UPPSALA
UNIVERSITET

Grekiska och bysantinologi: självständigt arbete 15 hp

Juni 2024

A Macronizer for Ancient Greek

Albin Thörn Cleland

UPPSALA
UNIVERSITET

A Macronizer for Ancient Greek

Albin Thörn Cleland

Abstract

Much philological work on Ancient Greek texts hinges on correctly disambiguating the vowel length of the so-called dichrona vowels, α, ι and υ, a process known as macronization. Inspired by the Latin macronizer of Winge (2015), it is shown how the vowel lengths of Ancient Greek tokens can be programatically compiled into a dictionary using digital grammars, lexica and other sources, a first step towards a macronizer for Ancient Greek. Given three hundred thousand morphologically tagged tokens from all the extant Ancient Greek tragedies, around forty-two thousand tokens containing the ambiguities in question are culled out. Vowel length data from two dictionaries, a Ruby lemmatizer and an online repository of scanned verse is crawled and carefully collated, disambiguating roughly half of the tragic corpus. Finally, a number of ways in which the percentage of macronized dichrona can be increased algorithmically are presented, with three of them solely "shuffling around" the data already contained in the sourced dictionary, and two requiring formalization of morphological rules, such as those governing prefixes and the endings of nominal forms.

Institutionen för lingvistik och filologi

Uppsala universitet

Handledare: Eric Cullhed

Examinator: David Westberg

... while it is not difficult to learn by heart the prosodic length of a restricted number of inflections and small words, it takes an incredible intellectual effort to memorize the prosodic length of thousands and thousands of words.

Lauxtermann 2022, p. 125.

... λέγω μετὰ δακρύων,
Ἀνάθεμαν τὰ γράμματα.

Ptochoprodromiká 3

Caveat Emptor

If the present presentation appears to be on the (excessively) shorter side of a master thesis, the reader is urged to remember the words of Apollo to Callimachus:

> [...] “ἀοιδέ, τὸ μὲν θύος ὅττι πάχιστον
> θρέψαι, τὴν Μοῦσαν δ᾽ ὠγαθὲ λεπταλέην”

If this written thesis is my slender song, my fat sacrificial lamb (spaghetti code does make you fat!) is the accompanying software, which I humbly present to the community of digital Greek scholarship. The majority of my effort lies ploughed down there, not here.

Furthermore, I am not a software engineer, and this report mirrors my efforts to come to grips with the technologies of digital philology as much as it mirrors the research eventually accomplished by way of them. Precisely because of this, I think it may prove useful and inspiring for scholars who have yet to use digital tools.

Contents

1 Introduction

1.1 Prosody and Dichrona

... τὴν ϑ' ὁμηρείαν φύσιν
ἀλλοπροσάλλως διχρόνοις χρῆσϑαι λέγει,
τὸ «Ἄρες, Ἄρες» πανταχοῦ βοῶν μέγα
(Βρεντησίου μένδητος ὦ πατρὸς τέκνον!).

He [...] keeps saying that Homer uses the dichrona as it suits him, while bellowing «Ἄρες, Ἄρες» on every occasion (oh that bastard from Brindisi!).

Tzetzes, *Schol. Ar. Plut.* 43.31-44.2 Massa Positano.
Transl. in Lauxtermann 2022.

Ancient Greek (hereafter, simply 'Greek') prosody[1] is distinguished by two key characteristics.[2] The first is pitch accent, as found in e.g. Swedish, Norwegian, Lithuanian, Serbian and Japanese.[3] The second is phonemic vowel length, as found in e.g. Latin, Biblical Hebrew, Sanskrit, Japanese, Finnish, Hungarian and Thai.

Each of the ten vowels of Greek has one of two phonemic vowel lengths: long or short. During the history of the Greek alphabets and diacritics, different solutions have abounded as to how to represent these vowels graphically.[4] When the goal is to fully disambiguate their lengths, the following set of letters is preferred:

$$\alpha,\ \bar{\alpha},\ \varepsilon,\ \eta,\ \iota,\ \bar{\iota},\ o,\ \upsilon,\ \bar{\upsilon},\ \omega \qquad (1)$$

ᾱ, η, ῑ, ῡ, ω are long, the rest are short. The bar above three of the letters is known as a *macron* (using the Greek term; plural *macra* or macrons) or *longum* (using the Latin; plural *longa*).[5] In some early Greek scripts, all ten vowels could be down to sharing as few as five letters[6]:

$$\alpha,\ \varepsilon,\ \iota,\ o,\ \upsilon \qquad (2)$$

[1]Already in the ancient grammarians, there were two concepts of prosody, one wider and one narrower. Narrowly conceived, prosody solely refers to the pitch variations of the language and the signs used to indicate them (this sense of προσῳδία is translated into Latin as *accentus*). I here use prodosy in the wider sense, which includes not only pitch accents, tone contours and their three diacritics, but also (most importantly for us) vowel length, aspiration, and in a word, everything that relates to the suprasegmental aspects of pronunciation and their diacritical apparatus (see the very readable introduction to Probert 2003, especially p. 3).

[2]Note on the epigraph: 'Ἆρες, Ἄρες βροτολοιγὲ μιαιφόνε τειχεσιπλῆτα...' (*Il.* 5, 31); see LSJ s.v., A, III, "epith. of Zeus, as the avenger of perjury".

[3]Pitch accent is distinguished from phonemic tone by not being lexical and by often being suprasegmental (spanning more than one phoneme or one syllable); pitch accents depend on inflection and context of use and hence do not carry absolute semantics in the way, say, the tone of a Chinese lemma does. If, say, Swedish were tonal, we would have to concede that the lexical meaning of 'äter' was different from 'äta', since the root has different pitch.

[4]In the following, the history of the actual looks and shapes of the letters is not relevant, only the number of distinct vowel letters and the number of vowels of differing length classes referred to by each. I'm hence using the standard small letters throughout.

[5]As we shall see, there's also a diacritic indicating short vowel length, e.g. ᾰ; it is referred to as a *breve*, plural *brevia*.

[6]Transliterations of Linear B typically use exactly the following set. Although a large syllabary, Linear B C+V graphemes can nonetheless only end on one of five different vowels. This is true for the Japanese syllabary as well, and it makes sense to be frugal: disambiguating a single new vowel would add one new grapheme for every consonant.

Here each letter could represent either a long or a short vowel, and it was up to the reader to disambiguate based on context whether, say, an inscribed ο was "in fact" an ω or an α an ᾱ.

For historical reasons, a middle ground was eventually established as canonical and is, its shortcomings notwithstanding, still the most common:

$$\alpha,\ \varepsilon,\ \eta,\ \iota,\ o,\ \upsilon,\ \omega \tag{3}$$

In this system, the three vowel letters α, ι, υ still underdetermine vowel length. Collectively, they are known as the *dichrona* (singular *dichronon*), literally 'two-timed', or sometimes, as in French *ancipites* and Italian *ancipiti*, as *anceps vowels.*

It is imperative to note that it is the letter, the grapheme, that is *dichronon* and not the vowel. It is a question of underdetermination of reference by referee, not of any intrinsic ambivalence in the underlying phonemes. As vowel length disappeared from the Greek language, the strictly unrelated phenomenon of poets' substituting long root vowels for short ones to facilitate the meter (cf. Homer's Ᾱ̓ρες, Ἄρες in the initial quote above), a product of an oral culture, was predictably confounded with the strictly graphical concept of the *dichronon*, leading Byzantine scholars to waste rivers of ink and much gusto on metaphysical hypotheses regarding the esoteric ways in which *dichrona* wobbled and waved between lengths, like vowel versions of Schrödinger's cat, including inventing ridiculous words like ἀδιαφοροδιχρονισταί.(Lauxtermann 2022).

What these scholars did get was that dichrona are primarily relevant to *the study of meter*, which provides a segue to the next section.

1.2 Dichrona and Meter

1.2.1 Research Question and *Dionysus Recomposed*

The impetus for the present investigation is intimately tied to the planning for a potential digital-humanities research project called *Dionysus Recomposed*, led by Eric Cullhed. With the goal of harnessing the power of large language models to efficiently generate, and then promptly sift through and eliminate, enormous numbers of conjectures for gaps in papyri fragments of ancient tragedies, *Dionysus Recomposed* will need to formalize as many stilometric strictures as possible for the evaluation of tragic verse. Among the most important strictures is *metre.* Since meter is nothing but sequences of heavy and light syllables [7], a prerequisite for metrical analysis is that all dichrona in open syllables are disambiguated as long or short.[8]

In the unpublished description of his project, Cullhed writes that

> A key problem in automating the analysis of these patterns is the ambiguous vowel length of α, ι, and υ. Therefore, a comprehensive metrical analysis system requires the inclusion of a macronizer. The Makron program will parse sentences with a part-of-speech and morphological tagger (e.g., *odyCy*; see Kostkan et al. 2023), decompose words into their stem, prefix, and endings, and then accurately mark all instances of α, ι, and υ with either a macron or a breve, guided by **a dictionary compiled from grammars and lexica**. The format of vowel length markers often varies from entry to entry (for example, in LSJ, *macra* and *brevia* may appear in the lemma or as elliptical notes within the article). Manually

[7] *Light* syllables are open, have short vowels and do not stand at line end. A syllable that's not light is *heavy.*

[8] Strictly, the length of open-syllable dichrona at line end does not effect the meter. They can however add to stilometrically interesting statistics regarding the treatment of anceps position.

sifting through this prosodic information on a word-by-word basis would be labor-intensive. However, large language models with large context windows (such as GPT-4, Gemini 1.5, or Mixtral 8x7b) can be fine-tuned to sift through each entry and methodically extract the relevant information as structured data. Makron, which will be released as open source, will have a wide range of applications in the automatic analysis of prose rhythm and meter.

The present thesis describes my attempt at creating that dictionary for Makron by programming. Hence, the research question answered is:

How does one programmatically compile a dictionary of macron and breve data for a large set of Ancient Greek tokens?

