Historical Language Comparison with LingPy and EDICTOR

Johann-Mattis List

2017-11

Historical language comparison for the purpose of identifying cognates and sound correspondences in multilingual wordlists which can later be used to infer phylogenetic trees can be conveniently carried out in a framework of *computer-assisted language comparison* (see http://calc.digling.org), using tools for automatic inference, like LingPy (http://lingpy.org, List and Forkel 2016, and tools for data annotation and curation, like EDICTOR (http://edictor.digling.org, List 2017. In this tutorial, we will guide the users through all important steps ranging from data preparation, via data curation, to data export.

1 Preliminaries

So far, the majority of all work done in historical linguistics is carried out manually. Even when people use automatic approaches to infer phylogenetic trees, using sophisticated algorithms, ranging from distance-based approaches via parsimony frameworks to likelihood frameworks, the original data fed to the algorithm is based on manually annotated cognate sets provided by linguistic experts. While it is clear that complicated tasks like the identification of cognate words are difficult to fully automate, specifically when it comes to distinguishing homology due to borrowing (*xenology* in the biological literature, see List 2016) from homology due to true inheritance (*cognacy* in the classical linguistic literature, ibid.), it is also clear that manual annotation which is not based on consistent frameworks for data curation bears many shortcomings:

1. Linguists use a large variety of different formats (in fact, researchers often come up with their own ad-hoc format when they start to prepare wordlists annotated for cognacy), ignoring that well-established ways for cognate annotation have been around since the 1990ies (e.g. the ones used in the STARLING package, Starostin 2000).
2. Even if linguists know the languages under investigation extremely well, ad-hoc formats for annotation often lack the flexibility to document all steps which lead to a certain decisions. As a result, literature for individual cognate sets is barely cited, and both outsiders and insiders often do not even know which parts of the words asssigned to the same cognate sets are indeed cognate.
3. When converting the manually annotated cognate sets into the format required for phylogenetic software packages, linguists do rarely make use of automated approaches, but rather often type off the data manually, which is not only tedious, but also extremely error-prone, especially when dealing with larger datasets, not to speak of the fact that it often decouples phylogenetic analysis from the original data, making it impossible to trace which words contributed to the decisions of the phylogenetic algorithms.

These shortcomings in *transparency*, *efficiency*, and *accuracy* of currently available datasets call for the use of more consistent frameworks in which data annotation follows strict rules. The obvious solution is to use specific *interfaces* for data annotation and curation, which force scholars to be consistent in their annotation, thus increasing transparency and accuracy of cognate coding. In addition, since ad-hoc formats are not specialized for the specific task of data curation we are dealing with in historical linguistics, specialized interfaces will also greatly increase the efficiency of data curation and annotation.

A last advantage to be listed in this context is the fact that scholars who use improved frameworks for cognate coding can profit a lot from automatic software packages which may help them to carry out certain tasks in a semi-automatic fashion, e.g., by first searching for cognates automatically, and then correcting the errors. This again greatly increases the efficiency of cognate coding, while at the same time it may even improve the accuracy, since scholars can conveniently compare automatically inferred cognates with their manually inferred ones, thus finding cases where they may have committed an error.

The drawback – at least following from the feedback given by scholars from personal communication – is that scholars have to acquaint themselves with new tools and even acquire few programming skills. While the EDICTOR tool for cognate annotation does not require any programming skills, it requires scholars to understand the major design principles of cognate coding, i.e., the basic framework proposed by the EDICTOR. Advanced approaches which make use of LingPy or other software packages to preparse the data require a basic knowledge of the Python programming language.

Scholars are at times reluctant of taking these efforts, even when it comes to mastering the minimal requirements of the EDICTOR tool, which is written in plain JavaScript and can be launched in a simple webbrowser. While it is clear that not every linguist wants or should learn Python, it is not clear – given that linguists conveniently use software packages, like Endnote or Zotero for reference managment, word for text-editing, Excel for ad-hoc annotation of data – why they should not be able to acquaint themselves with the few largely intuitive operations required by tools like STARLING or EDICTOR. Linguistics is a data-driven discipline, and we are entering the digital age. Historical linguistics can of course be carried out by writing texts in prose that argue that some languages are related or for the phylogeny of a certain language family, but if we want to convince our colleagues and also render our findings in such a way that not only experts can follow our reasoning, we need to put our methodology on formal grounds, increasing the transparency of our judgments.

The following tutorial will be divided into two parts. The first parts requires some basic Python knowledge. By this, we mean that the users have a freshly installed version of Python3 on their computer (Python2 is not supported for multiple clashes on Mac-systems where the Python distribution is corrupted with libraries distributed across non-existing folders), and that they know how to open a terminal and run a Python script. We also assume that the relevant Python packages, like LingPy, Segments package (https://github.com/bambooforest/segments), and the Concepticon API (https://github.com/clld/concepticon-data) have been installed in *development mode* on the users computer. Instructions for installation can be found online for all operation systems.

The second part introduces the EDICTOR and does not require any programming knowledge, apart from the willingness to dive into the major ideas of the tool, as well as the patience to explore it.

This tutorial is further supplemented by sample files containing data, and scripts which run the code described here. We illustrate the major aspects for different language families to emphasize that the tools are applicable to virtually all language families, although certain aspects may require further development in the future.

2 Preparing the Data

As mentioned before, linguists who prepare their own collections of cognate sets for the purpose of phylogenetic reconstruction often come up with their own, seemingly convenient, formats for data representation. Judging from our experience, we can say that these formats usually have huge disadvantages in terms of transparancy and inter-operability, and we recommend all users to take the time to read more about the formats underlying LingPy, EDICTOR, but also the *Cross-Linguistic Data Formats* initiative (Forkel et al. 2017, see http://cldf.clld.org). These formats are to a large degree compatible, for LingPy and EDICTOR, they are almost identical, and will be introduced below.

The most obvious failure in data annotation that many scholars commit when preparing their data in Excel or other spreadsheet software is that they include multiple different types of information into one cell. Thus, if a word has a variant, scholars will place it into one cell in their spreadsheet software and separate the entries by a comma, a colon, a tilde, a dash, or at times even by a back-slash, often even using all of these separators inconsistently for the same dataset. A first and general rule that people creating data must understand and follow, is that

1. Only one type of information should be put into one cell in a spreadsheet.

This rule is non-negotiable, as in our experience with a huge number of differently coded datasets, scholars necessarily make annotation errors, even if they try to be consistent. Computers are not like humans, and if you want to profit from computers to ease your work, assume that they cannot interpret whether you use a comma and a colon without semantic difference when listing word variants or whether you do it on purpose. In fact, humans are also unlikely to understand this, unless it was them who created the data.

A more general rule deriving from this first rule is the rule that

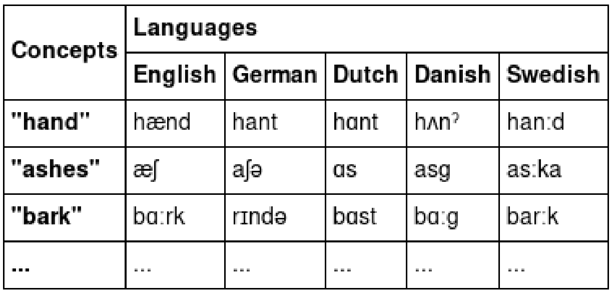
2. All information valid for a given analysis needs to be consistently annotated.

This means, for example, that, if root alternation is important for your reconstruction and cognate decisions, you need to think how to model this in consistent markup. If your data contains reflexes of an alternating protoform **ka-* vs. **ku-*, for example, it is not sufficient to simply write `*ka- ~ *ku-` and listing the reflexes, assuming that your readers will understand which reflex stems from which of the two alternants. Instead, Two proto-forms should be listed, the variants should be assigned to the correct proto-form from which they evolve, and the additional information should be given that **ka-* and **ku-* are variants of the same root. This practice is rarely followed systematically in etymological dictionaries, and therefore also often disregarded in databases, but it is clear that it is the only way to transparently list what reflex stems from which proto-variant. In fact, this is not a matter of a more computational approach to historical linguistics, but rather a matter of improving on our common practice in historical linguistics, which has for too long a time been based on lax guidelines.

2.1 Data Formats in LingPy and EDICTOR

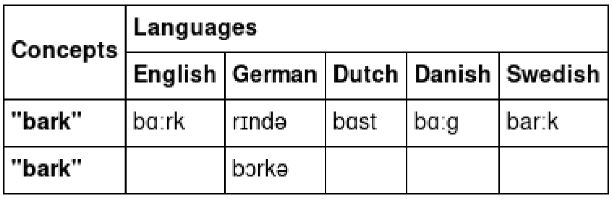
Following our rule 1, which claims that only one type of information should be put into one cell, it is clear that the *language x concept*-format which scholars use most frequently when representing word list data is not sufficient. This format essentially reserves one column for one language in the spreadsheet, and one row for one concept. The language names are given in the first row, and the concept labels are given in the first column of the spreadsheet.

This format does not only lack flexibility, as there is only one piece of information that we can give for



Concepts	Languages				
	English	**German**	**Dutch**	**Danish**	**Swedish**
"hand"	hænd	hant	hɑnt	hʌnˀ	hanːd
"ashes"	æʃ	aʃə	ɑs	asg	asːka
"bark"	bɑːrk	rɪndə	bɑst	bɑːg	barːk
...	...	...	...	...	...

Table 1: Tabular data format with languages in columns and concepts in rows.



Concepts	Languages				
	English	**German**	**Dutch**	**Danish**	**Swedish**
"bark"	bɑːrk	rɪndə	bɑst	bɑːg	barːk
"bark"		bɔrkə			

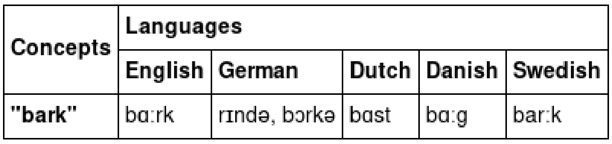
Table 2: Tabular data format with additional rows for synonym rendering.

each concept ni a given language, it is also getting more and more impractical if we are dealing with many different languages, as it will be extremely hard to inspect them on a screen (scrolling horizontally is always harder for inspection than scrolling vertically).

Despite the shortcomings, this format, or any variant of it, is one of the most widespread forms in which language data is annotated nowadays. The problem of adding essential information on cognacy, or allowing for synonyms is again mostly handled in an ad-hoc manner. Some scholars add additional rows for the same concept in order to allow to add more than one word per meaning and per language (see Table 2). Some scholars use commata or other separators to add the same entry in the same cell (Table 3). And some scholars add another column for the language which shows the synonym (Table 4).

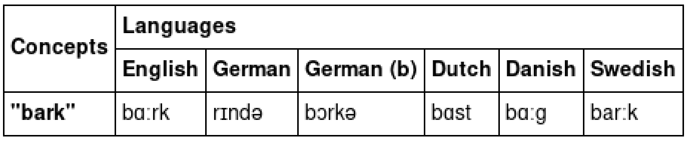
For people concerned with a consistent representation of knowledge, this is a nightmare, but the nightmare gets even more frightening, when it comes to the annotation of cognate sets. Here, people have been proving an incredible amount of phantasy in creating solutions that are computationally not only difficult to track, but also extremely prone to errors. Scholars have been using colors (Table 5). They often even just put the information on cognacy in a separate sheet, which makes it incredibly difficult to compare their judgments, especially when then number of language exceeds a handful (Table 6). At times, they may even binarise the data manually, which is even more dangerous, as it is almost guaranteed that manually binarising cognate sets will yield errors (not to speak of the waste of time and the fact that one cannot trace the characters back when carrying out phylogenetic analyses, Table 7).

Software packages like STARLING try to circumvent this problem by allowing for additional columns



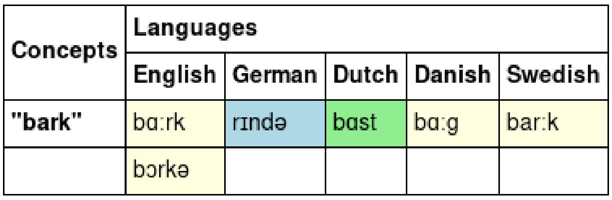
Concepts	Languages				
	English	**German**	**Dutch**	**Danish**	**Swedish**
"bark"	bɑːrk	rɪndə, bɔrkə	bɑst	bɑːg	barːk

Table 3: Multiple synonyms in the same cell.



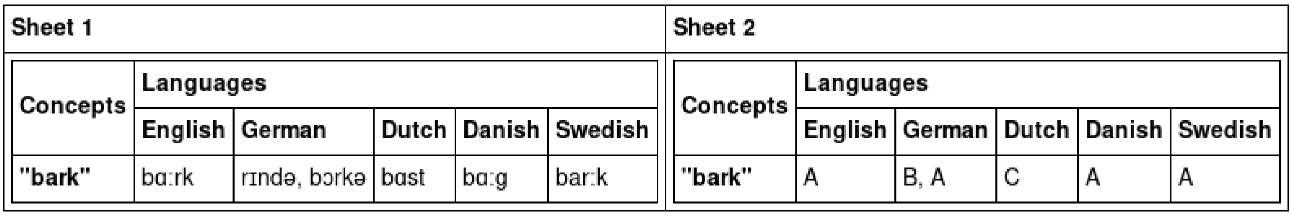
Concepts	Languages					
	English	German	German (b)	Dutch	Danish	Swedish
"bark"	bɑːrk	rɪndə	bɔrkə	bɑst	bɑːg	barːk

Table 4: Additional column for language to render synonyms.



Concepts	Languages				
	English	German	Dutch	Danish	Swedish
"bark"	bɑːrk	rɪndə	bɑst	bɑːg	barːk
	bɔrkə				

Table 5: Color-based annotation of cognate sets.



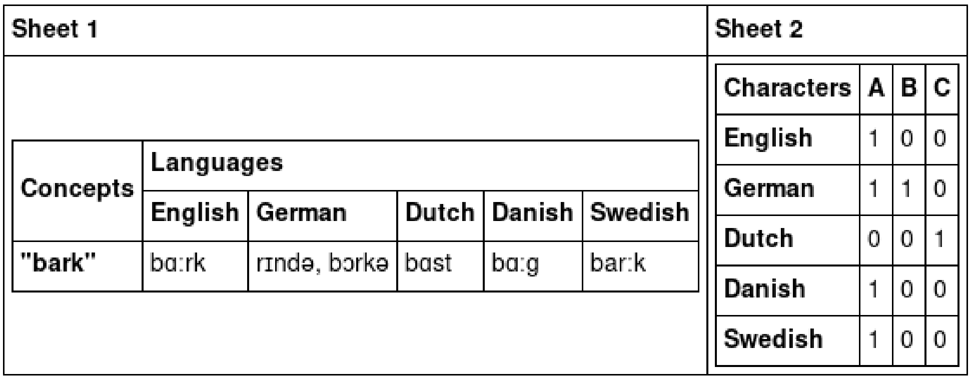
Sheet 1

Concepts	Languages				
	English	German	Dutch	Danish	Swedish
"bark"	bɑːrk	rɪndə, bɔrkə	bɑst	bɑːg	barːk

Sheet 2

Concepts	Languages				
	English	German	Dutch	Danish	Swedish
"bark"	A	B, A	C	A	A

Table 6: Multi-sheet-based annotation of cognate sets.



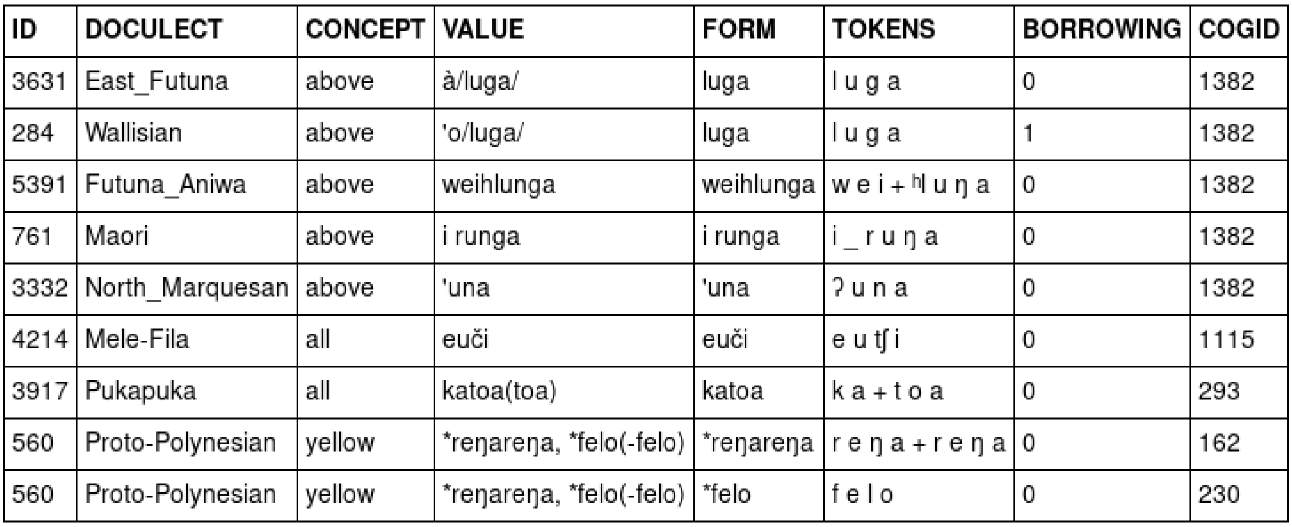
Sheet 1

Concepts	Languages				
	English	German	Dutch	Danish	Swedish
"bark"	bɑːrk	rɪndə, bɔrkə	bɑst	bɑːg	barːk

Sheet 2

Characters	A	B	C
English	1	0	0
German	1	1	0
Dutch	0	0	1
Danish	1	0	0
Swedish	1	0	0

Table 7: Binary annotation of cognate sets in multiple sheets.



ID	DOCULECT	CONCEPT	VALUE	FORM	TOKENS	BORROWING	COGID
3631	East_Futuna	above	à/luga/	luga	l u g a	0	1382
284	Wallisian	above	'o/luga/	luga	l u g a	1	1382
5391	Futuna_Aniwa	above	weihlunga	weihlunga	w e i + ʰl u ŋ a	0	1382
761	Maori	above	i runga	i runga	i _ r u ŋ a	0	1382
3332	North_Marquesan	above	'una	'una	ʔ u n a	0	1382
4214	Mele-Fila	all	euči	euči	e u tʃ i	0	1115
3917	Pukapuka	all	katoa(toa)	katoa	k a + t o a	0	293
560	Proto-Polynesian	yellow	*reŋareŋa, *felo(-felo)	*reŋareŋa	r e ŋ a + r e ŋ a	0	162
560	Proto-Polynesian	yellow	*reŋareŋa, *felo(-felo)	*felo	f e l o	0	230

Table 8: Polynesian data example for standard format in LingPy and EDICTOR.

which add additional information for the same language. LingPy and EDICTOR, however, employ a different approach which greatly increases the flexibility of the format. The major principle of this approach is to reserve one row in the spreadsheet for exactly one word form. Additional information for each word form is provided in additional columns (which can be flexibly added by the user both in LingPy and EDICTOR). The content of each column in a LingPy/EDICTOR-spreadsheet is given in the header of the file, with the first column being reserved for a numeric ID which should be greater than 0 (Table 8).

