Jon MILLS, University of Luton

Lexicon Based Critical Tokenisation: An Algorithm

Abstract

In some languages, spaces and punctuation marks are used to delimit word boundaries. This is the case with Cornish. However there is considerable inconsistency of segmentation to be found within the Corpus of Cornish. The individual texts that make up this corpus are not even internally consistent. The first stage in lemmatising the Corpus of Cornish, therefore, involves the resegmentation of the corpus into tokens. The whole notion of what is considered to be a word has to be examined. A method for the logical representation of segmentation into tokens is proposed in this paper. The existing segmentation of the Corpus of Cornish, as indicated by spaces in the text, is abandoned and an algorithm for dictionary based critical tokenisation of the corpus is proposed.

Keywords: lexicography, tokenisation, lemmatisation, segmentation, Cornish

1. Introduction

We tend to take for granted the unit that we loosely call a word. In languages such as English, words are delimited by spaces and punctuation marks. The task of tokenisation may, consequently, seem trivial. Some languages such as Chinese and Japanese, however, do not have explicit token delimiters. In such languages a sentence is a string of characters with no English blank space equivalent. Languages with conjunctive orthography such as Finnish and Shona have few word delimiters. Such orthographic practices make tokenisation a serious problem for Natural Language Processing in these languages. The problem also exists when working from the medieval manuscripts which are the source of the Corpus of Cornish.

Sentence word tokenisation is the process of converting a sentence into a string of words. Because most Natural Language Processing applications take words as basic processing units, it is a common stage in the preprocessing of a text. Given a string of characters generated by removing blank spaces that function as word delimiters from a natural language sentence, a natural problem is to discover a way to restore these blank spaces (Guo Jin 1996:1).

2. Types, Tokens and Tones

The token-type distinction originates with the philosopher Charles Peirce. However Peirce discriminates between tokens, types and tones. These three emerge as manifestations of three modes of reality: existential reality, the reality of law, and the reality of qualities (Winder 1996).

Tokens equate with existential reality. In other words, a token belongs to the existential world. When we say that a sign is a token, we are indicating what is absolutely unique in its occurrence, its place in time and space (Winder 1996). With regard to a text corpus, tokens are, thus, simply text positions.

Types equate with the reality of law. A type is a sign that represents the law-like generality of a class. Unlike tokens, we cannot point to a type anymore than we can point to the law of gravity. Although types are real, they do not belong to the existential world in which pointing is possible (Winder 1996). With regard to a text, one can say that it has a vocabulary of X number of word types.

Tones equate with the reality of qualities. In any given investigation, there are certain perceptual units that cannot or will not be analysed. These are qualities. Although we may recognise the general form of a scrawl, we may not distinguish individual letters. We cognise some quality whether we recognise the form or not. In other words, whatever interpretation we may finally bring to something, our first impression has a value which is distinct from time and space and distinct from law. That value is tone. With regard to text, one has an awareness of more than one actually uses for reading purposes. For example a letter may be smeared and the spacing of the text may vary. Such qualities may be perceived but the reader can choose to ignore them. The tones of a text are, thus, defined as those qualities which one does indeed wish to consider as fundamental yet unanalysable in a given analysis. For example, usually alphanumeric characters are considered unanalysable. In other words, the reader does not analyse them into bars and curves, or distinguish them according to pitch in proportional spacing (Winder 1996).

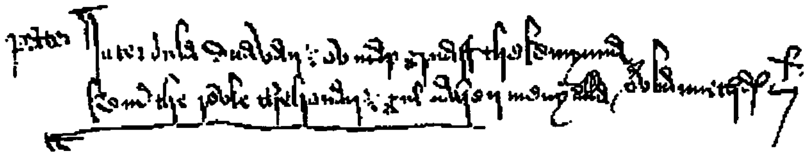

Figure 1

Figure 1 shows an extract from the 16th century Cornish miracle play, *Beunans Meriasek* (Ton 1504: 19). Below is a transcription and translation of the extract in Figure 1.

