Natural Language Processing: Programming Project I – Tokenisation

Reykjavik University – School of Computer Science

September 2009

1 Description

In this project, you will develop a tokeniser and a sentence segmentiser for a language of your choice. Indeed, it would be best if your program could handle a family of languages (instead of a single language), for example some of the Germanic languages like English, German, Icelandic, Danish and Swedish. With language independence in mind, regular expressions for matching abbreviations are preferred to an implementation using a list.

As for implementation languages, you should use **JFlex** and/or **Perl** (both of which we have introduced in this course) Note that it may be appropriate to use both JFlex and Perl. One might, for example, use JFLex to do "simple" tokenisation and then read its output into a Perl program which does more complicated tokenisation/sentence segmentation, not easily solved by using regular expressions.

Hence, you could implement your solution by constructing a set of units running in a sequence. In that case, each unit reads as input the output from the preceeding unit in the sequence. However, the exact division of units should be hidden from the user, i.e he/she should only need to enter a command like:

```
tokenise input.txt output.txt
```

where *input.txt* is the input file and *output.txt* is the corresponding tokenised output text.

This project is in two parts, described below in Sections 1.1 and 1.2.

1.1 Word tokeniser

- The input to this program is a text file with a "free format", i.e. the file can contain one sentence per line, many sentences per line, one token

per line, several tokens per line, etc.

- The output should be one token per line.

1.2 Sentence segmentiser

- The input to this program is a text file with one token per line, i.e. the input is a file generated by the word tokeniser.
- The output is one token per line along with empty lines denoting sentence boundaries.

2 The goal

Your goal should be to develop a tokeniser/segmentiser which is reasonably accurate, but it does not have to be perfect (which indeed is a difficult task!).

3 Testing

For testing you should gather text in your own language (and related languages) from the web, for example, text from newspapers, personal pages, university pages, etc. The text should contain at least 10,000 tokens and make sure that it contains **different types of tokens**, e.g. words, personal names, numbers and abbreviations.

4 Due date

Return you program code, along with the input text and the corresponding output, no later than Thursday, October 8th, at 23:59.