While this format seems to be rather redundant on first sight, it offers a so much greater degree of flexibility that all linguists who started to seriously test this kind of data representation quickly understand the advantages. What you need to keep in mind is that the number of columns is theoretically unlimited. So you can easily add your own columns which you want to use for either enhanced ways to annotated and model your data, or to add notes in prose which you can later include in your publication. You can add sources, and you can be very detailed, listing the page number for each word form to trace from which source it was originally taken. The possibilities are virtually unlimited, once you get a clearer understanding of this way to handle linguistic data.

What is also important to know, in order to understand the superiority of the representation of data in the way in which it is supported by EDICTOR and LingPy compared to the ad-hoc format of tabular representation discussed above, is that the columns are very explicit in what they require as input. We list eight different columns in our example, namely `ID`, `DOCULECT`, `CONCEPT`, `VALUE`, `FORM`, `TOKENS`, `BORROWING`, `COGID`, but EDICTOR and LingPy allow to add more if needed, and they do not need all of these columns in order to work properly.

Let us give a brief introduction to the most important aspects regarding these basic fields (columns) in our example:

1. **`ID`**: requires a numerical value (integer) greater than 0, but not necessarily consecutively ordered (i.e., if you have 5, 10, 10000, all is fine, but you cannot use floats, negative values, or 0).
2. **`DOCULECT`**: requires an arbitrary language name and can take virtually any value, but if users plan to export data from LingPy and EDICTOR to other formats, users should make sure that their language names do NOT include any brackest, spaces, and ideally also no other characters than the standard characters from the Latin alphabet. Spaces can be conveniently replaced by

underscores _, and brackets can be simply deleted. If users insist on having specific language names, we recommend to add an additional column to their data, where they list the language names as they prefer (with brackets and spaces) but that they allow the software to deal only with those language names which can be easily exported to other data formats.

3. **CONCEPT**: This value can also be arbitrary, but ideally, users should avoid inconsistencies. Given that EDICTOR works in JavaScript, which means that certain characters may be meaningful which are also included in concepts, we recommend to restrict the concept labels to alphanumerical entries, ideally not using too many brackets, and especially avoiding characters like the greater-smaller-sign (<>) or literal quotation marks ".
4. **VALUE**: This column is not required by neither LingPy nor EDICTOR, but it is important to show how the data was presented in the original source. As you can see in our example, the values are considerably modified in the columns FORM and TOKENS. For our format, following the specifications developed for the CLDF initiative (Forkel et al. 2017), we assume that the value is the entry that you find in a source, i.e., a dictionary, and that this may consist of multiple forms. As a result, different FORM-data for the same concept and the same language may show the same value.
5. **FORM**: This column is the single entry extracted from a potentially more complex VALUE in a given source. As you can see from the last two rows in our example in Table 8, the complex value for Proto-Polynesian *yellow* was split into two different forms, both being assigned to one row in our data, thus overriding the entry given in the original source, the ABVD (Greenhill et al. 2008).
6. **TOKENS**: This is the most important entry for all aspects of sequence comparison. It represents the form in both a corrected transcription system as well as in *space-segmented form*. Scholars have often problems when hearing the first time about the segmentation requirement, given that they are confident that they can easily guess themselves where the sounds are. However, a computer cannot easily do so, especially in ambiguous cases, like affricates spelled out with two characters (<ts>), and for this reason, our software insists on an unambiguous representation of segments. As an additional layer, we allow for a user-defined (as this cannot be automated so far) segmentation into morphemes, using the + as a marker, as well as a user-defined segmentation into words, using the _ as a marker. For morpheme boundaries, linguists often use the dash-symbol (-) which we cannot accept since it is required to represent gaps in alignment analyses. For word boundaries, the space is usually used, which we need to replace by the underscore, since the space is already used as a marker for segment boundaries.
7. **BORROWING**: This column is a simple binary column in our example which serves the purpose to indicate whether a word has been borrowed. More elaborate ways to handle borrowing exist without doubt, but for the purpose of turning cognate sets into the binary format which is passed to other data formats and fed into phylogenetic reconstruction software, the binary format is usually all that is needed so far. Users can, however, think of their own ways to provide an enhanced codding of borrowings and get in contact with us. If sufficient examples are available, we may consider adding it to both the LingPy-EDICTOR-formats as well as the CLDF standards.
8. **COGID**: This column is crucial for annotating which words are related. The annotation is fairly simple, following the STARLING principle as used in older STARLING versions: if two words are given the same numeric ID (which must be greater than 0), they are judged to be cognate. What is extremely important for users to know is that both LingPy and EDICTOR assume that cognate sets are *globally* assigned. That is: if you give the same cognate ID to words which have different meanings, they will still be regarded as cognates. STARLING (in its most recent version which underlies the Global Lexicostatistic Database, Starostin and Krylov 2011) has given up this principle. That means, that cross-semantic cognacy can no longer be assigned. Although cognates are primarily assigned per meaning class in LingPy and EDICTOR, we are trying to develop ideas which allow for improved identification of cognates across different meanings. For

this reason, we do not allow for a **local identifier**. Problems resulting from the fact that identifiers in STARLING need to be (to our knowledge) manually assigned and could therefore lead to errors when scholars forget which numbers they already used, can be easily avoided when using the EDICTOR tool, as it automatically searches for the smallest available identifier. Other database projects like IELex (Dunn 2012) use alphabetic letters to assign words to cognate sets. We do explicitly not follow this practice, as it does not enhance readability and will also make the computation of new values for cognate sets much more difficult.

If you want to prepare your data in this format to test how it could be read in by LingPy or EDICTOR, we recommend to start with a spreadsheet (using Excel, LibreOffice, or GoogleSheets) and inserting the values as shown above. Once this has been done, you can either export the spreadsheet to text-format ("csv", *comma-separated value*), with a tabstop as delimiter and without quote characters (see the instructions for this on the web, they are numerous), or, what is often saver, you copy-paste all columns and cells (only the ones that you assign, not empty rows or columns should be included!) to an empty text file which you open with a text-editor of your preference. If you copy-paste your data in this form, it will automatically be in the format required by LingPy/EDICTOR.

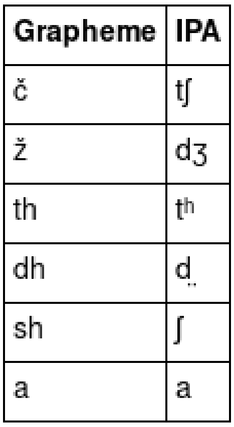
2.2 Orthography Profiles

It is probably due to the formulaic attitude to linguistic reconstruction imposed by Saussure (1916) that linguists often do not seem to worry if they base their reconstructions on highly inconsistent transcription systems, mixing various transcription practices with transliteration and even pure orthography. No matter whether one follows Pulgram's (1959) skepticism regarding the realism of linguistic reconstruction, or Hall's (1960) optimism, for a consistent investigation of linguistic data, the transcriptions need to be harmonized, and we need to assume that each segment which is represented as such represents some valid distinction in a given language. LingPy and EDICTOR offer close support for what could be called a "broad version" of the International Phonetic Alphabet, that is, they allow for the usage of symbols which are synonymous, pointing to identical sounds, such as <ʦ> vs. <ts>, <ʣ> vs. <dz>, <ʧ> vs. <tʃ>, or <ʤ> vs. <dʒ>, and often going even beyond that, accepting symbols which are used in alternative transcription transcriptions, like <č> or <ž>, which are internally treated as <tʃ> and <dʒ>. Nevertheless, judging from our experience with both computer-based and computer-assisted language comparison in the past, we highly recommend to all users to closely follow the IPA, ideally in the explicit version developed as part of the *Cross-linguistic transcription systems* initiative (List et al. in preparation), for which an online demo is available (http://calc.digling.org/clts/).

If you want to prepare your data adequately, starting from the VALUES, you have basically three choices: You can (a) prepare and segment the data manually, by adding spaces and correcting wrong transcriptions, deleting spaces, splitting values into forms, etc., you can (b) hope that your data is more or less in a good state and trust that LingPy automatically segments your data properly enough, and (c) you can follow a semi-automated workflow in which you use some Python code to split your values into different forms which you then automatically segment and modify with help of *orthography profiles* (Moran and Cysouw forthcoming).

An orthography profile can be thought of as a simple text file with two or more columns in which the first represents the values as you find them in your data (i.e., non-IPA transcriptions, etc.), and the other columns allowing you to convert the sequence of characters that you find in the first column. So in brief, you have a source-pattern and a replacement pattern:

If you save this profile in tab-separated form (as explained in the end of the previous section), you can easily use it to convert all words which have nothing but the required segments in the profile into the



Grapheme	IPA
č	tʃ
ž	dʒ
th	tʰ
dh	d̤
sh	ʃ
a	a

Table 9: A simple orthography profile.

target transcription (the profile above is provided with this tutorial as file `simple-profile.tsv`:

```
# import relevant modules
from segments.tokenizer import Tokenizer

# load the tokenizer object
tk = Tokenizer('P_simple-profile.tsv')

# convert a string to test it
print(tk('čathashadhža'))
print(tk('čathashadhža', column='IPA'))
```

```
č a th a sh a dh ž a
tʃ a tʰ a ʃ a d̤ dʒ a
```

What you can see from this small example is that the orthography profile code does essentially two things: it segments the word by treating all those characters as one segment which you listed in the `Grapheme` column, and it can further be used to convert those segments automatically, in case you provided an additional column (in our case called `IPA`).

If your profile is not sufficient to list all relevant characters or character combinations, however, the code will yield erroneous output:

```
print(tk('catapura'))
print(tk('catapura', column='IPA'))
```

```
� a � a � � � a
� a � a � � � a
```

You can see from this example, that only the “a” is recognized as a valid variant in the string `catapura`, as the relevant characters are not given in our profile. As a result, you may imagine that it is quite tedious to write an initial orthography profile on your own, specifically when you have not a sound knowledge of the variation in your data.

To ease the pain, we have written a script that uses LingPy’s rather well-informed algorithm for segmentation and can be used for the initial creation of orthography profiles. We demonstrate the usage with the small example file `P_input-file.tsv`, which contains a couple of Germanic languages and only two concepts. The file which LingPy creates will be called `P_profile.tsv`. This code can be run in any commandline, so in contrast to the Python code above, you only need to open a terminal, and type in the following commands:

```
$ lingpy profile -i P_profile-creation.tsv -o
  P_created-profile.tsv --column=ipa
```

The resulting profile will look as shown in Table 10.

You can see that in additoin to the IPA column, the code also lists the frequency and the Unicode codepoints. If the IPA column contains the character sequence <?>, it means that LingPy cannot interprete the sound segment. If it contains the characters sequence <???>, it means that the full entry could not be resolved.

Once having created an initial profile in this way, you can then easily modify the characters yourself, and apply it to your original data. In order to illustrate this, we have updated the profile, replacing the questionable cases as shown in Table 11.

Note that in this case, we decided that the `???` should no longer be rendered in the data, which is why we replace them by `NULL` a specific term used to indicate that something should be deleted upon conversion. In the other case we decided to take the first form, and we list only this form, but we manually segment it, following the basic principle of segmentation which says that sounds which form a unit should be grouped into one segment each.

With this corrected profile, which you find with the filename `P_modified-profile.tsv`, we can now enhance our data with help of some short Python code:

```python
from lingpy import *
from segments.tokenizer import Tokenizer
wl = Wordlist('P_input-file.tsv')
op = Tokenizer('P_modified-profile.tsv')
wl.add_entries('tokens', 'ipa', op, column='IPA')
wl.output('tsv', filename='P_output-file', ignore='all',
    prettify=False)
for idx, doculect, form, tokens in wl.iter_rows('doculect', 'ipa', 'tokens'):
    if form != tokens.replace(' ', ''):
        print('{0:10} {1:10} {2:15}'.format(doculect, form, tokens))
```

```
German     ???ɪx      ɪ x
English    ɔːl,a:l    oː l
```

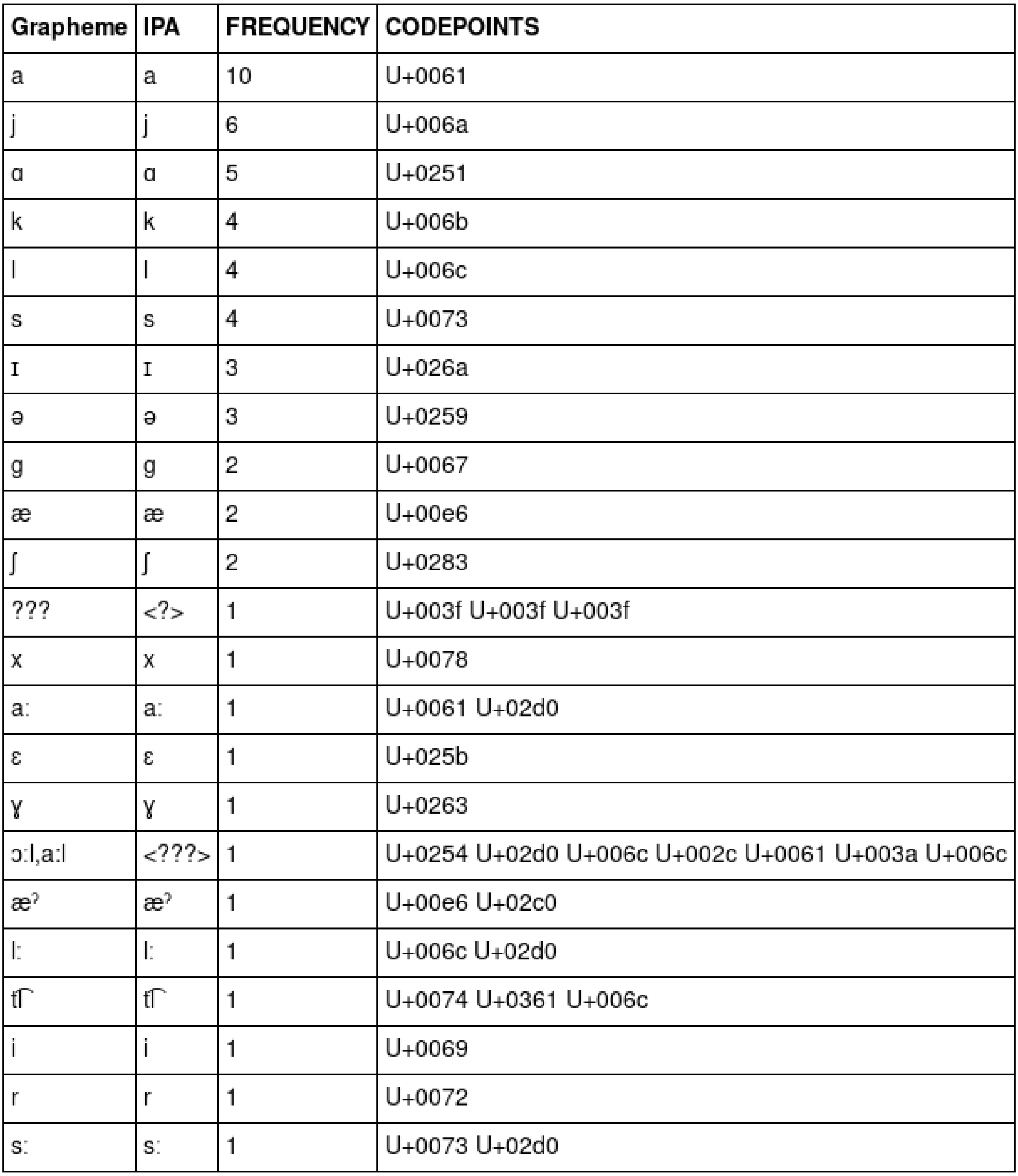
You can see, the profile code has successfully converted the data in the file, and if you inspect the file `P_output-file.tsv` you will see that an additional column with `TOKENS` was added to the file.

There is an enhanced way to make use of orthography profiles, namely by adding simple context, such as "occurs in the beginning" and "occurs in the end". You will need to tweak the code a little bit, but context-sensitive profiles may come in handy if you deal with more complex data:

```
$ lingpy profile -i P_input-file.tsv -o P_context-profile.tsv --column=ipa
  --context
```

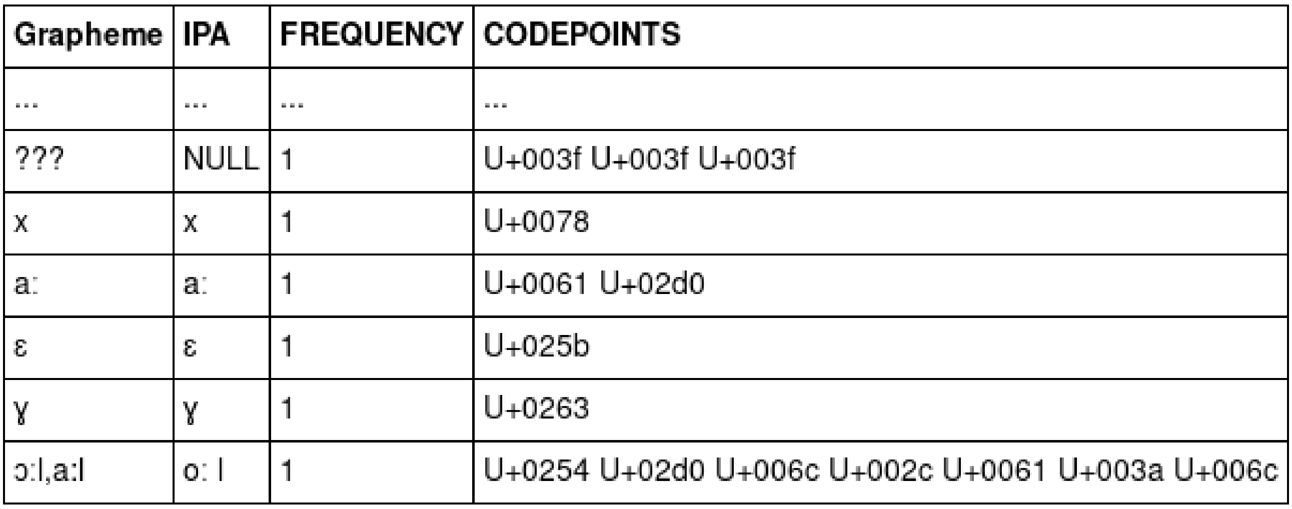
The resulting profile has the following appearance:

As you can see, the number of rows has slightly increased, since this time, each character is given at least three times: one time as the beginning of a word form (marked by the character ^), one time in the middle of a word, and one time as the end of a word form (marked by the Dollar character $). The profile is also more verbose in so fwar as it gives examples and the languages in which these



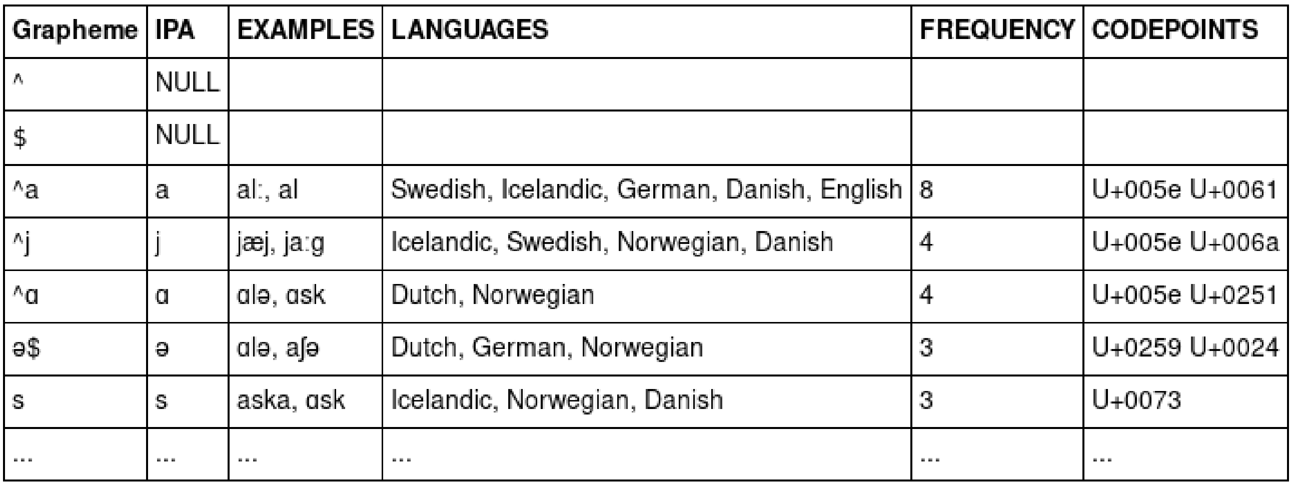
Grapheme	IPA	FREQUENCY	CODEPOINTS
a	a	10	U+0061
j	j	6	U+006a
ɑ	ɑ	5	U+0251
k	k	4	U+006b
l	l	4	U+006c
s	s	4	U+0073
ɪ	ɪ	3	U+026a
ə	ə	3	U+0259
g	g	2	U+0067
æ	æ	2	U+00e6
ʃ	ʃ	2	U+0283
???	<?>	1	U+003f U+003f U+003f
x	x	1	U+0078
aː	aː	1	U+0061 U+02d0
ɛ	ɛ	1	U+025b
ɣ	ɣ	1	U+0263
ɔːl,a:l	<???>	1	U+0254 U+02d0 U+006c U+002c U+0061 U+003a U+006c
æˀ	æˀ	1	U+00e6 U+02c0
lː	lː	1	U+006c U+02d0
t͡l	t͡l	1	U+0074 U+0361 U+006c
i	i	1	U+0069
r	r	1	U+0072
sː	sː	1	U+0073 U+02d0

Table 10: Preliminary orthography profile with help of LingPy.