Inter dula du avan	*Between the hands of God above*
ov map gruaff the kemynna	*My son, I do commend thee*
kemmer the roule the honan	*Take thine own rule:*
gul nahen me ny alla	*Do aught else I cannot*
ov banneth dis	*My blessing to thee*

From this extract it can be seen that the two lines of handwriting in the original actually represent five lines of verse. That it is indeed verse is evident from the rhyming of alternate lines and the metre of seven syllables per line. The transcription shows three instances of the word *the*. These occurences represent three word tokens but only one word type. Figure 2 shows the tone of the first occurrence of *the*.

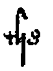

Figure 2

In order to read the original manuscript, it is necessary firstly to define the alphabet that is employed, and secondly to segment text into lines and words. Segmenting the text into words is known as word tokenisation.

3. Character Based Tokenisation

There are essentially two approaches to word tokenisation, character based tokenisation and lexicon based tokenisation. Character based tokenisation assumes that there are certain characters, such as the letters of the alphabet, that occur within words. It is similarly assumed that certain other characters, such as spaces and punctuation, occur between words. There can be, however, certain problems with these assumptions. In a language such as English, for example, some characters occur both within and between words. The hyphen occurs both within compounds and between parts of a sentence. The ' symbol is used both within words as an apostrophe and as a closing inverted comma. The space can occur within a single lexeme such as *New York*. The comma can occur within a number such as *1,000*. The fullstop can occur within a time such as *12.45 pm*. Conversely, in English, words are not always delimited by characters, for example *gimme* and *haven't*. Character based tokenisation does not provide a foolproof way of segmenting text into tokens. It does, however, act as a guide to the eye when reading a text. Character based tokenisation may be fine tuned by employing a lexicon of exceptions. This approach is a combination of character based tokenisation and lexicon based tokenisation. Whilst character based tokenisation works fairly well for some languages such as English, other languages, such as Chinese and Japanese do not have explicit word delimiters. In the extract from *Beunans Meriasek* in Figure 1, it can be seen that spaces between words are frequently omitted or not clear.

4. Lexicon Based Tokenisation

Lexicon based tokenisation is allied to Peirce's concepts of token, type and tone. The very notion of type implies a lexicon. The lexicon is, thus, derived from the text. Whilst, conversely and simultaneously, knowledge of the lexicon enables interpretation of the text. Figure 3 illustrates an algorithm for lexicon based tokenisation.

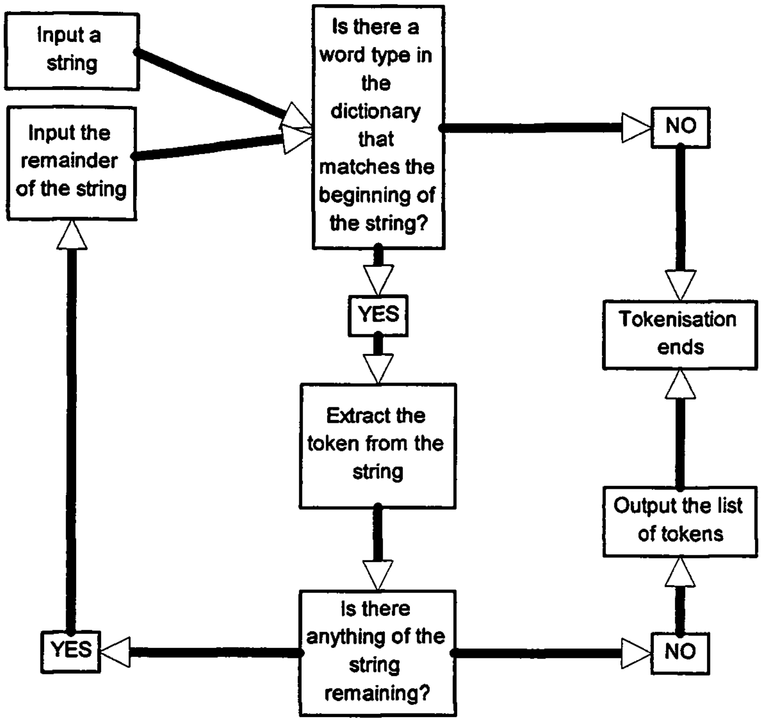
Input a string
Input the remainder of the string
Is there a word type in the dictionary that matches the beginning of the string?
NO
YES
Tokenisation ends
Extract the token from the string
Output the list of tokens
Is there anything of the string remaining?
YES
NO

