Synthesis of Reading Editions for Greek
Lee Butterman   May 2006

Reading Greek at the collegiate level has stayed the same for 100 years: text+dictionary+criticalnotes=translation. Dictionaries are inevitable for reading Homer, whose vocabulary is one-third hapax; later authors strove for even more nuance and *variatio*. Repetitive dictionary lookups remove much of the joy of decoding a language. Alternately, there are picture books full of notes on vocab, grammar, syntax, culture, and such. These remove much of the effort of decoding the language. This research explores how an edition of Greek text can be optimized for fast comprehension, but still stay academically honest. The editions improve comprehension speed, and invite interdisciplinary analogies.

History

SRE’s design comes from the best features of many different editions.

When I was in 9th grade, we started to read Caesar, from the textbook, to xeroxes of a critical edition for half a month, back to the textbook. The critical edition was my first introduction to authentic Roman texts and writing, difficult initially. When we moved back to the book, it was faster going. I compared the book’s Caesar with the xeroxed Caesar, to see how far off it was, and it was identical. The textbook was undeniably easier, even with its few notes.

In 11th grade, we used Pharr's Aeneid. The on-page vocab and notes were very helpful, though re-inking the same plates for decades made the tiny type of some notes difficult to read, and the vocabulary at the back was partitioned by popularity, so a word more frequent than expected could take two lookups. And many other Latin editions had their benefits and drawbacks: Garrison's Catullus had copious whitespace for marking notes, Cambridge's font was easy to read in any light, etc.

Greek I started just last summer, and my experience is an order of magnitude less. But I liked Hanson and Quinn's excerpts, vocabulary and notes at the botton of the page, I enjoyed Draper's Iliad 1, good textual criticism with sometimes-too-helpful grammatical notes, and I love the Greek font of the Oxford and Loeb editions.

This reading edition synthesis has been for my own benefit thus far, so it is primarily designed to accomodate my own weaknesses: short attention span, low vocabulary retention rate, poor vision. A dictionary is fantastically distracting—a list of a language's words is an index of the thoughts expressed in the culture, each a garden path of diversion. English and Greek vocabulary is internalized best through reading good books, but taught through sitting down with lists of words.[1] The texts have been designed to be very legible, using a thicker font than the default, and arranging notes for easiest visual searching.

[1]An SAT coach came to my high school, and told of a girl who had recently come in from Russia, having read Dickens in Russian, who then read all of Dickens in English, for several months, and aced the Verbal portion of the SAT. He said that her case was unusual, and advised memorizing word tables. Cf. steroids.

The following is a sample input to SRE, with its output.

```
w(\s ei)pw\n qa/mnwn u(pedu/seto di=os *)odusseu/s,@Always \bcode{dios}.@
e)k pukinh=s d' u(/lhs pto/rqon kla/se xeiri\ paxei/h|
fu/llwn, w(s r(u/saito@He is a concealer.@peri\ xroi\+ mh/dea fwto/s.
bh= d' i)/men w(/s te le/wn o)resi/trofos a)lki\ pepoiqw/s,
o(/s t' ei)=s' u(o/menos kai\ a)h/menos, e)n de/ oi( o)/sse
dai/etai: au)ta\r o( bousi\ mete/rxetai h)\ o)i/essin
h)e\ met' a)grote/ras e)la/fous: ke/letai de/ e( gasth\r
mh/lwn peirh/sonta kai\ e)s pukino\n do/mon e)lqei=n:
w(\s *)oduseu\s kou/rh|sin e)uploka/moisin e)/melle
mi/cesqai, gumno/s per e)w/n: xreiw\ ga\r i(/kane.
smerdale/os d' au)th=|si fa/nh kekakwme/nos a(/lmh|,
tre/ssan d' a)/lludis a)/llh e)p' h)io/nas prou)xou/sas:
oi)/h d' *)alkino/ou quga/thr me/ne: th=| ga\r *)aqh/nh
qa/rsos e)ni\ fresi\ qh=ke kai\ e)k de/os ei(/leto gui/wn.
sth= d' a)/nta sxome/nh: o( de\ mermh/ricen *)odusseu/s,
h)\ gou/nwn li/ssoito labw\n e)uw/pida kou/rhn,
h)= au)/tws e)pe/essin a)postada\ meilixi/oisi
li/ssoit', ei) dei/ceie po/lin kai\ ei(/mata doi/h.
```