Grapheme	IPA	FREQUENCY	CODEPOINTS
...	...	...	...
???	NULL	1	U+003f U+003f U+003f
x	x	1	U+0078
aː	aː	1	U+0061 U+02d0
ɛ	ɛ	1	U+025b
ɣ	ɣ	1	U+0263
ɔːl,a:l	oː l	1	U+0254 U+02d0 U+006c U+002c U+0061 U+003a U+006c

Table 11: Corrected orthography profile.



Grapheme	IPA	EXAMPLES	LANGUAGES	FREQUENCY	CODEPOINTS
^	NULL				
$	NULL				
^a	a	alː, al	Swedish, Icelandic, German, Danish, English	8	U+005e U+0061
^j	j	jæj, jaːg	Icelandic, Swedish, Norwegian, Danish	4	U+005e U+006a
^ɑ	ɑ	ɑlə, ɑsk	Dutch, Norwegian	4	U+005e U+0251
ə$	ə	ɑlə, aʃə	Dutch, German, Norwegian	3	U+0259 U+0024
s	s	aska, ɑsk	Icelandic, Norwegian, Danish	3	U+0073
...	...	...	...	...	...

Table 12: Extended orthography profile with context.

examples occur. Correcting this profile is a bit more tedious, but it has the advantage of allowing you to be much more precise in your conversions. When converting the data with help of Python later, you must remember to add the start- and end-markers to each sound, which renders the code a bit more complex (assuming a corrected profile `P_corrected-context-profile.tsv`):

```
wl = Wordlist('P_input-file.tsv')
op = Tokenizer('P_modified-context-profile.tsv')
wl.add_entries('tokens', 'ipa', lambda x: op('^'+x+'$', column='IPA'))
wl.output('tsv', filename='P_output-file', ignore='all', prettify=False)
for idx, doculect, form, tokens in wl.iter_rows('doculect', 'ipa', 'tokens'):
    if form != tokens.replace(' ', ''):
        print('{0:10} {1:10} {2:15}'.format(doculect, form, tokens))
```

```
German     ???ɪx      ɪ x
English    ɔːl,a:l    oː l
```

2.4 Automatic Analysis with LingPy

Once you have assembled your data in the form discussed above, it is straightforward to use LingPy to carry out an initial search for cognates. Thus, using the file `P_output-file.tsv`, we can easily search for cognates in our data.

```
lex = LexStat('P_output-file.tsv')
lex.cluster(method='sca', threshold=0.45, ref='cogid')
lex.output(
    'tsv',
    filename='P_cognate-file',
    subset=True,
    prettify=False,
    ignore='all'
)
```

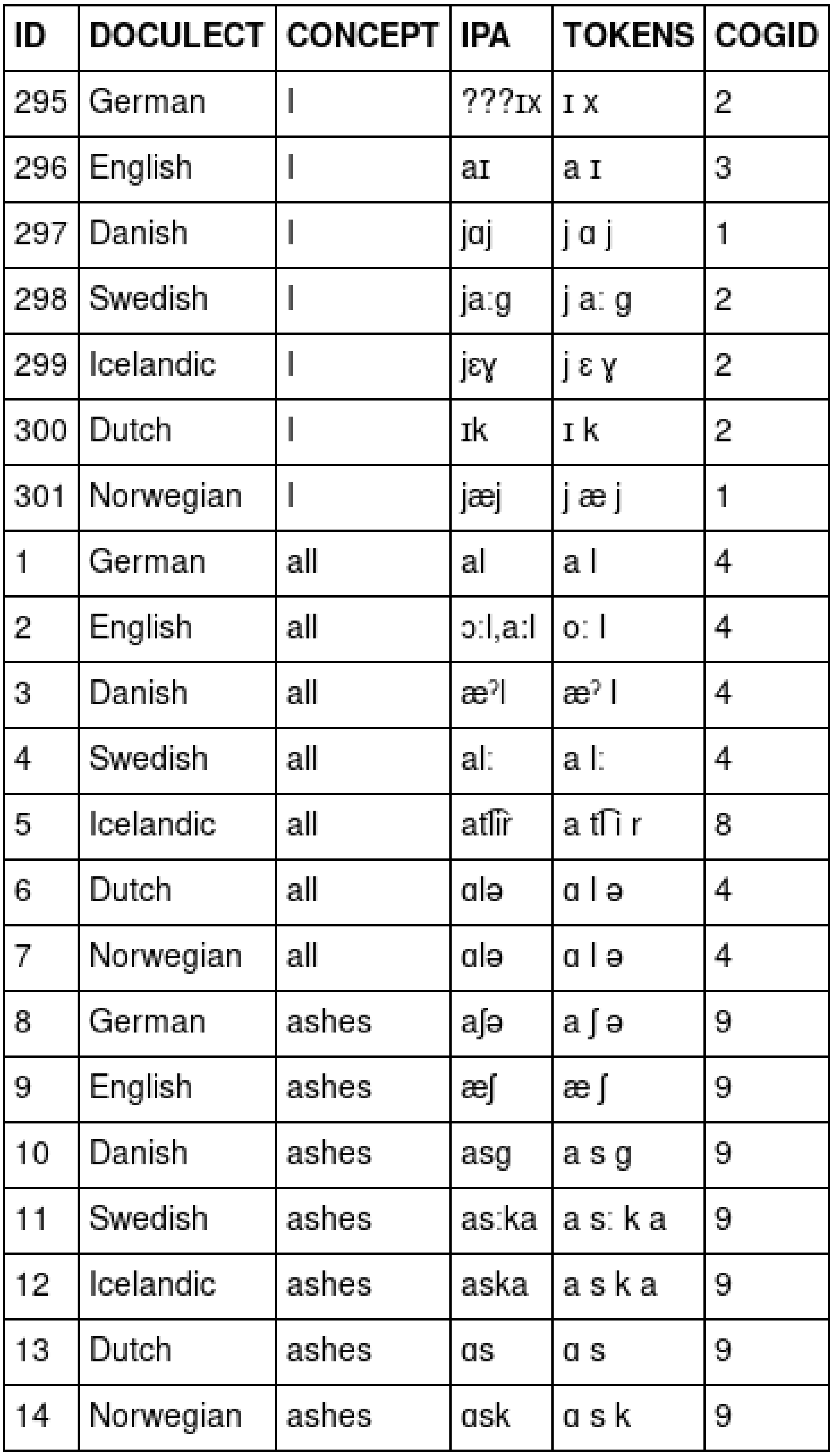
The resulting file looks as shown in Table 13.

Thus, you can see that LingPy added the `COGID` column which contains the automated cognate judgments made by the algorithm we used.

You can also align the data, which can be easily done as follows:

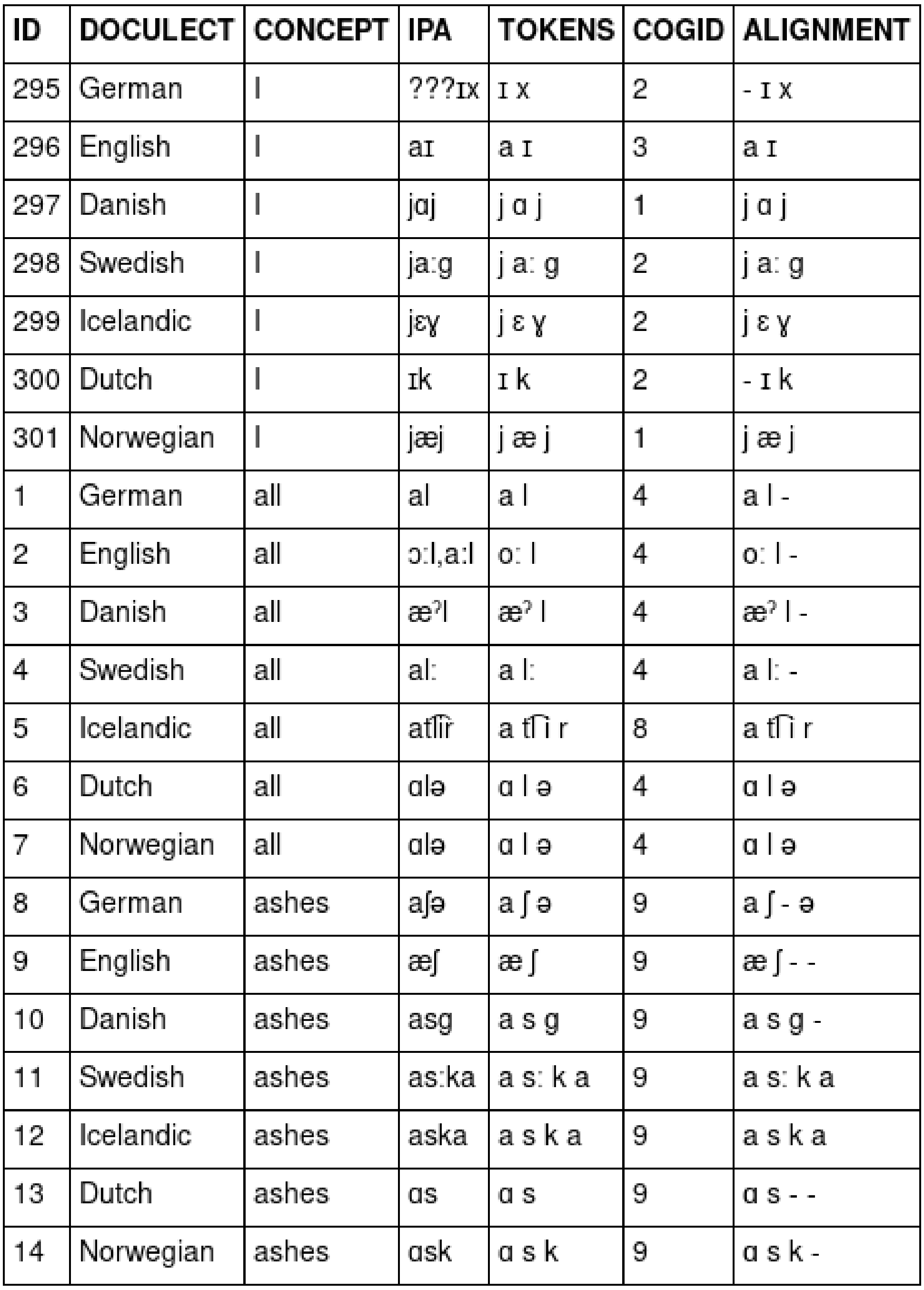
```
alm = Alignments(lex, ref='cogid')
alm.align()
alm.output(
    'tsv',
    filename='P_alignment-file',
    subset=True,
    cols=['doculect', 'concept', 'ipa', 'tokens', 'cogid', 'alignment'],
    prettify=False,
    ignore='all'
)
```

The output now looks as shown in Table 14.



ID	DOCULECT	CONCEPT	IPA	TOKENS	COGID
295	German	I	???ɪx	ɪ x	2
296	English	I	aɪ	a ɪ	3
297	Danish	I	jɑj	j ɑ j	1
298	Swedish	I	jaːg	j aː g	2
299	Icelandic	I	jɛɣ	j ɛ ɣ	2
300	Dutch	I	ɪk	ɪ k	2
301	Norwegian	I	jæj	j æ j	1
1	German	all	al	a l	4
2	English	all	ɔːl,aːl	oː l	4
3	Danish	all	æˀl	æˀ l	4
4	Swedish	all	alː	a lː	4
5	Icelandic	all	at͡ɬir	a t͡ɬ i r	8
6	Dutch	all	ɑlə	ɑ l ə	4
7	Norwegian	all	ɑlə	ɑ l ə	4
8	German	ashes	aʃə	a ʃ ə	9
9	English	ashes	æʃ	æ ʃ	9
10	Danish	ashes	asg	a s g	9
11	Swedish	ashes	asːka	a sː k a	9
12	Icelandic	ashes	aska	a s k a	9
13	Dutch	ashes	ɑs	ɑ s	9
14	Norwegian	ashes	ɑsk	ɑ s k	9

Table 13: Output of automatic cognate detection in LingPy.



ID	DOCULECT	CONCEPT	IPA	TOKENS	COGID	ALIGNMENT
295	German	I	???ɪx	ɪ x	2	- ɪ x
296	English	I	aɪ	a ɪ	3	a ɪ
297	Danish	I	jɑj	j ɑ j	1	j ɑ j
298	Swedish	I	jaːg	j aː g	2	j aː g
299	Icelandic	I	jɛɣ	j ɛ ɣ	2	j ɛ ɣ
300	Dutch	I	ɪk	ɪ k	2	- ɪ k
301	Norwegian	I	jæj	j æ j	1	j æ j
1	German	all	al	a l	4	a l -
2	English	all	ɔːl,aːl	oː l	4	oː l -
3	Danish	all	æˀl	æˀ l	4	æˀ l -
4	Swedish	all	alː	a lː	4	a lː -
5	Icelandic	all	atl͡ir	a tl͡ i r	8	a tl͡ i r
6	Dutch	all	ɑlə	ɑ l ə	4	ɑ l ə
7	Norwegian	all	ɑlə	ɑ l ə	4	ɑ l ə
8	German	ashes	aʃə	a ʃ ə	9	a ʃ - ə
9	English	ashes	æʃ	æ ʃ	9	æ ʃ - -
10	Danish	ashes	asg	a s g	9	a s g -
11	Swedish	ashes	asːka	a sː k a	9	a sː k a
12	Icelandic	ashes	aska	a s k a	9	a s k a
13	Dutch	ashes	ɑs	ɑ s	9	ɑ s - -
14	Norwegian	ashes	ɑsk	ɑ s k	9	ɑ s k -

Table 14: Aligned output in LingPy.

You can see, that LingPy added an ALIGNMNENT column which essentially has the same structure as the TOKENS with the difference that it contains aligned data, thus, for each cognate set, gaps may have been introduced for sounds which do not correspond to any other sounds. Needless to say that this kind of data is best inspected with the EDICTOR, where you can then directly start to correct wrong alignments or cognate sets.

2.5 Enhancing Your Data

You should always be keen on making your data as transparent as possible. This means as well that you should give other people the chance to immediately know which concepts you were investigating and which language varieties. For language varieties, you should provide a `languages.csv` file in which you list the name you use in your dataset for each language variety along with the Glottocode (if you can find one).

To make sure that your data is comparable in terms of the concepts that you investigated, you should link your questionnaire to the Concepticon (List et al. 2016). Many scholars still have a huge problem in understanding what the Concepticon actually is. We won't go into the details here, but if you are interested in selecting comparable questionnaires (e.g., words less prone to borrowing) for your language sample, you should definitely have a close look at the Concepticon website at http://concepticon.clld.org, since it is highly likely that your specific questionnaire has already been linked. In this case, you should download the concept list in the form in which it is provided by the Concepticon project, as this will spare you the time of typing it off yourself (which may introduce new errors), and you will get a lot of meta-information which may be useful. For example, if you download the Leipzig-Jakarta list (Tadmor 2009, Tadmor-2009-100), you may first learn a lot about how it was constructed, but you can also directly compare it with lists that may be similar. If you want to know how stable the concepts in this list are, for example, you could have a look at the basic list underlying the original project (Haspelmath-2009-1460), where you will receive explicit ranks for all concepts.