Figure 3

<u>Dictionary</u>
a
bird
black
blackbird
peter
saw
Figure 4

Figure 4 represents a very simple dictionary to be used with the algorithm in Figure 3. Given the algorithm in Figure 3 and the dictionary in Figure 4, if we input the string,
"petersawablackbird",
the system will return the output,
[peter,saw,a,black,bird].
There are two problems with this algorithm. Firstly, if the string contains items which are not in the lexicon, tokenisation fails. Thus the string,
"marysawablackbird",

will not tokenise. Secondly, a given critical segment may contain one, or more than one, word type. Alternative readings are, thus, possible. The algorithm, however, finds only one solution. The order that items are listed in the dictionary determines the outcome. So since *bird* and *black* are both listed in the dictionary before *blackbird*, the system selects [black,bird] and [blackbird] is not chosen.

There are two types of ambiguity involved, combinatorial ambiguity and overlapping ambiguity (Guo Jin 1996). Combinatorial ambiguity refers to critical segments that consist of one or more than one word type. Figure 5 shows some examples of combinatorial ambiguity.

Blackbird	black bird
Below	be low
Today	to day

Figure 5

Overlapping ambiguity may be defined as follows. Given a character string *ABC*, if the substrings *A*, *AB*, *BC* and *C* are all words in dictionary, the string *ABC* is said to have overlapping ambiguity, as there exists an overlap between the word *AB* and the word *BC*. Thus, in English, the string "fundsand" can be tokenised as either "funds and" or "fund sand". Similarly the string "toplace" can be tokenised as either "to place" or "top lace" (Guo Jin 1996: 3).

The algorithm in Figure 3 was implemented in the Prolog programming language. Prolog is a declarative programming language which uses unification to find all the solutions to a goal. If we run our Figure 3 algorithm in Prolog by setting the goal

```
        ?- tokenise("petersawablackbird",X).
```

then the system will find all the possible tokenisations of the string, thus

```
        X = [peter,saw,a,black,bird];
        X = [peter,saw,a,blackbird].
```

However, tokenisation will still fail if the string contains a word that is not in the dictionary. A complete dictionary is, therefore, needed. A complete dictionary is one in which all valid words are included and there are no unknown words. Guo Jin (1996: 2-3) points out that although a linguistically complete dictionary is never within reach, an operationally complete dictionary is trivial to compile. One simple way is to add all the characters in the alphabet to the dictionary as single character words. These then spell out unknown words which can be glued back at a later stage. Guo Jin's solution thus combines lexicon based tokenisation and character based tokenisation.

Figure 6 shows all the possible solutions to the string when all the characters in the alphabet have been added to the dictionary as single character words. The solutions have been sorted so that the solutions with the least number of tokens occur at the top of the list. **blackbird** is, thus, listed before **black,bird**, and **black,bird** is listed before **black,b,i,r,d**. This manner of sorting I have named 'Longest First Tokenisation'. Its purpose is to present what are possibly the more plausible solutions first.

```
?- tokenize("petersawablackbird").
[peter,saw,a,blackbird]
[peter,saw,a,black,bird]
[peter,saw,a,black,b,i,r,d]
[peter,saw,a,b,l,a,c,k,bird]
[peter,saw,a,b,l,a,c,k,b,i,r,d]
[peter,s,a,w,a,blackbird]
[peter,s,a,w,a,black,bird]
[peter,s,a,w,a,black,b,i,r,d]
[peter,s,a,w,a,b,l,a,c,k,bird]
[peter,s,a,w,a,b,l,a,c,k,b,i,r,d]
[p,e,t,e,r,saw,a,blackbird]
[p,e,t,e,r,saw,a,black,bird]
[p,e,t,e,r,saw,a,black,b,i,r,d]
[p,e,t,e,r,saw,a,b,l,a,c,k,bird]
[p,e,t,e,r,saw,a,b,l,a,c,k,b,i,r,d]
[p,e,t,e,r,s,a,w,a,blackbird]
[p,e,t,e,r,s,a,w,a,black,bird]
[p,e,t,e,r,s,a,w,a,black,b,i,r,d]
[p,e,t,e,r,s,a,w,a,b,l,a,c,k,bird]
```