ὣς εἰπὼν θάμνων ὑπεδύσετο δῖος Ὀδυσσεύς,[1]
ἐκ πυκινῆς δ᾽ ὕλης πτόρθον κλάσε χειρὶ παχείῃ
φύλλων, ὡς ῥύσαιτο[2] περὶ χροῒ μήδεα φωτός.
βῆ δ᾽ ἴμεν ὥς τε λέων ὀρεσίτροφος ἀλκὶ πεποιθώς,
ὅς τ᾽ εἶσ᾽ ὑόμενος καὶ ἀήμενος, ἐν δέ οἱ ὄσσε
δαίεται: αὐτὰρ ὁ βουσὶ μετέρχεται ἢ ὀίεσσιν
ἠὲ μετ᾽ ἀγροτέρας ἐλάφους: κέλεται δέ ἑ γαστὴρ
μήλων πειρήσοντα καὶ ἐς πυκινὸν δόμον ἐλθεῖν:
ὣς Ὀδυσεὺς κούρῃσιν ἐυπλοκάμοισιν ἔμελλε
μίξεσθαι, γυμνός περ ἐών: χρειὼ γὰρ ἵκανε.
σμερδαλέος δ᾽ αὐτῇσι φάνη κεκακωμένος ἅλμῃ,
τρέσσαν δ᾽ ἄλλυδις ἄλλη ἐπ᾽ ἠιόνας προὐχούσας:
οἴη δ᾽ Ἀλκινόου θυγάτηρ μένε: τῇ γὰρ Ἀθήνη

αγροτερος	wild
αημι	to breathe hard, blow
αθηναι	the city of Athens
αλκι	might, strength
αλλυδις	elsewhither
αλμη	sea-water, brine
βαινω	to walk, step
βη	baa
βους	cow
γαστηρ	the paunch, belly
γυμνος	naked, unclad
δαιω	to light up, make to burn, kindle
δαιω	to divide
δομος	a house
ειδομαι	are visible, appear
ελαφος	a deer
ευπλοκαμος	with goodly locks, fairhaired
η	or
ηε	or, whether.
ηιων	a sea-bank, shore, beach
θαμνος	a bush, shrub
ικανω	to come, arrive
κελομαι	to urge on, exhort, command
κλαω	to break, break off
κορη	a maiden, maid, damsel
λεων	a lion
μετερχομαι	to come or go among
μηδος	counsels, plans, arts, schemes
μηδος	the genitals
μηλον	a sheep or goat
μηλον	tree-fruit
μιγνυμι	to mix, mix up, mingle, properly of liquids
οιος	alone, lone, lonely
οις	ram
ορεσιτροφος	mountain-bred
οσσα	a rumour
παχυς	thick, stout
προεχω	to hold before
πτορθος	a young branch, shoot, sucker, sapling
πυκνος	close, compact
ρυομαι	to draw to oneself
σμερδαλεος	terrible to look on, fearful, aweful, direful
τρεω	to flee from fear, flee away
υλαω	to howl, bark, bay
υλη	forest-trees
υω	to send rain, to rain
φαος	light, daylight
φυλλον	a leaf;
φως	a man
χειρις	a covering for the hand, a glove
χρεω	want, need;
χρως	the surface of the body, the skin

[1]Always διος.
[2]He is a concealer.

θάρσος ἐνὶ φρεσὶ θῆκε καὶ ἐκ δέος εἵλετο γυίων.
στῆ δ᾽ ἄντα σχομένη: ὁ δὲ μερμήριξεν Ὀδυσσεύς,
ἢ γούνων λίσσοιτο λαβὼν ἐυώπιδα κούρην,
ἦ αὔτως ἐπέεσσιν ἀποσταδὰ μειλιχίοισι
λίσσοιτ᾽, εἰ δείξειε πόλιν καὶ εἵματα δοίη.