If you want to check the overlap between the Leipzig-Jakarta list and Swadesh's (1955) list of 100 items, you can use the Concepticon API, querying for the intersection of both lists:

```
$ concepticon intersection Tadmor-2009-100 Swadesh-1955-100
  1   ARM OR HAND            [2121] HAND (1, Swadesh-1955-100)
  2   ASH                    [646 ]
  3   BIG                    [1202]
  4   BIRD                   [937 ]
  5   BITE                   [1403]
  6   BLACK                  [163 ]
  7   BLOOD                  [946 ]
  8   BONE                   [1394]
  9   BREAST                 [1402]
 10   BURN                   [2102] BURNING (1, Tadmor-2009-100)
 11   COME                   [1446]
 12   DOG                    [2009]
 13   DRINK                  [1401]
 14   EAR                    [1247]
 15   EARTH (SOIL)           [1228]
 16   EAT                    [1336]
 17   EGG                    [744 ]
 18   EYE                    [1248]
 19   FIRE                    [221 ]
 20   FISH                    [227 ]
 21   FLESH OR MEAT           [2615]
 22   FLY (MOVE THROUGH AIR)  [1441]
 23   FOOT OR LEG             [2098] FOOT (1, Swadesh-1955-100)
 24   GIVE                    [1447]
 25   GO                      [695 ] WALK (1, Swadesh-1955-100)
 26   GOOD                    [1035]
 27   HAIR                    [1040]
 28   HEAR                    [1408]
 29   HORN (ANATOMY)          [1393]
 30   I                       [1209]
 31   KNEE                    [1371]
 32   KNOW (SOMETHING)        [1410]
 33   LEAF                    [628 ]
 34   LIVER                   [1224]
 35   LONG                    [1203]
 36   LOUSE                   [1392]
 37   MOUTH                   [674 ]
 38   NAME                    [1405]
 39   NECK                    [1333]
 40   NEW                     [1231]
 41   NIGHT                   [1233]
 42   NOSE                    [1221]
 43   NOT                     [1240]
 44   ONE                     [1493]
 45   RAINING OR RAIN         [2108] RAIN (PRECIPITATION) (1, Tadmor-2009-100)
 46   RED                     [156 ]
 47   ROOT                    [670 ]
 48   SAND                    [671 ]
 49   SAY                     [1458]
 50   SEE                     [1409]
 51   SKIN                    [763 ]
 52   SMALL                   [1246]
 53   SMOKE (EXHAUST)         [778 ]
 54   STAND                   [1442]
 55   STAR                    [1430]
 56   STONE OR ROCK           [2125] STONE (1, Swadesh-1955-100)
 57   TAIL                    [1220]
 58   THIS                    [1214]
 59   THOU                    [1215]
 60   TONGUE                  [1205]
 61   TOOTH                   [1380]
 62 * TREE OR WOOD            [2141] WOOD (1, Tadmor-2009-100),
                                     TREE (1, Swadesh-1955-100)
 63   WATER                   [948 ]
 64   WHAT                    [1236]
 65   WHO                     [1235]
```

From this output, you can learn that Leipzig-Jakarta lists “arm or hand” as a concept, while Swadesh is more concrete, listing only “hand”. You can also learn that Swadesh is not very concrete regarding the concept “rain” where he fails to inform us whether it was intended as a noun or a verb. From the match 62, you can further see that “tree” and “wood” are both judged to be subsets of the meta-concept “tree or wood”, and indeed, there are quite a few languages which do not distinguish between the two.

There are more possibilities: The `concepticon union` command allows you to calculate the union of different lists, thus allowing you to create your own questionnaires based on different concept lists. By typing the following command in the command line, for example, you can learn that the union of Leipzig-Jakarta and Swadesh’s 100-item list are 135 concepts:

```
$ concepticon union Tadmor-2009-100 Swadesh-1955-100 | wc -l
135
```

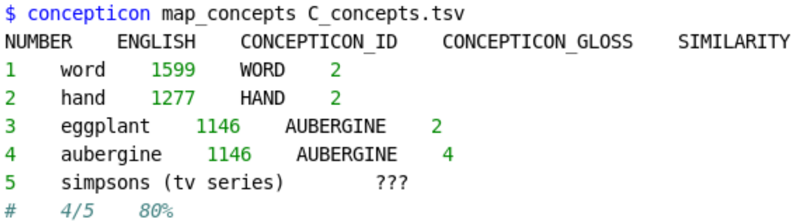
And if you add the 200-item list by Swadesh (1952), you will see that the union has 222 concepts:

```
$ concepticon union Tadmor-2009-100 Swadesh-1955-100 Swadesh-1952-200 | wc -l
222
```

More importantly, if you want to merge data from different questionnaires or datasets where your do not know to which degree concepts overlap, you can use the automatic mapping algorithm provided by the Concepticon API to get a first intelligent guess which concepts your data contains. This works even across different languages, as we have so far assembled concept labels in quite a few different language varieties which we can use to search for similar concepts. The command is a simple as typing `concepticon map_concepts <yourconceptlist>` in your terminal, where you replace `<yourconceptlist>` with your filename. We have prepared three files, one in English, one in Chinese, and one in German, all showing the following tabule structure (the following being from the file `C_concepts.tsv`):

```
NUMBER  ENGLISH
1   word
2   hand
3   eggplant
4   aubergine
5   simpsons (tv series)
```

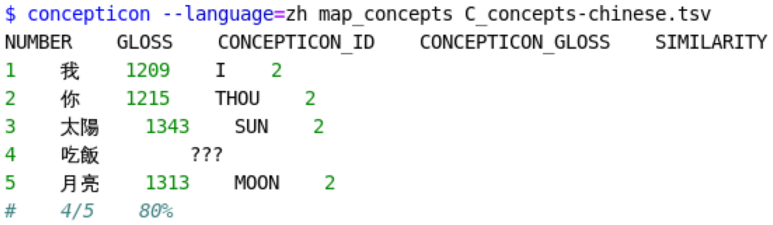
In order to link this English file to the Concepticon, all we have to do is to type:



```
$ concepticon map_concepts C_concepts.tsv
NUMBER    ENGLISH    CONCEPTICON_ID    CONCEPTICON_GLOSS    SIMILARITY
1    word    1599    WORD    2
2    hand    1277    HAND    2
3    eggplant    1146    AUBERGINE    2
4    aubergine    1146    AUBERGINE    4
5    simpsons (tv series)        ???
#    4/5    80%
```

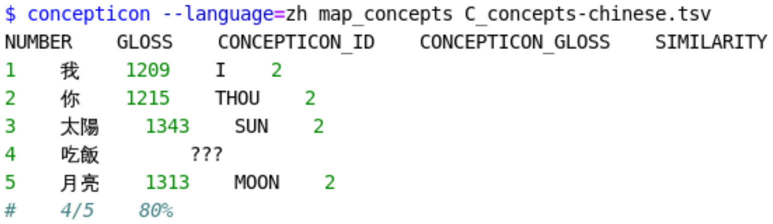
The output tells us first, whether the Concepts can be linked to Concepticon, and second, it gives us the overall percentage for inferred links. You can see that the mapping algorithm is not based on simple string identity, as it correctly links “eggplant” to the concept set `AUBERGINE`.

Similarly, we can try to link our file with Chinese concepts, the file `C_concepts-chinese.tsv`:



```
$ concepticon --language=zh map_concepts C_concepts-chinese.tsv
NUMBER    GLOSS    CONCEPTICON_ID    CONCEPTICON_GLOSS    SIMILARITY
1    我    1209    I    2
2    你    1215    THOU    2
3    太陽    1343    SUN    2
4    吃飯        ???
5    月亮    1313    MOON    2
#    4/5    80%
```

And accordingly also our file `C_concepts-german.tsv`:



```
$ concepticon --language=zh map_concepts C_concepts-chinese.tsv
NUMBER    GLOSS    CONCEPTICON_ID    CONCEPTICON_GLOSS    SIMILARITY
1    我    1209    I    2
2    你    1215    THOU    2
3    太陽    1343    SUN    2
4    吃飯        ???
5    月亮    1313    MOON    2
#    4/5    80%
```

The Concepticon is a collaborative effort that is supposed to render our linguistic data more comparable. The more questionnaires we can add to our collection, the easier it will be for future research to build on these resources. Even if you think that you do not need to link your data to Concepticon, since you anyway use the "standard list" by Swadesh, you should at least provide a `concepts.tsv` file in which you list your explicit links. In this way you guarantee that other can re-use your data and also contribute to the collaborative efforts which are currently being done in the context of the CLDF initiative.

3 Annotating and Data with EDICTOR

3.1 Overview

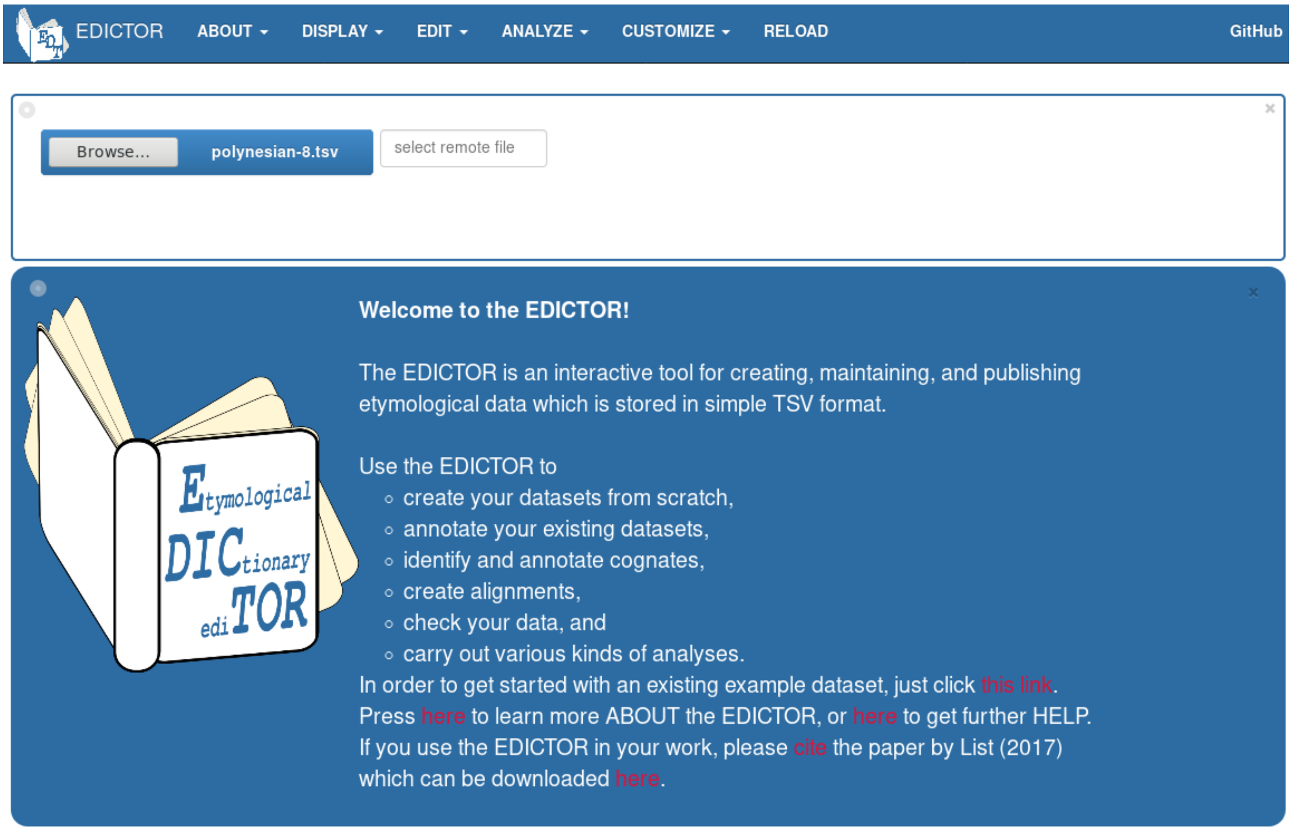
The Etymological DICtionary ediTOR (EDICTOR, http://edictor.diglin.org, List 2017) is a free, interactive, web-based tool that was specifically designed to serve as an interface between quantitative and qualitative tasks in historical linguistics. Inspired by powerful features of STARLING (Starostin 2000) and RefLex (Segerer and Flavier 2015), expanded by innovative new features, and based on a very simple data model that allows for a direct integration with quantitative software packages like LingPy, the EDICTOR is a lightweight but powerful toolkit for computer-assisted applications in historical linguistics.

The EDICTOR is written in JavaScript and can be used through a simple webbrowser (GoogleChrome and Firefox are the currently supported variants, but we mainly test on Firefox). The fact that the EDICTOR runs in a webbrowser means that it runs virtually on all modern operation systems (Windows, Linux, Mac-OS). It can also be used without an internet connection in Firefox. In order to do so, one only has to download the relevant software from the GitHub repository (https://github.com/digling/edictor) and open the `index.html` file in the main package.

3.2 Getting Started

The EDICTOR is based on a modular structure which is organized in form of *panels*, that is, windows which open once data was loaded into the tool. The basic panel, the *Wordlist* panel, is used to edit data, similar to the way in which this can be done in Excel or other spreadsheet software. Additional panels help to cluster words into cognate sets, investigate the morphological structure of words, or even sound correspondence patterns.

What users need in order to use the tool is a text-file encoded in the form in which it was discussed above, that is, a file in the standard format in which each word is given a row, and a header informs which type of data a certain column contains. In order to use the tool, users need to open the website (http://edictor.digling.org) in their browser (preferably firefox) and drag their file into the BROWSE button which which shows up on the top right of the window:



Welcome to the EDICTOR!

The EDICTOR is an interactive tool for creating, maintaining, and publishing etymological data which is stored in simple TSV format.

Use the EDICTOR to

- create your datasets from scratch,
- annotate your existing datasets,
- identify and annotate cognates,
- create alignments,
- check your data, and
- carry out various kinds of analyses.

In order to get started with an existing example dataset, just click this link.
Press here to learn more ABOUT the EDICTOR, or here to get further HELP.
If you use the EDICTOR in your work, please cite the paper by List (2017) which can be downloaded here.

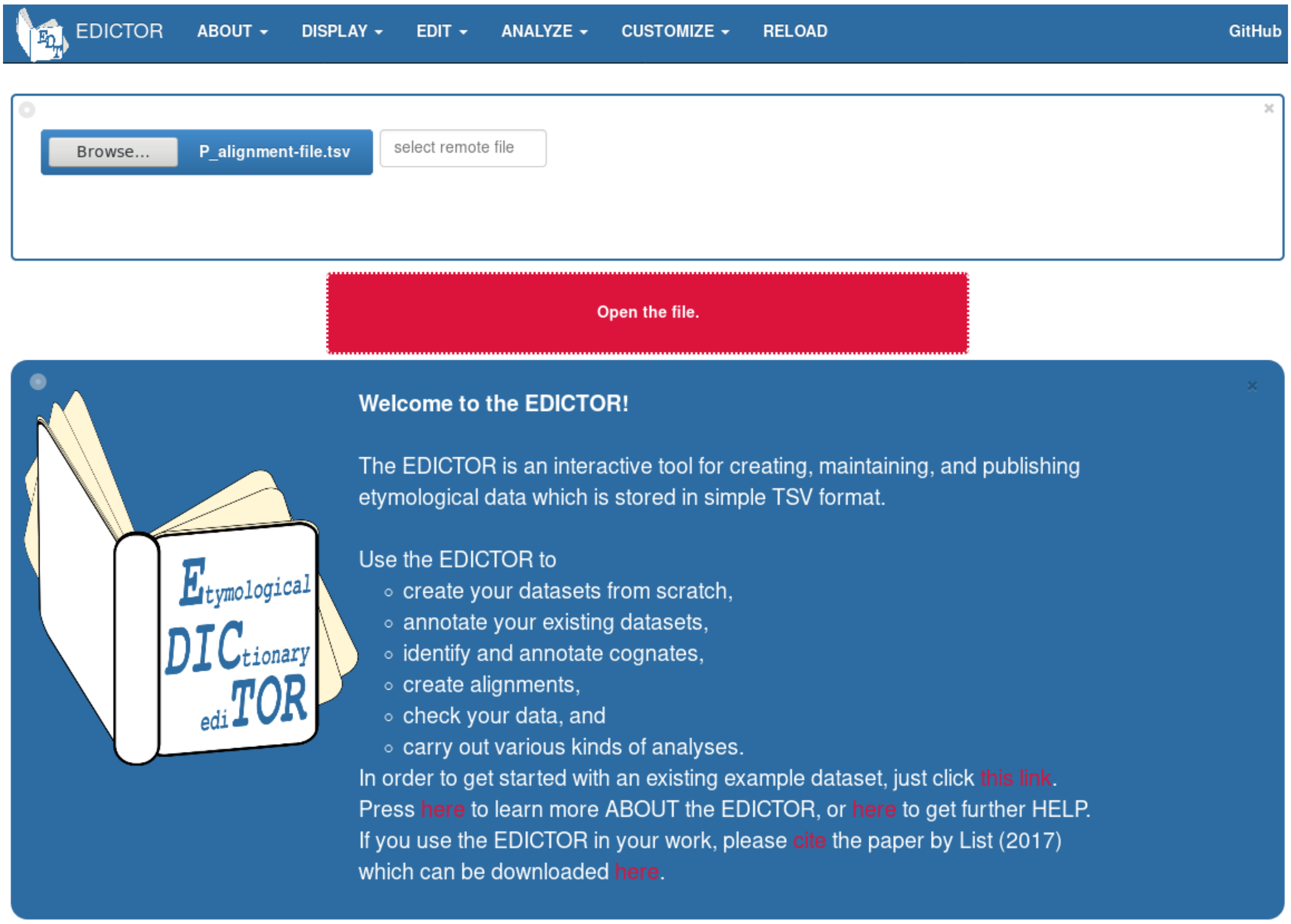
Figure 1: Welcome page of the EDICTOR.

Since the EDICTOR is written in plain JavaScript, no data will be send to the server, but all data will stay on the computer of the user alone. Thus, essentially no data is uploaded somewhere else. Linguists are sometimes afraid that their data might be stolen by their colleagues. Apart from the fact that this is unlikely, given that everybody will know where the original data came from, and that not many linguists would papers on languages where they do not know the sources, and where they do not know on what judgments the cognate assignments were based, not to speak of the fact that normally, the data people work on does not list all the sources, users of the EDICTOR should know that although the BROWSE procedure by which they can load their data looks like an upload button in other software applications, no upload is carried out. Data will rest on the client computer and not leave it during the whole time the EDICTOR is being used.

Once you have browsed your file, a red window will open, showing that the file was successfully loaded:

By clicking on the red field, you initiate your EDICTOR session, and the Wordlist panel will pop up.

As you can see from the figure, the data that we just created in the previous section is now rendered



Welcome to the EDICTOR!

