9 TOKENIZATION*

Gregory Grefenstette

9.1 INTRODUCTION

The linguistic annotation of naturally occurring text can be seen as a progression of transformations of the original text, with each step abstracting away surface differences. *Tokenization* is one of the earliest steps in this transformation during natural language processing.

Tokenization means dividing up the input text, which to a computer is just one long string of characters, into subunits, called *tokens*.[1] These tokens are then fed into subsequent natural language processing steps such as morphological analysis, wordclass tagging and parsing. Since these subsequent treatments are usually designed to work on individual sentences, a subsidiary task of tokenization is often to identify sentence boundaries as well as token boundaries.[2] Though rarely discussed, and quickly dismissed, tokenization in an automated text processing system poses a number of thorny questions, few of which have completely perfect answers.

Tokenization is not the first step in the abstraction process. If you consider an original printed document, most typesetting distinctions (e.g. font size, font style, page layout, pictures and graphics) are filtered out of the text that will be tokenized

[1] Here a *token* means the individual appearance of a word in certain position in a text. For example, one can consider the wordform *dogs* as an instance of the word *dog*. And the wordform *dogs* that appears in, say, line 13 of page 143 as a specific *token*.

[2] Sentence identification is not always considered as a part of tokenization.

*The following work was accomplished at Xerox Limited.

H. van Halteren (ed.), Syntactic Wordclass Tagging, 117-133.

and further analysed. It is not that these elements do not carry meaning (cf. Holstege *et al.* 1991) that could be exploited by a machine, but rather that the great variety of different typesetting conventions make it difficult for any general processing system to take them into account. Preferably, however, the information is not deleted altogether but rather translated into *markup*, so that the possibility remains to exploit it during later processing.

The input to a tokenizer, then, is a stream of characters which consists of *graphic tokens* separated by layout (after the previous step probably only space and newline characters) and possibly enhanced by markup symbols. Unfortunately, the graphic tokens, usually defined as anything between two layout symbols, need not coincide with the *linguistic tokens*.

The most obvious exception is formed by the concatenation of words and punctuation marks. Other cases where graphic tokens may be split are enclitic forms, such as "he's", and contractions, such as "'twas". The opposite is also possible: two or more graphic tokens may have to be combined into one linguistic token. Examples are proper names, such as "John Jones", and compounds, such as "in spite of". Usually the decision about what constitutes a token will have been made during the definition of the tagset (cf. 4.3.1). There is, however, one special case where two graphic tokens can almost always be combined, viz. *hyphenated words* generated by line breaks. We will examine this case in detail below.

If markup is present in the input, it may or may not be useful. Most of the markup can be ignored and there are even cases where it is necessary (for some operations) to ignore the markup, e.g. markup within a word. Some markup, however, can be extremely useful during tokenization.[3] Font information can help decide whether or not to combine tokens (e.g. a company name in bold font) or split them (e.g. a footnote reference in smaller font and superscripted), layout information can be a great help in pinpointing sentence boundaries and there are many more examples. However, no systematic investigation of such exploitation of markup has been made so far, mostly due to the already mentioned variety in typesetting conventions. As a result, we too will neglect markup in the rest of this chapter.[4]

The tokenization process depends strongly on the type of text which is being processed, so that an analysis of the tokenization problems in the specific type of text must be done (with ways of checking the separator/non separator status of characters). Learning from the corpus being tokenized is also very important as we demonstrate in the sections below.

[3] And during later stages of tagging as well, e.g. the convention to italicize foreign words and technical terms in English text can help identify them.

[4] An internet search for *dehtml* will lead to pages such as http://www.math.utah.edu/~beebe/dehtml which provide code for eliminating HTML markup from files.