αντα	over against, face to face
ανταω	to come opposite to, meet face to face, meet with
αποσταδον	standing aloof
αυτως	in this very manner, even so, just so, as it is
γονυ	the knee
γυιον	a limb
γυιοω	to lame
δεος	fear, alarm, affright
ειμα	a garment
ενγυαω	to give or hand over as a pledge
ευωπις	fair to look on
θαρσος	courage, boldness
λισσομαι	to beg, pray, entreat, beseech
μειλιχιος	gentle, mild, soothing
μερμηριζω	to be full of cares, to be anxious or thoughtful, to be in doubt
φρην	the midriff or muscle which parts

Characteristics of SRE

SRE is oblivious to the meanings of words. The tag for ‘he walks’, *βῆ δ′ ίμεν*, glosses *βη* as the onomatopoetic sheep sound ‘baa’, which makes little sense in context. The goddess Athena is glossed as Athens. But its thoroughness means that it includes *counsels/plans/schemes* for *μήδεα*, as well as *genitals*—he drew his genitals close to his skin, but soon he will debate whether he will grab her knees as briny as he is, or entreat her with words, keeping his schemes close to his body as well. And similarly, *ύλη* ‘forest-tree’ sounds like *υλάω* ‘to howl’, emphasizing the wildness of Odysseus when he washes up. A dictionary would not facilitate finding such an allusion, when looking up a word that scansion and context predict is a genitive noun. Indeed, this is an important first step to enumerating wordplay, especially in poetry, where the meaning is larger than the sum of the words. Note that it is not beneficial to include everything a computer can determine about a word: good design is as much about exclusion as inclusion, especially for a static medium like a printed page; also, as a computer analyzes forms in greater detail, this approaches the information content of a Loeb, and Loebs are not designed for a reader to focus on Greek text.

The reading editions, above all, are wonderfully usable.

I was the sole usability tester; still, I read 45-60 lines an hour with SRE’s texts. I was reading 15-20 lph with paper books, 25-30 lph with Perseus online. I achieved a three-fold increase in reading speed from any other physical book, with perceptibly greater satisfaction after reading a passage, and a much better ability to sightread.

Perseus' morphological analysis cannot resolve all words, and I did use Autenrieth. But I was mostly able to use the reading editions as is, jotting down notes, completely focused on the page. Part of it comes from reading at a higher level, and part of it comes from preservation of flow.

Interdisciplinarities

Alan Perlis, a famous computer scientist, observed:

> *A programming language is low-level*
> *when its programs require attention*
> *to the irrelevant.*

Similarly, a reading edition is low-level when its readers require attention to the irrelevant. Translation is looking at a page of Greek text and generating a mental parse tree in colloquial English of the meaning. Thumbing through many different books is irrelevant. Computers are much better at extracting data from an ordered table. If a computer creates a text with useful vocab marked, a reader who knows the grammar can focus on how the words fit together.

The effect of focus while reading and understanding Greek is similar to the effect of focus while reading and understanding computer code.

Reading one line of code means understanding the lines of code preceding it, the lines of code following it, the variables it accesses, the data structures that it works on, the functions that it calls, the library routines it uses, the peculiarities of its language's syntax, the comments around it — this is all relevant to the meaning of one line of code.

A well-designed function has one specific task, which each of its lines of code collaboratively achieve. It in turn calls other functions (even perhaps itself) and, understanding how the system works from the ground up, a programmer can grasp the nuances of how the code works, how the parts join together.

This is a difference in degree, not in kind, from translating Greek. The *μήδεα/ὑλης* wordplay, the ironic nature of Odysseus' epithet *δῖος* as he is washed up in tatters like driftwood, the non-lethality of him as a lion now versus in Book 22, the confused meter of line 12 where the girls scatter: poetry synthesizes beauty with meaning, and grasping the meaning as soon as possible means a reader can focus on enjoying the beauty, ultimately having more fun.

This is similar to what Eastern thought would term 'being at one with a text', and what Western philosopher Mihaly Csikszentmihalyi called 'flow'. Flow is full immersion in a task, with a limited field of attention, a loss of conscious self-awareness, a control over the task and rewards from the task. Turning pages, frequently changing between languages, grasping different physical books, frustration in lack of note or dictionary entry: these break flow. They expand the field of attention by requiring attention to the irrelevant, they make one self-aware of the act of translation, they take control away from the act of translating, they (besides the rare notes that change one's understanding of a text) are ultimately without reward.