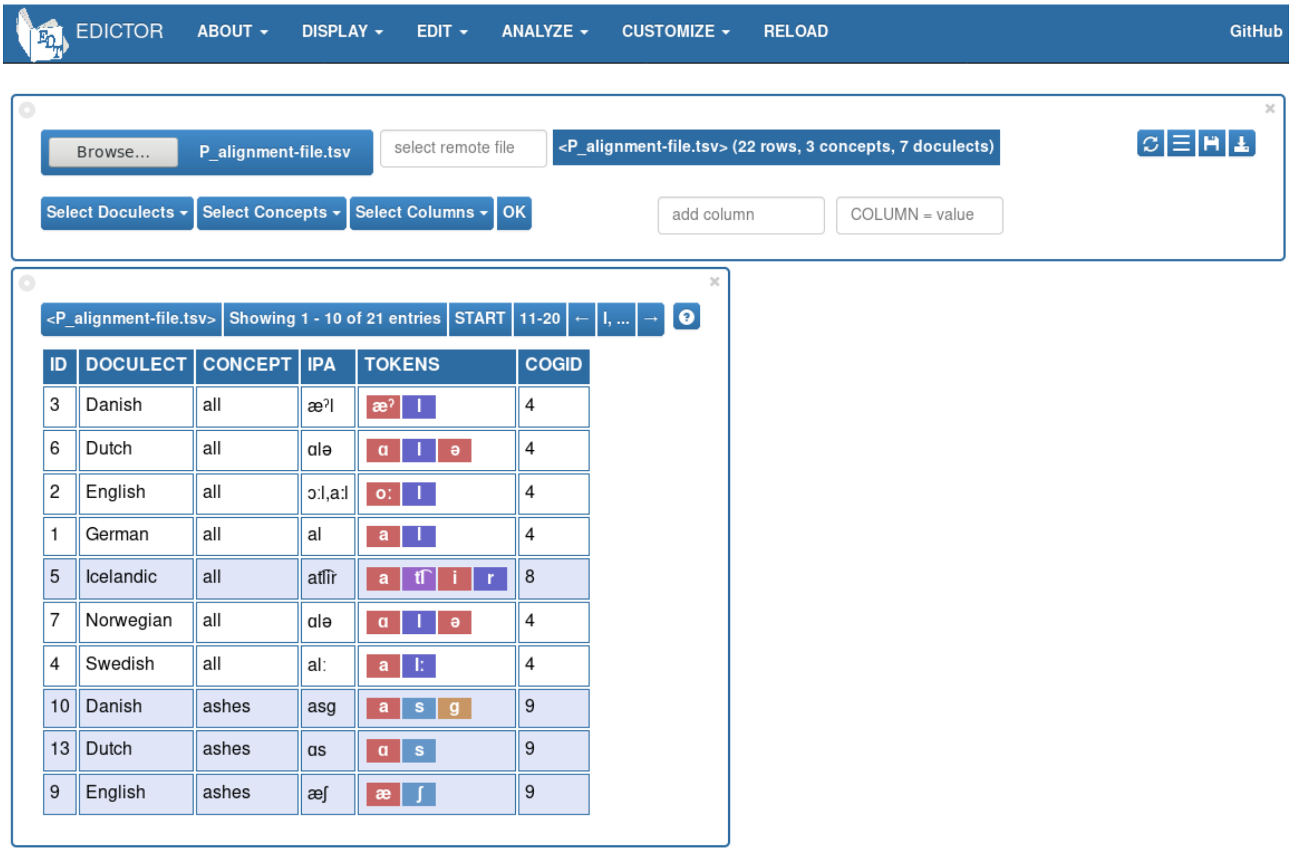
The EDICTOR is an interactive tool for creating, maintaining, and publishing etymological data which is stored in simple TSV format.

Use the EDICTOR to

- create your datasets from scratch,
- annotate your existing datasets,
- identify and annotate cognates,
- create alignments,
- check your data, and
- carry out various kinds of analyses.

In order to get started with an existing example dataset, just click this link.
Press here to learn more ABOUT the EDICTOR, or here to get further HELP.
If you use the EDICTOR in your work, please cite the paper by List (2017) which can be downloaded here.

Figure 2: Loading data with EDICTOR.



ID	DOCULECT	CONCEPT	IPA	TOKENS	COGID
3	Danish	all	æˀl	æˀ l	4
6	Dutch	all	ɑlə	ɑ l ə	4
2	English	all	ɔːl,aːl	oː l	4
1	German	all	al	a l	4
5	Icelandic	all	at͡ɬir	a t͡ɬ i r	8
7	Norwegian	all	ɑlə	ɑ l ə	4
4	Swedish	all	alː	a lː	4
10	Danish	ashes	asg	a s g	9
13	Dutch	ashes	ɑs	ɑ s	9
9	English	ashes	æʃ	æ ʃ	9

Figure 3: Wordlist panel in EDICTOR.

through the EDICTOR. In the top panel, the Menu panel, basic statistics about the file is displayed (i.e., that it contains 22 rows, 7 languages, and 3 concepts). On the top-right of the Menu panel, you can see four different buttons:

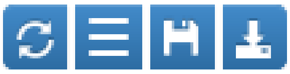


Figure 4: Buttons for refreshing, saving, and loading files.

If you hover over the buttons with your mouse, information on their function will be displayed. The first button serves to refresh the data. If you have the impression that the EDICTOR is not doing the right thing, it may be useful to press this button to make sure all data is correctly analysed. The second button will switch of the three basic Filters provided by the Menu panel (see below for more details on the filters). The third and fourth button are essential to export the data after you have edited them. In general, the first button stores the data internally in your webbrowser and does not save them forever (especially if you have rigid privacy options that delete cookies and internal storage after closing your webbrowser). The fourth button will "download" the data, i.e., it will export them in text-form and usually store them in your "Download" folder of your system. When working with the EDICTOR, I recommend to press both buttons regularly all 10 minutes to make sure no data is lost. This can be conveniently done with help of the shortcuts `<CRTL>S` and `<CTRL>E`. When re-entering an EDICTOR session, all you have to do is then to load the file you last edited into the system, and you can keep on working where you ended.

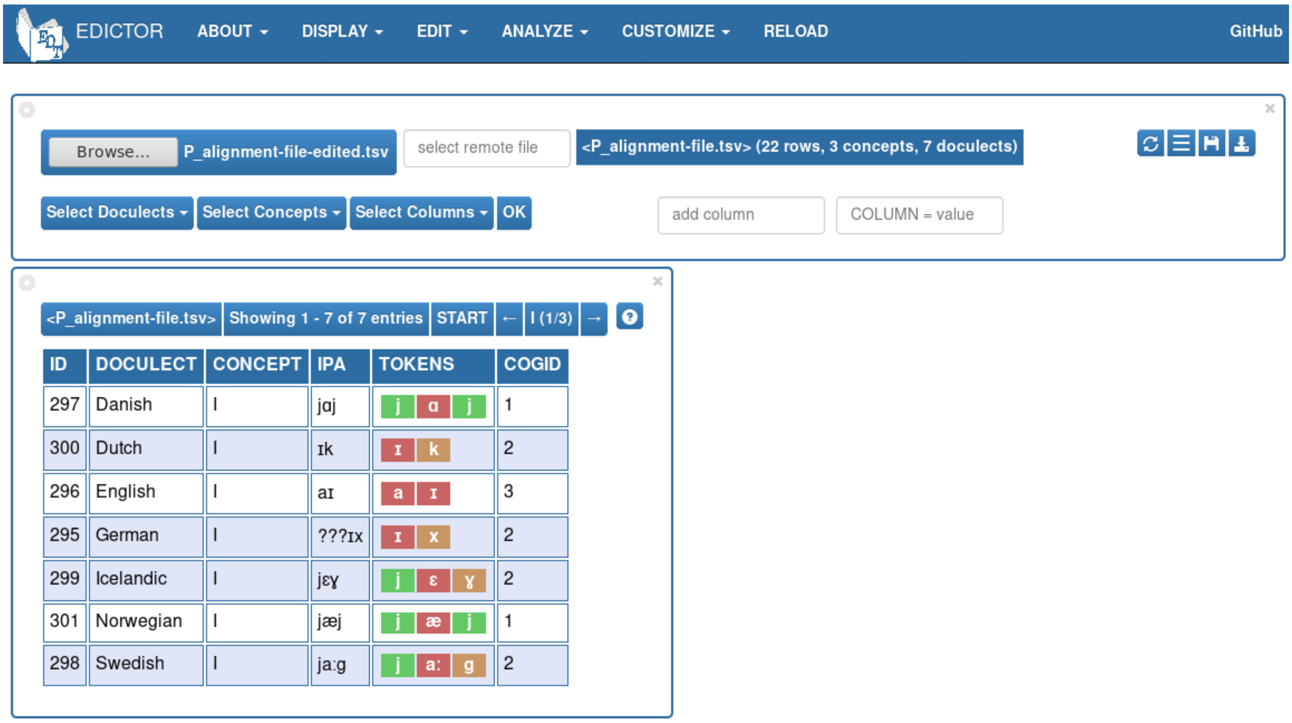
This is further enhanced by the fact that the EDICTOR will remember certain operations and write them to the text-files. For example, when you selected only one concept with help of the Filter options in the Menu panel and then export your file, the EDICTOR will remember this and upon re-import it will only render those concepts which you selected. You can test this behaviour by taking the file `P_alignment-file-edited.tsv` and importing it into the EDICTOR. You will see that it only displays the first concept ("I") if you use this file, but all concepts if you use the `P_alignment-file.tsv`.

Each panel has a little circle symbol on its top left. By clicking on this symbol, you can drag the panel and arrange them to your needs. Each panel can further be switched off by clicking on the cross in the top-right. In order to switch a panel back on, you need to go to the menu in the very top of the EDICTOR and click on the respective panel. Most panels also offer quick help. In order to view it, press on the top-right button with the question-mark. Press on it again to display the panel instead of the help message.

If you click in the top-level menu on DISPLAY->SETTINGS, a new window will open which allows you to modify or set certain options.

By hovering with the mouse over the optionso on the left of this menu, more information on the different functions of these options will be shown. What is crucial for cognate annotation is to tell the EDICTOR in which column your cognate sets are stored, and in which column your alignments can be found. Usually the EDICTOR guesses independently, but you can likewise specify your own preferred column for cognate sets or alignments.

The EDICTOR will also "highlight" certain values, i.e., render them differently than they appear in your original data, as you can see when looking at the TOKENS column. To change this behaviour, you can modify the values for the Highlight option. Similarly, if you do not want that the data you type is automatically treated as X-SAMPA input, you should empty the values in the X-SAMPA option. Having modified your preferred options, you should press on the REFRESH button on the top-right of the Settings panel.



ID	DOCULECT	CONCEPT	IPA	TOKENS	COGID
297	Danish	I	jɑj	j ɑ j	1
300	Dutch	I	ɪk	ɪ k	2
296	English	I	aɪ	a ɪ	3
295	German	I	???ɪx	ɪ x	2
299	Icelandic	I	jɛɣ	j ɛ ɣ	2
301	Norwegian	I	jæj	j æ j	1
298	Swedish	I	jaːg	j aː g	2

Figure 5: Loading options along with the file in EDICTOR.

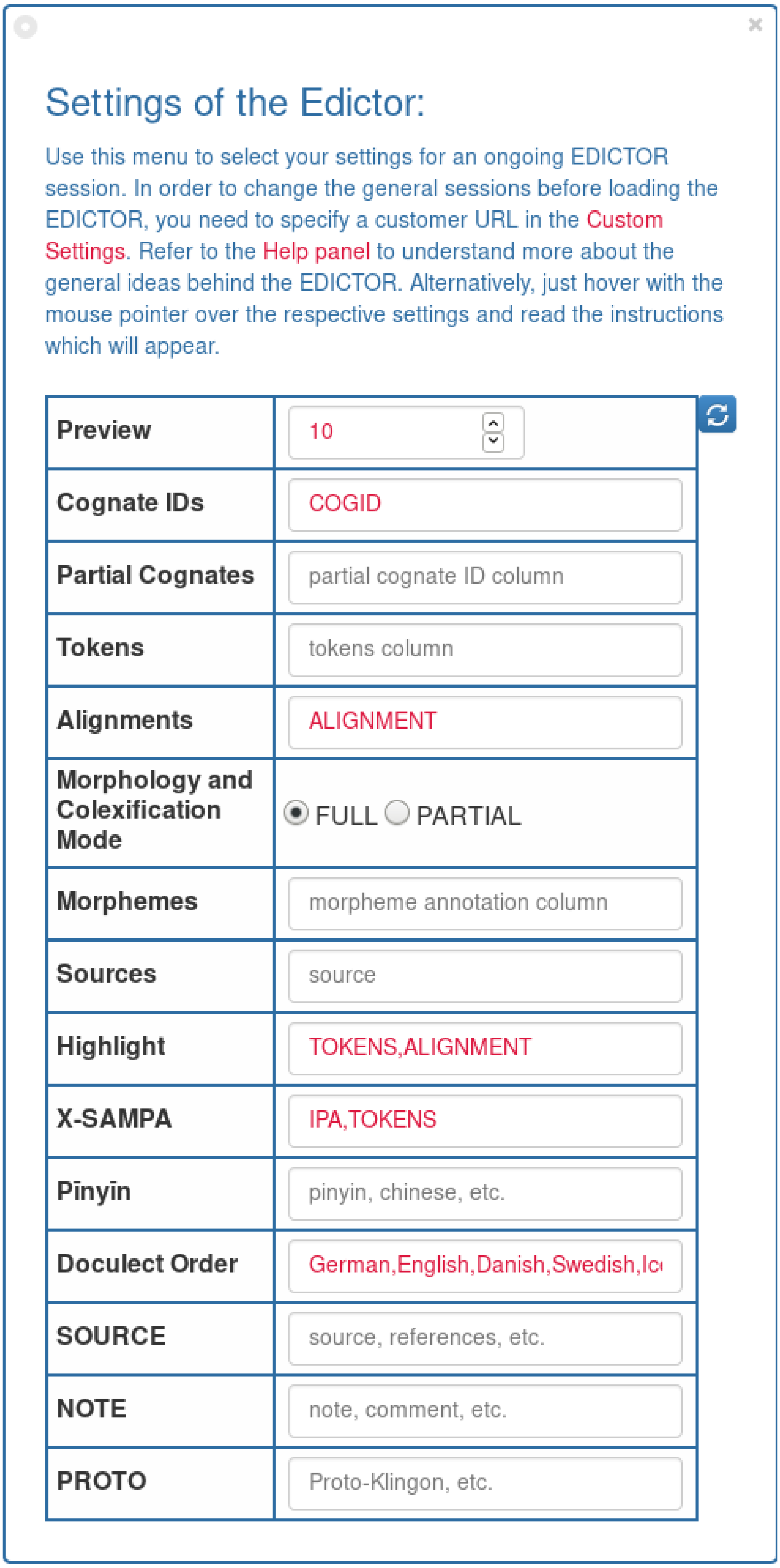
3.3 Hands on the Data: The Wordlist Panel

The Wordlist is the central view of the EDICTOR and all other ways to inspect and edit the data are centered around the Wordlist panel. If you open a file the first time, you may be surprised that you see only 10 items. This is the default preview of your data. For a quick browsing of the data, you can click on the buttons to the left and the right of the START symbol on top of the Wordlist panel. If you just loaded your data, there won't be a browse-button on the left of START, but only on the right, indicating the next number of items which you can inspect. In this way, you can go forward and backward through your data. You can conveniently scroll through the data with help of the page-up and page-down keys on your keyboard. If you prefer a larger preview, you can change this in the Settings. As a new feature, we currently test a new way to browse the wordlist data on a concept-per-concept basis. In order to do so, just click on the arrows in the top menu of the Wordlist panel.

Editing Data

A general feature of the Wordlist panel is to offer a quick way to edit the data. Essentially, each cell in the Wordlist can be edited by clicking one time on the cell and modifying the entry. In order to save the data, you need to press either the ENTER key, or one of the UP and DOWN keys, which will automatically bring you to the next cell above or beyond the cell you just edited.

Instead of editing the data, you can also use the keys to navigate quickly through your data. While UP and DOWN key work as expected, you need to press CTRL in addition to "LEFT" and "RIGHT" to jump from one cell to the cell at the left or the cell at the right. This is important, since otherwise you would have difficulties in editing the content of a cell, where LEFT and RIGHT are also needed to navigate.



Settings of the Edictor:

Use this menu to select your settings for an ongoing EDICTOR session. In order to change the general sessions before loading the EDICTOR, you need to specify a customer URL in the Custom Settings. Refer to the Help panel to understand more about the general ideas behind the EDICTOR. Alternatively, just hover with the mouse pointer over the respective settings and read the instructions which will appear.

Preview	10
Cognate IDs	COGID
Partial Cognates	partial cognate ID column
Tokens	tokens column
Alignments	ALIGNMENT
Morphology and Colexification Mode	● FULL ○ PARTIAL
Morphemes	morpheme annotation column
Sources	source
Highlight	TOKENS,ALIGNMENT
X-SAMPA	IPA,TOKENS
Pīnyīn	pinyin, chinese, etc.
Doculect Order	German,English,Danish,Swedish,Ic
SOURCE	source, references, etc.
NOTE	note, comment, etc.
PROTO	Proto-Klingon, etc.

Figure 6: Settings panel in the EDICTOR.

Interaction between Columns

One of the most crucial features of the EDICTOR is the interaction between different columns in your data. As a general rule, if a cognate set identifier is found in your data (default `COGID`), right-clicking in a cell with your mouse will open an alignment window in which you can align the data by simply clicking on it (see below). Right-clicking on other columns will store the data in cache. Right-clicking again will insert the stored value in the field on which you clicked.

Adding and Deleting Rows

If you want to add a new row to your data or delete an existing row, you can click on the values in the ID column. This will open a dialog from which you can select your options.

Filters

A crucial and important function for convenient data-editing are the filters provided by the Menu panel. Three filters are offered here, plus one largely experimental filter based on string comparison.

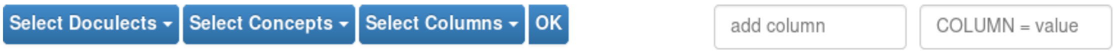


Figure 7: Filters for the Wordlist panel of the EDICTOR.

When applying a filter, you first carry out your selection by pressing on the respective buttons, and then need to press on the OK button on the right of the three main filters to submit the changes. To use the column-filter on the right of the filter menu, write your filter-query in the form `column=value` exclude all rows in which the column you selected does NOT contain the string, or write `column==value` to filter only those rows which have the exact same string as content. Thus, to filter, for example, all cognate sets which contain a 1, you could write `cogid=1`. In order to filter all cognate sets which are identical with 1, you should write `cogid==1` and then press enter (note that both capital and lowercase spelling are supported).

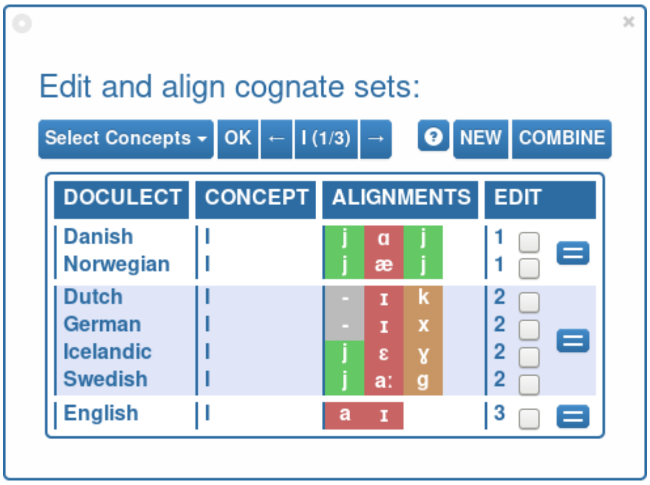
Sorting

By double-clicking on a given column (apart from the ID column), the data will be automatically sorted. Double-clicking again will sort the data in the opposite direction, double clicking a third time will restore the original order.

3.4 Finding Cognates: The Cognate Sets Panel

In order to offer an efficient and convenient way to identify cognate sets, the EDICTOR offers the Cognate Sets panel. This panel can be opened by clicking on EDIT in the main menu and then on Cognate Sets. It will open a new window in which, starting from the first concept in your data, all languages are listed in tabular form along with their alignments and cognate sets.