For reasons of scope, the work described in the following report has been done under the following restrictions:

- Only tokens from the surviving ancient tragedies are considered.
- The exact integration into the larger metrical software is not considered (but briefly discussed in the conclusion).
- The main focus is on sourcing macrons that cannot easily be inferred from either the prosodical rules or the general declension and conjugation tables.

The first restriction was suggested to me by Eric Cullhed, as the domain of *Dionysus Recomposed* is tragedy. The precise import of the last restriction will become clear when the work process is described, and indications for how the restriction may fairly easily be lifted through future research are given in the conclusion.

1.2.2 Epistomology of Quantity

The problem of alpha in lyrics is insoluble; we cannot expect consistency or accurate etymological knowledge from the poet.

Lloyd-Jones, *Sophoclis Fabulae*, xiv

In the field of dichrona, scholars have since antiquity met with two obstacles worthy of being called *epistemological* in the philosophical sense, since they pertain not only to that transient lack of knowledge, the border of which is constantly pushed back, characteristic of all sciences at any one instance, but to the presence of permanent and immutable underdetermination.

The first obstacle has already been mentioned, namely that switching between vowel length classes in general, and not only with regard to dichrona, is subject to poetical licence. I would like to say a few more words on the topic here. The second one, which I will treat afterwards, is what Allen called "hidden quantities".

I have chosen the above quote by Lloyd-Jones as symptomatic of the dilemma faced by the vowel length inquirer: what is the goal, what is the "real" vowel length? The one intended by the poet or the "etymologically correct" one? If the former: what does it mean, or would it mean, that a poet makes a mistake in the identification of vowel lengths in his own poem?

Of course, whether we are concerned with later poets from the Imperial era and beyond or with classical authors makes a big difference: when Gregory of Nazianzus treats a dichronon as short that was spoken long in ancient Attica, it arguably implies a failure to conform to *his own* project of reviving a by his time obsolete prosodical feature, and hence a failure of intention. In other words, one may readily imagine Gregory thankful and considering the point well-taken, would he have been presented with evidence of the correct ancient usage.

A classical or archaic poet, an Attic tragedian say, on the other hand, arguably most often based what he *wrote* on what he *heard*. Opinions differ, though, regarding the extent to which such a poet's prosody and phonology was archaic even by the standards of his own day. A quick look at the data collected in *The Phonology of Attic Greek in the Hellenistic Period* (Teodorsson 1978, p. 21, 24) for example shows EI as an orthographic error substituting both for /i/ and /iː/.

However, it is far from certain——indeed, it is implausible *prima facie*——that when "poetical licence" invites a poet to at one time write ῠ̔δωρ (*Il.* 18.347) and at another ῡ̔δωρ (15.37), all the variants reflect words he has actually heard spoken outside of music (just like nobody would expect the melismas of a Monteverdi aria to reflect the absurd syllables of some outlandish dialect of Italian).

At the end of the day, if we are primarily interested in vowel length *qua* evidence for verse scansion, then vowel length partakes of an hermeneutic circle: it's meaning and intention will be vicarious in the sense that it depends on the meaning and intention of the metre of the whole line, which in its turn depends on a system of stanzas that may exhibit responsion and other interlinear phenomena. It is only in the situation where there is a draw between two or more fully fleshed-out metrical interpretations, that we have to bite the sceptical bullet.

The second obstacle regards the status of dichrona in closed syllables, where the weight of the syllable as it were "hides" the length of the vowel. Unlike dichrona in open syllables, we cannot rely on metrics out (cf.2.2.2). Allen gives the following philological sources for hidden quantities:

a. **Orthographical errors**: EI for long I in Hellenistic inscriptions (cf. Teodorsson *loc. cit.* above on short I), "particularly after about 100 B.C." E.g. ῥίπτω is long because we see ΡΕΙΠΤΩ.

b. **Dialects**: Attic α is long where there is a corresponding Ionic η

c. **Derivates with open syllables:** e.g. ῥι-πή is found in *arsis* position, ergo ῥίπτω is long.

d. **Prosodical rules:** Prosodical rules work on a vowel level, unlike the syllabic rules of metrics. Circumflexes hence indicate both that the vowel it is on is long and that the eventual following ultima is short. Inflectional generalizations are shaky, however: for example we have gen. κήρῡκος in spite of nom. κῆρυξ.

e. **Cases specifically mentioned by the ancient grammarians:** e.g. "τὸ ῥᾶξ [...] ἐκτεταμένον ἔχει τὸ α." (Herodian (?), < Περὶ διχρόνων > 1, Pontani 2020, p. 169)

I may add a further tentative item to the list, namely comparative Indo-European etymology (cf. Lloyd-Jones remark).

It should follow immediately from this scanty list of sources, that a relatively large number of hidden quantities are not presently known, and that even though a subset of them will certainly be disambiguated in light of new findings (every few years a new Περὶ διχρόνων seems to be found on some half-rotten palimpsest), a good few of them will have been lost to time.

Although hidden dichrona do not play any role in mainstream metrics, in a more fine-grained analysis they do decide whether a syllable is heavy or *superheavy*, i.e. has a long vowel followed by a consonant cluster,[9] which provide a way to find stylistic variation within the same scansion pattern. Their use for the field of historical phonology need hardly be mentioned, as well as the obvious guidance they provide for whoever wants to read a Greek text aloud, e.g. a participant in a modern staging of an ancient play where it is important that all singers and instrumentalists keep time in the same way.

For all of the above reasons, I found it worthwhile not to filter hidden dichrona out of the "wishlist" I brought to the crawling[10], even though I judge my results mainly on the basis of the macronization of dichrona in open syllables.

2 Technical Report: Problems, Solutions, and Results

This report chronicles the step-wise creation of a dictionary containing as many as possible of the vowel lengths of the dichrona vowels appearing in tokens from the tragic corpus. There were four main steps: firstly, preparation of a list of the tokens that needed disambiguation, secondly, crawling for the vowel lengths of said tokens, thirdly, collation of the crawled data, and last but not least indications of how the collated data can be expanded and generalized algorithmically.

Apart from the interface with *ifthimos* which I had to write in Ruby, all the coding was done in Python, a relatively readable and pedagogical programming language that is the standard for a wide range of scientific uses, including philological. The code is available on the dedicated public GitHub repository found here.[11] An excerpt from the latest version of the resulting macron dictionary is included as appendix I.

The reader who wishes to clone the repository and try to run some of the scripts needs to install the latest version of Python. The file *readme.md* contains a list of most of the third-party packages that the code depends on.

2.1 Preparing the Tokens

2.1.1 Mending Tokens

I started out with the result of odyCy (Kardos and Kostkan 2023) lemmatizing and parts-of-speech tagging all the ancient tragedies, a 300 595 line long tab-separated file. Here is the beginning of the file, a tokenization of the first line from Euripides *IT*:

```
NOUN Πέλοψ n-s---mn-πέλοψ
DET ὁ l-s---mn-ὁ
ADJ Ταντάλειος a-s---mn-Ταντάλειος
ADP ἐς r--------εἰς
NOUN Πῖσαν n-s---fa-Πῖση
VERB μολὼν v-sapamn-βλώσκω
```

The tabs from left to right are *part of speech*, *token*, *tag* (or *postag*) and *lemma*, where the tag n-s—mn- is read as "noun singular masculine nominative" (I've added a full list of how

[9]Dionysus of Halicarnassus noted the many possible lengths of closed syllables, and the different stylistic ends they can be put to. An example of a modern philological analysis repeatedly referring to superheavy syllables is Devine and Stephens 1994.

[10]In the special sense used here, *to crawl* means to digitally go through files or websites in order to index the information contained therein, with a view to facilitating searching for and retrieving data.

[11]Github is an online service to keep track of the versions of code repositories. This means that every single step of the coding of my project is documented and can be retraced.

to read the tags as an appendix). Though this particular sentence is fine, a large part of the headache of the project relates to bugs and oddities introduced by the tokenization. Hence, to understand the way in which the tokens have been extracted and the artifacts created in the process is crucial. Take as an example of a problematic line Soph. *Ajax* 699f,

```
Νύσια a-p---na-Νύσιος
Κνώσι n-p---na-Κνώσι
’ n-p---na-’
ὀρ n-p---na-ὀρ
-u--------
χήματ n-p---na-χήματ
’ n-p---na-’
αὐτοδαῆ a-p---na-αὐτοδαής
ξυνὼν v-sppamn-ξυνὼν
ἰάψῃς v2sasa---ἰάπτω
```

Obviously, words followed by a line with an elision mark, need to be reunited with this mark (Κνώσι, χήματ + ’), and words encircling a line-break mark need to be joined together (ὀρ + χήματ).[12]

Now, the use of dashes for ellipses could have created confusion, as e.g. Eur. *IT* 1, 23, ”τίκτει — τὸ καλλιστεῖον εἰς ἔμ᾽ ἀναφέρων —”

```
— u--------—
τὸ l-s---na-ὁ
καλλιστεῖον n-s---na-καλλιστεῖον
εἰς r--------εἰς
ἔμ p-s---ma-ἔμ
’ p-s---ma-’
ἀναφέρων v-sppamn-ἀναφέρω
— u--------—
```

Luckily, the two usages have (hopefully consistently!) separate unicodes: the line-break is a short en dash, same as in the tag, while the ellipsis is surrounded by two em dashes.