"A grad student who's too lazy to use a dictionary? Back in my day..." is a valid response. But programming has similar parallels. Consider the problem of taking the sum of the numbers in a file. A program has to ask the operating system for a file handle to read in the file, save the file handle, allocate a buffer, read the bytes from the file into the buffer until encountering one of three possible newline sequences into the buffer, make sure that the buffer does not overflow which is the source of 70% of all serious security holes, make sure that the file was read successfully, convert the list of bytes read into a machine number, close the file handle. And in that list, the only thing left out was the only important step: add a value to an accumulator value. The details are irrelevant to the task at hand. Little tricks to optimize on a case-by-case basis (fractional numbers take much longer to process than integers, usually the newline character stays constant in a file) are vastly outweighted by the ease of writing `{ sum += $0 }`. A study from 30 years ago, about a project 10 years earlier, still holds true: humans have a fixed tolerance for complexity, programmers generally code the same number of lines per hour regardless of language, and the higher the level at which one works, the more efficient one can be. Similarly,

there is no need to temporarily stop thinking about the parse tree of the sentence being read, visually search for a dictionary, grasp it in hand and open it, flip quickly to the general area of the lemma, sequentially scan a page for the lemma, follow all *αίξασκον*: *see αίσσω* entries and such and repeat until finding the lemma, even if the spine is broken to the section around *τοιο*–, or if a rolodex of cards serves as a dictionary.

Programmers also love to have notes about the language as they read and write it. Most programming languages have documentation that can easily be imported to a development environment. But the most authoritative answer to what some code means is to evaluate it, to type it into a Read-Evaluate-Print Loop. Almost all languages used in successful software startups (which, by necessity, work quickly to deliver the most features in the least time) have REPLs and online documentation—the equivalent of vocabulary notes and commentary on the page.

Coding at a higher level improves flow, limiting the attention field to the specific problem, and REPLs improve flow, creating a dialogue between programmer and computer in code. When a language becomes a medium of thought, awareness of the dissimilarity of the language drops greatly. And I felt enormously productive reading—my attention was only on the page, eyes darting back and forth, jotting down notes beside the words, hexameters gliding along.

SRE is primarily a framework for annotating text; it automatically marks vocabulary, and anything delimited by ampersands is inserted as a numbered footnote below the lemma-definition notes. SRE builts input for a very powerful typesetter, TEX, and these ampersand commands can use the full functionality with accentuation, formatting, and mathematical symbols. (TEX was initially designed as a typesetting system for math papers; this document is typset in TEX.)

This summer, SRE will have input filters to automatically fetch texts from Perseus, and, integrated somehow therein, it could extract all cross-references to a particular text and include the most helpful ones. Also, there is the possibility of a Wikipedia-style collaborative generation of a commentary, increasing on Perseus' grammar marking experiment ('voting' on the morphological analyses of a word). Or SRE could even not make PDFs, and generate HTML tables with text on one side and definitions (and notes) on the other; this would make it into another way of formatting Perseus output, the others being the old white/beige/blue Perseus 3 and the new white/grey/blue Perseus 4. And, if coupled with a similarly robust way of inserting annotations to HTML and changing the look of output, this would be enoirmously helpful. SRE now is just a first step, with its output and its internals designed for making PDF files.

Graphic Design

SRE necessitated several design decisions that strayed from TEX defaults. These are decisions that no editor of an edition of Greek poetry would have had to make; they arose from new limitations on fonts, page shapes, and usage.

SRE does not create any printable documents. It creates input to TEX, which in turn creates PDFs. TEX's default font is Computer Modern, which has an air of elegance at first, but after hours of reading, start to look fragile and spindly. The same can be said of the corresponding Greek font: μῆνιν ἄειδε ϑεὰ Πηληϊάδεω Ἀχιλῆος, οὐλομένην, ἧ μυρία βίβλια ϑῆκεν ἄχρηστα. In bold or italics it's a disaster: *ηϱάμην μὲν εγὼ, σέϑε, Φῶντα, παλαί ποτα* **χἅπαντα τὰ πράγματα οὔποτε ποιεῖ τὶ καλὸν**. The omicron is off-kilter, the rho's head is sideways, the breathings are ridiculous, the tildes are wispy, just a complete mess. The sans serif font is a little better. The glyphs, the letter forms, are clearer indivdually; combined they make words hard to read, because there's no lines at the baseline and x-height of the letters guiding readers' eyes along. The Greek font, though, does not have any serifs (or readability) to lose. μῆνιν ἄειδε θεὰ Πηληϊάδεω Ἀχιλῆος, οὐλομένην, ἣ μυρία βίβλια θῆκεν ἄχρηστα. Better not stray out of upright text, though: *ηράμην μὲν εγὼ, σέθε, Φόντα, παλαί ποτα*. Look how far the mu is from the alpha and eta, or the lambda from the alphas; it's badly kerned. Fonts for polytonic greek are much more difficult to find, compared to fonts for ancient Latin. (This font, Vera, was lovely for doing Latin.) I'm still not happy with the font situation, and I'm working to remedy it. The sans serif body / bold sans lemmata / roman definition is workable, but a hodgepodge. (Why not use the going Greek font, like $\mu\acute{\eta}\delta\epsilon\alpha$? That was typed as `\mu\acute\eta\delta\epsilon\alpha`, which is much more difficult than just `mh/dea`. This is also a potential summer project, to find beautiful fonts for polytonic greek.)

