

PC-Kimmo

Mike Rosner

May 18, 2008

Introduction

Morphology

PC-KIMMO

Englex

Word Grammar

Morphology

Morphology concerns how words are formed from morphemes. A morpheme is a meaning-bearing word-part, e.g. a final “s” which denotes plural.

Every language has two kinds of word formation process:

- ▶ **Inflection**, which provides various forms of the same word (e.g. *bark, baked, barking*).
- ▶ **Derivation**, which creates new words from old ones, e.g. *anti + climax = anticlimax*.

The distinction is not always that clear since it is often difficult to distinguish different forms of the same word from different words.

Different Words

Generally words undergoing morphological change are regarded as different if:

- ▶ They have different meaning eg. *bank* (of a river) and *bank* (financial institution).
- ▶ They have different syntactic category e.g. *observe* and *observation*
- ▶ They sound different even if they are spelt the same way e.g. *object* (noun - stress on first syllable) and *object* (verb - stress on second syllable)

If they have the same category e.g. *observe* and *observed* then they are different forms of the same word,

Computational Morphology

- ▶ Computational morphology defines computations concerning the internal structure of words.
- ▶ We broadly distinguish between *analysis* - which takes a complex word apart, and *synthesis*, which creates a whole word from its constituent morphemes.
 - ▶ A *segmentation* problem: how to get from a piece of written text to the sequence of morphemes that make it up.
 - ▶ A *morphotactic* problem: how to combine the individual morphemes together in a legitimate way.
 - ▶ A *spelling* problem which must account for phenomena which occur at morpheme boundaries.
 - ▶ An *interpretation* problem: how to compute the syntactic category of the result.

PC-KIMMO

- ▶ PC-KIMMO is a 1985 implementation of a program dubbed KIMMO after its inventor Kimmo Koskenniemi (see Koskenniemi 1983).
- ▶ The program is designed to generate (produce) and/or recognize (parse) words using a *two-level model* of word structure in which a word is represented as a correspondence between its *lexical level form* and its *surface level form*.

Two-Level Morphology

- ▶ The theoretical model of phonology embodied in PC-KIMMO is called two-level Morphology
- ▶ In the two-level approach, phonology is treated as the correspondence between the lexical level of underlying representation of words and their realization on the surface level.
- ▶ For example, to account for the rules of English spelling, the surface form spies must be related to its lexical form 'spy+s as follows (where ' indicates stress, + indicates a morpheme boundary, and 0 indicates a null element):

Lexical Representation: ' s p y + 0 s

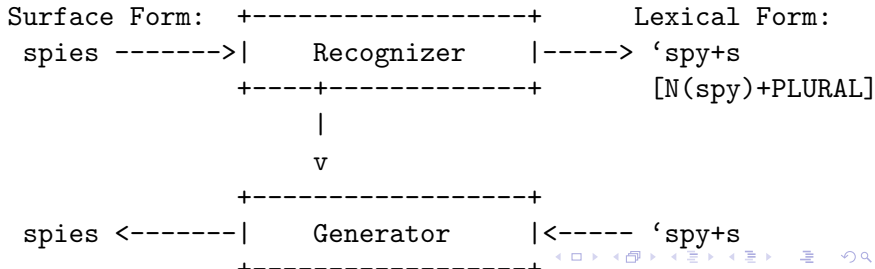
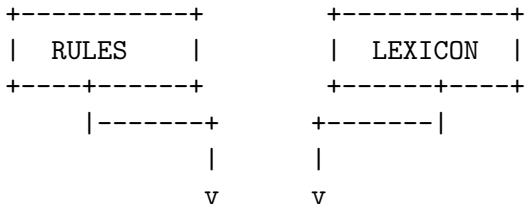
Surface Representation: 0 s p i 0 e s

Structure of a PC-KIMMO Description

A PC-KIMMO description of a language consists of two files provided by the user:

1. a *rules* file, which specifies the alphabet and the phonological (or spelling) rules, and
2. a *lexicon* file, which lists lexical items (words and morphemes) and their glosses, and encodes morphotactic constraints.

PC-KIMMO Organisation



Englex

- ▶ Englex is a description of English morphology and lexicon using the two-level model as it is implemented by PC-KIMMO. 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.

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.
- ▶ **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).

Inflection and Derivation

Morphological processes are traditionally divided into two types: inflection (producing e.g. *compute*, *computes*, *computing*, and *computed*) and derivation (producing e.g. *compute*, *computer*, *computerize*, *recompute*). The distinction has ramifications for building a morphological parsing system.

- ▶ To interface with a syntactic parser, the system must provide all the inflectional categories of a word as well as its part-of-speech
- ▶ Second, Englex must be able to recognize words formed by derivation. These can either be listed explicitly as separate entries, or derived from the respective roots and affixes.
- ▶ Whilst the second approach is has advantages, it is difficult to avoid spurious analyses (e.g. cabbage = cab + age).

Other Issues

Multiple Senses: when a word has several senses, how many dictionary entries should there be? Example, the word *fair* meaning (a) light coloured (b) impartial and (c) a festival.

In general answer depends on purposes. Englex's lexicon is a parsing lexicon, not a full dictionary, so it only distinguishes homonyms having different parts of speech.

Lexical conversion. when a word belongs to more than one part-of-speech (e.g. *ride* can be either verb or noun). Since in this case the words share a sense, the derivational relation between them is often described as zero derivation or conversion.

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.

Compounds

- ▶ **Solid.** e.g. *bedroom*
- ▶ **Hyphenated.** e.g. *moth-eaten*. 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.
- ▶ **Open.** e.g. *rose bush*. Not handled

Lexicon Contents

Englex's lexicon contains approximately 20,000 lexical entries (with its productive morphology, it will recognize several times this number of English words.