The third needed fix is exemplified by Soph. *OC* 1733,

```
"ἄγε με, καὶ τότ᾽ ἐπενάριξον":

Ἄγε v3siia---ἄγω
με p-s---ma-ἐγώ
,u--------,
καὶ c--------καί
τότ’ d--------τότ
<d--------<
ἐπ r--------ἐπ
>d-------->
ενάριξον v1saia---ενάριξον
. u--------.
```

[12]What is elision in Greek? In polysyllabic words, all final short vowels except ypsilon (CGCG 1.36) may be elided sometimes: α (ἄρ᾽), ε (ὅτ᾽), ι (ἔστ᾽), ο (ἀφ᾽ οὗ). In monosyllabic words, only epsilon. For a striking example in action from the Aeschylean corpus, see:
ἔτ᾽ ἆρ᾽ Ἀθηνῶν ἔστ᾽ ἀπόρθητος πόλις; (*Pers.* 348)

An editor has conjectured to add the prefix 'ἐπι' to 'ἐνάριξον' (ἐναρίζω, killing someone to take their ἔνᾰρα, "spoils"), and the conjectural part is written with <ἐπι>. This means the rest of the word lacks a spiritus, which is how this bug is found. We need to remove the <> and join the two lines. The result should thus be

```
Ἄγε v3siia---ἄγω
με p-s---ma-ἐγώ
,u--------,
καὶ c--------καί
τότ' d--------τότ
ἐπενάριξον v1saia---ἐπενάριξον
. u--------.
```

Another example is:

```
ξ d--------ἔξ
<u--------<
ίφει v3spia---ίφω
>u-------->
```

which clearly should simply be

```
ξίφει v3spia---ξίφει
```

An extra complicated case is Soph. *Ant.* 836

```
Καίτοι d--------καίτοι
φθιμένῃ v-sapmfn-φθιμένῃ
μέγ a-s---na-μέγ
<d--------<
α p-p---na-α
κ>ἀκοῦσαι v--ana---κ>ἀκοῦω
```

To reiterate, three salient problems that depended on the sentence order of the context of tokenization and that would have to be addressed to prevent problems downstream were:

- elided tokens whose elision signs have become separate tokens,
- words with intra-word line breaks which have been triply tokenized: first part of the word + en dash + last part,
- words with parts that are conjectures in angular brackets[13]

2.1.2 Unicode

A general problem for any digital treatment of Ancient Greek is the representation of the staggeringly rich set of Unicode characters. For modern Greek you are pretty much set with characters from the *Greek and Coptic* code chart, covering the subspace 0370–03FF.

With Ancient Greek, however, matters quickly get entangled. To begin with, we have *Greek Extended* in subspace 0370–03FF, which is supposed to be the go-to subspace for polytonic Greek. However, Unicode characters with diacritics can always be reached in two

[13]My functions solving these three situations are found at https://github.com/Urdatorn/greek-macronizer/blob/master/prepare_tokens/fix_elision_and_line_break.py.

ways: by pre-composed characters (say ᾧ) or with composition of base character + diacritics (e.g. ῾+ ῀+ ͺ + ω). The funny thing is that the diacritics needed for polytonic Greek are all in a third subspace, *Combining Diacritical Marks*, ranging between 0300–036F.

Unicode character naming convention is slightly idiosyncratic (it tends to use Modern Greek linguistic vocabulary that may not be the most current in Anglophone philology) and calls for a quick exposition before we continue:

- acute = ’oxia’ for polytonic and ’tonos’ for monotonic (only the small ones overlap with polytonic)
- grave = ’varia’ (this is downright obfuscating)
- circumflex = ’perispomeni’
- spīritūs asperī and lēnēs = ’psili’ and ’dasia’
- iota adscriptum = ’prosgegrammeni’
- iota subscriptum = ’ypogegrammeni’
- Greek diaeresis/trema = ’dialytika’ in precomposed characters, although on its own the diacritic is called ’combining diaeresis’)
- longum = ’macron’
- breve = ’vrachy’

Now to the problems. Unaccented Greek base characters are all from *Greek and Coptic*, and there are no base characters in *Greek Extended*, so that’s fine. The first problem is that Modern Greek does have two series of accented vowels, the precomposed *tonos* characters plus the precomposed *tonos* and *dialytika* characters (e.g. ΐ), that in most but not necessarily all fonts look identical to the polytonic *oxia* characters, and that nonetheless should be functionally equivalent to them. This clearly makes any text search and string matching unreliable, as one καί with *tonos* could end up not registering as identical to a καί with *oxia*. Confusingly, expert sources do not all agree on which of the two should be the “normal”. In his handbook article, Taubner implicitly endorses oxia-to-tonos normalization (Tauber 2019, p. 154), whereas the CLTK (with which the same Taubner has been closely associated) supplies a normalization function that defaults to tonos-to-oxia normalization.

I think the matter is easy to settle from a pragmatic point of view. Polytonic keyboards tend to enter acutes as *tonos* characters, and all corpora I’ve checked, including the tokenization corpus dealt with in this thesis, have *tonos* acutes. CLTK seems to have understood the complications, and provide a “backwards” options for their acute normalization, which I’ve used on the corpus to be on the safe side.

Differences between precomposed and composed characters may likewise present problems, and not only for string matching. In general, if you enter accented text through a polytonic keyboard, you will probably get precomposed characters (that is, from the 1Fxx subspace) rather than compositions. The Unicode FAQ for Greek (Unicode 2024a) has the following to say:

> Also, if you examine the code charts for the U+1F00..U+1FFF block of “extended” Greek carefully, you will note that all the polytonic Greek pre-composed characters have canonical mappings. This means that they are canonically equivalent to sequences consisting of the basic letters plus sequences of the basic letters plus combining voicing and accent marks. Any properly constructed Unicode search operation should treat canonical equivalents the same, so *it should not matter whether one specifies a target match in terms of the pre-composed characters or in terms of the sequences of basic letters and combining marks.* This situation for Greek is no different from the requirement for the Latin script that

a search for a pre-composed Latin letter and the same letter with a combining accent mark produce the same results. [My emphasis]

One can't but agree, but it's wishful thinking, and if you program a Python function to search for a precomposed string, it will at any rate certainly not match composed strings. The solution is to normalize the corpus. In Unicode terms there are two possible normals, NFC and NFD, "normal form composed" and "normal form decomposed" respectively. Luckily, python provides a `unicodedata.normalize(form, unistr)` method, where 'form' is NFC or NFD. Throughout my work flow, I have consistently and repeatedly used NFC normalization, avoiding problems pertaining to the divergent formattings of the source corpora for macron crawling.

It needs to be added, that certain combined characters simply do not have a Unicode points. Specifically, all accented polytonic characters with *macron* or *vrachy* are irreducibly decomposed, that is, they cannot be represented using less than two points. In the Wiktionary corpus, there are no less than 62 (!) unique decomposed combined characters with macronization.[14] The problem that arises here has to do with determining string length and the position of a given character within a string. For unless told otherwise, from the perspective of Python each Unicode escape code (e.g. 0370) counts as a character. Hence, the final α in Αἰγυπτῐ́ᾱ[15] risks being counted as the ninth character of the string. We will return to the precise solution of this problem in the chapter on macron crawling.

2.1.3 Sieving Tokens

My sieving pipeline aimed to create a tab-separated values (*.tsv*) file with columns *token*, *tag* and *lemma*, containing only well-formed and well-formatted tokens containing truly undecided *dichrona*, to serve as a maximally short wish list for the coming data crawling. It contained ten subscripts:

1. Removing punctuation
2. Splitting double tokens with final sigma at non-final positions
3. Removing duplicates
4. Normalizing delimiters
5. Removing lines with first column empty of Greek
6. Adding oxytone versions of every barytone token
7. Sorting unicode alphabetically with *pyuca*
8. Filtering truly undecided dichrona
9. Removing the last accent of words with two accents
10. Removing words lacking obligatory *spīritūs*

[14] See `https://github.com/Urdatorn/greek-macronizer/blob/master/crawl_wiktionary/macrons_map.py`

[15] Yes, the Greek font I'm typesetting this document with is not a "smart font" like New Athena Unicode, which "contains OpenType ligature instructions that allow the display of well-formed precomposed glyphs in response to input of two or more Unicode code points. This capability allows the use of combinations like alpha with macron and acute and smooth breathing depending only on official Unicode code points." (Classical Studies 2021) So it will indeed display on the typeset page like Python interprets it, and not like I see it when I type it, which is as an iota with both a *breve* and an acute accent.

The division into subscripts was partly practical, as each script could be tested in isolation until yielding the desired bug-free results. But it was also due to the fact that I could run them all from a master script, with a single in- and output, enabling the whole process to be easily reiterated, no matter how early in the chain changes were made or bugs found. For example, most of the pipeline was already written by the time I realized the need to perform the fixes outlined in the section on mending the tokens.

1, 2 and 9 perform token mending that does not rely on the original sentence context. For 2, I rely on the fact that since ς should only appear at word end, its presence inside a token indicates that two tokens have merged and should be split into two separate lines. In 9, I standardize tokens that were originally followed by enclitics, as double-accented forms will be searched for in vain in the large context-free dichrona corpora like Wiktionary.

3, 5 and 10 removes lines that either could not be saved by the mending or simply did not contain any Greek.

4 double-checks that all of the tab stops are well-formatted.

6 is included because oxytone tokens are way more likely to be found in the crawled corpora (cf. 9) and barytone tokens can then easily be made to algorithmically inherit the macrons from their oxytone brothers.

7 is of some special interest. As anyone trying their hands on Greek lexigcography knows, the precise alphabetic ordering of diacritics is far from trivial. For example, how are ἇ, ᾂ and ᾰ́ to be ordered? Or what about α and ᾱ? Python has no native support for this task. Luckily, James Tauber has ported the *Unicode Collation Algorithm*[16] to python as the package *pyuca* (Tauber 2017). While the UCA is certainly not the be-all and end-all of Ancient Greek string ordering, it was enough to render my files more readable, and crucially to enable me at a later stage to at a glance see the progress of macronization of a given root.

The most subtle and most important work went into 8, however. The filtering has two main components: (i) filtering for tokens with at least one "true" dichronon (not every α, ι, υ is born equal!), and (ii) filtering for certain combinations of accentual word class and dichronon placement.