Pagination turned out to be useless, so I pushed the bottom margin down to fill out each column. Another option is to use the page footer to mark the line numbers of the text on the page, and remove the numbers from the left margin. This suddenly becomes practical if there are 15 lines of text a page instead of 50. Intuitively, it seems difficult, but interesting to see the results of.

SRE creates reading editions optimized for Flow. To glance away from the text breaks concentration, so the goal is to minimize the time spent finding and reading English. Autenrieth has bold lemmata, which is a good idea, but it also has etymologies, principal parts, scansions, a dagger indicating each hapax, all of which are unnecessary most of the time. Is there any imminent need to know the gender of $\epsilon\lambda\acute{\alpha}\phi o\upsilon\varsigma$, or the pluperfect of $\delta\alpha\acute{\iota}\omega$? A dictionary does not consider that: it is reference, non-context-specific, comprehensive for its audience. SRE's vocab is locally chosen, and non-authoritative (it takes whatever Perseus has got)—intentionally throw-away, because the context of a word colors its meaning better than a dictionary can. The origins of Autenrieth suggest that this sort of experimentation with presentation is a good thing: *“The editor’s own experience leads him to believe that a pupil with this dictionary in his hands will easily read two pages of Homer in the time which, with the large lexicon, would be required for one page.”* My own experience leads me to believe that a reader with a dictionary-in-footnotes will read ten pages of Homer in the same time.

Source Code

All of the source code is divided into three parts. The `bigfoot` footnote package shuffles footnotes, the TEX code establishes the typesetting environment, and the Ruby program transforms the text.

Easiest things first. The footnote package’s style file, `bigfoot.sty`, has the code

```
\stepcounter{FN@totalid}
```

which increments the number of the footnote each time one pops up in the text. If you override this counter, you could put any footnote in running text, give it a number, and it’s in that order at the bottom of the page. And each word can be read as a base 26 number (a–z → 0–9a...).[2] The downside is that numbers in TEX have to be less than a billion or so, so we can only use the first six letters to determine sorting order: $26^6 \approx$ 300M. The other option, much more difficult, would be to count

[2]`tr('a-z','0-9a-z').to_i(26)` is valid Ruby code.

the lines of poetry, to count the number of footnotes passing through, doing some funny math with how far separated the baselines are, giving some leeway for lines that are too long, putting the footnotes in all in one shot, starting a new column, and before you know it, it’s an re-implementation of TEX and the footnote package, which is not the goal of the exercise.

Therefore, the line becomes

```
\setcounter{FN@totalid}{\value{vocab}}
```

which sets it to whatever value the counter vocab has, and we assume that we’ll make a command to set the counter vocab and call a footnote defintion at the same time. (All variables are globally accessible.)

(They are only 16 letters different, but the expressive power is increased tremendously. TEX is a high-level language for writing papers, and a language should promote much more thought than typing.)

Thus the footnote package. The TEX template is a bunch of page definitions and a few functions.

```
\documentclass[10pt]{article}
\pdfpageheight=11in\pdfpagewidth=8.5in
\usepackage{bigfoot}
\usepackage[margin=0.5in,top=0.35in,bottom=0.5in]{geometry}
\usepackage[english]{betababel}
\DeclareNewFootnote{df}\DeclareNewFootnote{cr}
\newcounter{vocab}\newcounter{notes}\setcounter{notes}{0}
\def\df#1#2{\setcounter{vocab}{#1}\Footnotedf{\kern-1.3ex}{\kern-4.2ex #2}}
\def\comm{\stepcounter{notes}\setcounter{vocab}{\value{notes}}\footnotecr}
\begin{document}\obeylines\twocolumn\parindent=0pt
\sf\pagestyle{empty}

\input sreout

\end{document}
```

