T-(538|725)-MALV, Natural Language Processing Tokenisation

Hrafn Loftsson¹  Hannes Högni Vilhjálmsson¹

¹School of Computer Science, Reykjavik University

Ágúst 2007
Outline

1. Tokenisation
2. Sentence segmentation
3. Lexical analyser
4. Unix/Linux tools
Outline

1. Tokenisation
2. Sentence segmentation
3. Lexical analyser
4. Unix/Linux tools
Tokenisation (í. tilreiðing)

- Breaking a text into smaller units – each unit having some particular meaning.
- In most cases, into separated words and sentences.
- Carried out by finding the word boundaries, the points where one words ends and another begins.
- Tokens/lexemes: the words identified by the process of tokenisation.

Word segmentation
- Tokenisation in languages where no word boundaries are explicitly marked
  - E.g. when whitespaces are not used to signify word boundaries.
  - Chinese, Thai
- We focus on tokenisation in “space-delimited languages”.

Loftsson, Vilhjálmsson Corpora
Tokenisation

**Programming Languages**

- Part of the lexical analysis in the compilation of a programming language source.
- Programming languages are designed to be unambiguous – both with regard to lexemes and syntax.

**Natural Languages**

- The same letter can serve many different functions.
- The syntax is not as strict as in programming languages.
Tokenisation

Programming Languages

- Part of the lexical analysis in the compilation of a programming language source.
- Programming languages are designed to be unambiguous – both with regard to lexemes and syntax.

Natural Languages

- The same letter can serve many different functions.
- The syntax is not as strict as in programming languages.
Is tokenisation an easy task?

“Clairson International Corp. said it expects to report a net loss for its second quarter ended March 26 and doesn’t expect to meet analysts’ profit estimates of $3.9 to $4 million, or 76 cents a share to 79 cents a share, for its year ending Sept. 24.”

The period is used in three different ways. When is a period a part of a token and when not?

’ used in two different ways.
Abbreviations

- Abbreviations need to be recognised.

- “Leitarvefurinn dohop.com hefur sett nýjustu lausn sína á ferðaáætlunum á markað í Bandaríkjunum. En veftaun íslenska hugbúnaðarfyrirtækisins dohop ehf. auðveldar fólk að gera ferðaáætlanir á netinu.” (mbl.is, 08.02.2006)

- [http://is.wikipedia.org/wiki/Listi_yfir_algengar_skammstafanir_%C3%AD_%C3%ADslensku](http://is.wikipedia.org/wiki/Listi_yfir_algengar_skammstafanir_%C3%AD_%C3%ADslensku)

Multiword expressions (í. fleiryrt orð)

- In some cases, a sequence of tokens needs to be handled as one token.

- *in spite of, aftur á móti, að auki, 26. mars*
Tokenisation

Abbreviations

- Abbreviations need to be recognised.

- “Leitarvefurinn dohop.com hefur sett nýjustu lausn sína á ferðaáætlunum á markað í Bandaríkjunum. En veﬂausn íslenska hugbúnaðarfyrirtækisins dohop ehf. auðveldar fólki að gera ferðaáætlanir á netinu.” (mbl.is, 08.02.2006)

- http://is.wikipedia.org/wiki/Listi_yfir_algengar_skammstafanir_%C3%AD_%C3%ADslensku

Multiword expressions (í. fleiryrt orð)

- In some cases, a sequence of tokens needs to be handled as one token.

- in spite of, aftur á móti, að auki, 26. mars
Outline

1. Tokenisation
2. Sentence segmentation
3. Lexical analyser
4. Unix/Linux tools
Broken into sentences.

Requires an understanding of the various uses of punctuation characters in a language.

The boundaries between sentences need to be recognised.
- The boundaries occur between words.
- "Sentence boundary detection"

At first sight, this seems simple.
- Can’t we just search for “.”, “?” and “!”?
- And sometimes “.”, “;”?

What about: “Ertu frá þér maður, og sjálfur sjómannadagurinn framundan!”, segir prestsfrúin . . .
Sentence segmentation

- Is a simple rule not sufficient?

- delim = “.” | “!” | “?”

  IF (right context = delim + space + capital letter OR
delim + quote + space + capital letter OR
delim + space + quote + capital letter)

  THEN sentence boundary

- Abbreviations can make sentence segmentation difficult:
  - “The contemporary viewer may simply ogle the vast wooded vistas rising up from the Saguenay River and Lac St. Jean, standing in for the St. Lawrence River.”
  - “The firm said it plans to sublease its current headquarters at 55 Water St. A spokesman declined to elaborate.”
A simple sentence segmentation

- If a period preceding a space is used as an indication of sentence boundaries, then one can recognise about 90% of the periods which end a sentence in the Brown corpus (http://en.wikipedia.org/wiki/Brown_Corpus).

- One can get quite far by using simple regular expressions without using a list of abbreviations.

- Let us assume three kinds of abbreviations in English:
  
  A., B., C. \[A-Za-z]\.
  U.S., m.p.h. \[A-Za-z]\.(\[A-Za-z]\.)+
  Mr., St., Assn. \[A-Z]\[bcdfghj-np-tvxz]+\.

