Chapter 1

Appendix: NLP in Python vs other Programming Languages

Many programming languages have been used for NLP. As explained in the Preface, we have chosen Python because we believe it is well-suited to the special requirements of NLP. Here we present a brief survey of several programming languages, for the simple task of reading a text and printing the words that end with `ing`. We begin with the Python version, which we believe is readily interpretable, even by non Python programmers:

```
import sys
for line in sys.stdin:
    for word in line.split():
        if word.endswith('ing'):
            print word
```

Like Python, Perl is a scripting language. However, its syntax is obscure. For instance, it is difficult to guess what kind of entities are represented by: `<>`, `$`, `my`, and `split`, in the following program:

```
while (<>) {
    foreach my $word (split) {
        if ($word =~ /ing$/) {
            print "$word\n";
        }
    }
}
```

We agree that "it is quite easy in Perl to write programs that simply look like raving gibberish, even to experienced Perl programmers" (Hammond 2003:47). Having used Perl ourselves in research and teaching since the 1980s, we have found that Perl programs of any size are inordinately difficult to maintain and re-use. Therefore we believe Perl is no longer a particularly suitable choice of programming language for linguists or for language processing.

Prolog is a logic programming language which has been popular for developing natural language parsers and feature-based grammars, given the inbuilt support for search and the *unification* operation which combines two feature structures into one. Unfortunately Prolog is not easy to use for string processing or input/output, as the following program code demonstrates for our linguistic example:

```
main :-
    current_input(InputStream),
```

```
    read_stream_to_codes(InputStream, Codes),
    codesToWords(Codes, Words),
    maplist(string_to_list, Words, Strings),
    filter(endsWithIng, Strings, MatchingStrings),
    writeMany(MatchingStrings),
    halt.

codesToWords([], []).
codesToWords([Head | Tail], Words) :-
    ( char_type(Head, space) ->
        codesToWords(Tail, Words)
    ;
        getWord([Head | Tail], Word, Rest),
        codesToWords(Rest, Words0),
        Words = [Word | Words0]
    ).

getWord([], [], []).
getWord([Head | Tail], Word, Rest) :-
    (
        ( char_type(Head, space) ; char_type(Head, punct) )
    ->  Word = [], Tail = Rest
    ;   getWord(Tail, Word0, Rest), Word = [Head | Word0]
    ).

filter(Predicate, List0, List) :-
    ( List0 = [] -> List = []
    ;   List0 = [Head | Tail],
        ( apply(Predicate, [Head]) ->
            filter(Predicate, Tail, List1),
            List = [Head | List1]
        ;   filter(Predicate, Tail, List)
        )
    ).

endsWithIng(String) :- sub_string(String, _Start, _Len, 0, 'ing').

writeMany([]).
writeMany([Head | Tail]) :- write(Head), nl, writeMany(Tail).
```

Java is an object-oriented language incorporating native support for the Internet, that was originally designed to permit the same executable program to be run on most computer platforms. Java has replaced COBOL as the standard language for business enterprise software:

```
import java.io.*;
public class IngWords {
    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new
          InputStreamReader(
                 System.in));
        String line = in.readLine();
        while (line != null) {
            for (String word : line.split(" ")) {
```

```
                if (word.endsWith("ing"))
                    System.out.println(word);
            }
            line = in.readLine();
        }
    }
}
```

The C programming language is a highly-efficient low-level language that is popular for operating system and networking software:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv) {
   int i = 0;
   int c = 1;
   char buffer[1024];

   while (c != EOF) {
       c = fgetc(stdin);
       if ( (c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')
           buffer[i++] = (char) c;
           continue;
       } else {
           if (i > 2 && (strncmp(buffer+i-3, "ing", 3) == 0 || strncmp(buffer+i-3, "ING
               buffer[i] = 0;
               puts(buffer);
           }
           i = 0;
       }
   }
   return 0;
}
```

LISP is a so-called functional programming language, in which all objects are lists, and all operations are performed by (nested) functions of the form `(function arg1 arg2 ...)`. Many of the earliest NLP systems were implemented in LISP:

```
(defpackage "REGEXP-TEST" (:use "LISP" "REGEXP"))
(in-package "REGEXP-TEST")

(defun has-suffix (string suffix)
  "Open a file and look for words ending in _ing."
  (with-open-file (f string)
     (with-loop-split (s f " ")
        (mapcar #'(lambda (x) (has_suffix suffix x)) s))))

(defun has_suffix (suffix string)
  (let* ((suffix_len (length suffix))
   (string_len (length string))
    (base_len (- string_len suffix_len)))
    (if (string-equal suffix string :start1 0 :end1 NIL :start2 base_len :end2 NIL)
```

```
          (print string))))

(has-suffix "test.txt" "ing")
```

Ruby is a more recently developed scripting language than Python, best known for its convenient web application framework, *Ruby on Rails*. Here are two Ruby programs for finding words ending in *ing*

```
ARGF.each { |line|
  line.split.find_all { |word|
    word.match(/ing$/)
  }.each { |word|
    puts word
  }
}


for line in ARGF
  for word in line.split
    if word.match(/ing$/) then
      puts word
    end
  end
end
```

Haskell is another functional programming language which permits a much more compact (but incomprehensible) solution of our simple task:

```
import Data.List
main = putStr . unlines . filter ("ing" `isSuffixOf`) . words =<< getContent
```

The unix shell can also be used for simple linguistic processing. Here is a simple pipeline for finding the *ing* words. The first step transliterates any whitespace character to a newline, so that each word of the text occurs on its own line, and the second step finds all lines ending in *ing*

```
tr [:space:] '\n' | grep ing$
```

(We are grateful to the following people for furnishing us with these program samples: Tim Baldwin, Trevor Cohn, David Duke, Rod Farmer, Andrew Hardie, Aaron Harnly, Edward Ivanovic, and Lars Yencken.)