The document will be an article, at 10 point body text (and 8 point footnote text). Set the pdf page size to 8.5x11; it defaults to A4, but letter-sized paper is much more common in the US. Use the bigfoot package. Use the page geometry package to shrink the margins to as little as my printer can manage. Use the betababel package to be able to turn beta code into polytonic greek, and set the default language as English. Declare a new footnote, df. Declare a new footnote, cr. Make a new counter, vocab. We changed bigfoot to rely on a vocab counter, and defined the vocab counter after we included the bigfoot package. No problem—TEX is a macro language. It replaces control sequences (delimited by backslashes) as needed, so the value of the counter for vocab is not evaluated until a footnote is used. Make another new counter, `notes`, and set it to zero. Define a control sequence `df` to take two parameters immediately following each other, and replace it by a command to set a counter, `vocab`, to the value of the first parameter, and then a lower-level command to make a footnote. This lower-level `Footnotedf` takes two parameters, a symbol to use as the number, and text to insert at the bottom of the page. A kern is a blank space, so the footnote command is to insert a negative space in the running text (to compensate for the footnote's automatic space), and to insert a negative space at the bottom of the page followed by the second parameter of our function. The next function, `comm`, is a function of no variables, and replace it with the following commands: add one to the counter `notes`, set the `vocab` counter to the value of `notes`, and start a high-level footnote for *c*ommentar*y*. This footnote takes one parameter, so the function `comm` is actually a function of one implicit parameter. (Leave as much as possible implicit; no sense in asking an operating system for a file handle, allocating a buffer, and so on.) Begin the document. Obey the line separations, and make new paragraphs each time a new line is seen.[3] Make the output two-column; a line of hexameter is usually under 4 inches, so the page will be packed more densely, and thus less page turning, and better flow. Because each line of text is a new paragraph, and the convention is to indent a paragraph, set the paragraph indent to 0 points, 0 seventy-seconds of an inch. Switch to *s*ans seri*f* font, and make the pages empty of line numbers. Input the SRE output file (which always stayed as `sreout.tex`), and end the document.

[3]The default is to make new paragraphs every time two new lines are seen—for old computers with 80x25 character screens (for which this was originally designed), and for papers with long paragraphs (for which this was originally designed), paragraphs of more than one screen line of text are the norm, and TEX is designed to be unintrusive by default, with easily changeable defaults.

```
load 'sym2proc.r'
load 'inetfile.r'
load 'texcode.r'

class String
  def flatscan(rgx) scan(rgx).flatten end
  def cleanup() tr('_','').gsub(/&.+?;/,' ').gsub(/\n+/,"\n") end
end

$ouija = 2
$morph_lang = "la"
$lem_def_rgx = %r%<G><foreign lang="la">(.*?)[#0-9]*?</foreign></G></font></td><td>(.*?)</td><td>%
$freq_rgx = />([0-9.]+)</
$text_word = /[[:alpha:]]+/
$slidewin = [nil] * 100
Gensym = memoize {|s| (0..19).map {rand 10}.join}

def defn_to_tex lemma,meaning,freq10k
  $slidewin.shift; $slidewin.push lemma+meaning
  TeX.df(word_to_num(lemma),TeX.textbf(lem_to_tex(lemma))+' '+meaning)
end
def lem_to_tex(word) word end
def word_to_url(word) word end
def word_to_tex(word) word end
def word_to_num(word) (word.downcase + 'a' * 6).tr('a-z','0-9a-z')[0,6].to_i(26) end

def morphhtms word
  HTM["http://www.perseus.tufts.edu/cgi-bin/morphindex?lang=#{$morph_lang}&embed=2&lookup="+word_to_url(word)].split('<p>')[1..-1].grep($lem_def_rgx)
end

def avgfreq(frequencies) [0.5 * (frequencies[2].to_f + frequencies[4].to_f)] end

Defs = memoize {|word|
         morphhtms(word).map {|s|
           s.flatscan($lem_def_rgx) + avgfreq(s.flatscan($freq_rgx))
         }.uniq.sort_by(&:slice(0))
       } # [lemma, definition, (max+min/2)freq/10k]

def anydefs(word)
  Defs[word].find_all {|lemma,meaning,freq10k| freq10k < $ouija && !$slidewin.include?(lemma+meaning)}.map(&method(:defn_to_tex)).join
end

def maketex(file)
  wolinenums = File.read(file).gsub(/@.+?@/) {|s| Gensym[s]}.gsub($text_word) {|w| word_to_tex(w)+anydefs(w)}.cleanup
  wolinenums.zip(1..wolinenums.count("\n").next).
             map{|line,num| (num%10==0 ? TeX.leavevmode+TeX.llap(TeX.scriptsize+num.to_s+'~~') : "")+line}.join.
             gsub(/\d{20}/) {|v| TeX.comm Gensym.index(v)[1..-2]}
end
```

```
load ’srelatin.r’
class String
 def gsubarr(a) a.inject(self) {|s, params| s.gsub(*params)} end
end

$morph_lang = "greek"
$lem_def_rgx = %r%">([^0-9<]+)[0-9]?</foreign></font></td><td>(.+?)</td><td>%
$text_word = /[a-z()*\/\\|+=]+’?/

def word_to_url(word) CGI.escape word.tr(’\\’,’/’) end
def word_to_num(word) (betacodify(word) + ’a’*6).downcase.tr(’^a-z’,’’).tr(’abgdezhqiklmncoprstufxyw’,’0-9a-z’)[0,6].to_i(24) end
def betacodify(word) word.gsubarr(%w%th ph ch ps h &ecirc; &ocirc; x 7%.zip(%w%q f 7 y 0 h w c x%)) end
def lem_to_tex(word) TeX.sffamily + word_to_tex(betacodify(word)) end
def word_to_tex(word) TeX.bcode word.gsub(’aa’,’a0a’) end

maketex(’odatnau’).writeout(’sreout.tex’)
```

