

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 QfyV\_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 manuall 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 wors, 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.

### 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[[1]](#footnote-1) (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. or it is less rule based but the rules are easier to understand? [↑](#footnote-ref-1)