The following assumes that the reader is familiar with the broader outline of my proposed project as set out in my Oxford application-documents, specifically the main research proposal and/or either of the two UKRI funding application forms submitted alongside it.

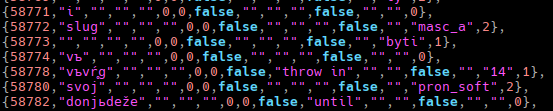
The stuff is largely unchanged since it was first done in 2021, when I was very new to programming, so as code it is pretty atrocious, but I've done a lot more foreign-language-focused programming since then so would easily be able to redo it more sensibly.

Why do I need an Autoreconstructor in the first place?

Since my goal is to enable quick, easy and comprehensive analysis of manuscript spellings, I need to produce LCS reconstructions of every form in the texts such that they will show up in searches for specific LCS phonemes or phoneme-class sequences (a crude search-mechanism for the already-reconstructed texts can still be found at [http://slavtexts.infinityfreeapp.com](http://slavtexts.infinityfreeapp.com/search.php)).

Doing this by hand isn’t really that much effort, but since the TOROT corpus contains high-quality lemmatisation and 10-place morphology-tagging, it’s more fun to take the list of TOROT lemmas, reconstruct for each the invariant part of the stem[s], label each one for inflectional-category, then just stick the correct inflections onto the end of the correct stems according to what the morphology tag tells us.

To hold this and a lot of other possibly relevant information (loanword-origin and source-language, whether something is a blatant non-integrated post-LCS loan, whether a differently-formed doublet of the stem occurs which TOROT has no separate lemmatisation for, etc.), I have a sort of ‘master-spreadsheet’ (<https://docs.google.com/spreadsheets/d/1qw75jk_oLbjohmdh_qlC6B1yb9xRhwqY>) which started life just as a list of all the TOROT Church-Slavonic lemmas. To fully reconstruct the Marianus and (TOROT’s part of) Euchologium Sinaiticum I just filtered the lemmas to only display those which occur in Mar. and Euch., then manually reconstructed the stems and added the various pieces of auxilliary information. The necessary parts of the spreadsheet then get converted into a C++ std::set data-structure of what I’ve called Lemma structs, and it’s from this std::set that the program retrieves the correct Lemma struct and inspects its stem-form and inflection-class fields to start producing an LCS form.[[1]](#footnote-0)



If you look at the third row you can see that the stem of byti is blank and its inflection-class is ‘byti’, because it essentially has its own whole paradigm. Indeed, the number of separate inflection-classes is pretty obscene for something which claims to be “automatic”: I counted 47 verb-classes and 40 nominal-classes (and I haven’t even added \*dǫti yet because it doesn’t occur in Mar. or Euch.), but there are actually more class-names than inflection-tables because, for example, all o- and a-stem nouns and all hard-stemmed adjectives get passed to the same list of 63 (3 numbers \* 7 cases \* 3 genders) endings, and similarly I reuse the pronoun \*jь’s table for the long-adjectives and participles.

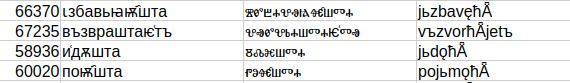
Case study: class I verbs

My inflection classes 11, 14 and 15 correspond to Diels (1963)’s class 1.1, 1.4 and 1.5, i.e. this is class I verbs which add their endings directly to a consonantal-stem with no intervening jot (they differ from classes 12 and 13, which are respectively nasal and liquid-stemmed, in that their 2nd and 3rd pers. singular bare aorists add \*-e rather than nothing: \*nes**e** vs \*prijьn > \*priję > (in OCS also with a -tъ 3rd sg. marker added) приѩ[тъ], \*umer > оумрѣ[тъ]).

How do I deal with morphological restructuring?