The first page of code is `srelatin.r`, the Ruby code for SRE in Latin. The second page of code loads that, and redefines functions to produce Greek. It's helpful to read backwards. [4]

The last line of the Greek code looks like it should make TEX output, from the file `odatnau`, and write it out to `sreout.tex`, which is exactly what it does. Now we move to Latin code, which is where the bulk of the processing is, and we'll see later how complications arise in Greek. The last function, `maketex`, is defined as taking one parameter, which it calls `file`. `File.read()` reads a file, taking as a parameter a filename, and returns a character string of the entire contents of the file. (`File` is a object, it is a class, and `.read` calls its `read` method. This is object-oriented programming; objects can perform certain functions on themselves.) So now we have a string, with the file contents. Tell the string to globally substitute, every time it sees in itself anything that matches the Regular Expression of an ampersand `@` and any character `.` 1 or more times `+` (as few as possible though) `?` and an ampersand `@`, globally substitute for each instance the result of the expression in braces. The function in braces takes one parameter, `s`, through the

[4]One very nice property is that if functions don't modify any global variables, then they produce the same output given the same input. If you're in a REPL, always evaluating expressions, you don't need to worry about functions relying on what you've done before, and you can just figure out which combination of operators returns the right value. In addition to calling combinations of operators, the next level of abstraction is to create combinations of operators, which is what functions are. So REPLs almost always go with languages that make it easy to create functions. Here, much of this Ruby code, is just functions that join together, and the highest levels of abstraction, the last lines of code, are the easiest to start out with.

‘chute’ (as it’s lovingly called), and gsub passes to its associated function block the whole text that matched the Regular Expression (regexp). This function returns `Gensym` of `s`.

`Gensym` is a memoized/cached function; memoize means to store a value once computed. `memoize` takes a function, within braces, and returns a lookup table, and every time a value is looked up that does not exist, the function is run with that value, and the return value is stored in the lookup table. When `Gensym` looks up a value, its default value is a string of twenty random numbers from 0 to 9: the range `0..19`, twenty numbers, each mapped to the value of the associated function, which is a random number less than 10, then joined together into a string. So we replace every ampersand-footnote with twenty random numbers, but Gensym’s lookup table (hash table) records all incoming strings, so we can replace it back, after we’ve finished fiddling with the words. (`memoize` is in `sym2proc.r`[5]).

So `gsub` returns the string, with the substitutions made. Then we gsub that, matching every `$text_word` (call it `w`), to the TEX version of the word `w` (which is just itself) plus any definitions of `w`: `anydefs(w)`. Clean up this second substituted string, and call it `wolinenums`; it is the

[5]`sym2proc.r`:
```
class Symbol
 def to_proc(*args)lambda{|*a| a.first.send self,*(args+a[1..-1])}end
 alias [] to_proc
end
def memoize(&b) Hash.new {|h,nu| h[nu] = b[nu]} end
```

TEX file without line numbers.