What is a "true" dichronon? It is dichronon whose length is not immediately and mechanically given by its diacritics or adjacent vowels. For α, ι, υ with circumflex are always long, as are α, υ with iota subscript or iota adscript. Regarding diphthongs, they are always constituent of metrically heavy syllables, regardless of the length of the constituing vowels, so for our intents and purposes α, ι, υ need not be disambiguated.[17]

However, programmatically it's easier to say what a dichronon *is*, rather than what it is *not*:

```
DICHRONA = {
# CAPITALS
"\u1f08",  # Ἀ Greek Capital Letter Alpha with Psili
"\u1f38",  # Ἰ Greek Capital Letter Iota with Psili

"\u1f0c",  # Ἄ Greek Capital Letter Alpha with Psili and Oxia
"\u1f3c",  # Ἴ Greek Capital Letter Iota with Psili and Oxia

"\u1f0a",  # Ἂ Greek Capital Letter Alpha with Psili and Varia
"\u1f3a",  # Ἲ Greek Capital Letter Iota with Psili and Varia

"\u1f09",  # Ἁ Greek Capital Letter Alpha With Dasia
```

[16]See http://unicode.org/reports/tr10/

[17]If we were more interested in the reconstructed pronunciation than in metrics, we would have to distinguish between long and short diphthongs, and furthermore decide for long diphthongs the length of their "phthongs" (arguably, short diphthongs ought have short constituents).

```
"\u1f39",  # Ἱ Greek Capital Letter Iota With Dasia
"\u1f59",  # Ὑ Greek Capital Letter Upsilon With Dasia

"\u1f0d",  # Ἅ Greek Capital Letter Alpha With Dasia And Oxia
"\u1f3d",  # Ἵ Greek Capital Letter Iota With Dasia And Oxia
"\u1f5d",  # Ὕ Greek Capital Letter Upsilon With Dasia And Oxia

"\u1f0b",  # Ἃ Greek Capital Letter Alpha With Dasia And Varia
"\u1f3b",  # Ἳ Greek Capital Letter Iota With Dasia And Varia
"\u1f5b",  # Ὓ Greek Capital Letter Upsilon With Dasia And Varia

# LOWER-CASE (NB 3 overlapping tonos-oxia)
"\u03b1",  # α Greek Small Letter Alpha
"\u03b9",  # ι Greek Small Letter Iota
"\u03c5",  # υ Greek Small Letter Upsilon

"\u03ac",  # ά Greek Small Letter Alpha With Tonos
"\u03af",  # ί Greek Small Letter Iota With Tonos
"\u03cd",  # ύ Greek Small Letter Upsilon With Tonos

"\u1f71",  # ά Greek Small Letter Alpha With Oxia
"\u1f77",  # ί Greek Small Letter Iota With Oxia
"\u1f7b",  # ύ Greek Small Letter Upsilon With Oxia

"\u1f70",  # ὰ Greek Small Letter Alpha With Varia
"\u1f76",  # ὶ Greek Small Letter Iota With Varia
"\u1f7a",  # ὺ Greek Small Letter Upsilon With Varia

"\u1f00",  # ἀ Greek Small Letter Alpha With Psili
"\u1f30",  # ἰ Greek Small Letter Iota With Psili
"\u1f50",  # ὐ Greek Small Letter Upsilon With Psili

"\u1f04",  # ἄ Greek Small Letter Alpha With Psili And Oxia
"\u1f34",  # ἴ Greek Small Letter Iota With Psili And Oxia
"\u1f54",  # ὔ Greek Small Letter Upsilon With Psili And Oxia

"\u1f02",  # ἂ Greek Small Letter Alpha With Psili And Varia
"\u1f32",  # ἲ Greek Small Letter Iota With Psili And Varia
"\u1f52",  # ὒ Greek Small Letter Upsilon With Psili And Varia

"\u1f01",  # ἁ Greek Small Letter Alpha With Dasia
"\u1f31",  # ἱ Greek Small Letter Iota With Dasia
"\u1f51",  # ὑ Greek Small Letter Upsilon With Dasia

"\u1f05",  # ἅ Greek Small Letter Alpha With Dasia And Oxia
"\u1f35",  # ἵ Greek Small Letter Iota With Dasia And Oxia
"\u1f55",  # ὕ Greek Small Letter Upsilon With Dasia And Oxia

"\u1f03",  # ἃ Greek Small Letter Alpha With Dasia And Varia
"\u1f33",  # ἳ Greek Small Letter Iota With Dasia And Varia
"\u1f53",  # ὓ Greek Small Letter Upsilon With Dasia And Varia

# DIAERESIS/TREMA/DIALYTIKA (NB 2 overlapping tonos-oxia)
"\u03ca",  # ϊ Greek Small Letter Iota With Dialytika
"\u03cb",  # ϋ Greek Small Letter Upsilon With Dialytika

"\u0390",  # ΐ Greek Small Letter Iota With Dialytika And Tonos
"\u03b0",  # ΰ Greek Small Letter Upsilon With Dialytika And Tonos
```

```
"\u1fd3",  # ΐ Greek Small Letter Iota With Dialytika And Oxia; my addition
"\u1fe3",  # ΰ Greek Small Letter Iota With Dialytika And Oxia; my addition

"\u1fd2",  # ῒ Greek Small Letter Iota With Dialytika And Varia
"\u1fe2",  # ῢ Greek Small Letter Upsilon With Dialytika And Varia
}
```

It is worthwhile to include this complete Python dictionary, because it forms the most important and the most imported part of my code, and it exemplifies several points touched on in the section on Unicode. It is clear that if a token contains any of these characters, it requires at least some effort to settle its metrical word shape.

To understand the second filter, we need to note all of the logical relationships between the five Greek accentual word classes[18] and the length of the ultima, given that it is a dichronon:

- OXYTONE implies nothing without context (cf. ᾰ̆ν vs. ᾱ̆ν = ἐάν)
- PAROXYTONE with ≥ 3 syllables does NOT imply long vowel in ultima, because not all accents are recessive. However, **paroxytone + long penultima implies short ultima** as per the logical contraposition of the so-called **σωτῆρᾰ-rule**.[19].
- **PROPAROXYTONE implies short ultima** (an exception would have been the πόλις declination's εως, had it ended on a dichronon).
- PERISPOMENON implies long ultima (strictly, this is irrelevant, since I do not class α, ι, υ with circumflex as true dichrona).
- **PROPERISPOMENON implies short ultima**.

I have boldfaced the three cases where we actually get valuable new disambiguating information from the accentual word class, and I chose to formalize the former two of them to act as the second filter, to make the amount of tokens crawled more manageable. Simply put, we should not go through the hassle of searching external sources of macrons for words whose only true dichronon can easily be disambiguated based on their accentual word class alone. We will return to all three of these rules, when we go through how they were implemented to macronize the ultimas of words that survived the filtering because of containing more dichrona than just in the ultima.

The formalization required a series of dependent steps. Using the syllabifier extracted from Eric Cullhed's *Dionysus Recomposed* and making sets of all relevant acutes and circimflexes, I defined properispomenon and proparoxytone:

```
def properispomenon(word):
    '''
    >> properispomenonὖσον('')
    >> True
    '''
    list_of_syllables = syllabifier(word)
    if len(list_of_syllables) >= 2:
        penultima = list_of_syllables[-2]
        circumflexes = r'ᾶῆῖῦῶἇἆἧἦἶἷὗὖὧὦἧἦἆἇὦὧᾇᾆᾷᾗᾖᾧᾦῷῇ[]'
```

[18] *Oxytone* means acute on the ultima, *paroxytone* acute on the penultima, *proparoxytone* acute on the antepenultima, *perispomenon* circumflex on the ultima and *properispomenon* circumflex on the penultimate. When a final acute has been changed into a grave, we sometimes talk of the word as belonging to a sixth 'barytone' word class, which is however not lexical.

[19] The rule (CGCG 21.11) says that if the accent falls on a long penultimate and the ultima is short, the accent is a circumflex rather than an acute. In contraposition (viz. the negation of the apodosis implies the negation of the protasis), we get: if the accent is an acute rather than a circumflex, either the accent falls on a short penultimate or the ultima is long. Hence if we restrain ourselves to cases with long penultimate, acute on penultimate directly implies long ultima. Phew!

```
        if re.search(circumflexes, penultima):
            return True
        else:
            return False
    else:
        return False

def proparoxytone(word):
    '''
    >> proparoxytoneποτιδέρκομαι('')
    >> True
    '''
    list_of_syllables = syllabifier(word)
    if len(list_of_syllables) >= 3:
        antepenultima = list_of_syllables[-3]
        acutes = r'άέήόίύώἄἅἔἕὄὅἤἥἴἵὔὕὤὥΐΰᾄᾅᾴᾔᾕῄᾤᾥῴ[]'
        if re.search(acutes, antepenultima):
            return True
        else:
            return False
    else:
        return False
```

The first function to filter out tokens with could therefore be defined by the following criteria (and *mutatis mutandis* for the second one concerning proparoxytones:

1. The entire string is recognized by `word_with_real_dichrona`.
2. The accent type of the string is classified as `properispomenon`.
3. The ultima of the string is recognized by `word_with_real_dichrona`.
4. The part of the string before the ultima is NOT recognized by `word_with_real_dichrona`.

2.2 Crawling for Macrons

The sieving pipeline took the list of tokens from 300 595 lines to 41 825 (with 32 540 unique tokens), as 94% of the tokens were either punctuation or tokenization artifacts, or contained no true dichrona, or had its only dichronon as an ultima easily given by the rules of prosody.

I now had a feasibly-sized wish list of dichrona. So where does one go to find disambiguated dichrona? The first answer that comes to mind is probably the dictionary. As a substantial dictionary explicitly aiming to cover the entire vocabulary of tragedy, LSJ is a natural choice. It also happens, as we will see below, that one free online version, `LSJ.gr`, provides a website with features making it particularly ripe for crawling.

However, a lexicon like LSJ will as a rule only provide the vowel lengths of lemmas (i.e. the uninflected lexical search words) and, occasionally, their principal parts.

This is where Wiktionary comes in. The unsung king of digital morphological table carpentry, Wiktionary importantly provide ample support for and encourages the markup of vowel lengths in every single morphological form, and the website is easy to crawl. Wiktionary was the back-bone of the operation, and would come to provide more than one third of the crawled vowel lengths.

