  INFL
  |      +s
  ROOT  +PL
`fox
`fox

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

---

One important difference is that now only one result is returned, namely the one that correctly interprets the -s suffix as a plural marker. What has happened is this.

- First, the input form *foxes* was analyzed using the rules and lexicon only. This produced the same two results as first shown above. However, these results are retained internally and not displayed on the screen.
- Then, each result was passed to the word grammar component. The word grammar accepted the first result, in which +s is a plural suffix, but rejected the second, in which +s is a verbal suffix. The rejected result was discarded and only the result that satisfied the grammar was retained and displayed on the screen.

Thus the lexicon and grammar work together to produce the desired results. The lexicon serves to break a word into its morphemes using minimal morphotactic constraints, while the grammar applies a more powerful morphotactic mechanism that filters out any incorrect analyses allowed by the lexicon. The Recognizer result display consists of three parts: the tokenized lexical form, the parse tree, and the feature structures. The first part is always displayed, while the other two parts are displayed only if a word grammar is in use and certain options are turned on. In the display shown above, the first part of the result display is the same as it was before the word grammar was loaded (assuming that the *alignment* option is still *on*). The second part of the result display is the analysis tree. The nodes of the tree bear the category symbols used in the word grammar rules. The leaf nodes (ROOT and INFL) also display the lexical form and gloss of each morpheme. The *tree*

option determines how the tree is displayed. In the display above, the *tree* option is set to *full* by default. If the *tree* option is set to *flat*, then it would be displayed as a bracketed string like this:

```
(Word (Stem (ROOT `fox `fox')) (INFL +s '+PL'))
```

Setting the *tree* option to *off* will suppress display of the tree entirely. The third part of the result display consists of feature structures. The *features* option determines how feature structures are displayed. In the display shown above, only the feature structure for the top node of the tree is shown because the *features* option is set to *top*. If it is set to *all*, then the feature structure for each node of the tree is shown:

```
Word_1:
[ cat:   Word
  clitic:-
  head:   [ number:PL
            pos:   N ]
  root_pos:N
  root:   `fox ]
```

```
Stem_2:
[ cat:   Stem
  ajr8:   -
  head:   [ number:SG
            pos:   N
            proper:- ]
  root_pos:N
  root:   `fox
  reg:    + ]
```

```
ROOT_3:
[ cat:   ROOT
  ajr8:   -
  gloss:  `fox
  head:   [ number:SG
            pos:   N
            proper:- ]
  root_pos:N
  lex:    `fox
  reg:    + ]
```

```
INFL_4:
[ cat:   INFL
  from_pos:N
  gloss:  +PL
  head:   [ number:PL
            pos:   N ]
  lex:    +s
  reg:    + ]
```

Setting the *features* option to *off* will suppress display of feature structures entirely. In the example above using the word *foxes*, the lexicon returned two results, one of which was disallowed by the word grammar. In the next example, the lexicon returns one result which is expanded into three by the grammar. First, turn off the grammar component by typing *set grammar off*. This causes the Recognizer to behave just as if no grammar were loaded. Then type *rec deer*. One result is displayed.

```
-----
PC-KIMMO>set grammar off
PC-KIMMO>rec deer
`deer      `deer
-----
```

Now type *set grammar on* and *rec deer* again.

```

-----
PC-KIMMO>set grammar on
PC-KIMMO>rec deer
`deer      `deer

```

```

1:
Word_4
|
Stem_5
|
ROOT_6
`deer
`deer

```

```

Word:
[ cat:    Word
  clitic:-
  head:    [ number:SG
             pos:    N
             proper:- ]
  root_pos:N
  root:    `deer ]

```

```

2:
Word_1
|
Stem_2
|
ROOT_3
`deer
`deer

```

```

Word:
[ cat:    Word
  clitic:-
  head:    [ number:PL
             pos:    N
             proper:- ]
  root_pos:N
  root:    `deer ]

```

In this display, the single result from the lexicon has been given two analyses by the word grammar. While the two trees are identical, the feature structures for the top nodes of the trees differ: for the first tree, the feature *number* has the value SG, while for the second it has the value PL. In other words, the grammar has produced both a singular and a plural form for *deer*. The next example demonstrates that the prefix *un+* has two analyses (or there are two homophonous prefixes spelled *un+*). First, the negative *un+* as in *unclear* attaches to adjectives and negates their meaning. Second, the reversive *un+* as in *untie* attaches to verbs and reverses their action. A word such as *unlockable* has two readings due to the ambiguity of the *un+* prefix: either "not lockable" or "can be unlocked." To see how the word grammar distinguishes these readings, type *rec unlockable*:

```

-----
PC-KIMMO>rec unlockable
un+`lock+able      NEG4+`lock+AJR25a