Figure 6

5. Critical Tokenisation

It is necessary to represent a text that has been tokenised in such a way that both individual tokens and individual types can be identified. Lager (1996 : 34) describes how segments and points form a basis for tokenisation. A critical point in a text is that which delimits two adjacent segments. Conversely a critical segment of text is that which is delimited by two points in the text. Segments are, thus, located in time and space. Segments can be observed, pointed at and given unique names. Segments and strings are not the same. Segments are instances of strings. Two segments may be instances of the same string. Segments are, thus, tokens, whereas strings are types. Figure 7 shows how critical tokenisation can be applied to the first line of the fragment of the Cornish miracle play shown in Figure 1.

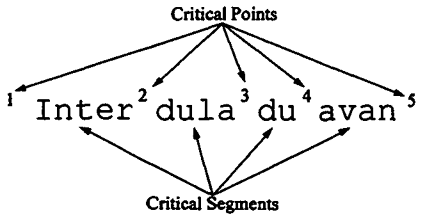

Figure 7

Figure 8 shows how the principles of critical tokenisation may be applied to the fragment of the Cornish miracle play shown in Figure 1 in order to create a Prolog database. The text is arranged vertically with each token on a separate line. Each token is represented by a three place predicate. The first and second arguments are the critical points which define the segment covered by the token. The third argument is the word type of which the token is an instance.

```
token( 1,  2, inter).
token( 2,  3, dula).
token( 3,  4, du).
token( 4,  5, avan).
token( 5,  6, ov).
token( 6,  7, map).
token( 7,  8, gruaff).
token( 8,  9, the).
token( 9, 10, kemynna).
token(10, 11, kemmer).
token(12, 13, the).
token(13, 14, roule).
token(14, 15, the).
token(15, 16, honan).
token(16, 17, gul).
token(17, 18, nahen).
token(18, 19, me).
token(19, 20, ny).
token(21, 22, alla).
token(23, 24, ov).
token(24, 25, banneth).
token(25, 26, dis).
```

Figure 8

With the text in the form of a database it is possible to conduct various types of search. For example, if one wants to know what word type is found between critical points 2 and 3, one sets the goal,

```
?- token(2,3,T).
```

The system returns,

```
T = dula
```

Conversely, if one wants to know in what text positions the word type *dula* is found, one sets the goal,

```
?- token(X,Y,dula).
```

The system returns,

```
X = 2
Y = 3
```

If one wants the phrase found between critical points 1 and 5, one sets the goal,

```
?- get_segment(1,5,Segment).
```

The system returns,

```
Segment = [inter,dula,du,avan].
```

If one wants an alphabetical list of all the word types attested in the text, one sets the goal,

```
?-setof(T,X^Y^token(X,Y,T),List).
```

The system returns,

```
List = [avan,du,dula,gruaff,inter,kemmer,kemynna,map,ov,the]
```

The process is, thus, seen to be founded on a type of quotation, which we call an 'attestation'. In this manner, routines can be written in Prolog to produce frequency wordlists, concordances, lists of collocations and other types of data.

6. References

Guo, Jin (1996) "An Efficient and Complete Algorithm for Unambiguous Word Boundary Identification" Available: http://sunzi.iss.nus.sg:1996/guojin/papers/acbci/acbci.rtf.

Grefenstette, Gr. & P. Tapanainen (1994) "What is a Word, What is a Sentence? Problems of Tokenization" Available: grefen@xerox.fr, tapanai@xerox.fr.

Lager, T. (1995) *A Logical Approach to Computational Corpus Linguistics* Gothenburg Monographs in Linguistics 14. Gothenburg: Department of Linguistics, Göteborg University.

Ton, R. (1504) *Beunans Meriasek.* Ms. Peniarth 105. National Library of Wales.

Winder, W. (1996) "Reading the Text's Mind: Lemmatisation and Interpretation from a Peircean Perspective" Available http://www.chass.utoronto.ca:8080/epc/chwp/winder/.