Interactive Workshop Measuring Language Complexity, Freiburg 2019

**Locus of marking and dependency length in possessive noun phrases**

Description of files that together constitute the analysis 30 June 2019

Viljami Haakana and Kaius Sinnemäki

General linguistics, University of Helsinki

[Viljami.haakana@helsinki.fi](mailto:Viljami.haakana@helsinki.fi); [kaius.sinnemaki@helsinki.fi](mailto:kaius.sinnemaki@helsinki.fi)

**tag\_chart.txt**

This file contains instructions on how to identify possessive NPs from different languages.

Column 1: ID (from workshop organizers)

Column 2: name of the language (from workshop organizers)

Column 3: lemma of the equivalent of English ‘of’ in possessive NPs

Column 4: POS of said lemma

Column 5: syntactic function of said lemma

Column 6: POS of the dependent (i.e. the possessed, said lemma’s head word if it exists)

Column 7: morphological tag of said dependent

Column 8: syntactic function of said dependent

Column 9: POS of said dependent’s head (the possessor)

Column 10: morphological tag of said head

Column 11: class to which a construction belongs if *all* conditions (columns 3–10) are met

Columns 3–10 contain instructions on how to identify whether a single condition is met. A *condition*, such as on.NOUN (“on” comes from Finnish “is/has”);on.PROPN, is first split between semicolons and then checked if *any* sub-condition is met. (If so, that cell yields a positive result). A condition that contains ampersands, such as sis.Poss=Yes&sis.PronType=Prs&sis.!Reflex=Yes, will yield a positive only when *all* its sub-conditions are met (“sis” comes from Finnish “includes”).

Sub-conditions start with either on. (equals, from Finnish *on* ‘is’) or sis. (contains, from Finnish *sisältää* ‘contains’) which is then followed by the string of characters that tag must contain or be equal to. If on. or sis. is followed by an exclamation point, it inverts the truth value of said condition.

Examples:

on.NOUN;on.PROPN means ‘must be equal to NOUN **or** to PROPN’

sis.Poss=Yes&sis.!Reflex=Yes means ‘must contain Poss=Yes **and not** contain Reflex=Yes’

If columns 3–5 are empty, that construction has no adposition, particle or determinant equivalent to ‘of’.

For head\_exist constructions (head\_exist = possessive noun phrase which contains only the head but no syntactically separate dependent), columns 3–7 are empty. In these cases, column 8 tells what kind of a dependent (nmod or nmod:poss) the head word must not have in order to be classified as a head\_exist. This is *not* preceded by on. or sis..

**tagchart\_script.py**

This is the main script that will count the number of different kinds of possessive NPs from each UD data set, including the average dependency lengths and the number of words in each dataset. It will save the findings to a tab-separated file called language\_stats.txt, whose contents it will then read and generate the main results file, Sinnemäki.csv. For it to work, it will need the following files in the same folder:

conllu\_reader.py

language\_importer.py

csv\_functions.py

Finnish.py

Swedish.py

Vietnamese.py

Chinese.py

RomanianHYPHENRRT.py

Hebrew.py

RomanianHYPHENNonstandard.py

German.py

KoreanHYPHENGSD.py

KoreanHYPHENKaist.py

Russian.py

PolishHYPHENLFG.py

PolishHYPHENSZ.py

tag\_chart.txt

ud\_tabmodel.txt

The files colored in blue contain certain general functions. The red files contain a function called possessive which returns a dict of each possessive type in a sentence in that particular language or dataset. In order to find possessive structures from a conllu sentence, the script will first look if that language or material has its own script and if not, it will check for instructions found in tag\_chart.txt. The file ud\_tabmodel.txt contains information on which ID corresponds to which folder and which language name.

The script expects itself to be in the same folder as the folder UDtrack, which contains the datasets.

***Note on encoding and ‘\ufeff’***

Whenever you save a UTF-8 encoded file using Notepad, the file will always start with the character U+FEFF. Even if that character is not present, Notepad will be able to detect that a file is UTF-8 encoded, but the Windows search tool will not. That means, if you can see a segment of a UTF-8 encoded file in Windows search results, it will appear as if it was latin-1 encoded.