The second type of source is a corpus of verse with the metrics marked up. With every syllable marked as long or short, it is immediately possible to macronize all open syllables. Luckily, David Chamberlain has created precisely the right kind of corpus on *hypotactic.com*. (Chamberlain 2023)

2.2.1 Wiktionary

Underneath its collaborative surface, the bulk of Wiktionary's Ancient Greek tables is generated by an ingenious set of code called *grc-conj* written in the Lua programming language by a single individual, an anonymous woman going by the username ObsequiousNewt.[20] (ObsequiousNewt 2017)

Because of the structured pages, Wiktionary was easy to crawl. However, because of the smart fonts used on the site, all macra and brevia are presented as diacritics in the text, as Unicode, which meant that they couldn't directly be compared to my prepared list of desired tokens.

As mentioned above, in my crawled list of macronized tokens there are sixty-two different combinations of polytonic characters with a macron or breve. My goal was i) to strip the vowel length diacritics off the polytonic characters, all without loosing their other diacritics (the accents, breathings, subscripts and dialytika etc. vertically stacked on top of the macronized base) and ii) reformat the information contained in the diacritics as a stand-alone macron column in the tsv.

Following Johan Winge's *latin-macronizer*, I chose to represent a macron as an underscore '_' and a breve as a caret '^', but unlike his inline markup my underscores and carets are followed by numerals indicating the ordinal position of the vowel they refer to in the token string. For example, '^1' is intended to be read as "with a breve on the first letter", "_1^3^5" as "with a macron on the first letter, and breves on the third and fifth letters".

Practically, then, I needed to take a line such as

```
νεᾱνῐᾰ́ς
```

and turn it into the two-column

```
νεανίας _3^5_6
```

As mentioned earlier, after a lot of fruitless attempts at smart text searches, I realized that by far the safest solution was to brute-force the problem with an explicit dictionary. So I made a script compiling all unique characters in the 400 000 line crawled corpus at my command, and manually specified how each should be stripped, as in the example below:

```
'\u1FD0\u0301': 'ί',  # ῐ́
'\u1FD0\u0300': 'ὶ',  # ῐ̀
'\u1FD0\u0308': 'ϊ',  # ῐ̈
'\u1FD1\u0313': 'ἰ',  # ῑ̓
```

There is one particularly subtle aspect of the dictionary: for full compatibility, the twelve macronized base letters ᾰᾸᾱᾹῐῘῑῙῠῨῡῩ with macron or vrachy as only diacritic had to be included in both precomposed and composed versions. To make this explicit, and in general to increase the future usability of this solution as a sort of quarry for macronized Unicode, the left-hand entries are given as explicit Unicode escape sequences, with their graphs indicated in the comment (I included their Unicode name as well, but it makes the comment to long to typeset here).

As for goal ii), to reformat the information that is being stripped, it presented its own challenges.

[20] *grc-decl* is the analogous code for nouns. See https://en.wiktionary.org/wiki/User:ObsequiousNewt for ObsequiousNewt's userpage and https://en.wiktionary.org/wiki/Module_talk:grc-conj for a page showing the interactions of other later contributors.

The main problem here was that diacritics often make Python count an extra character (as discussed in an earlier section). The solution was to leverage the CLTK function `base` to search for the completely stripped base character in a created base alphabet set, and only increment the character position counter for characters found in that set (free-floating diacritics do not have such a base, and will thus be skipped):

```python
def process_word(word):
    modifications = []
    i = 1  # Initialize character position counter
    for char in word:
        if re.search(base_alphabet, base(char)):
            char_length = length(char)
            if char_length == LONG:
                modifications.append(f"_{i}")
            elif char_length == SHORT:
                modifications.append(f"^{i}")
            i += 1

    processed_word = strip_length_string(word)
    return processed_word, modifications
```

As we'll return to in the chapter on collation, a staggering 14 731 lines of our tokens were eventually macronized when collated with the data from Wiktionary.

2.2.2 Hypotactic

Hypotactic is a Herculean effort, and a (probably lifelong) completely unfunded work-in-progress. I will not try to paraphrase Chamberlain's own words in the introduction to his scansion repository:

> This project arose from multiple years of failed experimentation with algorithms (my own and others'; no pun intended) for teaching a computer to scan ancient verse. The reason this was doomed to fail is explained here [...]. Eventually the realization hit that they're not writing any more ancient verse, so why not just *scan it all* and publish it for all to use? (Chamberlain 2023, Original hyperlink but my emphasis)

"Please take the data and see what you can do with it", says Chamberlain further down on the same page. Your will, my command! With 3 148 lines of our tokens macronized in the end, Hypotactic proved an irreplaceable supplement to Wiktionary.

Hypotactic contains a lot of information. Here is the first line of the *Argonautica*, ἀρχόμενος σέο, Φοῖβε, παλαιγενέων κλέα φωτῶν, in Chamberlain's HTML:

```html
<div class="line hexameter"
data-number="1"
data-metre="hexameter">
  <span class="word">
    <span class="syll long">ἀρ</span>
    <span class="syll short">χό</span>
    <span class="syll short">με</span>
    <span class="syll long">νος</span>
  </span>
  <span> </span>
  <span class="word">
    <span class="syll short">σέ</span>
    <span class="syll short">ο,</span>
  </span>
  <span> </span>
  <span class="word">
    <span class="syll long">Φοῖ</span>
    <span class="syll short">βε,</span>
  </span>
  <span> </span>
  <span class="word">
    <span class="syll short">πα</span>
    <span class="syll long">λαι</span>
    <span class="syll short">γε</span>
    <span class="syll short">νέ</span>
    <span class="syll long">ων</span>
  </span>
  <span> </span>
  <span class="word">
    <span class="syll short">κλέ</span>
    <span class="syll short">α</span>
  </span>
  <span> </span>
  <span class="word">
    <span class="syll long">φω</span>
    <span class="syll long">τῶν</span>
  </span>
  <span> </span>
</div>
```

Beyond syllable weights, meter (hexameter) and exact syllable lengths are provided. For certain texts, even more dimensions of analysis have been included in the markup.

All in all, the crawled Greek corpus is 60 MB's of text across 88 works in their separate HTML files. Importantly, it includes three tragedies by Aechylus (out of his extant seven), which means that all dichrona in open syllables from these works have been disambiguated.

My work on Hypotactic proved the most computationally intensive of the entire project, which implied a level of care for efficiency not necessary. When I tried to extract the vowel lengths immediately, the process was interminably long, and I had to break it down into steps, represented by the following scripts:

1. Crawling HTMLs:
   `crawl_hypotactic/crawl_hypotactic.py`

2. Creating huge SQL .db:
   `crawl_hypotactic/crawl_hypotactic_preprocessing.py`

3. Querying the .db:
   `crawl_hypotactic/crawl_hypotactic_db.py`

The first step, which I saw as "preprocessing", consisted in porting the information contained in the HTML files to an SQL, a format that not only can be read much faster but also reliably allows multithreading with the Python package `concurrent.futures`. Multithreading here means that the script can query information regarding several tokens at the same time, efficiently slicing processing time in half or more. The SQL database `metrical_patterns.db` consists of 1 456 400 (*sic*) pairs of tokens and metrical patterns, presented as in table 1. The script has simply entered underscores and carets after syllables depending on whether they were marked as long or short in the HTMLs.

With an efficient database and multithreading, I could start querying, using `crawl_hypotactic/crawl_`
The script has a number of notable subtleties: firstly, as we have noted in the epistemological

No.	Greek	Transcription
1	ὦ	ὦ_
2	παῖ,	παῖ,_
3	τέλος	τέ^,λος_
4	μὲν	μὲν_
5	Ζεὺς	Ζεὺς_

Table 1: First lines of `metrical_patterns.db`

section, poets cannot be trusted to provide canonical and consistent vowel lengths. Their job is to provide musical meters and not to be pedagogical or to respect etymology (leave that to Cratylus!). This means that only tokens that have the same vowel lengths in all of their appearances in the database should be used to disambiguate dichrona. Secondly we now wanted to filter out only tokens appearing in our wishlist.

The resulting was a list of 6 592 tokens, with at least one macron or breve each in a column of macrons using the same format as for Wiktionary.

2.2.3 Ifthimos

With merely 1 915 tokens macronized, the crawling of Ifthimos barely justified the fact that it was the most complicated and time-consuming crawl, taking me several weeks of dedicated work. However, Ifthimos is an interesting package, and if my work at all highlights its existence and potential, it will have been worthwhile.

Written in Ruby by classics enthusiast and professional physicist Ben Crowell, Ifthimos has a curious relationship to the internet. Launched on Github and advertised in a post by the author on the Ancient Greek subforum on Reddit less than a year ago, the package was subsequently moved from Github to less-known version control service Bitbucket, citing concerns about the ethics of Github, and the forum post was deleted by the author. I cannot speculate as to why Crowell has made the software less easy to find, but its by-now extreme online obscurity makes its mention here all the more worthwhile.

Ifthimos is harder to deploy than other packages I have mentioned and used. It is not included in RubyGems, the standard terminal package delivery software for Ruby, and depends on three other Ruby packages by Crowell, `tinycus`, `genos` and `lemming`, out of which only one is a gem in RubyGems. All of these packages hence have to be cloned manually from Bitbucket, with all the subtle file structure and import issues this entails. With that in mind, this exposition will be slightly more technical than my others.

Here is Crowell's own description, from the online docs (i.e. `README.md`), of how Ifthimos handles vowel length (my emphasis):

> The constructor tries to infer the lengths of *as many vowels as possible*, and for example if the part of speech data have been supplied, then it will know the correct lengths for any inflectional endings. Any vowel lengths that are unknown are recorded as unknown. If it's of interest, vowel lengths can also be set in the input string by supplying macronized characters. When the object is rendered into a string, vowel length is not shown explicitly by default. Facilities are supplied for showing it in several different human-readable ways, using the methods

```
puts w.to_s # ἄγνυμι
puts w.stringify # ἄγνυμι \n {}
puts w.describe # α- γνυ- μι with acute at 0
puts w.inspect # ἄγνυμι ( ῡ )
```

In the following example, the given part-of-speech (POS) information allows the library to infer by pattern matching that the upsilon in the inflectional ending is long. The inspect method returns a string representation of the word followed in parentheses by an upsilon with a macron, showing that the upsilon is long.

```ruby
require 'ifthimos'
genos = GreekGenos.new('classical')
w = Ifthimos::Mows.new(
    'ἄγνυμι',genos,[],
    pos:Ifthimos::Pos.new('v1spia---'))
print w.inspect # ἄγνυμι>> ( ῡ )
```

Despite the promising words about macronizing as much as possible, the package does not seem to know any roots or augments, and only macronized endings in my corpus. A further caveat is that the package does not include any brevia.