You can select concepts by using the filter on the top left (which also allows you to select more than one concept), or by clicking on the arrows to quickly navigate from one concept to the other. As a general



Edit and align cognate sets:

DOCULECT	CONCEPT	ALIGNMENTS	EDIT
Danish	I	j ɑ j	1 □
Norwegian	I	j æ j	1 □
Dutch	I	- ɪ k	2 □
German	I	- ɪ x	2 □
Icelandic	I	j ɛ ɣ	2 □
Swedish	I	j aː g	2 □
English	I	a ɪ	3 □

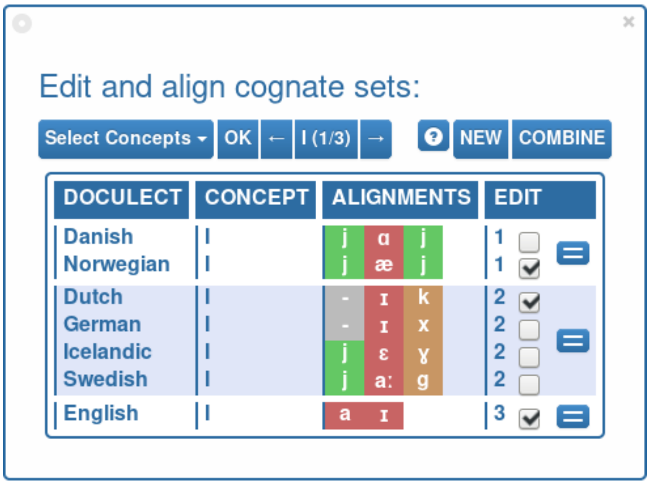
Figure 8: Cognate Sets panel of the EDICTOR.

rule, pressing the OK button will refresh your data, which is important when you carry out alignment analyses, as they will not be automatically refreshed after they were submitted.

Cognate Set Annotation

There are two basic operations that are supposed to help in editing cognate sets. One is called *new* and one is called *combine*. For each of the two procedures you need to select words by clicking on the selection fields on the right of the cognate sets table. By clicking on *new* you will create new cognate identifiers for the respective words. By clicking on combine, all cognate sets with the same cognate identifier as the ones you identified by clicking on them will be merged to form a large new group which retrieves the smallest of the cognate identifiers in the cluster.

Hence, if you press on one representative of each of the three cognate sets shown in the previous figure, and then on combine, it will merge all cognate sets into one class, automatically selecting the lowest value for the cognate ID.



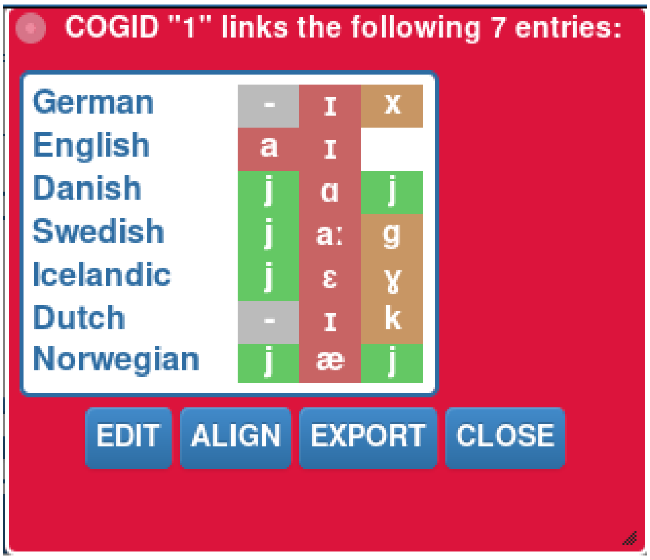
Edit and align cognate sets:

DOCULECT	CONCEPT	ALIGNMENTS	EDIT
Danish	I	j ɑ j	1 □
Norwegian	I	j æ j	1 ☑
Dutch	I	- ɪ k	2 ☑
German	I	- ɪ x	2 □
Icelandic	I	j ɛ ɣ	2 □
Swedish	I	j aː g	2 □
English	I	a ɪ	3 ☑

Figure 9: Annotation of cognate sets.

Alignment of Cognate Sets

Having assigned your words to cognate sets, you should align your data. This can be conveniently done by clicking on the alignment symbol next to each cognate set in the EDIT field. If you press this button, a window will pop up, offering different options for the alignment analysis.



COGID "1" links the following 7 entries:

German	-	ɪ	x
English	a	ɪ	
Danish	j	ɑ	j
Swedish	j	aː	g
Icelandic	j	ɛ	ɣ
Dutch	-	ɪ	k
Norwegian	j	æ	j

Figure 10: Alignment of cognate sets.

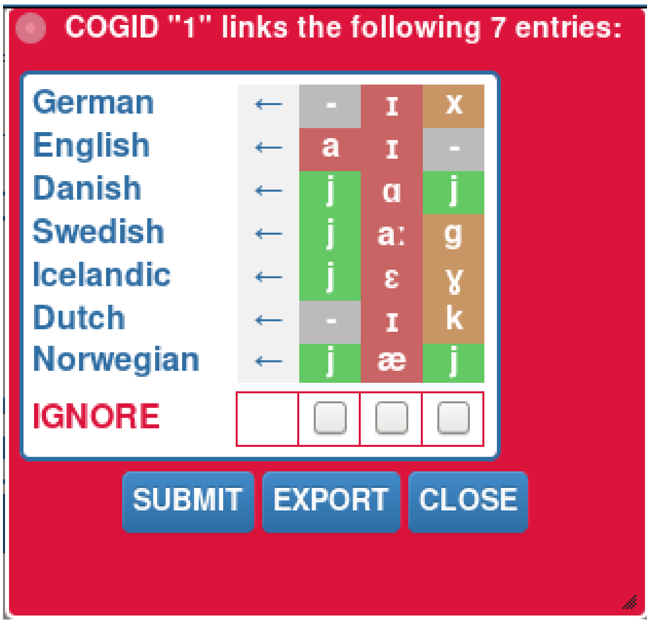
If you press on EDIT, you will enter the EDIT mode where you can push the sequences around by clicking on sounds or gaps. As a rule, if you click on a sound, this will insert a gap to the left of the sound. If you click on a gap, the gap will disappear. In order to drag a sound to the left, click on the arrow-symbol on the left of the sequence. If you want to push and pull more than one sequence at the same time, you can click on the language names in the left. This will mark them as being linked, and when you now click on any of those sequences, gaps will be inserted or deleted in all sequences assigned to the same group.

By clicking into the IGNORE row on the bottom of the alignment, you may mark parts of your alignment as irrelevant for your investigation. This may come in handy when you work with words where you can only identify common roots, while the morphology largely differs. It is also a convenient short-cut when working with compound words and high amounts of partial cognates, although we recommend to use the more powerful Partial Cognate Set panel for this kind of analysis.

As a final feature, you can have the EDICTOR align the data automatically, although the results may be disappointing. The algorithm used for this purpose is a very simple one that only takes the longest string and aligns all other strings to it. Nevertheless, especially when dealing with larger alignments, it may come in handy.

Integration of Wordlist and Cognate Set Panel

If you leave the Wordlist panel open while annotating your cognate sets, you will see that the Wordlist panel will change content when you switch to the next concept. In fact, it will always display the same concepts which you have selected in the Cognate Sets panel. Similarly, when using the concept-navigation in the Wordlist panel, the cognate sets will be synchronized accordingly. This makes it very



COGID "1" links the following 7 entries:

German	←	-	ɪ	x
English	←	a	ɪ	-
Danish	←	j	ɑ	j
Swedish	←	j	aː	g
Icelandic	←	j	ɛ	ɣ
Dutch	←	-	ɪ	k
Norwegian	←	j	æ	j
IGNORE		□	□	□

Figure 11: Editing alignments.

convenient to edit the data at the same time when carrying out the cognate judgments. For example, if you realize that an entry is wrongly segmented, you may just change it quickly in the Wordlist panel and then go on and align it again. The integration across panels is one of the crucial features of the EDICTOR and you will see, when exploring the tool further, that this interaction holds for many of the different panels.

As a further important point that facilitates working with cognate sets in the Wordlist panel is the fact that inserting any non-numeric character into the column which stores the cognate sets for a given analysis will automatically search for the next free identifier which has not yet been used. This prevents errors from assigning the same number multiple times and illustrates the usefulness of using specific interfaces rather than ad-hoc formats when preparing data in historical language comparison.

3.5 Enhanced Cognate Annotation: The Partial Cognate Sets Panel

With a few exceptions (Starostin 2013: 119-123), *partial cognacy* (Trask 2000:248), resulting from morphological processes such as derivation and compounding, has long been disregarded as a problem for phylogenetic reconstruction. Only recently more and more evidence has been assembled to show that it may even crucially impact on the results of phylogenetic reconstruction (List 2016, Hill and List 2017). Even if partial cognates are rare in a given dataset, it is useful to think of ways how to handle them consistently. While the Cognate Sets panel of the EDICTOR allows for quite some flexibility in assigning words to cognate sets, and especially the possibility to mark those parts which are relevant for a cognate set in the alignment, it may reach its limits if the amount of partial overlap in cognate words is large.

For this reason, the EDICTOR offers an explicit way to annotate partial cognates which is theoretically largely superior to binary cognate coding in so far as it allows to assign only parts of words to different cognate sets. The disadvantage is that unless the data can be obviously segmented into morphemes, as in the case of many SEA languages, users will have to do this manually. On the other hand, given that linguists who seek to reconstruct a language families history usually demand the highest level of expertise anyway, it seems that there is no serious way around the problem of segmenting the data morphologically, even if one only splits off the root from the rest.

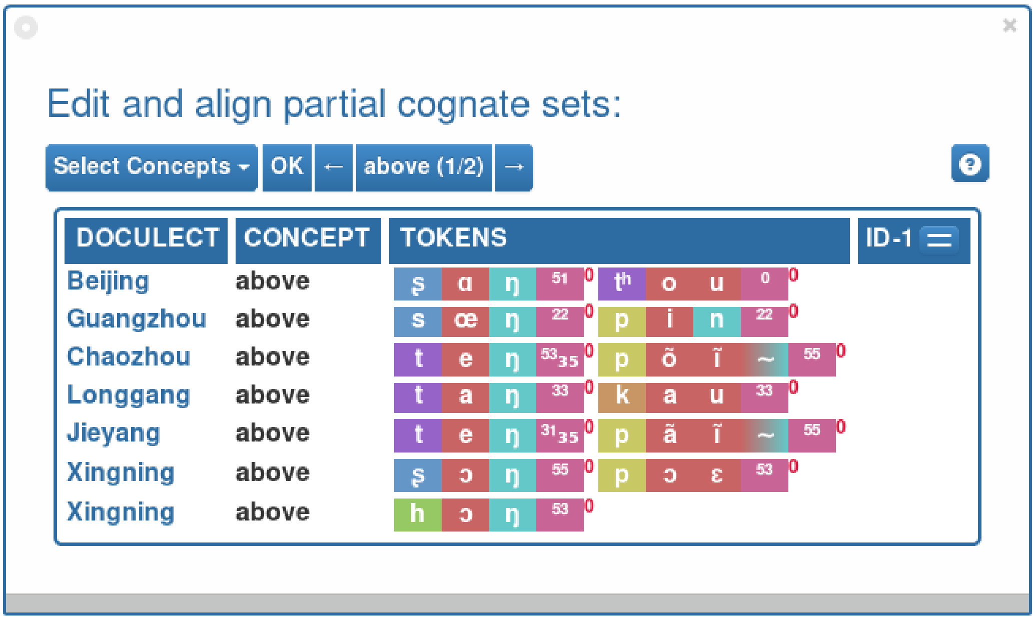
As mentioned before, morphological segmentation can be carried out by splitting the TOKENS with help of the plus-character (+) or the underscore (_). Although both characters may have a different meaning for linguists, they are not further distinguished when it comes to annotating partial cognates.

Preliminaries for Partial Cognate Identification

In order to annotate partial cognates with the EDICTOR, your need to specify the column in which you want to store partial cognate sets. In fact, even if your data is not morphologically segmented, you can just specify a column and annotated your data with help of the Partial Cognate Sets panel, and you may even think that it is even more intuitive and convenient to use than the classical Cognate Sets panel, given its improved operations. However, if you face problems in identifying cognates consistently in your data or aligning your words, you should definitely go through the pain of annotating your data for morpheme boundaries and assigning cognate sets on the basis of partial cognates.

The default name for partial cognate sets is `COGIDS`. If you prepared a file in which you add such a column, you can immediately use the Partial Cognate Sets panel. If this is not the case, you will have to specify your column in the Settings panel. To allow you to test this behaviour, we prepared a test-file, called `E_chinese.tsv`, which provides partial cognates and pre-segmented data for six Chinese dialects (prepared by Yunfan Lai and Johann-Mattis List as part of the CALC project).

If you open the panel, a window similar to the Cognate Sets window will pop up, with the difference being that cognate identifiers are now given in additional columns, with one empty column each time, showing you the next cognate identifier to which you can assign new data.

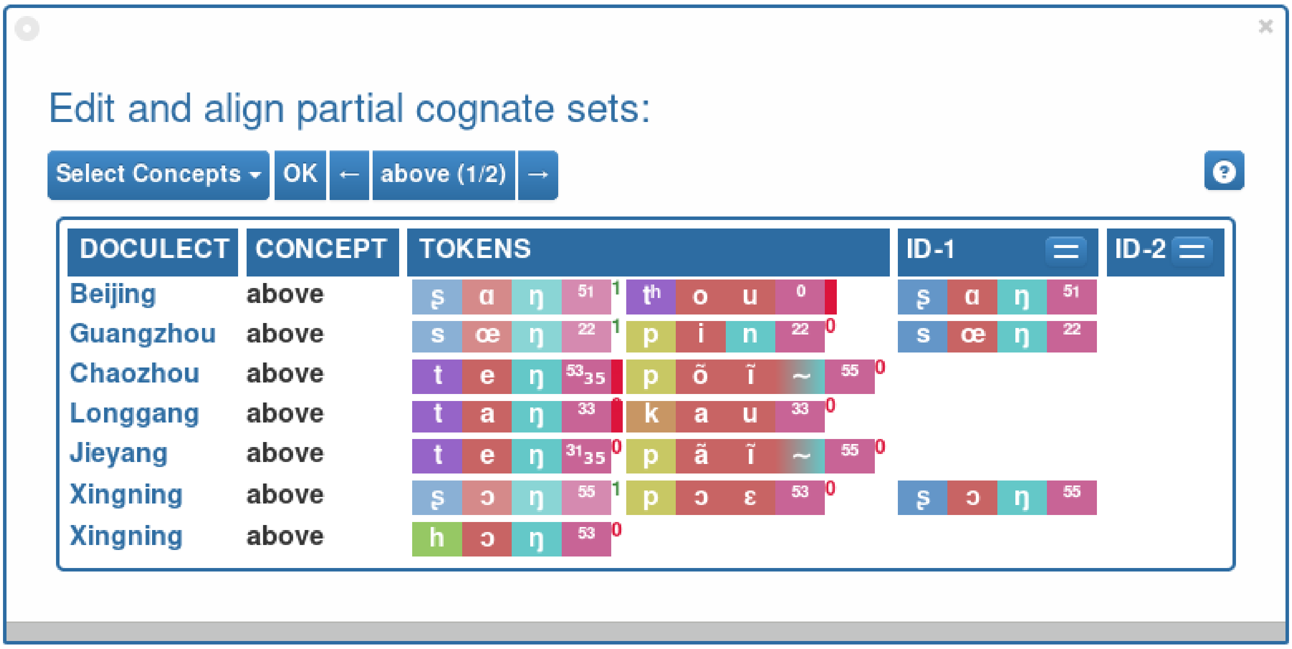


Edit and align partial cognate sets:

DOCULECT	CONCEPT	TOKENS	ID-1
Beijing	above	ʂ ɑ ŋ 51 0 tʰ o u 0 0	
Guangzhou	above	s œ ŋ 22 0 p i n 22 0	
Chaozhou	above	t e ŋ 5335 0 p õ ĩ ~ 55 0	
Longgang	above	t a ŋ 33 0 k a u 33 0	
Jieyang	above	t e ŋ 3135 0 p ã ĩ ~ 55 0	
Xingning	above	ʂ ɔ ŋ 55 0 p ɔ ɛ 53 0	
Xingning	above	h ɔ ŋ 53 0	

Figure 12: Partial Cognate Sets in the EDICTOR.

If no cognate sets have been distributed so far, all the morphological segments in your data will be given a little 0 in red to their top right. You can now select morphemes across the languages by clicking on them. Once you have done this, you can click on the next free ID on the right to assign the morphemes to this cognate set. If you want to refine an analysis, you can delete an item from a given cognate set by clicking on it inside the column with the cognate identifiers.



DOCULECT	CONCEPT	TOKENS	ID-1	ID-2
Beijing	above	ʂ ɑ ŋ 51 1 tʰ o u 0	ʂ ɑ ŋ 51	
Guangzhou	above	s œ ŋ 22 1 p i n 22 0	s œ ŋ 22	
Chaozhou	above	t e ŋ 5335 p õ ĩ ~ 55 0		
Longgang	above	t a ŋ 33 k a u 33 0		
Jieyang	above	t e ŋ 3135 0 p ã ĩ ~ 55 0		
Xingning	above	ʂ ɔ ŋ 55 1 p ɔ ɛ 53 0	ʂ ɔ ŋ 55	
Xingning	above	h ɔ ŋ 53 0		

Figure 13: Editing partial cognate sets.

You can also align the data, simply by clicking on the alignment button, and you can likewise sort the data in various ways, by rightclicking on a given column with a cognate set identifier. If you have a look at the Wordlist panel after assigning cognate identifiers, you will see that the COGIDS column will regularly contain as many numbers, separated by a space, as there are morphemes per word. The numeric annotation thus follows the order of morphemes in the base form of the word and provides a unique identifier similar to the cognate set identifiers used by EDICTOR for full cognates.

