Coptic Transliteration Tool

Michael Shehata[1]

Montclair State University

[1]shehatam3@montclair.edu

Introduction

Coptic Language is the final phase of the ancient Egyptian Language and contains borrowed words from Greek and other Semitic languages. The Greek alphabet was adopted in Coptic and seven letters were added from demotic Egyptian[1]. Coptic is the only phase of Egyptian that was written in a clear pronunciation way for modern scholars to dissect. (Funk & Wagnalls New World Encyclopedia, 2018). Coptic is a dead language, however, it is still preserved currently as the liturgical language of the Coptic church. Most research on Coptic is focused on pronunciation, which makes sense given the importance of the language in terms of researching more about Ancient Egypt. There is also research that is focused on the Phonological phenomenon of Coptic.

[1] Egyptian hieroglyphic writing of cursive form that was used in handwritten texts from the early 7th century BCE until the 5th century CE.

Objective

The vision of this project is to provide an easy, bulk transliteration service between Coptic and Latin script (perhaps bidirectional). The primary use case (Coptic --> Latin script) is to allow English speakers who do not read Coptic to follow along with church services by offering transliterated text with transparent pronunciations.

The transliteration script:

- Accepts a single text file as input
- Detects which text the input file is written in
- Transforms a copy of the input file into either Latin or Coptic script using rule-based character mappings (implemented in Python)
- Outputs the transformed copy of the input file (text file output format)

Pronunciation Rules

Coptic has very complex rules that govern different allophones of the same phoneme. Working with Radobice Fass (Speech Program Manager at Google), we put together a spreadsheet where there are pronunciation rules that are verified by senior readers/priests of the Coptic church community. This ensures having the most recent and up to date pronunciation currently used in everyday liturgical text. An example of some of these complex rules is the Veeta letter 'ⲃ'. The Veeta letter has two possible allophones [b] and [v] depending on the environment they appear in. Fig.1 shows the different environment where each of the two allophones appear.

[v]	as in **v**acation	before vowels (ⲁ, ⲟ, ⲱ, ⲓ, ⲏ, ⲉ) in most contexts before word-final ⲧ
[b]	as in **b**oy	before ⲣ before ⲥ word-final in word-final ⲃⲓⲙ

Fig.1 illustrating the rules governing each allophone environment

To be able to deal with this, the script first iterates over the Coptic letters and applies the exceptions in an order that avoids any conflicts, then executes a simple 1:1 mapping after all the exception rules execution has been exhausted. The order is declared in the script file as shown in Fig. 2.

```
#cdrewrite

cascade = pynini.optimize(rule_addwb_1@rule_addwb_2@rule_1@rule_2@rule_4@rule_5@
                          rule_6@rule_7@rule_8@rule_9@rule_10@rule_11@rule_12@
                          rule_13@rule_14@rule_18@rule_19@rule_20@rule_21@
                          rule_onetoone_1@rule_onetoone_2@rule_onetoone_3@
                          rule_onetoone_4@rule_onetoone_5@rule_onetoone_6@
                          rule_onetoone_7@rule_onetoone_8@rule_onetoone_9@
                          rule_onetoone_10@rule_onetoone_11@rule_onetoone_12@
                          rule_onetoone_13@rule_onetoone_14@rule_onetoone_15@
                          rule_onetoone_16@rule_onetoone_17@rule_onetoone_18@
                          rule_onetoone_19@rule_onetoone_20@rule_onetoone_21@
                          rule_onetoone_22@rule_onetoone_23@rule_onetoone_24@
                          rule_onetoone_25@rule_onetoone_26@rule_onetoone_27@
                          rule_onetoone_28@rule_onetoone_29@rule_onetoone_30@
                          rule_onetoone_31@rule_onetoone_32@rule_onetoone_33@
                          rule_onetoone_34@rule_onetoone_34@rule_removewb)
```

Fig.2 from the script file showing the execution order of the rules.

Method

Programming Language: Python3

Library: Pynini

Dependencies:

Pynini depends on:

- A standards-compliant C++ 11 compiler (GCC >= 4.8 or Clang >= 700)
- The most recent version of OpenFst (at the time of writing, 1.7.4) built with the far, pdt, mpdt, and script extensions (i.e., built with ./configure --enable-grm) and headers
- Python 2.7 or 3.6+ and headers

Script

The script is available through PyPI and the README and instructions are found in the Github repository for this project. Updates will be released as the script adds support for more pronunciation rules and will be announced on the Github page. The latest version will always be deployed on PyPI as it becomes available. Pynini is a Python extension module which allows the user to compile, optimize, and apply grammar rules. Rules can be compiled into weighted finite state transducers, pushdown transducers, or multi-pushdown transducers. For general information and a detailed tutorial. Pynini makes it possible to create grammar rules that execute in whatever desired order, this helps execute the exceptional specific rules that result in different allophones (see pronunciation rules section for examples). After all these pre-set rules finish executing, one-to-one execution then takes place.

Reference List

Coptic Language. (2018). *Funk & Wagnalls New World Encyclopedia*, 1; Retrieved from http://search.ebscohost.com.ezproxy.montclair.edu:2048/login.aspx?direct=true&db=funk&AN=co213800&site=eds-live&scope=site

K. Gorman. 2016. Pynini: A Python library for weighted finite-state grammar compilation. In Proc. ACL Workshop on Statistical NLP and Weighted Automata, 75-80.