My first step was to create a script *in Ruby*, to call the macronizing abilities of Ifthimos and serve as something that I could later call from outside Ruby:

```ruby
require './tinycus-1.1.0/tinycus'
require './ifthimos'

token = ARGV[0]
pos = ARGV[1]

def ifthimos_macronizer(token, pos)
  # token = ἄγνυμι''
  # pos = 'v1spia---'
  # ifthimos_macronizer(token, pos)
  # => ῡ
  genos = GreekGenos.new('classical')
  mows = Ifthimos::Mows.new(
    token, genos, [],
    pos: Ifthimos::Pos.new(pos))
  inspection = mows.inspect

  # Check if the inspection ends with parentheses
  # and extract the content
  match = inspection.match(/\((.*?)\)$/)

  if match
    puts match[1]
  else
    puts "No match"
  end
end

ifthimos_macronizer(token, pos)
```

As the reader may have glimpsed from the code, I took parentheses to herald the presence of a macron in the return. To make this run in a subfolder within my main `greek-macronizer` working directory, I relativized all relevant imports in the Ifthimos files needed.

I was now ready to write a Python interface using the package `subprocess`. Once again, the process was too slow to finish, so I ended up adapting the code to use multithreading. The core functionality is the following:

```python
ifthimos_folder_path = 'ifthimos'
# Adjust the command to call the Ruby script from within its directory
command = ['ruby', 'macronize.rb', token, pos]
```

```
# Use the cwd parameter to set the current working directory to the ifthimos subfolder
result = subprocess.run(command, capture_output=True, text=True, cwd=ifthimos_folder_path)
```

I could subsequently query Ifthimos for each of the tokens in the main wishlist, saving them as a TSV.

2.2.4 LSJ

Adding macra to a mere 181 lines, LSJ was not necessary to crawl from a pragmatic viewpoint. However, methodologically it made a lot of sense to do it, and the way the particular website `https://LSJ.gr` is designed, made it easy. Why? Because the lemma of each entry has been extracted from the dictionary text and presented in an HTML-wise easily identifiable table at the head of the page, whose upper left entry includes macra and brevia, in case they are mentioned in the entry. The process of reformatting the crawled macra was identical to the Wiktionary case.

2.3 Conscientious Collation

All in all, 27 607 macra (including breves) were crawled. Figure 1 shows the number of tokens containing at least one instance of each respective source.

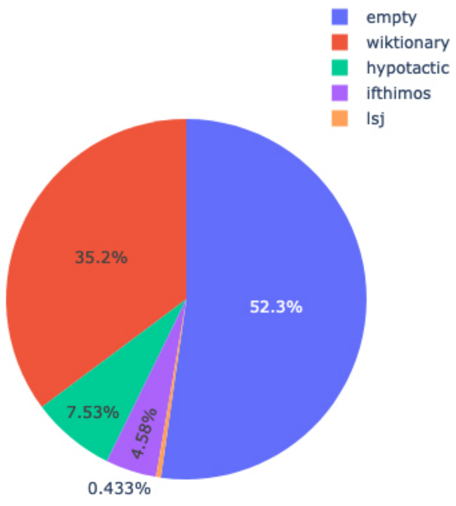

Figure 1: Macronized lines by provenience

However, all these macra were distributed among four different lists. Now, the goal was to create a single five-column TSV list, where beyond tokens, tags and lemmata, macra and brevia from all four sources would enter into a fourth column, with the sources listed in a fifth column.

The collation of the macron column had to be done orderly, though. As basic desiderata, I decided on the following:

- New macra (i.e. underscore/caret + one or two integers) should never overwrite existing ones
- A new macron is always inserted in the correct order among the existing ones
- The collation functions must be able to correctly handle inserting any number of new macra into a field with any number of macra, with the output containing the old macra plus those of the new macra that did not contradict the existing ones.

Practically, avoiding overwriting boils down to checking whether the integer part of a macron already exists before it is collated. To this end, I wrote a function that works like this:

```
    >>> ordinal_in_existing('^1_2_13', '^13')
    >>> True
```

Since the position counter of the new macron (13) matches one of the positions (1, 2, 13) of the existing macra, it returns True. Building on that function, I wrote a function that inserts a new macron in the correct order among existing ones, working in the following way:

```
    >>> insert_macron_in_order('_1^3^7', '^2')
    >>> '_1^2^3^7'
    >>> insert_macron_in_order('_1', '')
    >>> '_1'
    >>> insert_macron_in_order('', '^5')
    >>> '^5'
```

With the case of a single new macron solved, I made a function handling any type of complex updating of macron fields. Even though its parts had been unit tested (as all functions I make), the testing of this particular function demanded special attention. For even the slightest error in the collation algorithm could potentially corrupt and undo all the meticulous care that had gone into the sourcing of the data. The following test set aimed to represent all important case classes:

```
def test_collate_macrons():
    tests = [
        # Test cases: (existing_macrons, new_macrons, expected_result)
        ("", "^2", "^2", "Empty existing macrons"),
        ("^1_3^7", "", "^1_3^7", "Empty new macrons"),
        ("^1_3^7", "^2", "^1^2_3^7", "Single new macron addition"),
        ("_1^3^7", "^2_5", "_1^2^3_5^7", "Multiple new macrons addition"),
        ("^1_3^7", "^3", "^1_3^7", "Redundant macron addition"),
        ("^1_3^7", "^5^2", "^1^2_3^5^7", "Unordered multiple new macrons"),
        ("_1^3^7", "_4^2", "_1^2^3_4^7", "Complex case with all types"),
        ("", "^1^2^3", "^1^2^3", "All new macrons, none existing")
    ]
```

The usefulness of this robust collation code can not be overstated. Not only did it collate the four crawled sources, but it was imported and used anytime an algorithmic approach was used to create new macra, which is the subject of the next section. Indeed, given this code and the macron formatting code outlined previously, anyone who has a list of macronized tokens can safely and easily integrate it into my list (or vice versa).

2.4 Algorithmic Approaches

In this section I will outline some possible ways of generalizing the data collected in the previous chapter. I have no intention of being exhaustive; there are innumerable ways to go about and, as will be discussed, my hands are slightly tied, given that morpheme boundaries are not marked up in the tokens.

However, before we delve into grammatically knottier algorithms, three "easy wins" ought to be discussed. As a first measure, the prosodic rules (including the σωτῆρᾰ-rule) leveraged to filter for "true" dichrona in the sieving pipeline returned. A large number of tokens have dichrona both in the ultima and elsewhere, and these needed their ultimae macronized. That was the first easy win.

Remember that I made oxytone copies of all barytones early on, because I knew that none of the sources would string match barytones? The second easy win was to let all those barytones inherit the macra (if there were any) of their oxytone twins, yielding many freshly macronized lines.

The third maneuver can perhaps not be called “easy”, except for the fact that it required nothing over and above what was already in the token TSV.

For pairs of tokens

i) each with >2 syllables

ii) sharing lemma

iii) differing only with regard to the ultima and

iv) at least one of which has at least one macron not referring to a vowel in the last two syllables,

I wanted to generalize the macrons so that given, say,

```
μεγίστας a-p---fa-μέγας _7 ifthimos
μέγιστε a-s---mvs μέγας
μεγίστη a-s---fn-μέγας
μεγίστην a-s---fas μέγας
μεγίστης a-s---fgs μέγας
μέγιστον a-s---nas μέγας ^4 wiktionary
```

all lines would inherit the breve iota from the last line. In other words, in every qualifying pair the token with more macrons should bequeath whichever ones of them the one with fewer does not already have. The gist of a function realizing such a maneuver is captured by the following return statement (the variable names of which should be self-explanatory):

```python
return len(token_1) > 2 and len(token_2) > 2 and except_ultima1 and except_ultima2 and lemma_1 ==
    lemma_2 and only_bases(except_ultima1) == only_bases(except_ultima2)
```

However, that is easier said than done (or rather, easier written than run), since processing all possible pairs means processing a number of cases equal to the square of the number of lines. Since the number of lines is around 40 000, the number of pairs to be processed is in the order of magnitude of 1.6×10^9! Realizing that the processing would likely take longer than what was left of the semester, I once more turned to multithreading.

Having settled on a four-thread set-up (fitting my basic M1 processor), with each thread processing one fourth of the list, the Python script ran for more than two days straight before finishing. The result was inspiring: the source field of several thousand lines now contains ‘cognate’, indicating that they have been macronized by this script. The potential for improvement and extension of the algorithm is great, and further work along the same lines will be well worth it. Ideas for further experimentation include letting some token pairs that are highly alike share macrons even though they do not share lemma, given the many errors of lemmatization, and letting elided tokens count their elision character as an extra syllable.

With the “easy part” done, lets move on to the main algorithmic topic, morphology.

2.4.1 Morphology

Nominal Forms The easiest morphological pattern to formalize in unelided Greek nouns and verbs is the ending, since any final dichronon will regularly belong to the ending. This also means that the endings are in general well-handled on Wiktionary, and the code here merely serves to fill in a few holes (not more than five hundred of the tokens passing the functions developed below had not already been macronized by the sources). I will restrict myself to *nominal* forms, and specifically to those found in the table of endings on page 38 in CGCG.

The first step is to try to express the information regarding dichrona contained in the table in words, as compact as possible, here divided by declension:

- **1D:** Nom. and acc. and voc. sing. can be either, depending on whether or not it comes from ionian -η. Tricky...
  Acc. pl. => long -ας; however acc. pl. fem. of 3D are short: so if lemma is clearly 1D, ending is long.
- **2D: Nom. and acc. pl. (neut.): => short α** (the only dichronon; Same as neuter pl. 3D.)
- **3D: Dat. sing. => short ι** (all datives on iota are short) **Acc. sing. (masc.) => short α Nom. and acc. pl. (neutre) => short α**, i.e. if noun is masc. or neutre and ends on -α, that α is short[21] Dat pl: short ι; see dat sing. **Acc. pl. (masc) => short α**. Cf. 1D acc. pl.