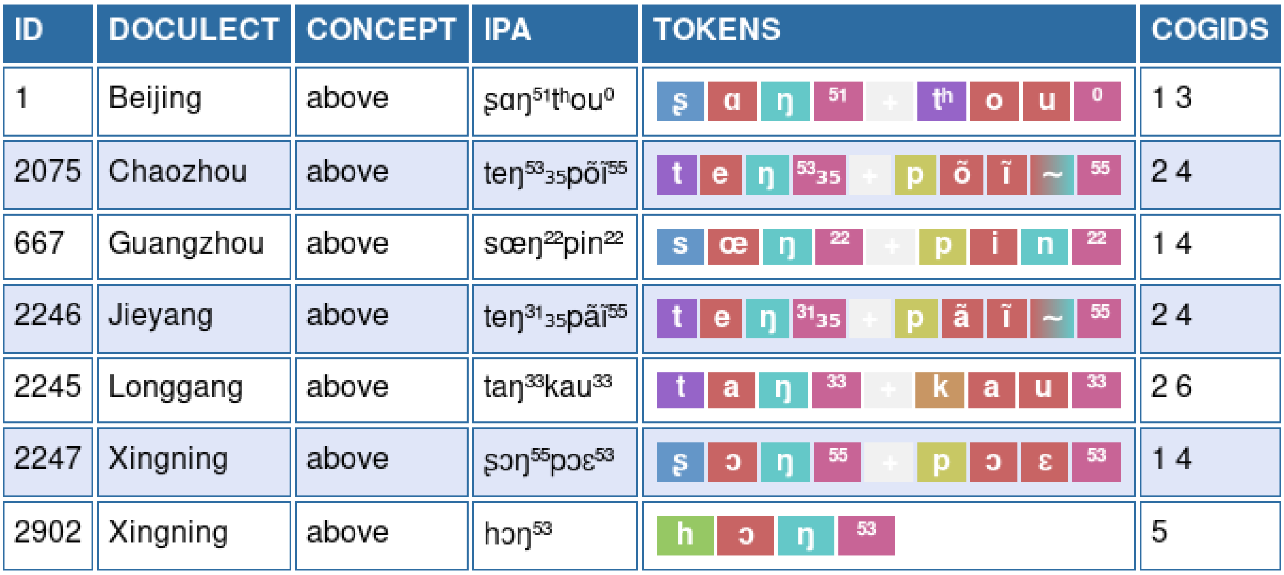
4 Analysing Data with EDICTOR

In addition to allowing for a consistent annotation of linguistic data for the purpose of historical language comparison, the EDICTOR also offers different ways to analyze the data. These procedures and methods are summarized under the ANALYZE header in the main menu of the EDICTOR. Using these additional features may require additional columns in your file, as you will see in the following.

4.1 Inspecting Sounds and Transcriptions: The Phonology Panel

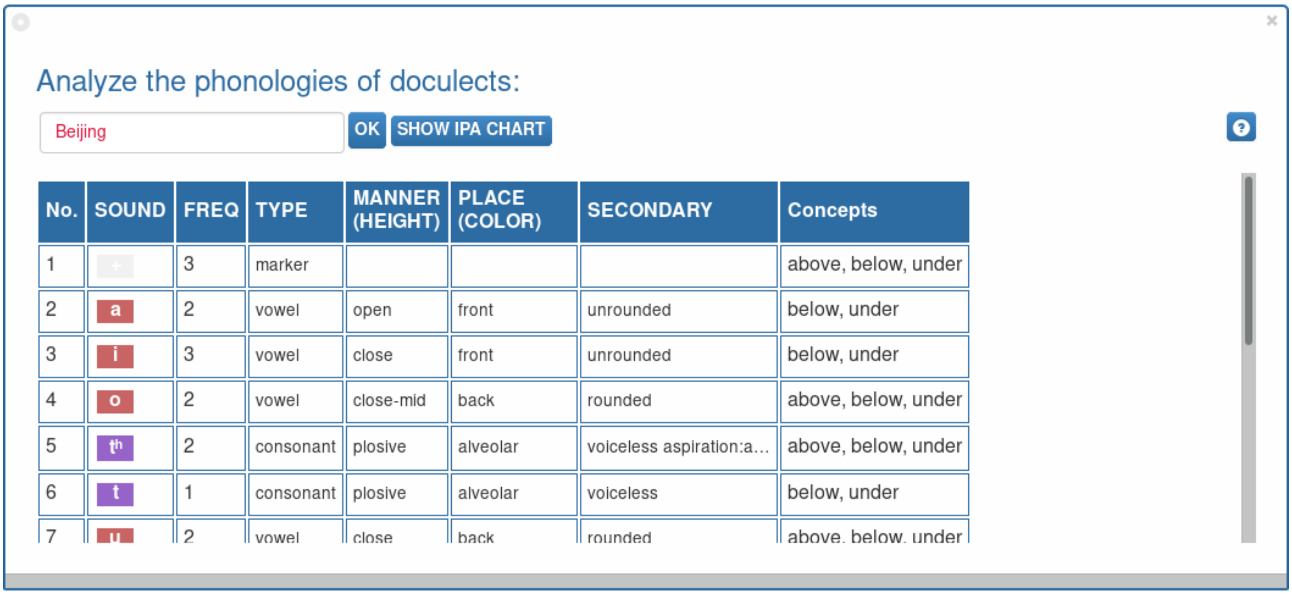
Let us quickly start by exploring the Phonology panel. When loading our sample file `E_chinese.tsv` and clicking on ANALYZE->Phonology, a new panel will open that allows you to select one of the doculects. Let us choose Beijing Chinese, as the phonology is farely well understood. The panel will now list all of the sounds encountered in the TOKENS column for Beijing Chinese:

By double-clicking on the different headers, you can sort the data accordingly. The columns list apart from the sound itself and its frequency of occurrence also some basic features (as far as the sounds are known to EDICTOR, which is again a reason why one should try to use the standards that are



ID	DOCULECT	CONCEPT	IPA	TOKENS	COGIDS
1	Beijing	above	ʂɑŋ^{51}tʰou^{0}	ʂ ɑ ŋ 51 + tʰ o u 0	1 3
2075	Chaozhou	above	teŋ^{53}$_{35}$põĩ^{55}	t e ŋ 53$_{35}$ + p õ ĩ ~ 55	2 4
667	Guangzhou	above	sœŋ^{22}pin^{22}	s œ ŋ 22 + p i n 22	1 4
2246	Jieyang	above	teŋ^{31}$_{35}$pãĩ^{55}	t e ŋ 31$_{35}$ + p ã ĩ ~ 55	2 4
2245	Longgang	above	taŋ^{33}kau^{33}	t a ŋ 33 + k a u 33	2 6
2247	Xingning	above	ʂɔŋ^{55}pɔɛ^{53}	ʂ ɔ ŋ 55 + p ɔ ɛ 53	1 4
2902	Xingning	above	hɔŋ^{53}	h ɔ ŋ 53	5

Figure 14: Representation of partial cognates in the Wordlist panel.



Analyze the phonologies of doculects:

Beijing

No.	SOUND	FREQ	TYPE	MANNER (HEIGHT)	PLACE (COLOR)	SECONDARY	Concepts
1	+	3	marker				above, below, under
2	a	2	vowel	open	front	unrounded	below, under
3	i	3	vowel	close	front	unrounded	below, under
4	o	2	vowel	close-mid	back	rounded	above, below, under
5	tʰ	2	consonant	plosive	alveolar	voiceless aspiration:a…	above, below, under
6	t	1	consonant	plosive	alveolar	voiceless	below, under
7	u	2	vowel	close	back	rounded	above, below, under

Figure 15: Phonology inspection with the EDICTOR.

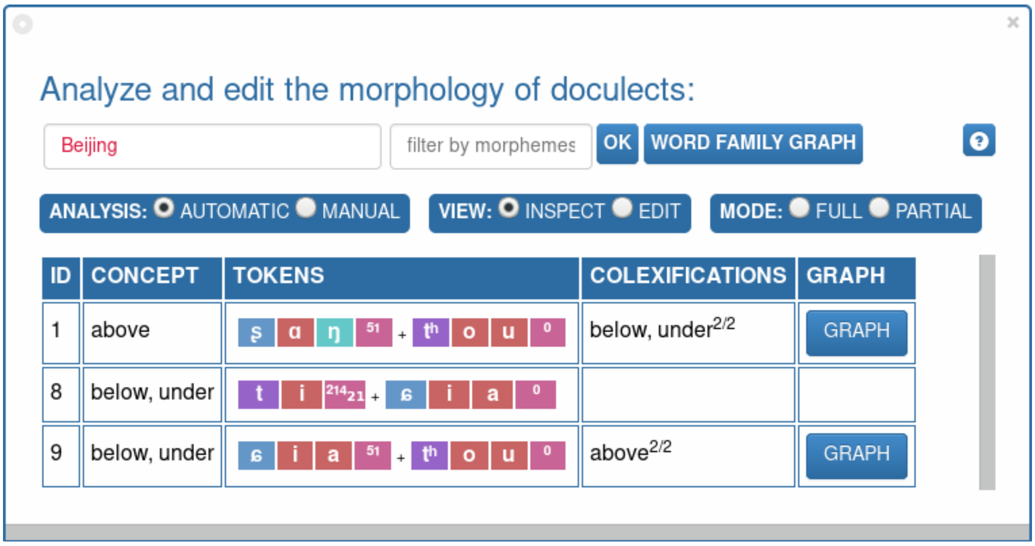
underlying the tool). The column on the right is called Concepts and it shows all the concepts in which the sound occurs. If you click on the cell with the concepts, the Wordlist panel will be automatically filtered, showing only those concepts for Beijing Chinese where the sound occurs. This is convenient for checking erroneous sounds in your data. By clicking on the button SHOW IPA CHART, you can see an IPA chart which assembles all sounds which can be identified in the relevant cells. Sounds which cannot be identified will be listed in isolation. In our sample, we don't have many words, so it is not entirely fun or enlightening to review the sounds, but in general, the IPA chart is useful to detect missing spots in regular series, be it due to sparse data or gaps in the phonological system.

4.2 Enhanced Morphological Annotation: The Morphemes Panel

We have mentioned partial cognates before, especially pointing to the importance of using this way of cognate annotation if the morphology of your languages is complex. An additional way to account for complex morphology is to use the Morphology panel to analyze full and partial colexifications in your data. This is extremely useful to detect dependencies of your cognate sets resulting from underspecification as in languages which have the same word for "arm" and "hand", but also to assign language-internal cognates.

Basic Inspection of Partial Colexifications

In order to carry out this kind of analysis, you will need a column called MORPHEMES (which is detected and assigned as such by default) or you will have to assign one of your columns to this group manually by using the Settings panel. In our case, we use the column MORPHEMES, which we already added to the file `E_chinese.tsv`. When clicking on ANALYZE->Morphology, a new panel will open and you can again specify a language variety. We will again use Beijing Chinese for this purpose. Make sure to select PARTIAL as MODE of inspection, and leave the other items at the default provided by the tool. The resulting analysis will look as follows:



Analyze and edit the morphology of doculects:

Beijing

ANALYSIS: ● AUTOMATIC ○ MANUAL

VIEW: ● INSPECT ○ EDIT

MODE: ○ FULL ○ PARTIAL

ID	CONCEPT	TOKENS	COLEXIFICATIONS	GRAPH
1	above	ʂ ɑ ŋ 51 + tʰ o u 0	below, under$^{2/2}$	GRAPH
8	below, under	t i 21421 + ɕ i a 0		
9	below, under	ɕ i a 51 + tʰ o u 0	above$^{2/2}$	GRAPH

Figure 16: Enhanced morphology inspection with the EDICTOR.

The information with which you are provided here can be read as follows: The entry "above" in Beijing Chinese is partially colexified with the entry "below, under". The superscript numbers divided by a slahs after the entry in the COLEXIFICATIONS cell indicate that we are talking about the second element of "above" and the second elemnt of "below, under". If you click on the GRAPH button, a pop-up window will open and visualize the structure:

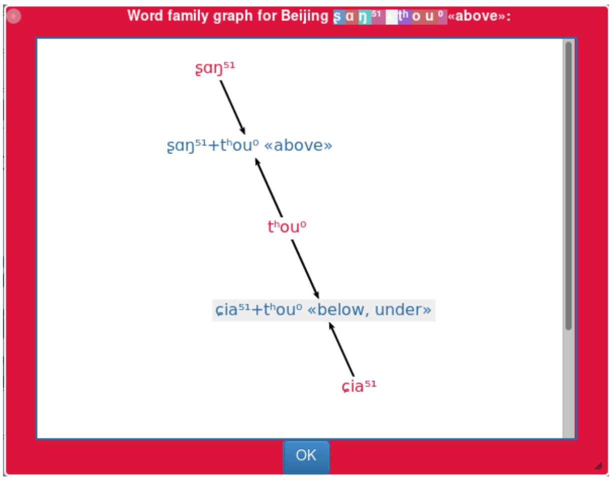

Word family graph for Beijing ʂ ɑ ŋ ^{51} + tʰ o u ^{0} «above»:
ʂɑŋ^{51}
ʂɑŋ^{51}+tʰou^{0} «above»
tʰou^{0}
ɕia^{51}+tʰou^{0} «below, under»
ɕia^{51}
OK

Figure 17: Graph representation of morphological relations.

This is bipartite graph constructed from shared elements in the data (see Hill and List 2017) for details on this datastructure.

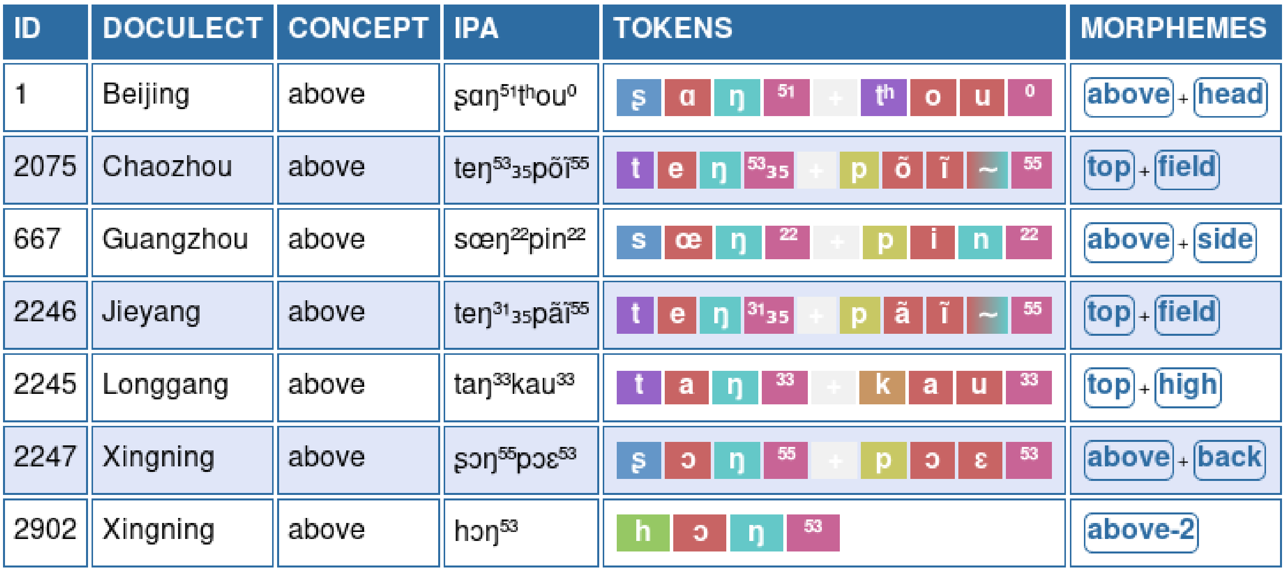
User-Annotated Partial Colexifications

An additional and more powerful way to not only analyze but also annotated your data is to provide your judgments on language-internal cognacy. In our data for the Chinese dialects, suffixes as well as root elements of the counterparts of "above" and "below, under" are recurring. In Beijing Chinese, for example, [tʰou^{0}] is a frequently recurring suffix, which formerly meant "head". In the word for "above" [ʂɑŋ^{51}tʰou^{0}], the first element usually functions as a noun, but it could be labeles as bearing the main meaning "above". We could thus characterize the word as "above" + "head", and if we insert these values, separated by a space into the MORPHEMES cell of the word, it will be rendered in a specific form that indicates the specific meaning of the elements of the word. The sample data has been annotated for all languages in this way, as you can see when selecting the column MORPHEMES in the major filter sections of the Wordlist panel:

As an additional important feature which helps to make binary cognate decisions for partial cognates more transparent, you can right-click on a given morpheme in the morpheme panel in order to indicate that it does not contribute to the overall cognate decision you make. If you do so, the labels in the MORPHEME column will be internally prefixed by an underscore. In the Wordlist panel, this will be rendered by decreasing the font-size of the relevant morpheme while at the same time increasing its transparency. As a result, you can easily spot which elements share the main morpheme in your data:

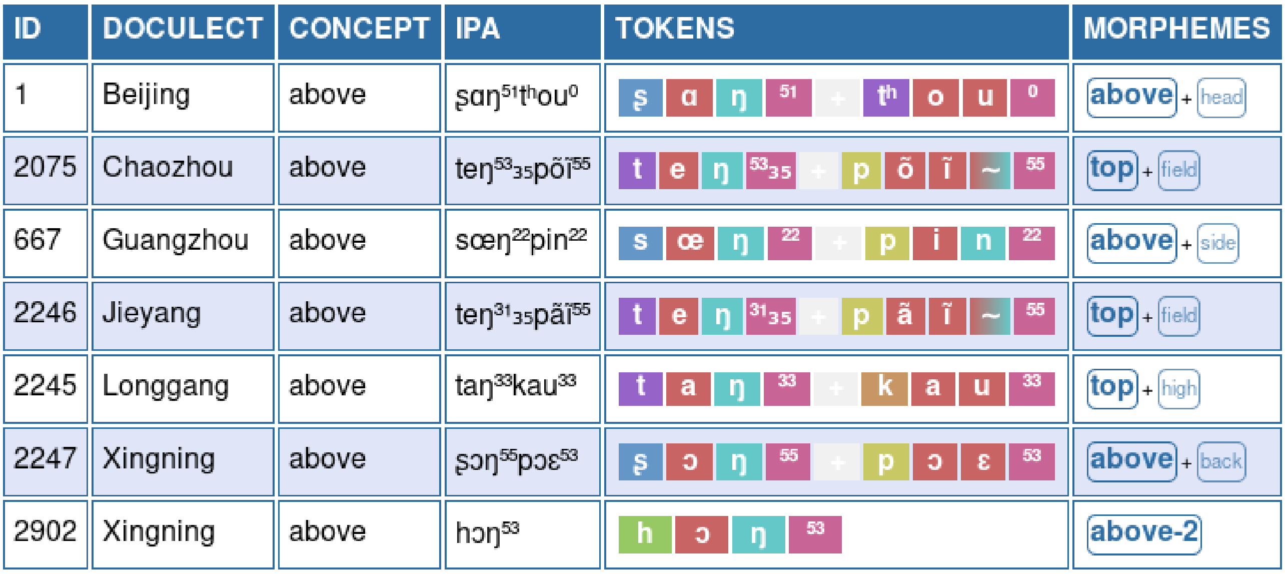
Browsing User-Annotated Partial Colexifications

You can also browse your user-annotated partial colexifications in the Morphology panel. In order to do so, just set the radio button in ANALYSIS to MANUAL and click on OK. The resulting table combines



ID	DOCULECT	CONCEPT	IPA	TOKENS	MORPHEMES
1	Beijing	above	ʂɑŋ^{51}tʰou^{0}	ʂ ɑ ŋ ^{51} + tʰ o u ^{0}	above + head
2075	Chaozhou	above	teŋ$^{53}{}_{35}$põĩ^{55}	t e ŋ $^{53}{}_{35}$ + p õ ĩ ~ ^{55}	top + field
667	Guangzhou	above	sœŋ^{22}pin^{22}	s œ ŋ ^{22} + p i n ^{22}	above + side
2246	Jieyang	above	teŋ$^{31}{}_{35}$pãĩ^{55}	t e ŋ $^{31}{}_{35}$ + p ã ĩ ~ ^{55}	top + field
2245	Longgang	above	taŋ^{33}kau^{33}	t a ŋ ^{33} + k a u ^{33}	top + high
2247	Xingning	above	ʂɔŋ^{55}pɔɛ^{53}	ʂ ɔ ŋ ^{55} + p ɔ ɛ ^{53}	above + back
2902	Xingning	above	hɔŋ^{53}	h ɔ ŋ ^{53}	above-2

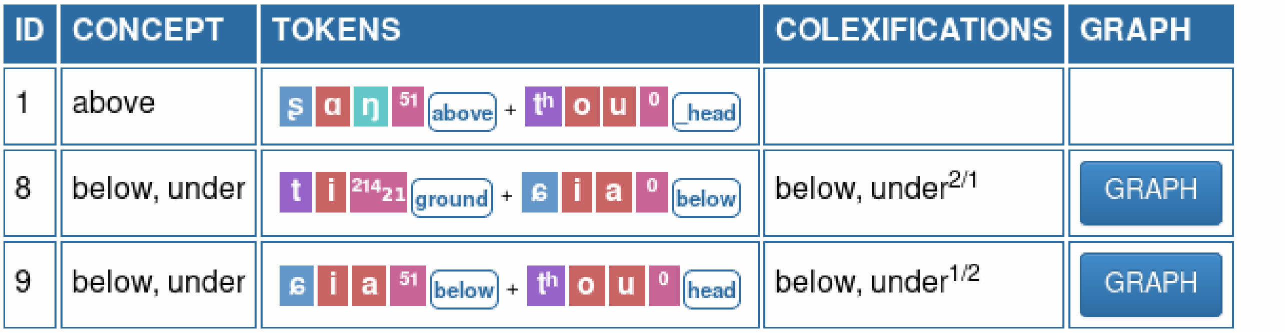
Figure 18: User-annotation of partial colexifications and language-internal cognates.



