

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

„Gothic Loanwords in Hungarian? Towards a computer-aided framework for borrowing detection“

verfasst von / submitted by

Viktor Martinović, BA MA

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der Philosophie (Dr. phil)

|  |  |
| --- | --- |
| Wien, 2020 / Vienna 2020 |  |
| Studienkennzahl lt. Studienblatt /  degree programme code as it appears on the student record sheet: | UA 792 382 |
| Dissertationsgebiet lt. Studienblatt /  field of study as it appears on the student record sheet: | Finno-Ugristik |
| Betreut von / Supervisor: | Univ.-Prof. Dr. Johanna Laakso |

Contents

[Chapter 1 Background 5](#_Toc40795645)

[1.1. Foreword 5](#_Toc40795646)

[1.2. Nonlinguistic aspects/ Working hypothesis 5](#_Toc40795647)

[1.3. Research Question 5](#_Toc40795648)

[1.4. Terminology 5](#_Toc40795649)

[1.5. Sources 5](#_Toc40795650)

[Chapter 2 Methods 6](#_Toc40795651)

[2.1. History of methods of loanword research 6](#_Toc40795652)

[2.2. State-of-the-art traditional methods of loanword research 6](#_Toc40795653)

[2.3. Computational models for loanword detection 7](#_Toc40795654)

[2.3.1. Computational models’ input 7](#_Toc40795655)

[2.3.2. Computational models’ semantic linking 7](#_Toc40795656)

[2.3.3. Computational models’ phonetic linking 7](#_Toc40795657)

[2.3.4. Problems of computational models from a traditional perspective 7](#_Toc40795658)

[2.4. A new framework for computer-aided borrowing detection 7](#_Toc40795659)

[2.4.1. Input (in principle, not concrete) 8](#_Toc40795660)

[2.4.2. Extracting Sound Change rules (in principle) 8](#_Toc40795661)

[2.4.3. Proto form generator 8](#_Toc40795662)

[2.4.4. NSE 8](#_Toc40795663)

[2.4.5. Creating substituions 8](#_Toc40795664)

[2.4.6. Slicing 8](#_Toc40795665)

[2.4.7. Semantic linking (in principle) 8](#_Toc40795666)

[2.4.8. Phonetic linking (in principle) 8](#_Toc40795667)

[2.5. English loanwords in Hungarian (Case study one) 9](#_Toc40795668)

[2.6. Applying current computer methods to find Gothic loans in Hungarian (Case study two) 9](#_Toc40795669)

[2.7. Applying the new framework to find Gothic loans in Hungarian (Case study three) 9](#_Toc40795670)

[2.7.1. General idea 9](#_Toc40795671)

[2.7.2. Getting the data 9](#_Toc40795672)

[2.7.3. Cleaning and editing the input 10](#_Toc40795673)

[2.7.4. Mapping and counting Sound Changes 11](#_Toc40795674)

[2.7.5. Generating Protoforms 12](#_Toc40795675)

[2.7.6. NSE 12](#_Toc40795676)

[2.7.7. Slicing 12](#_Toc40795677)

[2.7.8. Sound Substituion 13](#_Toc40795678)

[2.7.9. Slicing2 13](#_Toc40795679)

[2.7.10. Matching 13](#_Toc40795680)

[2.7.11. Semantic analysis 13](#_Toc40795681)

[2.7.12. Outlook 14](#_Toc40795682)

[2.8. Pseudo-Codes 14](#_Toc40795683)

[2.8.1. Uralonetparser1 14](#_Toc40795684)

[2.8.2. Uralonetparser2 14](#_Toc40795685)

[2.8.3. Zaicz extractor1 14](#_Toc40795686)

[2.8.4. Zaiczextractor2 14](#_Toc40795687)

[2.8.5. Wikilingparser 14](#_Toc40795688)

[2.8.6. Add links to Wikiling 14](#_Toc40795689)

[2.8.7. Hun2ipa 14](#_Toc40795690)

[2.8.8. Soundcounter 15](#_Toc40795691)

[2.8.9. Soundchangemapper 15](#_Toc40795692)

[2.8.10. Got2ipa 15](#_Toc40795693)

[2.8.11. Got data cleaner 15](#_Toc40795694)

[2.8.12. Substitutions 15](#_Toc40795695)

[2.8.13. Phonotactics 15](#_Toc40795696)

[2.8.14. Match 15](#_Toc40795697)

[2.9. Conclusion 15](#_Toc40795698)

[Chapter 3 Etymological Analyses 15](#_Toc40795699)

[Chapter 4 Outlook 15](#_Toc40795700)

[Chapter 5 Notes to myself 15](#_Toc40795701)

# Background

## Foreword

in this section I will briefly explain my motivation for this dissertation as well as sources for ideas and inspiration. The idea for this work came in 2015 in Paris while working on my Bachelor thesis titled *German loanwords in Hungarian*. There, I ask the question when speakers of Hungarian and speakers of German could have met the first time in history. I found out, slightly off topic, that Goths and Hungarians appear on the map of Europe in a similar time and space and that Gothic-Hungarian contacts could have predated the well-researched German-Hungarian ones. During my master thesis in 2018 I investigated this relationship more deeply. While I found out more about the historical connections, gaps in the methodology still remained. This work, among other things, aims to fill this gap by proposing a new framework for computer aided borrowing detection.

## Nonlinguistic aspects/ Working hypothesis

In this section I will Briefly touch upon-non linguistic aspects of this dissertaition. Even though I agree with Denis Sinor when he argues that

*“languages and not peoples are the proper subject of linguistics, and diachronic linguistics should deal with the history of one or several languages and not with the history of peoples who spoke them.” (The Uralic languages, Denis Sinor, 1988, p.18)*

arguments from other subfields such as archaeology, anthropology, or history can not be fully ignored, as they are crucial for contextualising this work.

## Research Question

Are there Gothic loanwords in Hungarian or not? The superordinate question whether there was *contact* between Hungarians and Goths is left out on purpose, as it is thematically outside the scope of this work. There have been several attempts to link nonlinguistic and linguistic data to gain deeper insights into the history of speaker-communities (quote1,2,3). These works raise a whole new set of complex questions (E.g. What is the definition of *speaker-community*? quote). Therefore, there is no space to deal with them in detail in this work. For possible conclusions for other scientific fields and societal impact see Chapter 4.

An important subquestion, that I am going to answer in Chapter 2, is: How to find loanwords?

## Terminology

Loanwordlayer – loanwords appear in clusters. These clusters are described as *stratigraphy* in their entirety.

False cognates – semanitcally linkable and phonetically similar but not linkable donor and loaned word (e.g…)

Gothic –

## Sources

/Breifly describe the main dictionaries and sources I am going to use

TESz, Ewung, Zaicz 2006, Wikiling, uralonet, UEW etc etc

# Methods

## History of methods of loanword research

Historical lexicology, as subfield of historical linguistics, can be divided into cognate and loanword research (quote). The often-quoted first etymolgy in Plato’s Socratic dialogue Cratylus. Already there, Socrates employs basically the same methods we use today: The starting point is a word in one’s own langue, which has no credible explanation to its origin. Then, the etymologist uses his knowledge about the vocabulary of one or more language and tries to find with the help of his intuition elements in this vocabulary base, that are phonetically and semantically linkable to the word of yet-unknown origin.

During the rise of historical-comparative linguistics in the 18th century (Sajnovics, William Jones quote) etymologists employed the same core concept found in Plato’s dialogue: They used their extensive personal (in a sence that they have to know everything by heart) knowledge of two or more languages (Sajnovics: Hungarian, Saami. Jones: Latin, Greek, Sanskrit), and used their intuition to find elements that are phonetically and semantically linkable. During this era it was discovered that phonetical changes happen regulary (quote). Today, 300 years later, not much has changed, regarding methodology (quote: Yiddish loans in polish, greek loans in coptic projects Berlin, Warsaw). Loanwords are found by so-called lexicographers: Scholars with extensive personal knowledge about the vocabulary of two or more languages, who use their intuition to find linkable words between languages (quote).

Since, however, the task of finding loanwords is highly systematic, it appears to be obvious that computeres could be helpful in this task. Nevertheless, computer-aided methods for borrowing detection are still in their infancy (List 2019), while significant steps have been made in developing a framework for automated cognate detection. (quote quote quote)

## State-of-the-art traditional methods of loanword research

/List all criteria when two words can be classified as donor and loanword.

Criteria for semantic linkability: In order for an etymology to be credible, the loaned and the donor word have to be linkable. Criteria for the linkabilitiy of semantics is described in 2.5.1. and for phonology in 2.5.2.

Criteria for semantic linkability (quote Oxford handbook). Borrowability (Haspelmath loanword database). Computational approaches: Princeton wordnet, nltk and gensim for python.

Criteria for phonetic linkability: Sound change laws and tendencies.

Input: Competing etymologies, multiple etymologies for one word, inner development, outside influences for inner developments, rejected etymologies, well established etymologies, semantic fields, time layers, biases,…

Output: Etymological dictionaries/databases.

Gothic words

Hungarian

words

Possible loans

Proto Forms

Sound Change

Sound

Substitution

comparison

## Computational models for loanword detection

/General outline of what has been done so far in automated borrowing detection. (linguarena.eu etc.) Detailed comments on those methods in the following chapters.

### Computational models’ input

### Computational models’ semantic linking

### Computational models’ phonetic linking

### Problems of computational models from a traditional perspective

Input: Concept lists: Ignore the biggest part of the vocabulary (longest concept list is only 1600 words long, quote). This is an extended version of the core vocabulary, where cogantes are found. For loanwords, however, it is common to appear on the periphery, in special socio-linguistic context, through word plays, jargon, thieves’ argot, or even in certain paradigms of otherwise non-loaned words (list examples, e.g. Hungarian *gyere* (Imperative ‘come’) could be a loaned paradigme of *jön* ‘to come’; Hungarian *gomolya* ‘certain type of goat cheese’ from Croatian *gomolj* (‘bundle’) with semantic development ‘bundle of cheese > certain type of goat cheese’. To my knowledge there is no wordlist that apriori contains a concept called ‘certain type of goat cheese’ or ‘Imperative of ‘to come’’. It is, however, exactly these kinds of words that typically get loaned, and this is also why it is hard to imagine that concept-list-based algorithms could be succesfull in finding any loanwords).