After further abstraction, this summary yields the following three fully generalizable rules for use in functions acting on our token dictionary; note that in stating restrictions on tags I use so-called regular expressions, where carets and dollar signs indicate string beginnings and ends, dots are wildcards and square brackets are used where there are multiple options for a single character:

1. for tokens with tag meaning "acc. pl. fem."

   ```
   (^n.p...fa.$)
   ```

   and lemma ending with -η or -α, ending -ας is long

2. for tokens with tag meaning "masc. and neutre nouns"

   ```
   (^n.....[mn]..$)
   ```

   ending -α is short regardless of case

3. for datives

   ```
   (^n......d.$)
   ```

   ending -ι is short

The third and final step is to formalize these rules as three Python functions. All the auxiliary functions referred to are imported from earlier scripts, and their names should be self-explanatory:

[21]Note that some *dual* forms (1D on -ης) can be masculine on long -α, e.g. τὼ προφήτᾱ, ὁπλῑ́τᾱ. Probably hyper rare or inexistent in the corpus and not the case for 2D/3D and the most common masc. duals like χεροῖν, χεῖρε.

```python
def long_fem_alpha(token, tag, lemma):
    tag_pattern_acc = re.compile(r'^[na].p...fa.$')
    tag_pattern_gen = re.compile(r'^[na].s...fg.$')

    base_token = only_bases(token)
    base_lemma = only_bases(lemma)
    endings = ('η', 'α', 'ος') # ος'' is to allow fem adj of 2D

    if base_token.endswith('ας') and (tag_pattern_acc.match(tag) or tag_pattern_gen.match(tag))
        and base_lemma.endswith(endings):
        macron = f"_{ordinal_last_vowel(token)}"
        return macron

    return None
```

```python
def short_masc_neut_alpha(token, tag):
    tag_pattern = re.compile(r'^n.....[mn]..$')

    base_form = only_bases(token)

    if base_form.endswith('α') and tag_pattern.match(tag):
        last_vowel_position = ordinal_last_vowel(token)
        breve = f"^{last_vowel_position}"
        return breve

    return None
```

```python
def short_dat(token, tag):
    '''
    Avoiding brevizing the iota of e.g. ἁβροσύνηι
    '''
    tag_pattern = re.compile(r'^n......d.$')
    base_form = only_bases(token)

    if base_form.endswith('ι') and tag_pattern.match(tag):
        # Find the position of the last vowel
        last_vowel_position = ordinal_last_vowel(base_form)

        # Check if the last ι'' is part of a diphthong or has adscriptum
        last_iota_index = base_form.rfind('ι')
        prev_pair = base_form[last_iota_index - 1: last_iota_index + 1] if last_iota_index > 0
            else ''
        next_pair = base_form[last_iota_index: last_iota_index + 2] if last_iota_index < len(
            base_form) - 1 else ''

        if not (is_diphthong(prev_pair) or is_diphthong(next_pair) or has_iota_adscriptum(
            prev_pair) or has_iota_adscriptum(next_pair)):
            breve = f"^{last_vowel_position}"
            return breve

    return None
```

Prefixes Leaving endings behind, we simultaneously pass beyond the end of what we can safely aim to exhaustively accomplish without first marking our tokens with the exact location of certain morphological borders. These borders are most importantly the ones separating i) the root from the rest of the word and ii) the ending from the stem. A special case of i) is separating the prefix from the rest of the word (which practically means separating it from either the root or some kind of augment). I chose to focus on this special case.

Why? Because it is clear that the following prefixes (not intended to be an exhaustive list) and their numerous variants due to apocope, elision, aspiration and assimilation both represent a considerable part of all open-syllable dichrona in the corpus and are comparatively predictable to macronize:

1. ἀνα- (e.g., ἀναφέρω)[22]
2. ἀμφι- (e.g., ἀμφιθέατρον)
3. ἀντι- (e.g., ἀντίθεσις)
4. ἀπο- (e.g., ἀποθνῄσκω)
5. δια- (e.g., διάλογος)
6. δυσ- (e.g., δυστυχής)
7. ἐπι- (e.g., ἐπίθεσις)
8. κατα- (e.g., καταστροφή)
9. μετα- (e.g., μεταμόρφωσις)
10. παρα- (e.g., παραβολή)
11. περι- (e.g., περίμετρος)
12. συν- (e.g., συμπάθεια)
13. ὑπερ- (e.g., ὑπεράνω)
14. ὑπο- (e.g., ὑπόθεσις)

I ran a simple test on one of them, excluding all sandhi variations, using the function

```python
def brevize_syn(word):
    base_form = only_bases(word)
    if base_form.startswith('συν'):
        return '^2'
    return None
```

which yielded 273 macra.

I do not know of any presently existing open-source algorithm that can mark-up prefixes, nor perform other morphological cuts.[23] However, as Eric Cullhed predicted in his description of Makron, it quickly become clear to me that the future of morphological analysis will most certainly be AI. But not without a caveat: even within a single family of models, like ChatGPT, a quick test revealed that performance is strikingly model-dependent.

I made a test consisting of a stretch of one hundred entries taken from a prefix-heavy portion of my tokens list, of which the following will give a good sense, with the ampersand (&) indicating the "cut":

[22]This is perhaps the trickiest, because in its elided form ἀνα coincides with the euphonic form of *alpha privativum*. *Alpha privativum* is among the naughtier Greek prefixes, as it is often short, but also often long (e.g. ᾱ̓θάνατος).

[23]I tried and investigated Tauber 2020 in detail, but came to realize that it basically only supported lexical (uninflected) forms.

```
ἀνδρῶνας
ἀνδρῶν
ἀν&έσχε
ἀν&εβόησε
ἀν&έβλεφ'
ἀν&εβόα
...
ἀνεμόεν
ἀνεμόεντι
ἀνέμοις
ἄνεμον
ἄνεμος
ἀνέμων
ἀνεμώκεος
ἀνέμων
ἀν&έξεται
ἀν&έξηι
...
```

The tricky part is, of course, to realize which ἀν is the actual prefix (ἀνα- or *alpha privativum*) and which is simply part of a root that incidentally shares letters with the prefix.

Here ChatGPT 4 did exactly the same cuts as I did, whereas the newer 4o ("omni") basically made the cut as ἀνε& throughout.[24]

Given the extremely many tricky interfaces (sandhi) that can appear between the root and these prefixes, an extensive bug testing of ChatGPT's capabilities would be necessary before implementing it on the whole corpus, which is beyond the scope of the present work. However, it is no doubt feasible, and the topic provides a segue to the discussion of the conclusion.

3 Conclusion

3.1 Summary

In this report I have described how I created "a dictionary compiled from grammars and lexica" (among other sources), serving to disambiguate "the ambiguous vowel length of α, ι and υ", the dichrona. Given three hundred thousand morphologically-tagged tokens from all the extant Ancient Greek tragedies, I culled out around forty-two thousand tokens containing the ambiguities in question. I crawled two dictionaries, a Ruby lemmatizer and a repository

[24] 4o is the first model to be trained not only on text, but also on audio and visuals. As with all AI technology, the correlation between performance and training is irreducibly unpredictable, so there is nothing surprising about a newer model behaving worse. (Cf. OpenAI 2023) For the record, my prompt looked like this:

> You are a pithy Ancient Greek morphological analysis machine, skilled in separating the prefix from the rest of the word with a single ampersand (&). E.g., given the prompt 'προσβαλοῦσα' you return only 'προσ&βαλοῦσα'. If there is no prefix, as ἤνεγκα, you return the word as-is. NB: You cannot add any other characters than ampersand (&), and only one & per word.
>
> Only analyse the following prefixes, and numerous variants due to apocope, elision, aspiration, assimilation and their combinations: ἀ-, ἀν- (so called alpha privativum, e.g., ἀθάνατος), ἀμφι- (e.g., ἀμφιθέατρον), ἀντι- (e.g., ἀντίθεσις), ἀπο- (e.g., ἀποθνήσκω), δια- (e.g., διάλογος), δυσ- (e.g., δυστυχής), εἰσ- (e.g., εἰσάγω), ἐκ-, ἐξ- (e.g., ἐκκλησία), ἐν- (e.g., ἔνδοξος), ἐπι- (e.g., ἐπίθεσις), κατα- (e.g., καταστροφή), μετα- (e.g., μεταμόρφωσις), παρα- (e.g., παραβολή), περι- (e.g., περίμετρος), προ- (e.g., προφητεία), προσ- (e.g., πρόσθεσις), συν-, συμ- (e.g., συμπάθεια), ὑπερ- (e.g., ὑπεράνω), ὑπο- (e.g., ὑπόθεσις).
>
> When given a tsv, you respond with a list of the analyses, exactly one word per line.

of scanned verse, and carefully collated the macron data from all four sources. I then showed a number of ways in which the percentage of macronized dichrona can be increased algorithmically, three solely “shuffling around” the data already contained in the sourced list, and two requiring formalization of morphological rules, taking as examples nominal forms and prefixes.

Relevant to metrical software projects like *Dionysus Recomposed*, there are 55 495 dichrona in open syllables (“non-hidden”) in the dictionary, out of which more than twenty thousand were macronized by the sources and the examples of algorithmic approaches together. The number of unique tokens in the list is 32 535, with the total number of macra and brevia roughly equal, which means that, in theory, each token has on average at least one of their dichrona disambiguated.

3.2 Whither Greek Macronizer?

In the best of worlds, the final *Makron* software will be akin to the Latin macronizer of Johan Winge, albeit with its own set of problems and possibilities.[25] Perhaps because every single Latin vowel is dichronon, Latin dichrona were already rather thoroughly and orderly compiled before Winge commenced his work, enabling him to focus on the morphologizers and to disambiguate minimal pairs.