11,000 nouns, 4,000 verbs, and 3,400 adjectives in the form of affixes, roots, indivisible stems and solid compounds.

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, demonstratives, interjections, ordinals, cardinals, quantifiers,

Morphotactic Constraints

- ▶ Morphotactic constraints are found in (a) the lexicon and (b) 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 according to the following positional analysis:
Prefix* Root Suffix* (Infl) (Clitic)
- ▶ **N.B.** This coarse analysis of morphotactic structure greatly overrecognizes, since it does not enforce any order among prefixes or suffixes.

Morphotactic Constraints

- ▶ For example, it will analyze the non-word *computerizer* into the parts
compute+ize+er
- ▶ Note that reversing the order of the suffixes produces the real word computerize.

Further Morphotactic Constraints

- ▶ To get the last example right, we must take the following constraint into consideration:
suffix +ize can only attach to a *noun stem*
- ▶ **Constraints such as these are difficult to implement efficiently with finite-state lexicon.**
cf. en+large+s
- ▶ In Englex, enforcement of any cooccurrence restrictions among the positional morpheme slots is performed by the **word grammar**.

Implementation of the Finite State Lexicon

The finite-state morphotactic analysis is implemented using declarations like this which associate names with sublexicons:

```

KEYWORD      NAME          SUBLEXICON
ALTERNATION Particle      AUX AUX-V PP CJ PP-CJ DT PR ...
ALTERNATION Prefix        PREFIX
ALTERNATION Root          N AJ V AV N-V N-AJ AJ-V AJ-AV ...
ALTERNATION Suffix        SUFFIX
ALTERNATION Compound      INITIAL
etc.
```

Each sublexicon contains entries. Every lexicon begins with the INITIAL Sublexicon.

The INITIAL Sublexicon

- ▶ 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 null entries must be empty.

The INITIAL Sublexicon

```
\lf 0          <-- Lexical Form (none)
\lx INITIAL    <-- Sublexicon (this entry)
\alt Particle  <-- Goto (see name table)
\gl           <-- Gloss
```

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

The Prefix slot can be filled by zero or more prefixes from the PREFIX sublexicon

The PREFIX Sublexicon

The first is a null entry whose alternation field names the Root alternation; this permits the prefix slot to be optional.

```

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

The PREFIX Sublexicon

The other two entries specify the Prefix alternation; thus if a prefix is recognized, the system keeps looping through the PREFIX sublexicon.

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

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

Examples from The Root Sublexicon

\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

Suffix Examples

```
\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
```

End Examples

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

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

Loading Englex

```

PC-KIMMO>load rules english.rul
Loading rules from english.rul
PC-KIMMO>load lexicon english.lex
Loading lexicon from english.lex
affix.lex                255 entries
noun.lex                 10461 entries
verb.lex                 4215 entries
adjectiv.lex            3345 entries
adverb.lex              400 entries
minor.lex                379 entries
proper.lex              1057 entries
abbrev.lex               127 entries
technica.lex            813 entries
natural.lex             435 entries
foreign.lex              88 entries
  
```

Simple Recognition

In the simplest usage we see one reading per line. The format of each line is lexical string/lexical gloss

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

The lexical string, the lexical gloss, and the respective sublexicon names of each reading can be visualised by setting alignment on.

Simple Recognition

```
PC-KIMMO>set alignment on
```

```
PC-KIMMO>rec foxes
```

```
'fox +s                <-- Lexical String
'fox +PL                <-- Lexical Gloss
N      INFL            <-- Sublexicon Names
```

```
'fox +s                <-- Lexical String
'fox +3SG              <-- Lexical Gloss
N      INFL            <-- Sublexicon Names
```

Word Grammar Rules

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.

Word = Word CLITIC

Word = PARTICLE

Word = Stem (INFL)

Stem = PREFIX Stem

Stem = Stem SUFFIX

Stem = ROOT

Using the Word Grammar

The word grammar introduces further morphotactic constraints. Using it we can eliminate one of the readings.

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

The output includes the parse tree for the word as well as the associated F-structure

Using the Word Grammar

```
'fox +s
'fox +PL
N      INFL
```

1:

Word

```
---|---
```

```
Stem  INFL
```

```
|      +s
```

```
ROOT  +PL
```

```
'fox
```

```
'fox
```

<-- TREE

Word Grammar Analysis: Shortcomings

1. **This tree does not tell us the part-of-speech of a word**, i.e. that the word *enlargement* is a noun.
2. Second, it still **does not adequately characterize important morphotactic constraints**.
 - ▶ For example, it would also return an analysis of the word *enlargement* with the bracketing
[en+ [large+ment]]
This of course is an incorrect analysis since it posits the nonexistent stem *largement*.
 - ▶ The analysis does not capture the fact that a suffix such as +ment can attach to *verb stems* only.

Rule Details

These rules are intended to (a) constrain the input and output categories of affixes, and (b) to determine the POS of a word. The grammar uses 2 features for this purpose

- ▶ Stems and roots have a *pos* feature standing for part-of-speech.
- ▶ Affixes have a *from_pos* feature as well as 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.

Examples

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]
```

Word Grammar Rules with 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>
  
```

Using the Word Grammar

Word:

```
[ cat:      Word
  head:     [ agr:      [ 3sg:  - ]
              number:PL
              pos:     N ]           <-- F-STRUCTURE
  root:     'fox
  root_pos:N
  clitic:-
  drvstem:- ]
```

1 parse found