Semantic linking: Only words are being linked that describe the same concept. This ignores semantic change, which is a significant factor, even over short periods of time (e.g. Hungarian or Finnish *ufo* is a loan from the 1970s meaining “unknown flying object”. Today it’s semanitcs has shifted towards “alien” in both languages). If we ignore semantic change the chances are therefore low to find loans.

Phonetic linking: The current approaches search for semantic similarity, through string similarity algorithms, such as the Levenshtein Distance. The problem with these is that they ignore the regularity of sound change, especially for well-researched languages as Hungarian, where we know sound-changes rules and tendecies in depth. The regularity of sound change implies that two non-similar words can be cognates and two similar words can be non-cognates (even if they are semantically similar, see false cognates in Terminology). Employing string-similarity thus seems to ignore this fact.

Concept lists: Concept lists are based on the hypothesis that two words can semantically be identical (quote). From a pragmatic point of view this hypothesis is quiestionable (quote). Even though word-disambiguation (quote) helps to compare the semanitc units, they are never able to capture the semantic nuances in their entirety (examples). And especially in loan word research it’s the nuances that matter, as explained in State-of-the-art traditional methods. Even though its in the nature of concept lists to be a simplification, their use nevertheless yielded good results in cognate detection (quote quote quote). One of the reasons for this is the fact that cognates are found in the core, or stable part of the vocabulary (quote), while loanwords appear on the periphery (qoute).

## A new framework for computer-aided borrowing detection

/After I’ve described case study three, here I could list the abstract principles of the framework so that they could be applied to any two languages.

In the following subchapters I will propose a new methodology for finding loanwords in which I try to tackle problems described in 2.3.4. The general idea is similar to the methods employed so far: The input are words of unknown origin. The intuition-based finding of linkability will be replaced by an algorithm. And the output will be a list of possible loans in the form of a .csv- or .tsv-File

### Input (in principle, not concrete)

Similarly to the first etymologists, I will use Hungarian words of unknown origin as an input. This is because words of unknown origin have no competing etymology. If I can find a Gothic etymology for a word of unknown origin, this is more credible than establishing competing etymologies to already well-established etymologies. Even though it would be of course also possible to establish competing etymologies, this would be the task of a much later stage.

I chose *Etmológiai szótár* (Gábor Zaicz, 2006) as a source for my input file. This Hungarian etymological dictionary is in its essence a compact summary of the currently (May 2020) most established such dictionary, the TESz (quote). In its annex, there is a list of words listed according to their first appearance in a text and also according to its origin. These lists in the annex are an advantage that other dictionaries lack and this is why I chose to use Zaicz dictionary as my input source.

I will, at this point, neither discuss the nature of unknown etymologies in general, nor to which degree they are unknown in Hungarian, but will for now take the list found in Zaicz 2006 as given. Even though it lacks already at first glance several words of unknown origin (e.g. “óriás”) and lists up words where the origin is fairly clear (e.g. “apát”), during my work this list turned out to be a useful approximation. For ways to improve the input see Chapter 4.

### Extracting Sound Change rules (in principle)

This is necesseary if (as in the case of Uralic) there is no comprehensive list of sound changes with their examples available.

### Proto form generator

### NSE

### Creating substituions

### Slicing

### Semantic linking (in principle)

After I described traditional criteria for semantic linking in State-of-the-art traditional methods, semantic linking in current computational models in Computational models’ semantic linking, and their problems in Problems of computational models from a traditional perspective, I will describe an own approach in this paragraph. Instead of pre-selecting allegedly semantically identical words and comparing them semantically, I will first pre-select phonetically linkable words and evaluate their semantic linkability afterwards.

### Phonetic linking (in principle)

After I described traditional criteria for phonetic linking in State-of-the-art traditional methods, phonetic linking in current computational models in Computational models’ phonetic linking, and their problems in Problems of computational models from a traditional perspective, I will propose an own approach for phonetic linking in this paragraph.

Loanword today

Loanword in the past

Donor word

Sound Changes

Sound substituion

In order to link a potential loanword to a potential donor word, it is important to capture both, sound changes, and sound substituions, rather than searching for sound similarities between them.

## English loanwords in Hungarian (Case study one)

As a general tendency, the number of sound changes increases, the further back into the past a word is traced. To evaluate the efficiency of LingPy’s string-similarity comparison for loanwords while ignoring sound change, I thus started out with the youngest loanword layer in Hungarian: English. Because it is poorly researched and no other sources were available, I had to retreat to the use of a list of English borrowings in Hungarian provided by [Wiktionary](https://en.wiktionary.org/wiki/Category:Hungarian_terms_borrowed_from_English). The goal was to evaluate how different methods for cognate detections perform compare to each other.

/Describe what I did exactly, false positives, describe the output, describe problems

what else could be done to improve results (Other,better input? Better means of comparison?) = Which other case studies I could put onto my list.

## Applying current computer methods to find Gothic loans in Hungarian (Case study two)

/Describe what I did: Input, method of comparison, output, problems, limitation of algorithm (max 10K wordpairs) slicing according to word type and word length.

/Maybe I should use this 1600-words concept list just to show that it doesn’t work at all? Or how about no?

what else could be done to improve results (Other,better input? Better means of comparison?) = Which other case studies I could put onto my list.

## Applying the new framework to find Gothic loans in Hungarian (Case study three)

### General idea

In this case study I will investigate whether or not there was any transfer of words from Gothic into Proto-Uralic that are or were present in the post-895-AD Hungarian lexicon. For this investigation I will firstly use a dictionary of Hungarian words and their corresponding Proto-Uralic ancestors. Second, from this dictionary I will map the sound changes within each and every word. Third, I will count those soundchanges. Fourth, I will generate all possible protoforms for a list of Hungarian words of unknown origin, thus creating a list of pseudo-proto-Uralic lexems. Fifth, I will create a table of possible sound substituions from Gothic words into Proto-Uralic. Sixth, I will slice both the Hungarian and the Gothic dataset according to phonotactic structure. Seventh, I will check if there are any phonetically identical words between pseudo-proto-Uralic and sound substituted Gothic. Eighth, I will semantically analyse the phonetic matches. Nineth, I will identify problems of this proves and suggest ways how to improve.

### Getting the data

#### Uralonet

To get a comprehensive list of Hungarian words and their Uralic proto-forms, I chose to rely on uralonet.nytud.hu. I retrieved the data by running code described in Uralonetparser. The parsed metadata is available at <https://github.com/martino-vic/Framework-for-computer-aided-borrowing-detection>. In principle, this data could also be extracted from etymological dictionaries, such as TESz or EWUng. However, since I have no access to a full pdf-version of those and the process of extracting the necessary data from them seems more complex, I chose uralonet.

#### Zaicz 2006

To get a list of Hungarian words of unknown origin, I used a wordlist provided in the annex of Zaicz 2006, pont G) of “grouping words accourding to their origin”. However, this dataset needs to be preprocessed. Already during my master-thesis (2018) the problem has arisen that this wordlist contains older and newer words. If we are searching for old loanwords, the newer terms have to be excluded. Now the question is how to define old and new? For example the word *csalamádé* (let’s ignore the fact that this word can phonetically not be of Uralic origin). This word appears in 1857 for the first time in a text. How can we know that this word has not been in use since Uralic times but has made it into a written source that survived until today, only in 1857? The answer is: The number of printed texts has strongly increased since 1600 and it is very unlikely that this word would have existed but was never printed into a work that has survived until today. So words that appear in 1957 or later for the first time in a text, can’t be of Uralic origin. But where to set this margin exactly? To answer this question I used the methods described in the following paragraph.

I used a code for R-Studio (2015) described in Zaicz extractor to extract the data from the annex and thereby create a new dataframe containing information on time of first appearance in a text and origin. I transformed this data into a pivot-table and took only words of Finno-Ugric origin. From this table it is clear, that words originating from the Finno-Ugric times all appear in texts from earlier than 1600 (only 1 word is an exception: ). Thus I set the margin to 1600. In a next step I used the pivot table to select a list of words of unknown origin appearing in texts from before 1600

#### Wikiling

I used the code described in Wikiling-parser to get the wordlists from wikiling.de

### Cleaning and editing the input

The webscraped csv-file had problems with the character encoding (especially accents: á would be the normal encoding, just one letter, but in the csv file it was encoded as two letters: a and ´ above each other). So I used ctrl+H in Excel to amend this encoding problem. (In the long run it would be probably nicer to find a way how to do this within Python. However, for the moment I couldn’t find out how to do this since whenever I typed the ominous ´ mark it would appear outside the quotation marks within python). I created two additional columns in the Excel-file: H\_extra and O\_extra. I moved comments and alternative forms to there, so that the columns Hun and Old would contain only one word per cell. The file is named uralonet\_webscraped\_editedinexcel.csv This file can be used as metadata so the uralonet doesn’t need to be scraped every time.

Then, I transcribed the Hungarian words to IPA using epitran for python, see hun2ipa. Resulting in Hun2IPA.csv

<ȣ̈> cannot be written with one symbol but always consists of two (ȣ+ ¨). However, to compare vowels, it is necessary that 1 Character represents 1 sound. Therefore I replaced <ȣ̈> with <¨> for denoting any front vowel. <ȣ> represents any back vowel, as usual.