When *reading* a file, these scripts will remove the initial U+FEFF character if present (not from the file but the in-read string and subsequent objects created from that string). When *writing* files, they will *not* add the character in. However, some files that have not been generated with Python but rather copied and pasted from a spreadsheet, they may and probably do contain that character.

***Note on command line arguments***

None of these scripts use command line arguments. Instead, they expect the needed files to be named the way described here and the UD materials to be in the UDtrack subfolder. The scripts will not automatically change the initial working directory to the location of the scripts, so it is advised to change the working directory within the command line before running the scripts. Another possibility is to use the IDLE shell to run them, in which case the initial working directory will always be the folder where the script is located.

**ud\_tabmodel.txt**

This file contains the following information on each dataset:

Column 1: id (e.g. grc\_perseus)

Column 2: the name of the language (e.g. Ancient Greek)

Column 3: the folder where that dataset is located in (e.g. UD\_Ancient\_Greek-Perseus)

Column 4: the British pronunciation of the language name (e.g. ˈeɪnʃənt ˈɡɹiːk)

Python scripts will read this file in order to deal with the differences in names and IDs.

**analyser\_GUI\_material.py**

*(Dis section was transla’ed from Standard* ***Bri’ish*** *English to Cockney w[[1]](#footnote-1)iv de elp ov*[*dis*](https://lingojam.com/EnglishtoCockneySlang) *online transla’ah. Changes ave bin made manually usin de elp ov da Cockney Wikipedia page an’ some guesses ov likely translation errahs.)*

Dis scrip’ is a graphical user in’ahface da’ allows da usah to browse analysed sen’ences ov each da’ase’. De in’ahface is wri’n in IPA wiv a Bri’ish accen’ (Received pronunciation, no’ Cockney). Instructions:

1. Choose a da’ase’ usin da box on da top righ’.

2. En’er a sen’ence id (or a star’ ov a sen’ence id, or an emp’y string if you jus’ wan’ da firs’ sen’ence) an’ press “ˈsɜːtʃ”. You can also omi’ dis par’ an’ do par’ 3 instead.

3. Click “ˈsɜːtʃ baɪ ˈmɑːkɪŋ...” to find a sen’ence wiv a’ leas’ one possessive NP ov a cer’n type. In da’ window, de en’ry box da’ as a defaul’ “1” is de awdinal da’ tells which sen’ence ov da’ kind you wan’. You can check da checkbox “ˈbɹaʊz ˈəʊnli ˈðiːz” if you wan’ to browse only dose sen’ences da’ con’ain da’ kind ov markin.

4. Da sen’ences can be browsed usin da “ˈpɹiːvɪəs” an’ “ˈnɛkst” bu’ns.

5. If you wan’ to modify a sen’ence (e.g. change a tag an’ see ow da’ would work) you can do de editions an’ click “ˈænəˌlaɪz”.

6. If you en’er a sen’ence da’ is presen’ in da material an’ click “ˈænəˌlaɪz”, da browsah moves to da’ sen’ence (so da “ˈnɛkst” an’ “ˈpɹiːvɪəs” bu’ns will move you to da sen’ences around da sen’ence you en’ahd *[=entered]* instead ov da one previously searched for aw *[=or]* browsed to). Dis is only true if a sen’ence as bin searched faw usin a search bu’n.

Da file SurrogatePair.py is needed faw complex charac’ers, such as emojis, no’ to cause de in’ahface to crash aw trea’ dem as two charac’ahs (Python’z defaul’ GUI library, tkinter, doesn’ suppaw’ non-BMP Unicode charac’ahs.). Mos’ likely none ov da “conllu”s contain emojis, bu’ da file is included jus’ faw sure.

1. In order to doublecheck whether our script had picked up possessive noun phrases and not some other constructions, we created a user interface. Just for the fun of it we used IPA for the interface and its commands. We thought to include this in the submission as well, but it is sort of an appendix. [↑](#footnote-ref-1)