Minimal pairs in Greek, on the other hand, are comparatively rare—the only *truly* common example being ᾶ̆ν = ἐὰν versus ᾰ̓ν simpliciter, whereas the salient examples from the present and imperfect indicatives of ῐ̔́στημι, darling conjugation of dichrona scholars, e.g. ῐ̔́στης (present ‘you’re putting’ versus imperfect ‘you used to put’), are not present at all in the tragic corpus. And when it comes to compilations of Greek dichrona—well, you can’t call the situation orderly and thorough.

Thus the weights must be shifted *mutatis mutandis*: the importance of morphological analysis for a Greek macronizer is less than for a Latin, whereas the importance of the quality control of the raw dichrona data is greater.

I believe that a combination of leveraging AI for automatic morphological cuts, programming using them, and simply manual sorting of the large amount of odd redundancies will quickly sort out the remaining dichrona in our corpus. But let’s not stop there. All the pipelines, scripts and work flows set up for the present project are ready to receive new larger corpora, ideally leading to the macronization of the open syllables of the entire Ancient Greek corpus, e.g. as represented by the TLG. Yes—not only verse! As scholars are becoming more and more aware of, already the ancients themselves marked the prosody of prose texts, whether simply disambiguating or as rhetorical and stylistic analysis.[26] If they could, we should.

I claim that this program is not idealistic; for one thing, there is a strong positive loop inherent in macronization, with the already macronized corpus serving to triangulate the remaining parts. The final proof of this claim will be part of my doctoral thesis.

[25]See Winge 2015, the web interface and the Github repository.

[26]As one recent scholar put it, “the interest in prosodic signs witnessed in grammatical treatises, annotated literary texts, and advanced school exercises spilled over into the realm of everyday texts, where, depending on the genre and purpose of the documents they were transcribing, writers found occasional practical use for their training in προσῳδίαι” (Ast 2017, p. 157). On vowel length diacritics, see specifically Colomo 2017.

4 Appendix I: macrons.tsv

Here is an example of fifty lines from the middle of the TSV (Tab-Separated Values) macron dictionary:

```
token tag lemma macron source
κατειργάσασθε v3paie---μικατειργάσκω ^2^8^10 wiktionary
μικρόν a-s---ma-μικρόν _2 wiktionary
μικρά a-s---fn-μικρός _2^5 wiktionary
μικρά a-p---na-μικρός _2^5 wiktionary
μικρὰ a-s---fn-μικρός _2^5 hypotactic,barytone
μικρὰ a-p---na-μικρός _2^5 hypotactic,barytone
μικράν a-s---fa-μικρός _2_5 wiktionary
μικρὰν a-s---fa-μικρός _2_5 barytone
μικρᾶς a-s---fg-μικρός _2_5 ifthimos
μικροῖς a-p---nd-μικρός _2 wiktionary
μικροῖς a-p---md-μικρός _2 wiktionary
μικρόν a-s---na-μικρός _2 wiktionary
μικρόν a-s---ma-μικρός _2 wiktionary
μικρὸν a-s---na-μικρός _2 barytone
μικρὸν a-s---ma-μικρός _2 barytone
μικρός a-s---mn-μικρός ^2 wiktionary
μικρὸς a-s---mn-μικρός ^2 barytone
μικροῦ a-s---mg-μικρός _2 wiktionary
μικροῦ a-s---ng-μικρός _2 wiktionary
σμικρά a-p---na-μικρός _3_6 wiktionary
σμικρά a-s---fn-μικρός _3_6 wiktionary
σμικρὰ a-s---fn-μικρός _3_6 barytone
σμικρὰ a-p---na-μικρός _3_6 barytone
σμικράς a-p---fa-μικρός _3_6 wiktionary
σμικρᾶς a-p---fa-μικρός _3_6 ifthimos
σμικρᾶς a-s---fg-μικρός _3_6 ifthimos
σμικροί a-p---mn-μικρός _3 wiktionary
σμικροὶ a-p---mn-μικρός _3 barytone
σμικροῖς a-p---nd-μικρός _3 wiktionary
σμικρόν a-s---na-μικρός _3 wiktionary
σμικρόν a-s---nn-μικρός _3 wiktionary
σμικρόν a-s---ma-μικρός _3 wiktionary
σμικρὸν a-s---ma-μικρός _3 barytone
σμικρὸν a-s---nn-μικρός _3 barytone
σμικρὸν a-s---na-μικρός _3 barytone
σμικροῦ a-s---ng-μικρός _3 wiktionary
σμικροῦ a-s---mg-μικρός _3 wiktionary
σμικρῷ a-s---md-μικρός _3 wiktionary
Σμικρῷ a-s---md-μικρός
σμικρῶν a-p---ng-μικρός _3 wiktionary
σμικρῶν a-p---mg-μικρός _3 wiktionary
μικροτέρων a-p---mg-μικροτέρων
μικρῶι a-s---md-μικρῶι
μίλακος n-s---fg-μίλακός
μίλακι n-s---fd-μίλαξ ^6 breve_ultima
μιμήσομαι v1sfim---μιμαίνω _2 wiktionary
Μίμαντα v-sapama-μίμας ^2_7 hypotactic
μιμεῖσθαι v--anm---μιμέομαι _2 wiktionary
μιμοῦ v2spme---μιμέομαι _2 wiktionary
μίμημ’ n-s---nn-μίμημ
μίμημ’ n-s---na-μίμημ
μιμήματα n-p---nn-μίμημα ^8 breve_ultima
...
```

5 Appendix II: Postag Legend

The nine positions of POS tags (- - - - - - - - -)[27]:

```
1:  part of speech

    n noun
    v verb
    t participle
    a adjective
    d adverb
    l article
    g particle
    c conjunction
    r preposition
    p pronoun
    m numeral
    i interjection
    e exclamation
    u punctuation

2:  person

    1 first person
    2 second person
    3 third person

3:  number

    s singular
    p plural
    d dual

4:  tense

    p present
    i imperfect
    r perfect
    l pluperfect
    t future perfect
    f future
    a aorist

5:  mood

    i indicative
    s subjunctive
    o optative
    n infinitive
    m imperative
    p participle

6:  voice

    a active
    p passive
    m middle
```

[27]Source: https://github.com/cltk/greek_treebank_perseus

```
    e medio-passive

7: gender

    m masculine
    f feminine
    n neuter

8:  case

    n nominative
    g genitive
    d dative
    a accusative
    v vocative
    l locative

9:  degree

    c comparative
    s superlative
```

Bibliography

Allen, W. Sidney (1987). *Vox Graeca.* 3rd ed. Cambridge.

APA (2024). *Technical Details about GreekKeys Unicode 2008.*

Ast, Rodney (2017). “Signs of Learning in Greek Documents: The Case of Spiritus Asper.” In: *Signes Dans Les Textes, Textes Sur Les Signes. Érudition, Lecture et Écriture Dans Le Monde Gréco-Romain.* Ed. by G. Nocchi Macedo and N. Scappaticcio. Papyrologica Leodiensia 6. Presses Universitaires de Liege.

Chamberlain, David (2023). *hypotactic.Com.*

Classical Studies, Society for (2021). *ABOUT NEW ATHENA UNICODE v. 5.008 (August 2021).*

Colomo, Daniela (2017). “Quantity Marks in Prose Papyri.” In: *Signes Dans Les Textes, Textes Sur Les Signes. Érudition, Lecture et Écriture Dans Le Monde Gréco-Romain.* Ed. by G. Nocchi Macedo and N. Scappaticcio. Papyrologica Leodiensia 6. Presses Universitaires de Liege.

Crowell, Ben (2024). *Ifthimos.*

Devine, A. M. and Laurence D. Stephens (1994). *The Prosody of Greek Speech.* New York-Oxford.

Emde Boas, Evert van et al. (2019). *The Cambridge Grammar of Classical Greek.* Cambridge: Cambridge University Press Cambridge.

Herodianus, Aelius (1870). *Περὶ Διχρόνων.* Ed. by Augustus Lentz. Vol. 3.2. Grammatici Graeci. Leipzig: Teubner. 7-20.

Johnson, Kyle P. et al. (2014–2021). *CLTK: The Classical Language Toolkit.*

Kardos, Marton and Jan Kostkan (2023). *odyCy.*

Kuhn, Friedrich (1892). *Symbolae Ad Doctrinae Περὶ Διχρόνων Historiam Pertinentes.* Breslau.

Lauxtermann, Marc (2022). “Buffaloes and Bastards: Tzetzes on Metre.” In: *Tzetzikai Epeynai.* Eikasmos: Studi Di Eikasmos Online 4. Bologna: Pàtron Editore, pp. 117–132.

ObsequiousNewt (2017). *Grc-Conj.*

OpenAI (2023). *GPT-4 Technical Report.*

Pontani, Filippomaria (2020). “A new Herodianic treatise on dichrona and a new fragment of Hipponax.” In: *Revue de philologie, de littérature et d’histoire anciennes* XCIV.2, pp. 163–191.

Probert, Philomen (2003). *A New Short Guide to the Accentuation of Ancient Greek.*

Tauber, James K. (2017). *Pyuca: Python Unicode Collation Algorithm Implementation.*

— (2019). “Character Encoding of Classical Languages.” In: *Ancient Greek and Latin in the Digital Revolution.* Ed. by Monica Berti. Berlin, Boston: De Gruyter Saur, pp. 137–158.

— (2020). *Greek-Inflexion.*

Teodorsson, Sven-Tage (1978). *The Phonology of Attic in the Hellenistic Period.* Studia Graeca et Latina Gothoburgensia.

Unicode (2023a). *Unicode Chart Combining Diacritical Marks.*

— (2023b). *Unicode Chart Greek and Coptic.*

— (2023c). *Unicode Chart Greek Extended.*

— (2024a). *Greek Language and Script.*

— (2024b). *Unicode FAQ on Greek Language and Script.* URL: https://www.unicode.org/faq/greek.html (visited on 02/28/2024).

Winge, Johan (2015). “Automatic Annotation of Latin Vowel Length.” Uppsala.