Then, I transcribed the Uralic protoforms, which are written in UEW’s own transcription (reference), to IPA (with strg+H in Excel, because it seemed more convenient). I chose IPA because it can be used as a common denominator to compare different sound-strings and it is probably most widely known, which contributes to the replicability and transparency of this research. Found in the file Hun\_old\_IPA.csv

Then, I used <https://www.zompist.com/sca2.html> to add wordfinal <∅> to Hungarian words ending on a consonant. This is because wordfinal vowels were lost in Hungarian. So <∅> stands for any lost vowel. I wrote these changes into a new column, named <H+∅>. Then I added word initial <0> to Hungarian words with a word initial consonant. This is to denote consonant loss. So <0> denotes any lost consonant. I added these changes to a new column named <0+H+∅>. Then I added <0> to all Proto-Hungarian words with a word initial consonant. In this case <0> doesn’t denote consonant loss but shows that there was no consonat there. I added these changes to a new column called <0+old>.

To be able to use zoompist, however, it was necessary to count and define all vowels and consonants first, for both: Hungarian and Proto-Hungarian. I counted the IPA-Sounds with Soundcounter. I saved those lists of Vowels and Consonants to a file named H\_phon.csv

I saved all these changes to a file named Hun\_old\_IPA4.csv

As to the Hungarian words of unknown origin: I added Englisch and German translations to column B and C. I corrected those translations based on my own knowledge of the Hungarian language. If a translation seemed unclear, I replaced it with the one to found in EWUng. Then, I created two more columns into which I wrote the word types of the word based on my own knowledge of the language (I tried to do this first the nltk package for python but this didn’t work, since I used the translations as input and English words often can have different word types than the Hungarian word). Most words just have one word type but some have two, hence two columns. In column F I inserted the IPA-transcriptions generated with hun2ipa. I named the file unknown.csv

As to the Gothic words: First, I tried to add the sources that are found in the additional column on wikiling.de. However, I found it too difficult to find a way how to webscrape those columns. So instead I decided to add a link to the original website to every entry with the code described in add\_got\_links.txt. Clicking on a link will redirect you to the website where you will be able to find sources and other information. I saved the output to a file named G\_raw\_links.csv. Then, I preprocessed the data using the code descibed in Gothic\_Preprocessing.txt. That means I converted all words to lower case, discarded second forms and explanations in brackets, and transcribed everything to IPA. The latter I inserted into a new column. Then I substituted <kᵂ, hᵂ and iu> with <kw, hw, and iw>. This substitution was necessary to make before I slice the data because I am going to slice according to word structure and it is only these three sound clusters whose transcription affects the phonotactical structure of the word. I saved the output in a file named G\_clean+ipa+subst1.csv.

### Mapping and counting Sound Changes

I divided the process of mapping sound changes between Hungarian and Uralic into two stages. First, vowels, second, consonant clusters. I used the code described in QfyV\_UFUUg.txt to map and count vowels of all timelayers. The mapped changes are written into the file named Hun\_old\_IPA5. The vowel changes themselves are counted in a file called Quantify\_∑\_Vowels.csv. The code described in QfyC\_UFUUg.txt takes the previously generated file Hun\_old\_IPA5 and H\_phon.csv as an input. The mapped consonant clusters (whereas a single consonant can also be a consonant cluster. So consonant cluster here means the sequence between two vowels) are written down in a file named AllChanges.csv. The consonant cluster changes themselves are counted in a file called Quantify\_∑\_Consonants.csv. Then I used the codes QfyV\_layers and QfyC\_layers to count the consonant cluster and vowel changes within three different timelayers: Uralic, Finno-Ugric and Ugric. These are saved in six separate csv-files, each named in the format Quantify\_timelayer\_Vowel/Consonant. I concatenated consonant clusters and vowels by manually copy-pasting one csv to the bottom of the other, thereby reducing the number of output files from eight to four (Soundchanges of all layers, Uralic sound changes, Finno-Ugric Sound changes and Ugric sound chages). I put all four csv into one xlsx-File (Quantify\_SoundChanges.xlsx), whereas one csv represents one sheet. In the end I created a Pivot-Table in Excel from the Uralic sound changes. I then manually rearranged the blocks of this pivot and saved those changes in a file named SCin.csv. The protoform generator will use this file as an input. Then I modified the file SCin.csv and called it SCin\_uew.csv. In this modified file the column names (or in a wider sense: keys of a dictionary in csv format) are in IPA but the cells in the column (or values) are given in the UEW transcription. Then, I modified SCin.csv again and called it SCin1.1.csv. In this modified file I removed all non-word initial <0>s and all <∅>s from the clumns (values). This is because <0 > and <∅> denote derivational affixes, which, in the case of Hungarian, all happen to be suffixes. Therefore, we can create, based on this fact, a second input in which we remove those derivational suffixes (Code: Subtract derivational suffixes), where there might be any. I called this file noderivIN.csv. Another file is also created, called withderiv.csv – this is the wordlist with the words still containing their derivational suffixes and can be concatenated after the protoforms were generated (The reason why we can’t put both into one file is that the protoformgenerator has a strict inputfile format, so if we add a column it won’t work. Instead we have to add this column after the protoforms were generated).

### Generating Protoforms

The protoform generator takes as input the Sound Changes saved in SCin1.1.csv, SCin.csv and SCin\_uew, generated in 2.7.4. and a list of unknown etymologies in unknown.csv, generated in 2.7.3. The Code is described in Protoformgenerator as pseudo-code. The Output files are Protoforms.csv and Protoforms\_noderiv.csv. They contain all the possible Uralic protoforms of Hungarian words of unknown origin appearing in texts before 1600. The protoforms are ranked according to their NSE, which I will describe in the next subchapter.

### NSE

I introduced the NSE (Normalised sum of examples) as a measure for the credibility of etymologies. The idea of NSE is explained in 2.4.4. I used the code described in NSE as pseudo-code. The input are the quantified sound changes for each timelayer generated in 2.7.4. The output are four csv-Files: NSEU.csv, NSEFU.csv, NSEUg.csv, NSE∑.csv, which I put together into one Excel-file: NSE.xlsx.

### Slicing

Now there both datasets are in fact almost ready to be compared: The Pseudo-Proto-Uralic and the Gothic. The list of pseudo-proto-Uralic words is around 50 000 words long. And the Gothic dataset would be of a similar size after applying sound substituion. Currently it contains around 5000 words. It would seem a waste of computing power, however, to compare every word with every word, resulting in 2 500 000 000 word pairs. So it seems counter-intuitive to compare such a large dataset, especially if there are word pairs of which you know apriori that they can’t be a match. So the goal is to group them or in other words to slice the dataset. If you divide your dataset into two equally big groups for example you reduce the total number of word pairs to be compared by half already. Also, in theory, slicing would allow you to assign more sound-substituted forms to every word, which makes the likelyhood of finding matches higher. There could be specific substitution rules that are only applied to a sliced dataset: E.g. there are syllabic consonants in gothic. So one could take only words with that specific characteristic and apply different substitution rules specifically for syllabic consonants. There could be many different types of slicing: According to word type, word length, phonotactic structure, semantics, or any other specific characteristics one chooses. For reasons of simplicity (The goal of this case study is mainly to show that the framework itself works, rather than necessarily finding Gothic loans), I chose to take only words of phonotactic structure CVCV (Consonant Vowel Consonant Vowel). And this time a consonant means an actual consonant, and not a cluster. Since the pseudo-protoforms of any given word can have different phonotactic structures (as we use consonant CLUSTERS for sound laws), we have to assign a phonotactic structure to every pseudo-protoform, not to every word of unknown origin. Therefore, the datatable has to transformed in such a way that every row contains only one pseudo-proto-form. I achieved this with the code described in oneperline.txt, creating a new file named root\_withderiv\_oneperline.csv. I also created a file named uew\_UFUUg.csv which contains a list of all sound types that appear in the Uralic, Finno-Ugric and Ugric dataset in column A and in column B a + or a – according to whether it is a consonant or not. Based on root\_withderiv\_oneperline.csv and uew\_UFUUg.csv the Code desribed in Hun\_phonotactics.txt adds an extra column to the dataframe in which the phonotactic structure of each word is depicted. I did the same with the Gothic dataset with the Code described in Got\_phonotactics.txt.

### Sound Substituion

Since the sound substitutions I chose would always replace one consonant with one consonant or one vowel with one vowel, these substitutions don’t change anything about the phonotactic structure. Therefore I first dropped all the Gothic words that did not have the structure CVCV and applied the sound subsitution only after that. The three sound subsitutions that changed the phonotactics were made before I droped non-CVCV words, see 2.7.3. Non-CVCV were dropped and she sound substitutions were generated with the code described in SoundSubstitution.txt. Input files are G\_clean+ipa+subst1.csv that was created in 2.7.3 and Substitutions.csv that I created on my own. I created this file comparing the proto-uralic and the Gothic phonetic inventory. If a sound existed in Gothic but not in Proto-Uralic, I listed up the phonetically most similar phonemes that were in the Uralic phoneme inventory as possible substituions. I am aware, that sound subsitutions are more complex than this. This is a simplification, to show that the framework itself works. In the long run, substitutions have to be more precise by looking into other loans from Germanic languages into Uralic languages and use the adequate and well established substitution laws and tendencies (esp. Läglos etc). I named the output-file G\_allSubstitutionsCVCV.csv.

<ch> has to be replaced by <h> and <c> by <k>. Pretty arbitrary decision. Can be amended later.

Sound substitution from Germanic into Finnic has been well documented in Läglos. There the changes are mapped from proto-germanic into proto-finnic. The task now will be to find the Gothic counter pair of each protogermanic sound and then the Uralic/Finno-Ugric/Ugric counter pair for every protofinnic sound. And to create three substitution input files from this. Also conclude which sounds would have surely NOT occurred in any loanword (and think about ways how to distinguish and display in the algorithm whether this sound didn’t exist in the protolanguage or merely couldn’t have existed in loans.