ID	DOCULECT	CONCEPT	IPA	TOKENS	MORPHEMES
1	Beijing	above	ʂɑŋ^{51}tʰou^{0}	ʂ ɑ ŋ ^{51} + tʰ o u ^{0}	above + head
2075	Chaozhou	above	teŋ$^{53}{}_{35}$põĩ^{55}	t e ŋ $^{53}{}_{35}$ + p õ ĩ ~ ^{55}	top + field
667	Guangzhou	above	sœŋ^{22}pin^{22}	s œ ŋ ^{22} + p i n ^{22}	above + side
2246	Jieyang	above	teŋ$^{31}{}_{35}$pãĩ^{55}	t e ŋ $^{31}{}_{35}$ + p ã ĩ ~ ^{55}	top + field
2245	Longgang	above	taŋ^{33}kau^{33}	t a ŋ ^{33} + k a u ^{33}	top + high
2247	Xingning	above	ʂɔŋ^{55}pɔɛ^{53}	ʂ ɔ ŋ ^{55} + p ɔ ɛ ^{53}	above + back
2902	Xingning	above	hɔŋ^{53}	h ɔ ŋ ^{53}	above-2

Figure 19: Inspecting user-annotation of morphological structure.

the information in the TOKENS and the MORPHOLOGY column, which maks it convenient to inspect the morpheme structure of the compounds:



ID	CONCEPT	TOKENS	COLEXIFICATIONS	GRAPH
1	above	ʂ ɑ ŋ 51 above + tʰ o u 0 _head		
8	below, under	t i 214 21 ground + ɕ i a 0 below	below, under$^{2/1}$	GRAPH
9	below, under	ɕ i a 51 below + tʰ o u 0 head	below, under$^{1/2}$	GRAPH

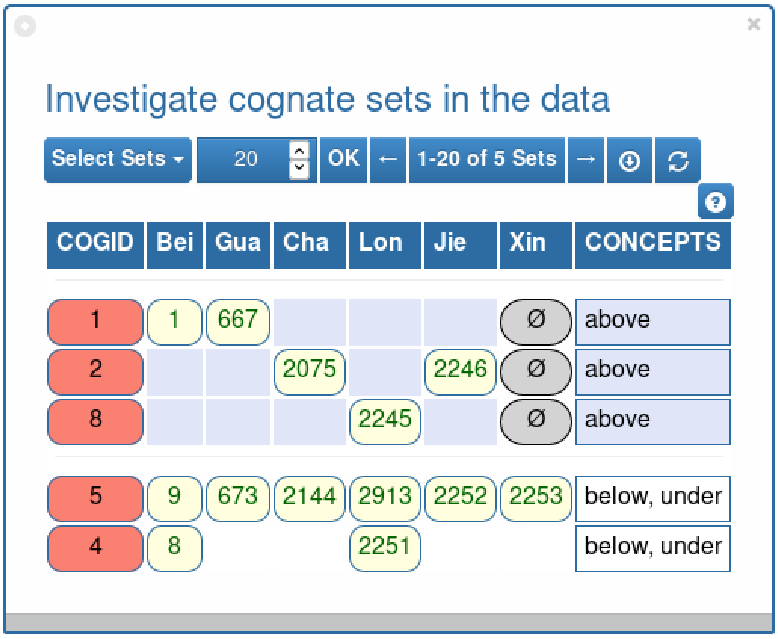
Figure 20: Browsing user-annotation and partial colexifications.

If you now set the radio button in VIEW from INSPECT to EDIT and click again on OK, you can edit the morphemes by clicking into the MORPHEMES cell without editing the data in the Wordlist panel. This will prove useful if you are working on language families whose morphology you do not yet fully understand. As a general rule, the algorithms underlying EDICTOR will identify all morphemes which you assign the same string (without spaces, so use an underscore or a dash, if you want to annotate longer phrases) as being judged to be cognate *inside* the given language. This principle will also prove extremely useful if you try to identify cognates across different meanings, as polysemy is the first step to identify cognates across meaning classes.

4.3 Analysing Cognate Sets: The Cognate Sets Panel

If you assign words to cognate sets, it is useful to check the overall patterning of the cognate sets in your data. In order to do so, make sure that you have selected *all* concepts in your data, using the concepts filter, and then click on ANALYZE->Cognate Sets. A pop-up window will open and ask whether you want to inspect full cognates or partial cognates. As we have already assigned our Chinese words to full cognate sets, following our principle of user-defined salience of morphemes in words briefly mentioned above, we select "full" for this test. In order to render the analysis more interesting, however, we will first delete the words in Xingning Chinese for the concept "above" by clicking on the ID field and selecting "delete row". If we now click on OK, a table will appear that shows all the cognate sets in the data:

This table shows the cognate sets in the first row (clicking on them will open the alignemnt view), and the words (represented by their unique IDs) in each cell in which a language has a valid reflex. Missing data is represented as shown for Xingning Chinese for the concept "above", which we deleted before, namely in gray background with a the Ø symbol, which serves to indicate gaps in the EDICTOR. When clicking on the entries in the table, you can view the word underlying the cognate set. By clicking on the export-button, you can even save the data in Nexus-format:



COGID	Bei	Gua	Cha	Lon	Jie	Xin	CONCEPTS
1	1	667				Ø	above
2			2075		2246	Ø	above
8				2245		Ø	above
5	9	673	2144	2913	2252	2253	below, under
4	8			2251			below, under

Figure 21: Analyzing cognate sets.

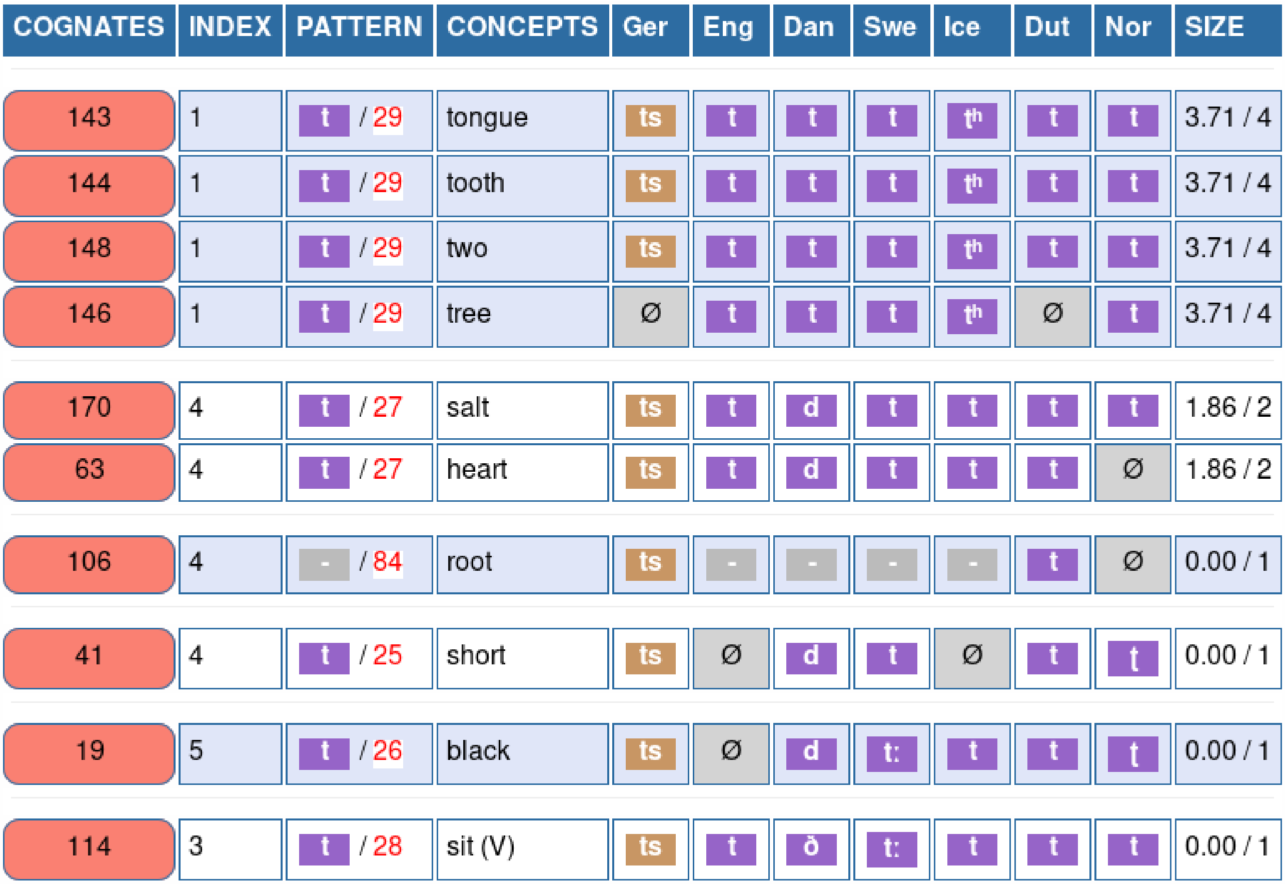
```
#NEXUS

BEGIN CHARACTERS;
    concept_1=1-3; [above]
    concept_2=4-5; [below, under]

END; [CHARACTERS]
BEGIN DATA;
DIMENSIONS NTAX=6 NCHAR=5;
FORMAT DATATYPE=STANDARD SYMBOLS="10" GAP=- MISSING=? INTERLEAVE=yes;
MATRIX
Beijing             10011
Guangzhou           10010
Chaozhou            01010
Longgang            00111
Jieyang             01010
Xingning            ??710
;
END;
```

This text file can be directly be used with software packages such as SplitsTree to create splits networks of distance-based trees. It illustrates another advantage of properly preparing your data with help of software tools: while you may have an extremely hard time in trying to convert your data in Nexus format when doing it manually, the EDICTOR just provides you with the export for free.

4.4 Analysing Sound Correspondences: The Correspondences Panel

As a last illustration of the multitude of possibilities which the EDICTOR offers for consistent data annotation and inspection, let us have a look at the sound correspondences in our data. Here the EDICTOR applies a new method which was developed in close collaboration with Nathan W. Hill (for an early account List and Hill 2017) and is currently being finalized. This new method allows to inspect the correspondence patterns in the alignments of the data by clustering all alignment sites into compatible correspondence sets, thereby automatically checking the regularity of a given alignment site and therefore also the regularity of a given cognante set. This panel, the Correspondences panel, uses a rough approximate solution for the problem of identifying correspondence patterns, which would be otherwise too expensive to be applied from within JavaScript software running in the browser.

For our illustration, we use the file `E_germanic.tsv` which takes cognate judgments as provided by the Tower of Babel project for a sample of 7 Germanic varieties along with phonetic transcriptions added by List (2014). For this experiment, the data was further automatically aligned with help of LingPy. When loading this file into the EDICTOR and opening the Correspondence Patterns panel via ANALYZE->Correspondence Patterns, a window will open and asking for the underlying mode (partial of full cognacy). For the Germanic data, we assume full cognacy, which means we can click the OK button directly. The resulting table which opens shows all correspondence patterns observed in the data and automatically clusters them into groups of compatible sites. Note that the algorithm is not accurate, but comes very close to a good result. Better analyses with LingPy are currently in preparation. By adjusting the PREVIEW, we can define how many correspondence patterns we want to watch at the same time. You can further select specific sounds by using the filter provided on the left of the menu above the table. The sounds selected are automatically those sounds of the first language in the sample, which is German in our case. If you have proto-languages in your data, you can select one of them via the Settings panel, and it will be displayed as the first language in this analysis. For our purpose, we filter all instances of [t͡s] in German (they are scattered over the data, as they occur in 6 distinct patterns which are not compatible with each other.



COGNATES	INDEX	PATTERN	CONCEPTS	Ger	Eng	Dan	Swe	Ice	Dut	Nor	SIZE
143	1	t / 29	tongue	ts	t	t	t	tʰ	t	t	3.71 / 4
144	1	t / 29	tooth	ts	t	t	t	tʰ	t	t	3.71 / 4
148	1	t / 29	two	ts	t	t	t	tʰ	t	t	3.71 / 4
146	1	t / 29	tree	Ø	t	t	t	tʰ	Ø	t	3.71 / 4
170	4	t / 27	salt	ts	t	d	t	t	t	t	1.86 / 2
63	4	t / 27	heart	ts	t	d	t	t	t	Ø	1.86 / 2
106	4	- / 84	root	ts	-	-	-	-	t	Ø	0.00 / 1
41	4	t / 25	short	ts	Ø	d	t	Ø	t	ʈ	0.00 / 1
19	5	t / 26	black	ts	Ø	d	tː	t	t	ʈ	0.00 / 1
114	3	t / 28	sit (V)	ts	t	ð	tː	t	t	t	0.00 / 1

Figure 22: Analyzing correspondence patterns.

As we can see from this view, we get an immediate impression regarding the amount of different patterns in the data in which German shows a [t͡s]. As a matter of fact, all these patterns go back to the same proto-sound, but the environments which trigger changes in the daughter languages are all different. Swedish, for example, shows a retroflex in the word for “black” due to the fact that it is preceded by an [r]. The words for “root” are swapped in some languages, and “salt” and “heart” have the Proto-Germanic *t in final position instead of the initial position. In fact, given that we know the development of the Germanic languages rather well, these facts are not surprising. What is interesting, however, is how quickly we can assemble the data in this convenient form if we only supply the aligments and a rather consistent phonetic transcription to the algorithm.

5 Concluding Remarks

I hope that this tutorial has illustrated the usefulness of the EDICTOR tool in combination with Python libraries such as LingPy, the Concepticon API, or the Segments package. More work has to be done and will be done, both in LingPy and the EDICTOR. For those who want to assemble data in form of cognate sets for the purpose of linguistics reconstruction or phylogenetic reconstruction, however, the tools are useful already in their current form.

If you come to use any of the software packages discussed here, it is important to quote them accordingly in order to pay respect to all the work that we have invested in order to create those tools. The EDICTOR package should be quoted by quoting the paper by List (2017), and the LingPy package should be quoted in its 2.6 (forthcoming) version (List and Forkel 2016).

References

- Dunn, M. (ed.) (2012): **Indo-European lexical cognacy database (IELex).** http://ielex.mpi.nl/.
- Forkel, R., S. Greenhill, and J.-M. List (2017): **Cross-Linguistic Data Formats (CLDF).** Max Planck Institute for the Science of Human History: Jena.
- Greenhill, S., R. Blust, and R. Gray (2008): **The Austronesian Basic Vocabulary Database: From bioinformatics to lexomics**. *Evolutionary Bioinformatics* 4. 271-283.
- Hall, R. (1960): **On Realism in Reconstruction**. *Language* 36.2. 203-206.
- Hill, N. and J.-M. List (2017): **Challenges of annotation and analysis in computer-assisted language comparison: A case study on Burmish languages**. *Yearbook of the Poznań Linguistic Meeting* 3.1. 47–76.
- List, J.-M. (2014): **Sequence comparison in historical linguistics**. Düsseldorf University Press: Düsseldorf.
- List, J.-M., M. Cysouw, and R. Forkel (2016): **Concepticon. A resource for the linking of concept lists**. In: **Proceedings of the Tenth International Conference on Language Resources and Evaluation**. 2393-2400.
- List, J.-M. and R. Forkel (2016): **LingPy. A Python library for historical linguistics**. Max Planck Institute for the Science of Human History: Jena.
- List, J.-M. (2016): **Beyond cognacy: Historical relations between words and their implication for phylogenetic reconstruction**. *Journal of Language Evolution* 1.2. 119-136.
- List, J.-M. and N. Hill (2017): **Computer-assisted approaches to linguistic reconstruction. A case study from the Burmish languages**. Paper, presented at the workshop “Regularity of Sound Change” (2017/07/20-21, Cologne, Universität zu Köln).

- List, J.-M. (2017): **A web-based interactive tool for creating, inspecting, editing, and publishing etymological datasets**. In: **Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. System Demonstrations**. 9-12.
- Moran, S. and M. Cysouw (2017): **The Unicode Cookbook for Linguists: Managing writing systems using orthography profiles**. Zenodo: Zürich.
- Pulgram, E. (1959): **Proto-Indo-European Reality and Reconstruction**. *Language* 35.3. 421-426.
- de Saussure, F. (1916): **Cours de linguistique générale**. Payot: Lausanne.
- Segerer, G. and S. Flavier (2015): **RefLex: Reference Lexicon of Africa**. Paris and Lyon.
- Starostin, S. (2000): **The STARLING database program**. RGGU: Moscow.
- Starostin, G. and P. Krylov (eds.) (2011): **The Global Lexicostatistical Database. Compiling, clarifying, connecting basic vocabulary around the world: From free-form to tree-form**. http://starling.rinet.ru/new100/main.htm.
- Swadesh, M. (1952): **Lexico-statistic dating of prehistoric ethnic contacts. With special reference to North American Indians and Eskimos**. *Proceedings of the American Philosophical Society* 96.4. 452-463.
- Swadesh, M. (1955): **Towards greater accuracy in lexicostatistic dating**. *International Journal of American Linguistics* 21.2. 121-137.
- Tadmor, U. (2009): **Loanwords in the world's languages. Findings and results**. In: Haspelmath, M. and U. Tadmor (eds.): **Loanwords in the world's languages**. de Gruyter: Berlin and New York. 55-75.
- Trask, R. (2000): **The dictionary of historical and comparative linguistics** . Edinburgh University Press: Edinburgh.