Matthew DeHass

6/24/2023

Treebank Search Tool Codebook

This is my tentative "codebook" for the Treebank Search Tool (hereafter **TST**), an XQuery module and script for searching the LDT and PROIEL treebanks at minimum.[1] The goal of the project is to understand how a feature of syntax varies between genre and register. I put "codebook" in quotes because this will be much more detailed than a finalized codebook should be, it is really a journal where I plan out certain processes in advance. First, I list the steps I need to take to make a fully functional querying system. Next, I will have a "journal" part where I work through the problems. Third, there will be some examples of what the code should look like and how the existing functions should be used. Fourth, there will be a guide for how comments should be structured. Fifth, there will be a list of goals which need to be implemented backwards over the code. This means there are a few steps to this process:

1) Find out how each process in querying the treebanks can be generalized to work with at least LDT and PROIEL; also figure out how it can be generalized to lemmatized texts.
2) Find words based on their pos, relation (i.e. the relation type), or lemma, and also produce a negative search for each of these things (i.e., return anything that is not punctuation, or not a noun)
3) Be able to take these results and find other words with a certain relationship (that is, child, parent, sibling, etc.) which are filtered based on the same criteria in 2). This search should return all the members of a relationship, so I can decide whether I want to look further at relationships of the heads or the dependents. The ability to exclude certain kinds of relationships should be baked into the 2) search algorithm.
4) Be able to nest the searches. The results of a search like 3) can go back as one part of another search like 3)
5) Have a function to find sentences which contain positive results to multiple different queries like that in 3) at the same time. If I want to find sentences where *hortari* is used with an *ut*-clause and an ablative absolute is also present, for whatever reason I would want to do something like that, it should be possible. This is not for **exploratory** work, but **confirmatory** work.

[1] David Bamman and Gregory Crane, "The Ancient Greek and Latin Dependency Treebanks," in *Language Technology for Cultural Heritage: Selected Papers from the LaTeCH Workshop Series*, ed. Caroline Sporleder, Antal van den Bosch, and Kalliopi Zervanou, Theory and Applications of Natural Language Processing (Berlin/Heidelberg: Springer-Verlag, 2011), 79–98 (citation requires verification); Dag T. T. Haug and Marius L. Jøhndal, "Creating a Parallel Treebank of the Old Indo-European Bible Translations," in *Proceedings of the Second Workshop on Language Technology for Cultural Heritage Data (LaTeCH 2008)*, ed. Caroline Sporleder and Kiril Ribarov, 2008, 27–34, http://www.lrec-conf.org/proceedings/lrec2008/workshops/W22_Proceedings.pdf#page=31; also note that the Late Latin Charter Treebank, and Corpus of the Epigraphy of the Italian Peninsula of the First Millenium are other possibilities (they are mentioned in "Treebanks and Corpora" in the Thesis Guide). However, I have not decided whether to include these yet.

6) For **exploratory** work, may be good to have a function which only returns the children of the PRED root (therefore, a general function for returning children, which can be used for this purpose)
7) There should be a function to print the raw results or put them in a CSV file. The CSV should contain
8) All of these need to be able to export to CSV. A few notes on the formatting:
    a. **Every** CSV output should have the sentence id, source document relative path, citation to location in the text as exact as possible, title, author, and URN. **There should be a function for reading and retrieving sentences from this list**, **and a function for exporting in treebank format.** These should be in the far left columns; since they are not necessary for text analysis, I want to easily be able to chop off the left 6 columns and put the data into some kind of software.
    b. The rest of the CSV output should vary by use case. If I am using the simple search in 2), it is as simple as all the info in the word's attributes (14 columns, since I want there to be one for each postag character). If a complex search (**this part is under construction!!!!**), it should have the forms of the two related words on one line, the relation and relationship of each, and also the word ID's, so they can be found with more info.
9) In line with 8), a function for exporting a series of word nodes in plain-text, as a treebank document, and as a CSV should be available.
10) Also have a function that can find the highest node in a sentence of a certain type (to cover situations where the PRED is not the highest because two main clauses are coordinated).

Also keep in mind that all code examples will be colored in "White, Background 1, Darker 50%" (a color at the far bottom left of the theme colors in my palette), such as the following:

```
declare %public function deh:sentence-lengths($docs as node()*) as item()*
{
  for $doc in $docs
    for $sentence in $doc//sentence
      let $words := $sentence/word[deh:check-punct(fn:string(@form)) ne true()]
        return <count sentenceid='{$sentence/fn:string(@id)}'>{fn:count($words)}</count>
};
```

Journal:

Issue 1: LDT and PROIEL Compatibility:

The tough decision is whether to leave each treebank in their own format or transfer both over to an independent format, so one function works for everything. Let us first look at the conversion options.

MODULE FORMATTING NOTES:

This section covers the rules for formatting the .xqm file for this project (titled agldt_search.xqm). I have currently not decided whether or not to have multiple .xqm files, as it is uncertain whether even XQuery will be necessary. I will either create a separate module for PROIEL's standard, or combine everything together (currently my preferred solution) and rename the files to something more helpful. As it stands, the AGLDT is the only treebank type this is compatible with.

At the top should be the relevant declarations, of course, such as the below:

```
xquery version "3.1";


module namespace deh = "https://www.youtube.com/channel/UCjpnvbQy_togZemPnQ_Gg9A";


import module namespace functx = "http://www.functx.com" at
"http://www.xqueryfunctions.com/xq/functx-1.0.1-doc.xq";
```

The Xquery version and module namespace's obligatory status should be obvious (the namespace gives a handle for us to import the module and reference the functions). We use several of the helpful FunctX functions here, so I have imported it in this .xqm file as well.

Sections of the file should be marked as so:

```
(:----------------deh:postag-andSearch AND DEPENDENCIES----------------------------:)

(: code :)

(:--------------------------END deh:postag-andSearch AND DEPENDENCIES-------------------------:)
```

A comment contains a line made up of a series of dashes (the number/length not being important). A description is placed in the middle, and it will start with END if it is marking the end of a section. Functions do not have to all be cordoned off into sections. Additionally, although I am currently using it to encapsulate important functions and their helpers/dependent functions, it does not have to be used that way. Just make sure the labels are clear and helpful.

FUNCTION LIST:

deh:mark-node:

- Takes an LDT node, and adds the document URI and sentence ID