* Vowels
  + Uralic

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Proto-Germanic | Proto-Finnic | Finnish | Gothic | Uralic (Proto-Hungarian) | Finno-Ugric (Proto-Hungarian) | Ugric (Proto-Hungarian) |
| a | a, long a, ä | a, aa, ä | a | a,ä |  |  |
| a from idg. o | o | o | a | a,ä |  |  |
| e | e | e | aí,ai | e,ä |  |  |
| e>i | i | i | i | i,e |  |  |
| long e | e,ä | ie,ä | e | e,i |  |  |
| long e> long a | long a, a | aa,a | e | e,i |  |  |
| i | i,long i | i, ii | i | i,e |  |  |
| long i | long i, i | ii, i | ei | i |  |  |
| long o | long o, o | uo, o | o | o,u |  |  |
| u | u,ü | u,y | u | u,o,ü |  |  |
| u>o | o | o | aú,au | o |  |  |
| u>long o | long o | uo |  | o,u |  |  |
| long u | long u, u | uu, u | u | u |  |  |
| ai | ai,äi,ei | ai,äi,ei | ái,ai | aj,a,i,ä |  |  |
| au | au | au | áu,au | a,o,u,aw |  |  |
| auj | aiv |  |  | aw,iw,äw |  |  |
| ei>long i | long i, i | ii, i |  | i |  |  |
| eu | eu,eü,iu | eu,öy,iu/yy | iu | i,u,ü |  |  |

Kommentare

* “Die Vokale der nicht-ersten Silben passen sich oft der urfí. Morphologie und Morphonologie an.” Das ist inkludiert weil der Generator auch „3“ als Zeichen für „any vowel“ mitreinnimmt. Andererseits produziert der Protoformgenerator nicht für alle sounds einen „3er“... Vielleicht sollte man das Datenset am Ende kopieren und alle nichtersten Silben mit „3“ ersetzen? Nein, weil das wäre intuitiv und nicht systematisch. Ich will in dieser Arbeit so systematisch wie möglich vorgehen. Vokale deren Substitution und rekonstruierte Protoform nicht den Regeln entspricht, sind weniger glaubhaft. Darum lasse ich sie auch fürs erste aus. Die würden ohne hin nicht dazu dienen eine neue Schicht zu entdecken, sondern wenn überhaupt, dann eine bekannte Schicht zu erweitern (eben weil sie nicht regelbasiert sind). Andererseits könnte ich ja die gotischen Wörter die in ihrer phonotaktik nicht als LW durchgehen in einen extra Topf tun und den mal später bearbeiten wenn ich Zeit habe. (Und dann zB die unerlaubten nicht-ersten Vokale nochmal so durchkombinieren, dass sie sich zwar nicht an die Substitutionsregeln, dafür aber an die phonotaktischen Regeln der Nehmersprache halten).
* Es ist anscheinend nicht egal welchen Ursprung ein germanischer Laut hat. Also a wurde mal als a/aa/ä entlehnt, mal als o, je nachdem was der idg. Ursprung des germ. Lautes war. Das selbe müsste man jetzt auch mit Got. machen. Das Problem ist, dass es sehr aufwändig wäre zu taggen, welcher gotische Laut in einem Wort von welchem idg. Laut kommt. Was ich machen kann ist, einfach alle drei Werte als Substitutionsmöglichkeit mitzunehmen und evtl. dann im Nachhinein, falls was gefunden wird, gegenzuchecken, was der idg. Ursprung des got. „a“-Lautes war.
* Manche Protogermanischen Laute werden in Läglos nicht behandelt. Wikipedia zB sagt dass es im protogermanischen (und vllt auch noch im Gotischen) Nasale gab. Diese Frage sollte ich noch klären.
* Theoretisch könnte man ja auch hier einen Datensatz hernehmen und den Lautwandel inkl. genauer Zahlen dokumentieren. Aber das wäre zu aufwending und würde sich auch wenig lohnen, da es sich ja nur um eine Approximation handelt. Wäre aber vllt eine Idee für ein paper, diese Lautwandel einfach zu quantifizieren. Nutzen: Forscher wissen welche Wandel genau wie gut belegt sind (nämlich in Form einer Zahl).
* Der Lautwandel der am besten Belegt ist, sollte als 100% dienen und alle anderen Lautwandel mit dem verglichen werden...
* Manche „e“s sind „e“s geblieben, manche sind noch im protogermanischen zu „i“ geworden. Davon hängt dann ab wie das Wort entlehnt wurde. Gotisches Pendant: ai (offenes e) kommt von protogerm. e und i.
* Auch bedenken dass sich Lehnwörter nicht 100% an die Regeln der Nehmersprache halten: Neue Phonotaktik, neue Phoneme können eingeführt, Vokalharmonie überhangen werden (e.g. ung. strand, soför, printszkrín, dzsungel). Das mit den neuen Phonemen ist fürs erste Hinfällig, weil wir ja wissen wie das Phonem *heute* lautet. Aber vielleicht wäre es gut die Vokalharmonie fürs erste zu ignorieren und auch die Phonotaktik nicht so ernst zu nehmen. Wenn der Code dann immer noch zu langsam ist, muss ich schauen wie ich df.iterrows() mit dieser anderen Variante ersetze die inetwa hundertmal schneller ist (#vectorization).
* Bei Verben muss ich dieses -jan Suffix von den got. Verben wegtun. Bei ung. Wörtern vllt solche offensichtlichen Suffixe oder Kofferwörter zuerst manuell entpacken.
* Auffällig eigentlich, dass in den ganzen germ. LW im Ostseefinnischen kein einziger Nasalvokal vorkommt. Irgendwas stimmt da nicht.
* Schwierig, weil Läglos behandelt Protogermanisch nach Protofinnisch. Ich habe mein Thema aber mit Gotisch nach Ungarisch bestimmt. Dh. Protogermanisch interessiert mich strenggenommen nicht. Ich muss jetzt also mir überlegen, wie eine spätere Phase des Protogermanischen (nämlich Gotisch) in eine frühere Phase des protofinnischen (nämlich uralisch, finnougrisch oder ugrisch) ausgesehen haben könnte. Was irgendwie verwirrend scheint, aufs erste. Aber ich denke man kann schon so argumentieren:
  + Uralisch>Finno-Ugrisch>Ugrisch>Ungarisch. Entlang dieser Achse kann man die lautlichen Veränderungen im Wortschatz mittels komparativer Linguistik nachverfolgen. Man weiß, welche Veränderungen passiert sind, aber man weiß nicht *wann* sie passiert sind. Das Protouralische kann ja weiter bestanden haben, während sich Proto-Finno-Permisch etc abgespaltet haben. Andererseits kann ich rein linguistisch gesehen nicht ausschließen, dass ungarisch, zum Zeitpunkt an dem es mit Gotisch in Kontakt trat, nicht sehr stark dem Protouralischen ähnelte. Es ist ja auch mehr ein theoretisches Gedankenexperiment. Das gleiche für Finno-Ugrisch und Ugrisch.
* Fuck, Wikipedia gibt komplett andere Vokale fürs Protogermanische an, als Läglos. Wtf. Im Zweifelsfall muss ich davon ausgehen dass Wikipedia Bullshit ist und Läglos aber richtig. Jetzt ist die Frage: In welchem Buch werden die germanischen Vokale korrekt dokumentiert? Und dann v.a. auch die Gotischen aus welchen germanischen Lauten sie stammen. Ich habs. Braune 2004 war das. Fand die Info immer redundant, auf welchen germ. Laut es zurück geht, wer hätte gedacht, dass es einmal so nützlich sein wird. Braune sollte eigentlich reichen. Damit sollte ich die Tabelle für Gotisch ausfüllen können. Unterschiede in der Aussprache zwischen Protogermanisch und Gotisch sollte ich aber auch notieren, weil das einen Einfluss auf die Art der Substitution hat.
* Auch sollte ich nicht vergessen, dass meine Datenbank strenggenommen aus westgotischen Wörtern besteht, nicht ostgotischen. Ich suche also eigentlich nach ungarisch-westgotischen Kontakten. Ostgotisch müssten man entweder irgendwie rekonstruieren, also ableiten, aus westgotisch und protogermanisch, oder man müsste wieder nach Ähnlichkeiten suchen und nicht nach exakten Übereinstimmungen. Letztere Option will ich aber nicht eingehen, weil dann die Etymologien wieder weniger glaubhaft werden, weil weniger regelbasiert. Notfalls könnte ich das aber probieren, falls ich nicht genug Matches anderweitig generieren kann.
* Die Unterscheidung a<idg.o und a<idg.a kann im Fall für Gotisch nicht mehr relevant sein, da das eine viel spätere Sprachstufe war. Eine Entlehung a>o ist daher auszuschließen.