```

```

1:
      Word
      |
      Stem
  _____|_____
PREFIX      Stem
  un+      _____|_____
  NEG4+    Stem  SUFFIX

```

```

      |      +able
ROOT  +AJR25a
`lock
`lock

```

```

Word:
[ cat:   Word
  clitic:-
  head:   [ aform: POS
            pos:   AJ ]
  root_pos:V
  root: `lock ]

```

un+`lock+able          REV1+`lock+AJR25a

```

1:
      Word
      |
      Stem
      |
  _____|_____
  |               |
Stem             SUFFIX
|               |
|               +able
|               +AJR25a
PREFIX Stem
un+          |
REV1+       ROOT
            `lock
            `lock

```

```

Word:
[ cat:   Word
  clitic:-
  head:   [ aform: POS
            pos:   AJ ]
  root_pos:V
  root: `lock ]

```

The two trees show how the two readings are produced. In the first tree, the negative *un+* attaches to the adjective *lockable* to give the reading "not lockable." In the second tree, the reversive *un+* first attaches to the verb *lock* to produce *unlock*, which in turn is suffixed with *+able* to give the reading "can be unlocked." Notice, however, that both trees have the same feature structure for their top nodes; in other words, *unlockable* is an adjective in either reading.

## Using the *Synthesizer* function

The *Synthesizer* function accepts as input a morphological form (a sequence of morpheme glosses separated by spaces) and returns one or more surface forms. In order to synthesize forms, you must first load a synthesis lexicon. This can be the same lexicon that you use for recognition, but it must be loaded again as a synthesis lexicon. You can have both a recognition lexicon and a synthesis lexicon loaded at the same time. They may or may not be the same lexicon. It is not necessary to load a recognition lexicon to synthesize forms. A rules file must be loaded before a synthesis lexicon can be loaded. Use of a grammar file is optional.

To try the *Synthesizer* function, first load the Englex lexicon as a synthesis lexicon (Macintosh users may first need to increase PC-KIMMO's memory partition):

```

-----
PC-KIMMO>load synthesis-lexicon english
Loading synthesis-lexicon from english.lex
-----

```

Now use the *synthesize* command with these morphological forms:

```

-----
PC-KIMMO>synthesize REV1+ `lock +AJR25a
unlockable

PC-KIMMO>syn NEG4+ `tie +ING
untying

PC-KIMMO>syn `fox +PL +GEN
foxes'
-----

```

To demonstrate that synthesis uses the grammar (if one is loaded), try this ill-formed input form:

```

-----
PC-KIMMO>syn `fox +3SG
*** NONE ***

PC-KIMMO>set grammar off

PC-KIMMO>syn `fox +3SG
foxes

PC-KIMMO>rec foxes
`fox `fox+PL
`fox `fox+3SG
-----

```

When the grammar is used, the form *fox +3SG* is rejected, since the grammar prohibits a verbal suffix on a noun. When the grammar is turned off, then the surface form *foxes* is returned, since this is permitted by the lexicon; this is demonstrated by recognizing the form *foxes* with the grammar off.

## 2.3 Changes to the rules component

### 2.3.1 Comment character declaration

In version 1, the comment character could be set either with a command line option (*-c*) or with the command *set comment*. In version 2 the comment character is declared in the rules file with the new **COMMENT** keyword. For example, this declaration sets the comment character to %:

```
COMMENT %
```

### 2.3.2 Support for multigraphs

In version 1, the alphabet declared in a rules file was restricted to single characters. In version 2, the alphabet can also include multigraphs--digraphs, trigraphs, and so on. For example, a description of Spanish would include the digraph *ll* in the list of alphabetic characters declared in the rules file. The digraph *ll* could then be used in the column header of a state table or in a **SUBSET** declaration. It is important to understand that a multigraph can never be interpreted as a sequence of characters. For example, the alphabet for a Spanish description will include both *l* and *ll*; but *ll* will always be treated as a multigraph, never as a sequence of *l* plus *l*.

### 2.3.3 Compatibility with Xerox's *twolc* compiler

Version 2 now can load a rules file (actually state tables) produced by Xerox's *twolc* rule compiler. For information on this compiler, visit Xerox Lexical Technology.

