Specifying orthography: harmonization, tokenization and transliteration

Michael Cysouw

2014-11-23

1 Introduction

Given any collection of linguistic strings, there are various issues that often arise in using these linguistic strings in the computational processing of such data. This vignette will give a short practical introduction to the solutions offered in the `qlcTokenize` package. For a full theoretical discussion of all issues involved, see Moran & Cysouw (forthcoming).

All proposals made here (and in the paper by Moran & Cysouw) are crucially rooted in the structure and technologies developed over the last few decades by the Unicode Consortion. Specifically the implementation as provided by the UCI and their porting to R in the `stringi` package are crucial for the functions described here. One might even question, whether there is any need for the functions in this package, and whether the functionality of `stringi` is not already sufficient. We see our additions as high-level functionality that (hopefully) is easily enough to be applied to also allow non-technically-inclined linguists to use it.

Specifically, we offer an approach to document *tailorder grapheme clusters* (as they are called by the Unicode consortium). To deal consistenly with such clusters, the official Unicode route would be to produce *Unicode Local Descriptions*, which are overly complex for the use-cases that we have in mind. In general, our goal is to allow for quick and easy processing, which can be used for dozens (or even hundreds) of different languages/orthographies without becoming a life-long project.

We see various use-cases for the `qlcTokenize` package, e.g.:

- checking consistency of the orthographic represenation in some data;
- tokenization of the orthography into functional units ("graphemes"), which is highly useful in language comparison (e.g. character alignment);
- checking for consistent application of a pre-defined orthography structure (e.g. the IPA);
- transliteration of orthography to another orthographic representation, specifically in cases in which the transliteration is geared towards reducing orthographic complexity (e.g. sound classes).

In general, our solutions will not be practical for ideosyncratic orthographies like English or French, nor for chracter-based orthographies like Chinese or Japanese, but is mostly geared towards practical orthographies as used in the hundreds (thousands) of other languages in the world.

2 Installing the package

The current alpha-version of the package `qlcTokenize` is not yet available on CRAN (*Comprehensive R Archive Network*) for easy download and application. If you haven't done so already, please install the package `devtools` and then install the package `qlcTokenize` directly from github.

```
# install devtools from CRAN
install.packages("devtools")
# install qlcTokenize from github using devtools
devtools::install_github("cysouw/qlcTokenize")
# load qlcTokenize package
```

```
library(qlcTokenize)
# access help files of the package
help(qlcTokenize)
```

3 Orthography Profiles

The basic object in `qlcTokenize` is the *Orthography Profile*. This is basically just a simple tab-separated file listing all (tailored) graphemes in some data. An orthography profile can be easily made by using `write.orthography.profile`. The result of this function is an R-dataframe, but it can also be directly written to a file by using the option `file = path/filename`.

```
test <- "hállo hállo"
```

```
write.orthography.profile(test)
```

graphemes	replacements	frequency	codepoints	names
		1	U+0020	SPACE
á	á	1	U+00E1	LATIN SMALL LETTER A WITH ACUTE
á	á	1	U+0061, U+0301	LATIN SMALL LETTER A, COMBINING ACUTE ACCENT
h	h	2	U+0068	LATIN SMALL LETTER H
l	l	4	U+006C	LATIN SMALL LETTER L
o	o	1	U+006F	LATIN SMALL LETTER O
o	o	1	U+043E	CYRILLIC SMALL LETTER O

There are a few interesting aspects in this orthography profile.

- First note that spaces are included in the orthography profile. Space is just treated as any other character in this bare-bones function.
- Second, note that there are two different “o” characters. Looking at the Unicode codepoints and names it becomes clear that the one is a latin letter and the other a cyrillic letter. On most computer screens/fonts these symbols look completely identical, so it is actually easy for such a thing to happen (e.g. when writing with a russian keyboard-setting you might type a cyrillic “o”, but when copy-pasting something from some other source, you might end up with a latin “o”). The effect is that some words might look identical, but that they are not identical for the computer

```
# the differenec between various "o" characters is mostly invisible on screen
"o" == "o"  # these are the same "o" characters, so this statement in true
```

```
## [1] TRUE
```

```
"o" == "o" # this is one latin and and cyrillic "o" character, so this statement is false
```

```
## [1] FALSE
```

- Third, there are two different “á” characters, one being composed of two elements (the small letter a with a separate combining acute accent), the second being a single “precomposed” element (called “small letter a with acute”). The same problem as with the “o” occurs here: they look identical, but they are not (always) identical to

the computer. For this second problem there is an official Unicode solution (called 'normalisation', more on that below). It might even happen that when you just copy-paste the above test-string into your own R-console, that the problem automagically vanishes (because the clipboard might automatically do so-called NFC-normalisation).