* Ante Aikio schreibt in Proto\_Uralic (<https://www.academia.edu/40193033/Proto_Uralic>), dass \*a in mit \*dieses komische i entlehnt wurden. Aber in UEW wird kaum ein Wort mit diesem Sound angegeben, zumindest fast keines, dass ein sicheres Reflex im Ungarischen hätte. Ich bilde mir ein nyíl ist die einzige Ausnahme. Ich gehe mal pauschal davon aus, dass dieser Laut wohl mit „irgendein Vokal“ bzw „irgendein Hintervokal“ geconvert sein muss. Protogermanisch \*a kann also mit \*a oder \*ä oder eben auch mit \*irgendein Hintervokal ersetzt worden sein. Aber dieses irgendein Hinter/Vordervokal füge ich eh *jedem* Vokal hinu, darum muss ich dieser Aikio-Beobachtung keine extra Beachtung schenken.
* Wichtige Frage: Welche Entsprechungen des Germanischen suche ich jetzt eigentlich im Gotischen? Die lautlichen oder die historischen? Weil historisch gesehen verändern sich die laute. Die vorungarischen Subsitutionen passieren aber lautbasiert. Oder? Nicht unbedingt. Die Geschichte eines Lautes, seine Zukunft und Vergangenheit, kann etwas über den lautwert und die damit verbundene Substitution aussagen. Die Subsitution selbst ist dann aber trotzdem auf jedenfall lautbasiert. Jetzt ist die Frage, bis zu welchen Grad ich diese Geschichte miteinbeziehen will. Das hängt vielleicht damit zusammen wie viel Computingpower ich habe und wie schnell mein Code ist.
* Manchmal captured die gotische Orthographie verschiedene Sounds mit verschiedenen Symbolen. Jetzt ist die Frage: Kann ich diese Unterschiede in die IPA-Transkription mitnehmen? Wenn ja, dann kann ich auch die Subsitutionen vom *Lautwert* und nicht vom Buchstaben abhängig machen. Sogesehen könnte ich den Datensatz schon bei der IPA-Transkription um zusätzliche Formen erweitern, die einen anderen Lautwert gehabt haben könnten. Zumindest bei jenen Lauten, die laut Braune 2004 wirklich nicht ganz Eindeutig sind, z.B. ai als Diphthong oder Monophthong. Der wäre dann nämlich auch unterschiedlich substituiert worden.
* Proof of concept1 hatte ich ja schon im Mai. Jetzt im Juli habe ich zum zweiten Mal zeigen können, dass die Idee und der Algorithmus im Prinzip funktionieren. Dazu habe ich halt *irgendwelche* Subsitutionen eingefügt. Die könnte ich jetzt der Einfachkeit halber zwar behalten. Aber ich schätze es wäre nicht besonders viel Aufwand, mehr Alternativformen im gotischen Datensatz beizubehalten. Gleichzeitig müsste das die Wahrscheinlichkeit, ein LW zu finden, signifikant erhöhen. Und jetzt weiß ich ja, dass der Code und die Rechengeschwindigkeit eigentlich kein Hindernis darstellen sollten. In diesem Sinne würde ich versuchen, diesen Datensatz diesmal genauer zu bearbeiten. Das heißt konkret:
  + Zweiformen die im Datensatz vorkommen, nicht rausschmeißen, sondern unten Anfügen.
  + Bei der ipa-Transkription Unsicherheiten beachten und Wörter auf jede mögliche Art transkrivieren (z.B. ai: Diphthong oder Monophthong)
  + Bei Substitutionen mehrere Möglichkeiten hinzufügen (Auch überlegen wie häufige Cluster ersetzt worden sein könnten)
* Realisierbarkeit der obengenannten unterpunkte:
  + Mühsam aber machbar: Evtl. Python Libraries zum cleanen von Daten anwenden. Prinzipiell aber keine Zauberei
  + Komplexer, da die replace-Funktion jetzt nicht mehr reicht. Müsste wsl die replace-Funktion um eine if-klausel erweitern und falls es sich um einen unsicheren Sound handelt, seine Zweit-, Dritt- etc formen einer Liste anhägen und diese Liste dann der Hauptliste anhängen. Auch bissi mühsam aber wahrscheinlich keine Hexerei.
  + Zu lange Wörter rausschmeißen
  + Infinitivsuffixe abschneiden (-jan, ung. -el, etc.)
  + Der Ausgangspunkt wäre dann ab hier der selbe wie bisher, nur wäre die Wortliste einfach etwas länger. Vielleicht doppelt so lang. Die Wirkliche Herausforderung werden die Substitutionen sein. Erstens müsste ich die Substitution von der Position im Wort abhängig machen. Das heißt ich könnte nicht wie bisher den IPA-Tokenizer verwenden. Oder vielleicht doch, müsste ihn dann aber um die Positionen 1,2,3 erweitern. Sollte gehen. Und dann in der Subsitutionstabelle diese dann auch berücksichtigen. Jetzt kommt die Frage mit den Konsonantenverbindungen. Wenn ich den ipa-tokenizer verwende, beachtet er einzelne Konsonanten und keine Cluster. Dann kann ich auch keine Cluster ersetzen. Verwende ich meinen eigenen (uralischen) tokenizer, kriegt er Cluster. Dann kann ich aber keine einzelnen Sounds subsitutieren. Ich muss also die Frage beantworten: Will ich meinen tokenizer verwenden, den von lingpy, beide, oder keinen von beiden? Das Problem ist, ich kann nicht für *alle* Cluster eine Subsitution vorschlagen. Oder vielleicht doch? Dazu müsste ich meinen eigenen Tokenizer auf den gesamten gotischen Datensatz anwenden, alle Vokale rausschmeißen und dann die Cluster zählen die übrigbleiben. Zumindest für die häufigsten Verbindungen könnte ich dann manuell/intuitiv eine Subsitution vorschlagen und in die Tabelle eintragen. Was passiert aber mit den Clustern, die ich so manuell nicht erfassen kann? Dort müsste dann jeden Konsonant einzeln subsituiert werden. Also tk>t+k, t>t,k>k =tk. Oder: Ich könnte schauen welche Konsonantenverbindungen überhaupt in UFUUG vorkommen und dann schauen welche im Gotischen vorkommen und dann nur solche Wörter behalten, deren Subsitution zu einem erlaubten Cluster geführt hätte. Nicht erlaubt Cluster ergeben ja so oder so einen KeyError dann später. Dann kann ich da noch einfließen lassen, dass zB im Anlaut *gar*keine Cluster stehen durften. Die Wörter könnte ich auch entweder direkt raushauen, oder eben manuell mit nur einem Sound subsituieren. Eben so wie Finnisch ranta<strand, str>r. Kann dann drei Optionen angeben: str1>s,str1>t,str1>r. Dh die Tasks wären: a. Ufuug Datensatz set(alle cluster incl. Position!), durchnummerieren. Danach dasselbe mit den gotischen Wörtern. Dann versuchen die manuell übereinzustimmen, unter Beachtung der Position. Es wird aber wohl kaum möglich sein (beziehungsweise sich kaum auszahlen), für *jeden* einzelnen gotischen Cluster einen Ersatz zu finden. Andererseits habe ich irgendwo gelesen, dass Wörter eintlehnt werden, auch wenn sie phonetisch noch so unterschiedlich sind. Also phonetische einfache Entlehnbarkeit macht ein lw keineswegs Wahrscheinlicher. In diesem Sinne müsste man für *alle* got. Cluster einen Ersatz finden. Manchmal wird es aber Cluster geben, die keinem der Erlaubten Cluster in der Protosprache entsprechen. Was heißt das dann? Natürlich können lw auch die phonotaktischen Regeln brechen. Vielleicht heißen diese phonotaktischen Regeln aber auch nicht „dieser und jener Cluster war erlaubt und dieser und jener verboten“ sondern eigentlich: „Für diese und jene Cluster gibt es eine Reihe an Beispielen von denen wir Regeln ableiten können“. dh. „unerlaubte“ Cluster waren wahrscheinlich erlaubt, aber es wäre reine Spekulation sie anzunehmen. Außer ich finde eine Reihe an gotischen Wörtern die den gleichen Change zeigen. Aber das wäre eine zu große Herausforderung für jetzt. Und Ich denke so würde man auch nicht eine neue lwschicht finden, sondern wie gesagt eine bekannte Schichte ausweiten. Dh. Changes müssen regelbasiert sein, nicht weil sie sonst unmöglich wären, sondern weil wir sonst nicht wissen zu welchen Laut sie sich heute entwickelt hätten. Das heißt jene got. Cluster die mit ihrer Subsitution keine Entsprechung in ufuug hätten, die müssen aus der Liste genommen werden. Dieser Schritt käme dann wsl nach der Subsitution, anstatt, oder als Ergänzung zum Struktur-Filter. Ganz bevor ich anfange sollte ich vllt. auch zu lange Wörter rausschmeißen und dieses Infinitivsuffix entfernen (das hat auch bei den protogerm. lw ins ostseefinnische gefehlt). Das heißt das Ziel wäre dann, in dieser Liste nur Wörter beizubehalten, die keinen KeyError ergeben würden.