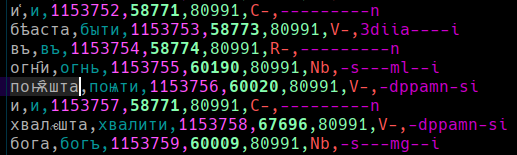
**Annotation-mistake detection**

Here we can see a random example of two bad autoreconstructions caused by incorrect TOROT lemmatisations in the Suprasliensis (ignore the Glagolitic):



In the first instance TOROT must simply have \*jьzbaviti rather than \*jьzbavĺǞti, which is understandable given that the imperfect tense of these class 4 verbs is indistinguishable from their secondary-imperfective class 5.4 counterparts (indeed that’s likely where this type of secondary-imperfective came from), but this is a present active participle so the lemmatisation should be that of the secondary-imperfective class 5.4 \*jьzbavĺǞti.

The second instance is a more extreme mistake, since we have ‘sing’ lemmatised as ‘grab’, as is clear from the context[[2]](#footnote-1) :



If we converted these erroneous reconstructions to the regular but unattested ‘normalised’ OCS system which is sometimes employed, i.e. избавѧща, поимѫща (a trivial task), then the “edit-distances” between the normalised-OCS forms and a cleaned-up version of the Suprasliensis forms should easily be large enough to pass whatever threshold we might set for alerting us of possible TOROT mistakes.

A potentially cool side effect of having such an Autoreconstructor is it could be combined with automatic lemmatisation and morphological-tagging (which Hanne can already do surprisingly effectively and which I think could possibly be improved further by more careful and targeted data-cleaning and potentially also neural-network approaches like the one described here [Morpheus reference]), to produce an LCS reconstruction straight from a raw text, potentially even a raw manuscript if Transkribus can be trained well enough (though that probably is pushing our luck).

A student should be able to point their phone at a past exam-paper or homework commentary-passage and get given back a bullet-point list of all the relevant phonological features in the text such that they can completely cheat; if I am successful in producing high-quality comprehensive phonological-annotation then all that would be necessary to achieve this is OCR good enough to locate the printed passage in my database, but it would be interesting to see if such cheating could also be enabled using nothing but the raw text.

The size of the corpus of relevant texts is small enough, though, that automated techniques are only ever envisaged as a starting point for manual correction: I don’t even trust Jagić’s editions and fully intend to read every single text I study (with the possible exception of Psal. Sin. which is a nightmare) cover-to-cover from manuscript-photographs before I sign anything off.

As a more computationally-competent but less linguistically-rigorous example of what is possible with databases and browser-technologies you can look at <http://57.128.170.157:5000/text_viewer>, which is a VPS-instance of an application to help serious language-learners learn vocab by reading and annotating texts. Many of my texts there are Danish, which as you’ll know is full of Germanic separable-verbs and multiword-idioms and identical forms that belong to different lemmas and multiple homomorphic lemmas, and it can deal with all of it in an efficient and scalable way, so problems such as how to display / encode phonological phenomena which span multiple lexical words (изд рѫкъі, Psal. съмѩ сѩ <\*sъmęsъ\_\_sę etc., clitics basically) will not be that hard to get around.

because those problems often point to significant events in the history of the language that you would never even think about

1. Compiling all this data directly into the C++ program (or using C++ at all for this task really) rather than reading it off of disk at runtime is an extremely stupid way of doing things and can cause problems with compilation. [↑](#footnote-ref-0)
2. Note also TOROT’s baffling sporadic replacement of Supr. ꙙ by the hooked Glagol. nasal ⱕ, which actually only occurs in Zogr. and Mar. (ⰳⱃⱔⰴⱕⰻ, ⱀⰵⱄⱕ etc.) and which Kortlandt (1979?) says is a remnant of \*y̨ corresponding to the NSl. \*-a endings on masc. Nsg. hard-stemmed class I PRAPs. Olander 2015:89-90 has more discussion, though beware his oversimplifying characterisation of it as “*ę* with a special sign” (Велчева and Trubetzkoy are crying out in pain). [↑](#footnote-ref-1)