- By default, this function lists all the Unicode codepoints and names. If you don't want them, add the option `info = FALSE`.
- By default, this functions adds a column "replacements" which will be used for transliteration later. If you don't want this columns, add the option `replacements = FALSE`
- Finally, note that the function also accepts vectors of strings:

```
test <- c("this thing", "is", "a", "vector", "with", "many", "strings")
```

```
write.orthography.profile(test)
```

graphemes	replacements	frequency	codepoints	names
		1	U+0020	SPACE
a	a	2	U+0061	LATIN SMALL LETTER A
c	c	1	U+0063	LATIN SMALL LETTER C
e	e	1	U+0065	LATIN SMALL LETTER E
g	g	2	U+0067	LATIN SMALL LETTER G
h	h	3	U+0068	LATIN SMALL LETTER H
i	i	5	U+0069	LATIN SMALL LETTER I
m	m	1	U+006D	LATIN SMALL LETTER M
n	n	3	U+006E	LATIN SMALL LETTER N
o	o	1	U+006F	LATIN SMALL LETTER O
r	r	2	U+0072	LATIN SMALL LETTER R
s	s	4	U+0073	LATIN SMALL LETTER S
t	t	5	U+0074	LATIN SMALL LETTER T
v	v	1	U+0076	LATIN SMALL LETTER V
w	w	1	U+0077	LATIN SMALL LETTER W
y	y	1	U+0079	LATIN SMALL LETTER Y

Normally, you won't type your data directly into R, but load the data from some file with functions like `scan` or `read.table`, and then perform `write.orthography.profile` on the data. Given the information as provided by the orthography profile, you might then want to go back to the original file and correct the inconsistencies, and then check again to see if everything is consistent now.

There is also a corresponding function `read.orthography.profile` in case you have a profile made yourself, or got a profile from somebody else. However, in most practical situations, you will only use both these low-level read/write-functions as part of a more powerful function called `tokenize` that will be decribed next.

4 Tokenization

In most cases you will probably want to use the function `tokenize`. Besides creating orthography profiles, it will also check orthography profiles against new data (and give warnings if there is something), it will separate the input strings into graphemes, and even perform transliteration. Let's run through a typical workflow using `tokenize`.

Given some data in a specific orthography, you can call `tokenize` on the data to create an initial orthography profile (just like with `write.orthography.profile` discussed above, though there are a few small differences. For example: spaces are not included by default in the profile, and spaces in the original are replaced by hashes).

The output of `tokenize` always is a list of three elements: `$strings`, `$orthography.profile`, and `$warnings`. The second element in the list `$orthography.profile` is the table we already encountered above. The first element `$strings`

is a table with the original strings, and the tokenization into graphemes as specified by the orthography profile (which in the case below was automatically produced, so there is nothing strange happening here, just a splitting into letters). The `$warnings` are just empty at this stage, but it will contain information about strings that cannot be tokenized with a pre-established profile.

```
tokenize(test)
```

```
## $strings
##    originals           tokenized
## 1 this thing t h i s # t h i n g
## 2         is                 i s
## 3          a                   a
## 4     vector         v e c t o r
## 5       with             w i t h
## 6       many             m a n y
## 7    strings       s t r i n g s
##
## $orthography.profile
##    graphemes replacements frequency codepoints                names
## 1          a            a         2     U+0061 LATIN SMALL LETTER A
## 2          c            c         1     U+0063 LATIN SMALL LETTER C
## 3          e            e         1     U+0065 LATIN SMALL LETTER E
## 4          g            g         2     U+0067 LATIN SMALL LETTER G
## 5          h            h         3     U+0068 LATIN SMALL LETTER H
## 6          i            i         5     U+0069 LATIN SMALL LETTER I
## 7          m            m         1     U+006D LATIN SMALL LETTER M
## 8          n            n         3     U+006E LATIN SMALL LETTER N
## 9          o            o         1     U+006F LATIN SMALL LETTER O
## 10         r            r         2     U+0072 LATIN SMALL LETTER R
## 11         s            s         4     U+0073 LATIN SMALL LETTER S
## 12         t            t         5     U+0074 LATIN SMALL LETTER T
## 13         v            v         1     U+0076 LATIN SMALL LETTER V
## 14         w            w         1     U+0077 LATIN SMALL LETTER W
## 15         y            y         1     U+0079 LATIN SMALL LETTER Y
##
## $warnings
## NULL
```

Now, you can work further with this profile inside R, but we find it easier to write the results to files, then correct/change these files, and use R again to process the data again. In this vignette we will not start writing anything to your disk (so the following commands will not be executed), but you might try something like the following:

```
dir.create("~/Desktop/tokenize")
setwd("~/Desktop/tokenize")
tokenize(test, file="test")
```