Nochmals kurz zusammengefasst: Die Protosprache in der wir uns bewegen hat phonotaktische und phonetische Regeln. Wenn wir eine *neue* Schicht entdecken wollen, sollten wir uns erst an diese Regeln halten. Das heißt: Die Geberwörter werden so substituiert, dass sie den Regeln entsprechen. Genauso werden auch die heutigen Wörter rekonstruiert, dass sie den alten Regeln entsprechen. Und genauso wie es heute Wörter gibt, die man aufgrund ihrer Phonetik oder Phonotaktik nicht als alte Wörter rekonstruieren kann, genauso gibt es Wörter, die man aufgrund ihrer Phonetik oder Phonotaktik nicht substituieren kann. Andererseits hat jede Lehnwortschicht seine eigenen Subsitutionsregeln. Vielleicht sollte ich einfach meine intuitive Tabelle um die Läglossubsitutionen ergänzen.

* Der konkrete Plan also nochmal:
  + Cleanen: Got. Wörter, Übersetzungen, Wortarten nach nltk transkribieren mit dictionary.
  + Zweitformen in der Datenbank beibehalten
  + Infinitiv- etc. suffixe abschneiden (Sich da reinlesen und Inspirationen aus Läglos mitnehmen. Auch ernsthaft überlegen, wie man andere Formen zB von Nomen generieren könnte, zB Akkusativformen, die entlehnt worden sein könnten)
  + Unterschiedliche IPA-Transkriptionen mitnehmen
  + zu lange Wörter rauswerfen
  + Auf C und V-Cluster aufteilen, Position hinzufügen (=posy)
  + Zählen welche Cluster wie oft vorkommen (sowohl im got., als auch im ufuug datensatz)
  + Alle Cluster (sowohl C als auch V, inkl. position) in die headers vom csv eintragen (nach häufigkeit geordnet)
  + Manuell für jeden Cluster überlegen, ob und wie er nach ufuug entlehnt worden sein könnte
    - Läglos+Intuition zur Hilfe heranziehen
    - Wenn nichts geht, dann leer lassen (nan Werte werden nicht eingelesen)
    - Wichtig: Für häufige Cluster viele verschiedene Subst.möglichkeiten vorschlagen
    - Häufige Cluster mit Entsprechungen in proto.germ. abgleichen und evtl. auch anderen germ. Sprachen und überlegen: Was ist ein „typisch gotischer“ Cluster. Damit ich dann später sagen kann: Das Wort muss gotisch sein und kann keine andere germ. Sprache sein.
    - Auch beachten, in welchem Wortteil welche Kombinationen erlaubt sind (steht in Sinor 1988 The Uralic Languages)
    - Wie könnten Hiatus und andere VV ersetzt worden sein? Je häufiger sie vorkommen, desto mehr Vorschläge.
  + Den shuffle Algorithmus für Subsitutionen etwas abändern:
    - ipa2tokens mit posy ersetzen (überlegen, was noch mit Null und Leeremenge passieren soll, evtl abschneiden?)
    - Rest gleichlassen: d.h. selbe Cluster in selber Position müssen gleich ersetzt werden. Selbe Cluster in unterschiedlichen Positionen werden wegen der 123-Markierung auch als unterschiedliche Zeichenkombinationen dann wahrgenommen
  + Als Bonus kann ich dann am Ende noch ein paar Zusatzformen hinzufügen, und zwar jene die Cluster übergreifende Veränderungen bedeutet hätten. Konkret betrifft das aus Läglos:
    - trV > tVr
    - klV > kVl
    - blV > pVl
    - Vielleicht könnte ich nach diesem Beispiel noch ein paar andere ähnliche Änderungen selbst hinzufügen, falls die sich für spezielle gotische Cluster gut ergeben würden.
    - Diese Substitution kann dann aber nicht mit dem Shufflealgorithmus passieren, sondern wahrscheinlich erst danach mit dem if (betreffender Cluster) -> replace, append to list und dann die liste an die Masterliste anhängen.
  + Die Masterliste beinhaltet dann, genauso wie bisher, nur präziser, alle möglichen substituierten Formen. Ist fixer Bestandteil dann vom Algorithmus.p
  + Die Funktionen, mit deren Hilfe ich diesen Datensatz erstellt habe, die lade ich in die Hauptfunktion rein.
  + Zwei unterschiedliche loan()-Funktionen haben: Eine für einzelne Wörter (-> Falls Übersetzung fehlt google Translate einschalten) und eine die ein df als Input nimmt und auch ein df wieder ausspuckt (dort kein Google translate, sonst wird die IP-Adresse wohl geblockt.
* Cleaning:
  + Reihenfolgen: nach Beistrich+Space aufspalten, Zahl, Stern, Fragezeichen
  + 1. Wenn Zahl am Ende der Zelle vorhanden: list.append(Zahl) else list.append(‘‘) / df[‚nr of occurence‘]=list.
  + Danach checken:
* IPA:
  + Zweitformen: **ai, au** könnten ein Diphthong gewesen sein, aber nur wenn er auf idg. ai zurückgeht. Vielleicht besser alle Wörter mit ai als Diphthong *und* als Monophthong zu transkribieren. Schlimmstenfalls bekommen ich dann false-positives die ich dann eh manuell gegenchecken kann. Aber besser so, als dass mit ein positives durch die Lappen geht.
  + **r** war auf jedenfall silbenbildend, postkonsonantisch im Auslaut oder vor s. Bei **l,m,n** ist man sich nicht sicher
  + **w** wurde in sehr komplizierten Fällen zu u vokalisiert. Würde diese Zweitform auch so bei allen Wörtern nach IPA transkribieren und dann false-positives nachher manuell aussortiern.
  + Es wird weithin angenommen, daß das g o t . **f** b i l a b i a l e r und nicht labiodentaler Spirant gewesen ist; p.63 braune 2004
  + **b** war je nach Kontext mal bilabialer Spirant, dann Verschlusslaut.
  + **h** je nach Kontext, entweder nichts, hauchlaut, oder spirant (wie ach-laut aber schwächer)
  + **g** Spirant oder Verschluss
  + **gg/gk/gq = ng.** Nicht hierher gehört die durch Assimilation entstandene Folge <gg> wiez.B. in jaggatrauada (< \*jah gatrauada). Ergo auch doppelt transkribieren? Oder drauf scheißen?
  + **<ggw> = /ngw,ggw/** man weiß nicht genau p.73
  + **d** je nach Kontext Spirant oder Verschlussl.
  + Versuchen den Kontext von **r,l,m,n,b,d,g,h** in die Transkription miteinzubeziehen, um nicht allzuviele false positives zu generieren. Gleichzeitig nicht zu viel Zeit für diesen Abschnitt wasten und immer überlegen: Wie würden diese abweichenden Aussprachen substituiert? Vielleicht würden sie gleich substituiert werden und dann ist muss die IPA-Transkription nicht so genau sein.
  + Suffixe vor Transkription abschneiden, da leichter die über die Orthographie zu detecten. Nachdenken wie und warum ich das ganze am besten angehen sollte.
  + Muss wohl den Shuffle-Algorithmus für die IPA-Transkription adaptieren, wenn ich mehrere Transkriptionsmöglichkeiten mitnehmen will. Selbes System, selber Code, selbe Idee, selber Input. Nur statt Input substitution.csv und Sound Gebersprache > Sound Nehmersprache, ist Input verschiedeneaussprachenderorthographie.csv und Orthographie > Aussprache
  + Suffixe: Teile die Wörter erst in Nomina, Verben, Adjektive. Da ich nicht nach Ähnlichkeiten, sondern exakten Formen suche, sollte ich die zusätzlichen Formen *hinzufügen* und nicht die alten ersetzen. Es gibt also die Wörterbuchform und dann gibt es wohl eine Reihe an flektierten Formen für die meisten Wörter. Bei Nomen wird es wohl die Form ohne Kasusendung geben. Die sollte ich mitnehmen und in einer idealen Welt sollte ich alle Kasusendungen automatisch rekonstruieren und anfügen. Das selbe bei Verben: Die Wurzel hernehmen und alle möglichen Endungen durchspielen. Auch bei Adjektiven (dort insbesondere maskulinum, femininum, neutrum Formen auch miteinbeziehen). In der Praxis wäre das wsl zu mühsam, v.a. auch wegen Sachen wie Ablaut und so. Aber ich könnte ja einen Schritt in diese Richtung machen und zB zmindest Akkusativ und Nominativformen bei Nomen mitnehmen. Natürlich könnte ich mich auch einlesen, wie das genau funktioniert, welche Formen die entlehnten Wörter entlehnen... Aber ich denke a) es würde ewig lange dauern was zu finden, b) wahrscheinlich weiß man sehr wenig darüber c) wenn es Fallbeispiele gibt, werden die wohl so unterschiedlich von meinem Beispiel sein, dass ich daraus kein Wissen ableiten kann und d) Was erwarte ich mir bestenfalls: Die Antwort wird wohl sein: Wir wissen sehr wenig, aber es scheint so, dass alles und noch mehr möglich ist bei Entlehnungen. Und genau das ist jetzt mein Ausgangspunkt. Auch in Läglos steht, dass auch Akkusativ- sg und pl Formen als Geberformen in Frage kommen könnten. Wenn diese Fälle, warum nicht jeder beliebige Fall. Und das selbe dann auf die anderen Wortarten angewendet.
  + Muss zu jedem Wort also automatisch ein paar flektierte Formen erstellen. Vielleicht geht das ja bei einigen schnell und einfach, wer weiß. ZB *t* anhängen sollte schnell gehen etc. Genauso die Infinitivsuffixe zu entfernen. Wichtig wäre, dass ich alle Wurzeln drin habe. Also flektierte Formen in eine neue Spalte einfügen, danach wieder dataframe.explode
  + Danach IPA, wieder explode
  + Also kurz: Zweiformen aus Datenbank->explode, flektierte Formen -> Explode, verschiedene IPA-Transkriptionen -> explode, verschiedene Substitutionsmöglichkeiten -> Explode
* Morphologie
  + Scheint für Nomen zumindest Kompliziert zu sein. Man müsste dafür einen morphological generator erstellen, so wie Jeremy für Marisch. Das wäre aber wahrscheinlich eine Dissertation in sich. Alternativ kann ich natürlich schauen, welche Formen ohne so einen Generator erstellbar wären. Diese würden dann das Ergebnis vielleicht um einiges verbessern und trotzdem wenig Aufwand kosten. Es ist aber trotzdem ein zusätzlicher Komplikationsfaktor, den ich vielleicht ganz am Ende als Bonus einfügen könnte, nach der Borrowability.
  + Devise: Alles in allem zu kompliziert und vielleicht sogar für zukünftige Projekte hoher Aufwand und wenig reward. Außerdem bin ich nicht gerade in der Germanistik beheimatet. Das einzige was ich schon machen würde, ist bei den Verben die Infinitivendung wegzutun, weil die sehr repetitiv ist. Und wie ich sehe sind die einzigen Endungen bei Verbe -an,-jan,-nan,-on. Ich kann ziemlich einfach diese entfernen und dem df hinzufügen. Das reicht dann auch schon. Nomen und Adjektive sehen extrem kompliziert aus, um etwas ähnliches zu vollbringen
* 14.08. Niko Partanen und Jeremy Bradley sind in engem Kontakt mit mir um einen morphological analyser für Gotisch herzustellen. Es scheint eine relativ einfache Sache zu sein. Das heißt ich fasse jetzt erstmal die Morphologie gar nicht an. Setzte mich aber mit Niko und Jeremy in Verbindung und dann kann ich den fertigen Algorithmus um die flektierten Formen erweitern um so die Ergebnisse etwas zu verbessern. Der Fahrplan lautet jetzt also: IPA Transkription, subsitution.csv erstellen, mit Position. Sound Changes manuell eintragen. Zuallererst aber muss ich zu lange Wörter rausschmeißen. Wie gehe ich das am besten an? Nach Clustern auf jeden Fall. Ganz am Ende sollte ich das mit der Länge auch dynamisieren, jetzt reicht es aber wenn es rigide ist, sag ich mal. Ich sollte dementsprechend also jetzt nicht die einzelnen Consonanten Zählen, sondern die Cluster. Strenggenommen würde man ja eigentlich die Silben zählen, aber das ist irgendwie komplizierter. Das heißt: ... Ziel: Zu lange gotische Wörter raushauen. Das längste was für Uralisch in Frage kommt ist CVCV (C=Ccluster, V=Vcluster). Fu und Ug geht länger, aber das interessiert mich jetzt fürs erste nicht, das kann ich später dynamisieren. Posy geht aber nur mit IPA. Dh: Erst alles nach IPA transkribieren, dann posy, dann alles raushauen was länger als 4 ist, danach substitution.csv rausbringen.