- By using these two simple methods one can correctly recognise about 98% of the sentence boundaries in the Brown corpus.
A paper about tokenisation

- See “Other material - Tokenisation” in MySchool.
Outline

1. Tokenisation
2. Sentence segmentation
3. Lexical analyser
4. Unix/Linux tools

Loftsson, Vilhjálmsson  Corpora
A lexical analyser (í. lesgreinir) is a program which breaks a text into lexemes (tokens).

A program which generates a lexical analyser is called a *lexical analyser generator* (í. lesgreinissmiður)

- Examples: Lex/Flex/JFlex ([http://jflex.de/](http://jflex.de/))
- The user defines a set of regular expression patterns.
- The program generates finite-state automata.
- The automata are used to recognise tokens.
JFlex

Java code is generated

- A tool which generates a lexical analyser given a set of regular expressions.
- Generates Java code, which contains a finite-state automaton (state transition table).
- Input: JFlex source program (e.g. Simple.flex)
- Output: Java code (e.g. Simple.java)

The Java code compiled and executed

- javac Simple.java (the output is Simple.class)
- java Simple <textfile>
Java code is generated

- A tool which generates a lexical analyser given a set of regular expressions.
- Generates Java code, which contains a finite-state automaton (state transition table).
- Input: JFlex source program (e.g. Simple.flex)
- Output: Java code (e.g. Simple.java)

The Java code compiled and executed

- javac Simple.java (the output is Simple.class)
- java Simple <textfile>
To make Java run (Windows)

- Set c:\jflex\bin into path.
- Change the file c:\jflex\bin\jflex.bat to:
  - set JFLEX_HOME="C:\JFLEX"
  - REM for JDK 1.2
  - java -Xmx128m -jar %JFLEX_HOME%\lib\JFlex.jar
**JFlex example**

%% A finite-state automata recognising (a|b)*abb

%public
%class Simple
%standalone
%unicode

%
    String str = "Found: ";
%

Pattern = (a|b)*abb

%%
{Pattern}   { System.out.println(str + " " + yytext());}
.          { ;}

Loftsson, Vilhjálmsson

Corpora
JFlex example

%% A good tokeniser for Icelandic?

%public
%class IceGood
%standalone
%unicode

{%
%}

WhiteSpace = [ \t\f\n]
Lower = [a-záéðíóúýþæö]
Upper = [A-ZÁÉÐÍÓÚÝÞÆÖ]
IceChar = {Upper}|{Lower}
IceWord = {IceChar}+

%%
{WhiteSpace} {;;}
{IceWord} { System.out.println(yytext());}
. { System.out.println(yytext());}
Outline

1. Tokenisation
2. Sentence segmentation
3. Lexical analyser
4. Unix/Linux tools
Various Unix tools exist which simplify the tokenisation and processing of texts:

- **grep** (general regular expression parser)
- **tr** (translate characters)
- **sed** (string/stream edit)
- As well as more tools that we will look at later.
“translate characters”

tr set1 set2 < inputfile > outputfile

Example (changes lower case to upper case):

tr ’[a-z]’ ’[A-Z]’ < inputfile > outputfile

With Icelandic letters:

tr ’[a-z\341\346\351\355\360\363\366\372\375\376]’
’[A-Z\301\306\311\315\320\323\326\332\335\336]’
< inputfile > outputfile

Octal values: http://en.wikipedia.org/wiki/Octal

Ascii-codes: http://www.ascii-code.com/
- `tr -d 'set1'`
  - Removes all the letters in the set `set1`.

- `tr -c 'set1' 'char2'`
  - Converts letters which are not in `set1` to the letter `char2`.

- `tr -s set1 set2`
  - Converts the letters in the set `set1` for letters in the set `set2` and suppress the output (each sequence of a repeated letter becomes one letter).

- Example (a tokeniser?):

  ```bash
  tr -s '' \012 < inputfile > outputfile
  ```

  ```bash
  tr -cs '[a-z\341\346\351\355\360\363\366\372\375\376
  A-Z\301\306\311\315\320\323\326\332\335\336
  0-9.,!?]' \012 < inputfile > outputfile
  ```
sed

- String/Stream editor:
- Processes one line at a time from the input file.
- Useful when the text of a line needs to be changed according to a regular expression.
- ’s’ for substitution:
  sed ’s/abc/(abc)/’ < input > output
  sed ’s/[a-z]*/(&)/’ < input > output
  & denotes the matched string
  sed ’s/[a-z]*/(&)/g’ < inntak > uttak
  ’g’ for “global replacement”, if all patterns in the line need to be changed, but not only the first one.
In sed: \n stands for newline

What does the following sed command do:

```bash
sed 's/\.$/\n\./' input.txt > output.txt
```

sed can be used for other things than changing text:

```bash
sed 5q < input.txt > output.txt
```
- Prints out the first 5 lines and quits ('q')

```bash
sed '/^$/d' < input.txt > output.txt
```
- Removes empty lines