We are going to add two new "tailored grapheme clusters" to the profile: open the file "test.prf" (in the folder "tokenize" on your Desktop) with a text editor like Textmate, Textwrangler or Notepad++ (don't use Microsoft Word!!!). First, add a new line with only "th" on it and, second, add another line with only "ng" on it. The file will then roughly look like this:

graphemes	replacements	frequency	codepoints	names
a	a	2	U+0061	LATIN SMALL LETTER A

graphemes	replacements	frequency	codepoints	names
c	c	1	U+0063	LATIN SMALL LETTER C
e	e	1	U+0065	LATIN SMALL LETTER E
g	g	2	U+0067	LATIN SMALL LETTER G
h	h	3	U+0068	LATIN SMALL LETTER H
i	i	5	U+0069	LATIN SMALL LETTER I
m	m	1	U+006D	LATIN SMALL LETTER M
n	n	3	U+006E	LATIN SMALL LETTER N
o	o	1	U+006F	LATIN SMALL LETTER O
r	r	2	U+0072	LATIN SMALL LETTER R
s	s	4	U+0073	LATIN SMALL LETTER S
t	t	5	U+0074	LATIN SMALL LETTER T
v	v	1	U+0076	LATIN SMALL LETTER V
w	w	1	U+0077	LATIN SMALL LETTER W
y	y	1	U+0079	LATIN SMALL LETTER Y
th				
ng				

Now try to use this this profile with the function `tokenize`. Note that you will get a different tokenization of the strings ("th" and "ng" are now treated as a complex grapheme) and you will also obtain an updated orthography profile, which you could also immediately use to overwrite the existing profile on your disk.

```
tokenize(test, orthography.profile = "test")

# with overwriting of the existing profile:
# tokenize(test, orthography.profile = "test", file = "test")

# note that you can abbreviate this in R:
# tokenize(test, o = "test", f = "test")
```

```
## $strings
##    originals        tokenized
## 1 this thing th i s # th i ng
## 2         is              i s
## 3          a                a
## 4     vector      v e c t o r
## 5       with           w i th
## 6       many          m a n y
## 7    strings     s t r i ng s
##
## $orthography.profile
##    graphemes replacements frequency     codepoints
## 1          a            a         2         U+0061
## 2          c            c         1         U+0063
## 3          e            e         1         U+0065
## 4          i            i         5         U+0069
## 5          m            m         1         U+006D
## 6          n            n         1         U+006E
## 7         ng           ng         2 U+006E, U+0067
## 8          o            o         1         U+006F
## 9          r            r         2         U+0072
## 10         s            s         4         U+0073
```

```
## 11         t            t         2         U+0074
## 12        th           th         3 U+0074, U+0068
## 13         v            v         1         U+0076
## 14         w            w         1         U+0077
## 15         y            y         1         U+0079
##                                        names
## 1                        LATIN SMALL LETTER A
## 2                        LATIN SMALL LETTER C
## 3                        LATIN SMALL LETTER E
## 4                        LATIN SMALL LETTER I
## 5                        LATIN SMALL LETTER M
## 6                        LATIN SMALL LETTER N
## 7  LATIN SMALL LETTER N, LATIN SMALL LETTER G
## 8                        LATIN SMALL LETTER O
## 9                        LATIN SMALL LETTER R
## 10                       LATIN SMALL LETTER S
## 11                       LATIN SMALL LETTER T
## 12 LATIN SMALL LETTER T, LATIN SMALL LETTER H
## 13                       LATIN SMALL LETTER V
## 14                       LATIN SMALL LETTER W
## 15                       LATIN SMALL LETTER Y
##
## $warnings
## NULL
```

Now that we have an orthography profile, we can use this orthography profile on other data, using the profile to produce a tokenization, and at the same time checking the data for any strings that do not appear in the profile (which might be errors in the data). Note that the following will give a warning, but it will still go through and give some output. The all symbols that were not in the orthography profile are simply separated according to unicode grapheme definitions, a new orthogrphy profile explicitly for this dataset is made, and the problematic string are summarised in the warnings of the output, linked to the original strings in which they occured. In this way it is easy to find the problems in the data.

```
tokenize(c("think", "thin", "both"), o = "test")
```

```
## Warning:
## The character(s):
##  k b
## are found in the input data, but are not in the orthography profile.
## Check output$warnings for a table with all problematic strings.
```

```
## $strings
##   originals tokenized
## 1     think  th i n k
## 2      thin    th i n
## 3      both    b o th
##
## $orthography.profile
##   graphemes replacements frequency     codepoints
## 1         b            b         1         U+0062
## 2         i            i         2         U+0069
## 3         k            k         1         U+006B
## 4         n            n         2         U+006E
## 5         o            o         1         U+006F
```

```
## 6        th           th         3 U+0074, U+0068
##                                        names
## 1                       LATIN SMALL LETTER B
## 2                       LATIN SMALL LETTER I
## 3                       LATIN SMALL LETTER K
## 4                       LATIN SMALL LETTER N
## 5                       LATIN SMALL LETTER O
## 6 LATIN SMALL LETTER T, LATIN SMALL LETTER H
##
## $warnings
##   original strings unmatched parts
## 1 "think"          "k"
## 3 "both"           "b"
```