* Eigentlich wäre es einfacher die phonetischen Unsicherheiten einfach im Substitutionsfile zu capturen. Dann wäre die IPA-Transkription leicht und straight forward. Man würde die Sachen nicht unnötig verkomplizieren. Es ist zB unnötig <w> als /w/ *und* als /u/ zu transkribieren, wenn eh beide als /w/ *und* als /u/ substituiert werden. Dann kann ich es ja gleich als w belassen. Das Problem bei <ai> ist, dass es auf zwei protogermanische Laute zurückgeht, den Diphthong und das offene e. Also vielleicht war <ai> *überall* /e/, vielleicht war es *mancherorts* Diphthong. Aber eben nicht überall. Und diese Unterscheidung kann ich nicht machen, außer es wäre konsequent mit Akzenten notiert. (496x<ai>, 299x<aí>, 5x<ái>) Also ich behalte die Akzente, transkribiere die richtig und bei denen ohne Akzent nehme ich false positives in Kauf. Selbes Schema bei <au>.
  + <ai,au> zu /ai,au/ (phonetisch falsch, wird in Substitution aber richtiggestellt)
  + <aí,aú> zu /e,o/ (phonetisch richtig, keine Extraformen bei Substitution)
  + <ái,áu> zu /ai,au/ (phonetisch vllt richtig, bei Subst. auch richtigstellen)
  + <r,l,m,n> zu /r,l,m,/ (phonetisch nicht lupenrein, kann dann aber selber in die Subst. mitreinnehmen ob silbenbildend oder nicht, muss das nicht extra in IPA notieren
  + <w> zu /w/, in Subst. dann zu /u/ oder /w/
  + <f> zu /bilabialer Spirant/. Möglichkeit als labiodental in Subst. mitreinnehmen.
  + <gg,gk,gq,ggw> zu /ng,nk,nq,ngw/. Möglichkeit von /gg,gk,gq,ggw/ in Subst.
  + <b,d,g,h> versuchen Kontext miteinzubeziehen.
* got. b im Inlaut nach Vokalen die Geltung eines stimmhaften (bilabialen) Spiranten habe, dagegen im Anlaut und inlautend nach Konsonanten den stimmhaften Verschlußlaut bezeichne. Wenn ich richtig verstehe ist in den seltenen Fällen wo es im Auslaut steht <b> ein Spirant.
  + if x[0]==‘b‘:
    - b=plosive
  + elif [b im Inlaut nach Konsonant]:
    - b=plosive
  + else:
    - b=Spriant
* for i in dfgot[‚Lemma’]
  + posylist=posy(got2ipa(i))
  + if posylist[0]==01:
    - remove 1st element from list
  + if posylist[...
  + Eigentlich sollte ich die Nullen und Leeremengen drin lassen. Bis auf ein einziges FU Wort endet jedes einzelne ufuug-Wort auf Vokal. Darum sollte auch jedes got. Wort auf Vokal (oder eben auf Leeremenge) enden. Selbe Idee mit dem Nuller am Wortanfang. Andererseits nein. Frage: Muss ich 0 und Leeremenge bei posy für Gotisch abschneiden oder nicht? Ja. Weil: Wenn diese Zeichen schon hinzugefügt werden, dann nach der Substitution, weil 0 und Leer eh durch nichts anderes substituiert werden. Aber: In den Protoformen die ich generiere, habe ich es so eingestellt, dass die 0 am Wortanfang mitgenommen wird. Die Leeremenge am Wortende aber nicht. So müssten die got. Wörter dann auch aussehen. Nullen am Wortanfang immer, Leeremengen am Wortende nie. Logisch: Wenn ich eine Leeremenge an ein urlaisches Wortende Anfügen würde, dann würde das ja heißen, dass das ein *uralisches* Ableitungssufix wäre. Aber über Ableitungssufixe im Uralischen denke ich wissen wir ziemlich gar nichts. Warum dann aber Nullen am Wortanfang? Weil h zB mit 0 subtituiert werden kann Und ich im substitutions.csv file nicht angeben kann ersetze h durch ‚‘. Da muss ein Zeichen stehen das bedeutet „hier ist was verschwunden“. Deshalb muss ich auch bei den ung. Protoformen diesen 0er am Wortanfang hinzufügen. Bleibt nur noch die Frage: Warum dann nur am Wortanfang? Gotisches h kommt ja überall vor. Apropos, h ist der einzige Buchstabe den ich noch nicht mit der IPA-Transkr. gecheckt habe. Tue ich jetzt. „Wortanlaut vor Vokal Aussprache als Hauchlaut an, in allen anderen Stellungen aber als Spirant (schwächer als der nhd. ach-Laut)“ Braune2004, p.68. Passt, Sache geklärt. Und jetzt wird es so aussehen, dass nur im Anlaut *h* stehen wird. Und das kann dann mit 0 replaced werden später. Nicht aber im Wortinneren. Das 01 und Leer brauche ich ja nur um Protoformen zu generieren. Die Protoformen selbst haben aber nie 0 oder Leer. Ausnahme ist Null am Wortanfang. Eben wegen dem h am Wortanfang. Das heißt keine einzige Protoform fängt mit Vokal an, sondern hat C oder 0 als erstes Zeichen. Darum sollte auch jedes got. Wort mit 0 oder mit C anfangen. Dh bei posy kann ich das 01 drin lassen und dann als einzige Subsitution <0> machen. Allerdings muss ich die Leeremenge am Ende abschneiden. Weil es da keine Geschichte mit Ausfallen gibt, wie bei h. Oder sollte ich das auch als Substitutionsmöglichkeit einkalkulieren? Was kann wegfallen? Ich könnte das selbe mit der Leeremenge auch machen. Das das zu jeder Protoform hinzugefügt wird, die auf C endet, und auch bei jedem got. Wort das auf C endet. Dann könnte ich, optional, später auch manche Auslaute in der Substituion wegfallen lassen. Was ist mit Schwund im Wortinneren? Generell gesehen, ein mögliches linguistisches Phänomen, allerdings kommt das in uralonet kein einziges Mal vor. Obwohl es in meinem Fall als 2 markiert ist (Inlaut), aber im Endeffekt sind es trotzdem immer nur Ableitungssufixe. Dh in der Wortmitte kann bei uns nie etwas wegfallen. Scheiß kompliziert. Das betrifft jetzt aber Soundchange. Aber in der Substituion kann ja alles mögliche Wegfallen? Also zumindest Hauchlaute und Vokale und Halbvokale. Was wenn jetzt ein Wortinneres <w> mit <leer> substituiert wird. Dann hab ich halt ein got. Wort mit <leer> in der Mitte. Das wird nie matchen mit Protoform. Weil Protoform nie <leer> in der Mitte hat. Warum nicht? Weil ich es so eingestellt habe? Warum? Weil die Lautregeln es nicht zulassen dass aus <leer> in der Mitte ein Sound im Ungarischen entsteht. Nur am Wortende. Aber das habe ich ja durch noder() gelöst. Aber jetzt die Kopfnuss: Es ist zwar unmöglich ein <leer> einem Sound zuzuordnen. Aber wenn jetzt im Gotischen ein Sound in der Wortmitte wegfallen würde, ich ihn nicht mit <leer> markiere, und jeder Sound mit jedem Sound matcht? Das sollte ja rein theoretisch gehen. Also zb got. <thaho> zu <thao> und dann zB Match <ta> und <ta>. Ja das sollte gehen. Dann darf ich aber nicht im subst.csv Nullen haben, weil wenn bei shuffle ein arg leer ist, ist alles leer. Ich müsste neue Inputformen generieren. Also sowas wie noder() nur für Wortmitte und Gotisch. Genau. Dh ich schreibe dann einfach bei Subst.csv auch manchmal 0 oder Leer als möglichkeit. Und dann mache ich noder() bevor ich shuffle. Und dann kann ich das <leer> am Ende mitnehmen. Muss aber auch in loanpy2 das so einstellen dass <leer> am Wortende von <leer> kommen kann. Dann wird auch das NSE vergleichbarer mit dem NSE der Wörterbuch formen, weil dort habe ich ja <leer> > <leer> auch immer mitgerechnet. Dann ist bei posy die max Länge aber 6 und nicht 4, wegen 0 und <leer>. Und dann schneide ich auch nichts ab von posy. Viel symmetrischer so.
* posylist=[]
* for i in dfgot[‚Lemma’]
  + posylist+=posy(got2ipa(i))
  + set(posylist) #also add count
  + Use this as headers of subst.csv
  + Write substis yourself there with help of läglos
  + Change sufy from ipa2tokens to posy
  + Apply noder() to gothic words before sufy-ing
* 866 verschiedene C und V Cluster habe ich jetzt. Hm. Blöd. Das schaff ich nicht manuell. Idee: Nur *einige* manuell eintragen, rest dann automatisch auf Basis vom manuellen eintragen. Was manuell eintragen? 1. Info aus Läglos. 2. Unsicherheiten von der Aussrpache mitreinnehmen. 3. Einzelne Sounds im Anlaut, Inlaut, Auslaut erledigen. Danach die Cluster nach folgendem Schema: Alles durchkombinieren: zB Cluster „abc“ könnte substituiert worden sein mit xy,yz,xz,x,y,z[[1]](#footnote-1). Wobei das mapping a>x,b>y,c>z aus diesem ersten File entnommen wird. Dh das erste File betrifft hauptsächlich nicht-Cluster. Der zweite File wird auf Basis des ersten erstellt. Danach wird alles concatenated. Dann muss ich nur das erste manuell machen, das sind wsl so um die 100 Einträge. Das sollte gehen. Bei den Clustern kann ich dann auch die Position in Betracht ziehen. Anlaut: <abc1>: lese ich im File a1>x. Auslaut: <abc2> lesen ich im File: c2>z.
* Was nehme ich konkret in substi1.csv mit?
  + Alle Läglos-Cluster
  + Alle ipa-Ungenauigkeiten
    - /ai,au/ als Monophthonge auch
    - /iu/
    - ng,nk,nq,ngw vllt gg,gk,gq,ggw
    - Alle einzelnen Sounds (w zu w oder u, f vllt f)
  + Anderes File:
    - **r** war auf jedenfall silbenbildend, postkonsonantisch im Auslaut oder vor s. Bei **l,m,n** ist man sich nicht sicher. Klartext: C\_r2,C\_rs2 -> Vokal substi. Detto bei l,m,n
* 15.08. Läglos ausnahmen muss ich nicht mitreinnehmen, da werden immer nur C-Cluster durch einen einzelnen Sound ersetzt. Das ist eh gecovert vom Algorithmus.
* Gerade Microsoft Keyboard Layout mit ach und krach installiert. Jetzt ein Keyboard erstellen dass: Alle got./ural ipasounds, sowie uew,got.orth beinhaltet und das keyboard phd nennen. 18:28 endlich funktioniert die Scheiße. Kurzer Gedankenflash: Substitution in unterschiedliche Sprachschichten muss auch unterschiedlich ausgesehen haben. Ich muss ergo für jede Schicht ein eigenes df.got erstellen mit einem eigenen substi\_Schicht.csv Ich beschränke mich jetzt erstmal auf subst. nach Uralisch. Aber wo sehe ich welche Sounds dort erlaubt waren? Und v.a. welche Cluster? Das steht in diesem einen Buch bzw. könnte ich es auch selbst schnell extrahieren. Und dann am Ende diesen Prozess dynamisieren. Keine Lust mehr. Genau: Ich kan die Subst. nicht ernsthaft durchführen wenn ich nicht genau weiß welche Sounds Uralisch hat. 19:14 endlich die erlaubten Uralischen Sounds extrahiert, ging eh schnell aber es muss schneller schneller schneller gehen. Fühle mich gehetzt! Scheiße ich habe in meinem uralischen Inputfile das <a> mal als Front mal als backvowel transkribiert wtf. 20:46 endlich die einzelnen Sounds transkribiert. Ist zwar wieder extrem *irgendwas* aber passt schon. Jetzt muss ich schauen dass ich aus diesem File dieses zweite mit den Clustern erstelle. Danach die beiden concaten.
* 16.08. Das neue dfgot.csv ist fertig. Sieht gut aus.
  + Verbesserungen in der Zukunft:
    - Code optimieren (schneller machen)
    - l,r,m,n einfügen
    - Morphologie einfügen
  + Jetzt:
    - 2. Struktur zu dfgot1.csv hinzufügen
    - 3. Automatisch ale Zeilen droppen die unerlaubte Struktur haben
    - 1. loan(): df concatenaten, nicht Zeilennamen „manuell“ kopieren
    - Das mit den timelayers *nicht* dynamisieren weils eh nur 3 sind
    - Präsentation finalisieren (zB Zwischenstufen erwähnen mp>mb>b
  + <https://www.shanelynn.ie/merge-join-dataframes-python-pandas-index-1/> Hier steht genau was ich vermutet habe: Zwei for-loops sind viel zeitintensiver als df.merge