## 2.4 Changes to the lexical component

In version 1, lexical entries looked like this:



```

LEXICON NOUN
`boy      Noun      "N(boy) "
`baby     Noun      "N(baby) "
`feet     Noun      "N(foot).PL"

```

Lexical entries were grouped into sublexicons declared with the keyword LEXICON; in the example above, these entries all belong to the NOUN sublexicon. Each lexical entry was composed of three fields, separated by white space and terminated by a new line. The three fields comprising an entry were the lexical item (or lexical form), the alternation name, and a gloss string. In version 2 of PC-KIMMO, these lexical entries look like this:

```

\lexform `boy
\sublexicon NOUN
\alternation Noun
\gloss N(boy)

\lexform `baby
\sublexicon NOUN
\alternation Noun
\gloss N(boy)

\lexform `feet
\sublexicon NOUN
\alternation Noun
\features pl irreg
\gloss N(foot).PL

```

Lexical entries are encoded in "field-oriented standard format." Standard format is an information interchange convention developed by the Summer Institute of Linguistics. It tags the kinds of information in ASCII text files by means of markers which begin with backslash. Field-oriented standard format (FOSF) is a refinement of standard format geared toward representing data which has a database-like record and field structure. Using FOSF to encode lexical entries has several advantages

- Version 2 of PC-KIMMO has added an optional *features* field to lexical entries (see the entry above for *feet*). The format for lexical entries used in version 1 would have required adding a fourth column to each entry which often would be empty. The new format makes adding another field easier.
- The user can define the codes used to mark fields by mapping them to fixed internal codes. For example, this declaration in the main lexicon file says that the field code *gloss* will mark the gloss field (where G is the internal code for the gloss field):

```

FIELDPCODE gloss G

```

This means that lexical entries can include alternative gloss fields, one of which is chosen for use when the lexicon is loaded. For example, a lexical entry might look like this:

```

\lexform `boy
\sublexicon NOUN
\alternation Noun
\eng boy
\sp muchacho

```

The field code *eng* and *sp* mark English and Spanish gloss fields. If the user wants English glosses then he includes this declaration in the main lexicon file:

```

FIELDPCODE eng G

```

and if he wants Spanish glosses, this declaration:

```

FIELDPCODE sp G

```

The same strategy can be used with any field used in lexical entries.

- Fields that haven't been declared in the main lexicon file are considered extraneous and ignored by PC-KIMMO. This means that lexical entries can contain fields of information intended for purposes other than use with PC-KIMMO without interfering with PC-KIMMO's operation. For example, a field linguist could develop a dictionary using FOSF where each entry contains many fields; with a bit of planning as to choice of field codes, the entries would still be compatible with PC-KIMMO.
- FOSF files are compatible with other software developed by the Summer Institute of Linguistics. This means that one can now use Shoebox (for PCs) or MacLex (for Macintosh) to manage lexicon files. It would also be easy to use a commercial database program to manage lexical entries and write a routine to export the entries in the required format for PC-KIMMO.
- Lexical entries belonging to a sublexicon do not have to be listed consecutively in a single file (as was the case for PC-KIMMO version 1); rather, lexical entries in a file can occur in any order, regardless of what sublexicon they belong to. Lexical entries of a sublexicon can even be placed in two or more separate files. This makes it possible now to optionally load files of entries that contain lexical entries belonging to various sublexicons. For example, you could optionally load a file of technical terms containing nouns, verbs, and adjectives.

## 2.5 Changes to the user interface

The following new commands are available in PC-KIMMO version 2. For detailed explanations of each command, see section 5.

### **clear**

Same as NEW command in version 1.

### **[file] compare synthesize** [*filespec*]

Reads morphological forms (a sequence of morpheme glosses separated by spaces) from *filespec*, submits them to the synthesizer, and compares the resulting surface form(s) with the expected results listed in *filespec*.

### **file synthesize** *input-filespec* [*output-filespec*]

Reads a list of morphological forms (a sequence of morpheme glosses separated by spaces) from *input-filespec*, submits them to the synthesizer, and returns each morphological form followed by the resulting surface form(s).

### **load grammar** [*filespec*]

Loads a word grammar from *filespec*.

### **load synthesis-lexicon** [*filespec*]

Loads a synthesis-lexicon from *filespec*.

### **save** [*filespec*]

Writes the current setting to a *take* file named *filespec*. If *filespec* is not specified, the settings are written to a file named PCKIMMO.TAK in the current directory. On start-up, PC-KIMMO automatically tries to load default settings from PCKIMMO.TAK (or PC-KIMMO.TAK).

### **set alignment** {on | off}

Turns alignment display mode on or off.

### **set ambiguities** *number*

Limits the number of analyses produced by the word grammar to *number*.

**set failures {on | off}**

Turns grammar failure mode on or off.

**set features {top | all | off}**

Sets the feature display mode.

**set features {full | flat}**

Sets the feature display style.

**set gloss {on | off}**

Turns gloss display mode on or off.

**set grammar {on | off}**

Turns the loaded word grammar on or off.

**set trim-empty-features {on | off}**

Turns trimming of empty features on or off.

**set tree {full | flat | indented | off}**

Sets the tree display style.

**set unification {on | off}**

Turns feature unification in the word grammar on or off.

**set warnings {on | off}**

Turns warning mode on or off.

**synthesize** [*morphological-form*]

Produces surface forms from a morphological form (a sequence of morpheme glosses).