5 Rules

There are various situations in which just a table with graphemes and grapheme clusters is not sufficient to get the right tokenization. To get the correct result, we offer the possibility to add some extra rules to be applied after the table with graphemes has been applied. Note that in many orthography systems there are situations that can still not be solved by adding rules. The underlying problem is that in some cases the proper tokenization depends on the morphological structure of the word. For example, in German it is impossible to decide (just on the basis of the strings of characters) that *Flaschen* ("bottles", morphologically *Flasche-n*) should be tokenized as "F l a sch e n" with a grapheme cluster "sch", while *Bläschen* ("small blister", morphologically *Bläs-chen*) should be tokenized as "B l ä s ch e n", without a grapheme cluster "sch". In such cases, the only solution is to list individual cases as 'rules'.

The basic idea of the rules is the following: tokenization will initially prefer to separate longer grapheme clusters (i.e. when both "sch" and "ch" are specified in the profile, then "sch" will be attempted first). The rules now have to specify all situations in which this basic "longest-first" tokenization leads to the wrong results. In practice, the rules work like a correction: whatever is tokenized wrongly can be changed by a rule, which assumes the wrong tokenization already has happened. Using the German example from above, *Bläschen* will be wrongly tokenized as "B l ä sch e n", so we add a 'rule' that changes "B l ä sch e n" into "B l ä s ch e n".

Rules simply consist of a two-column (tab-separated) file with the matching condition in the first column and the replacement in the second column (assuming regular expression syntax, as internally the function `gsub` will be used). The rules will be applied from top to bottom, so please watch out for any feeding/bleeding situations in which a rule influences the applicability of another rule!

In detail, tokenization thus works as follows:

- First, go through all graphemes in the orthography profile *ordered by size of the graphemes*, i.e. larger grapheme clusters will be tokenized first. The size of the grapheme cluster is measured in number of unicode codepoints.
- Equally-sized grapheme clusters are applied in the order as they appear in the orthography profile. For example, a string "abc" can be split into "ab c" or "a bc" depending on which bigraph "ab" or "bc" appears first in the orthography profile
- Then the rules are applied (in the order as provided in the file) to 'correct' the first-pass tokenization
- Only then the tokenized strings are possibly transliterated (see below)

The file with the rules should be in the same directory as the orthography profile and have the same name as the file with the orthography profile, but it should use the suffix ".rules" instead of ".prf". So, when we add the following file to our working directory `~/Desktop/tokenize`, then it will tokenize "rathome" not with a "th".

```
setwd("~/Desktop/tokenize")
cat("r a th o m e\tr a t h o m e\n", file = "~/Desktop/test.rules")
tokenize("rathome", o =  test)
```

```
## $strings
##   originals     tokenized
## 1   rathome r a t h o m e
##
## $orthography.profile
##   graphemes replacements frequency codepoints               names
## 1         a            a         1     U+0061 LATIN SMALL LETTER A
## 2         e            e         1     U+0065 LATIN SMALL LETTER E
## 3         h            h         1     U+0068 LATIN SMALL LETTER H
## 4         m            m         1     U+006D LATIN SMALL LETTER M
## 5         o            o         1     U+006F LATIN SMALL LETTER O
## 6         r            r         1     U+0072 LATIN SMALL LETTER R
## 7         t            t         1     U+0074 LATIN SMALL LETTER T
##
## $warnings
## NULL
```

6 Transliteration

After tokenization (possibly includinng the usage of rules), the resulting tokenized string can then be transliterated into a different orthographic representation by using the option `replace = TRUE`. Then the grapheme as specified in the column specified at the option `replacements` are used (by default this columns is also called "replacements", but other names can be used, and one orthography profile can include multiple replacement columns).

Note that to achieve contextually determined replacements (e.g. in Italian becomes /k/ except before , the it becomes /tʃ/), all combinations will have to specified in the orthogaphy profile, as there is currently no proviso for rules of transliteration. However, we expect that most contextually determined transliterations can be easily specified in a few written down tailored grapheme clusters, e.g. add

graphemes	replacements
c	k
ci	tʃi
ce	tʃe