### Slicing2

So the sound substitutions for Gothic were only done AFTER I sliced the dataset accourding to the phonotactics. Now I have to slice the Hungarian dataset again but this can be only done AFTER the Gothic sounds have been subsituted. Namely, I dropped all Hungarian words that contain sounds that don’t appear neither in Gothic nor in Sound-Substituted Gothic. I achieved this with the code described in droplines.txt. The output I named hun\_final.csv

### Matching

First, I transformed both input files, hun\_final.csv and G\_allSubstitutionsCVCV.csv into python-dictionaries using the codes described in huncsv2dict.txt and gotcsv2dict.txt, resulting in the files Gotdict.py and Hundict.py.

Then, I employed the code described in Match.txt to check if there are identical words in

I decided not to use a similarity measuring algorithm because there is currently no string-similarity measurement algorithm that capture sound change in an adequate way. After we have applied laws and tendencies of sound change and sound substitution to both our datasets it is not necessary anymore to search for similarities. Since two words that are slightly dissimilar won’t be in a valid donor-recipient relationship according to the rules of historical linguistics. Moreover, calculating the similiratity of two words is a heavy calculation and can take up lot of time and computing power. To compare roughly 700x5000=35 000 000 word pairs the coded needed about a second. To calculate the Levenshtein Distance of this many words would have maybe taken one or several days of computing with the same Computer (Microsoft Surface Laptop Generation 1, windows 64 bit). The output is saved in a file named result.csv

### Semantic analysis

This part still needs to be filled up. For now I’m using an intuitive approach, because the number of the output is still so small. In the long run I want to evaluate the matches automatically, using approaches described in Semantic linking.

### Outlook

/Don’t write what could be done better into each subchapter because it takes aways the flow and you can’t concentrate on what actually has been done. Write all ideas for improvement here.

* Better sound substitutions based on existing research into Germanic-Uralic loaning.
* Different slicing, according to different parameters
* Different sound substitutions specifically made for certain sliced layers
* Imply insecurities about Gothic pronounciation like nasalised /a/ or non-monophthongisised Diphthongs
* Check for matches for the forms without derivational suffixes
* Pack the whole code into one module where every part can be called with a function, name module findloans
* Store all sound types in one file, not in several
* Maybe transform input-csvs to python-dictionaries for higher efficiency?

## Pseudo-Codes

### Uralonetparser1

A few words about parsing in general and the legality. Ask maintainers of uralonet if it is okay if I use their data. List of “best practice” for parsing. I have to include this subchapter in my thesis because it describes a significant step of my research. I will not, however, include this code in the module. Instead I will provide the metadata I parsed myself. This way the servers of uralonet will not be unnecessarily burdened (best practice, quote).

This code depends on eight other packages: requests, BeautifulSoup, time, random, randint, pandas, mupy, os

#define variables:

I had to randomise a lot of things in order not to be kicked out of the page.

### Uralonetparser2

If you get kicked out of the page while you were scraping in a randomised sequence within a range then you need to use this program to webscrape only those pages now that you haven’t yet scraped yet with uralonetparser1. Note that those pages are scattered randomly and that’s why you need this program.

### Zaicz extractor1

This Code for R-Studio takes the Annex of Zaicz 2006 as an input. It takes the words as listed up according to their appearance and creates a dataframe: Column one words, columns two year of first appearance in a text.

### Zaiczextractor2

R-Studio Code. It takes the Output from Zaiczextractor1 and the Annex of Zaicz 2006 as an input to create a dataframe of four columns: Hungarian word, year of first appearance in a text, question mark, origin

### Wikilingparser

### Add links to Wikiling

### Hun2ipa

Based on epitran. Whenever I use modules I should probably quote the people who made them.

### Soundcounter

This is based on a modified ipa.csv from panfon. I added <ɜ> for any vowel, <¨>for any front vowel, <ȣ> for any back vowel, <∅> for any vowel and also <,> for any vowel (This latter one is needed for later).

First the program reads in the necessary csv, extracts the necessary wordlist, breaks down the words into their letters, and appends every letter to a list. Then for every type (as opposed to token) it is checked whether the sound is marked as consonant or as vowel in the ipa-file. Vowels are appended to a list called Vowels, Consonants to a list called Cons.

### Soundchangemapper

### Got2ipa

### Got data cleaner

### Substitutions

### Phonotactics

### Match

## Conclusion

/Compare the outputs of the three case studies

/None of the string-similarity algorithms is capable of capturing sound change and sound substituion in an adequate way. Therefore hypothetical protoforms of the recipient language (based on known sound change rules) and hypothetical sound-substituted words of the donor language (based on known similar sound substitutions) have to be generated and used as an input. Since those protoforms cover all possibilities of a loaning scenario, no similarity algorithms are needed. Much rather the algorithm will check if a loaned and a donor word are IDENTICAL or not.

/Which other case studies could be done in the future, how and by whom.

# Etymological Analyses

/Describe Matches.xlsx (Describe input, describe method of comparison)

# Outlook

/Conclusion: Are there Gothic loans or not? Expected outcome: There were most likely no 400-AD Gothic words loaned into Uralic, Finno-Ugric, or Ugric. However, if we take sound changes from Turkic into account this might change. Also if we consider different stages of Gothic, like one with more Protogermanic characteristics, or one with more Crimean Gothic characteristics, the results could change again.

# Notes to myself

The eventual goal is to give existing etymologies a semantic-phonetic linkability or score to evaluate the credibility of every etymology. At the current stage, however, I have first and foremost focused on phonetic similarity, since this is more systematic and rule based than semantics[[2]](#footnote-2) (quote) and thus more easily approached computationally.

While the most established comparative methods of today still rely on human intuition, I will propose a computational model to approach this task. One of the main difficulties is time depth. The further back in the past a hypothetical loanword layer lies, the more significant the role of sound change becomes. For reasons of simplicity,

I inserted this list into a column of an Excel-Dataframe. Into the second column I inserted the corresponding English donor word. Then I used different string similarity algorithms to etc etc

Then, I used this same concept for Gothic and Hungarian: Output. Problems

How to tackle these problems?

1. Ein C-Cluster in ufuug besteht ja maximal aus zwei Konsonanten. Ich sollte also so eine liste machen allowedclusters=[...] und bei dieser Kombinatorik nur solche Cluster mit reinnehmen die erlaubt sind. [↑](#footnote-ref-1)
2. or it is less rule based but the rules are easier to understand? [↑](#footnote-ref-2)