At the top, we put some definitions of functions in the `String` class; `cleanup` translates all underscores (common in Perseus' HTML) to the empty string, globally substitutes any `&ecirc;`-like HTML entities to an empty space, and globally substitutes all instances of one or more newline character to a single newline. (TEX does not create a blank paragraph when it sees a newline, but my *ad hoc* line numberer will treat it as a line, so we must take out all multiple newlines.) Once we define a function in the String class, it's callable just like the built in gsub.

Now for `anydefs`. For the definitions of a word, find all (lemma, meaning, frequency per 10k) triples whose frequency per 10K is less than the global variable `$ouija`—currently set at 2 up top—and for whom it is NOT TRUE!, the answer to the question, does the Sliding Window include this lemma+meaning?[6] Map each of these definitions with the function method to convert a definition to TEX code, and join them into one string.

The definitions `Defs` is a memoized function, but we're not going to need reverse lookups. The definition of a `word` starts with the morphological analysis html fragments of this word. `morphhtms` fetches a webpage (via `HTM`, defined in `inetfile.r`[7]): the language is interpolated

[6]Note Ruby's effective use of punctuation.

[7]`inetfile.r`:

```
require 'digest/md5'
```

into the string with sharp-brace syntax (`#{$morph_lang}`), because it is the same process for a Latin or Greek morphological analysis page, and reusing the same code to do the same thing is good. Split the page by HTML paragraph tags (each entry in the Perseus table starts with one), toss out the first (at position 0, get everything from element 1 to -1, the end), and Globally search for Regular Expressions Preserving/printing them, matching the lemma definitions regexp. Ok, so, take these morphhtms, map each, calling it `s`, to the scan for the lemma and definition in s, plus the average frequency of the scan of the frequencies of s. The regular expressions at top parenthesize values to save in the scan, and flatten makes the list easier to work with. After mapping, return only the unique definitions, sorted by the first slice of the list, the lemma (given two identical numbers, `bigfoot` will put the first one it sees first).

So. After we make the definitions, and filter them, we pass them to `defn_to_tex`, which takes, a lemma, meaning, and frequency per 10k. `$slidewin` is a sliding window; shift an old entry out (we start out with

```
require 'open-uri'
load 'sym2proc.r'

class String
  def writeout(f) File.open(f,'w',&:write[self]);self end #ersatz monad
  def md5() ".ip." + Digest::MD5.hexdigest(self)[0,16] end
end

HTM = memoize {|addr|
File.exist?(addr.md5)?open(addr,&:read).writeout(addr.md5):File.read(addr.md5)
}
```

the null value, so it never matches anything, which means our filter lets everything in at first), push in a concatenation of this lemma and meaning. Here we generate TEX code, from `texcode.r`:

```
class TeXCode
  def method_missing(sym,*args)
    "\\"+sym.to_s+args.map {|a| "{#{a}}"}.join
  end
end
TeX = TeXCode.new
```

So remember the object paradigm, you call a method on an object. What happens if an object doesn't have that method? Ruby defines `method_missing` to receive the symbol of the undefined function, and all of the arguments passed in. `TeXCode` is designed to make backslash sequences, with curly-brace-surrounded arguments. So instead of enumerating every possible sequence that SRE needs, just let Ruby's method handling do it, so every function call with arguments will map into a TEX function call. Recalling the code from a while ago, `df` takes two parameters, one the footnote number (which we decided would be the numerical equivalent of the word), and the other the contents of the footnote.

So to make a word into a number, make it lowercase, add 6 letter 'a's (to pad it if it's too short), translate a to z into 0 to 9 and a to (as high as z corresponds), start at position 0 and take 6 characters, and convert it to an integer base 26. (This changes, of course, if our letter order is different.) The text of the footnote is the boldfaced lemma, converted to TEX syntax somehow, plus a space plus the meaning.

All functions return the same outputs with the same inputs, except for `defn_to_tex`. In `maketex`, every time we find a `$text_word`, we call `anydefs` on it, which in turn calls `defn_to_tex`, so we are updating the state of the sliding window every time we insert a definition. If the 100-entry sliding window does not already have the definition, it will be processed, otherwise, filtered out.

So then, after getting the TEX code without line numbers, we insert the line numbers. A string is an enumerable value; Ruby counts a string by the number of newline characters it has, and `zip` merges enumerable values into a list of tuples of these values, much like a zipper, joining a pair of teeth. So we have a range, from 1 to the next higher number after the count of newlines in `wolinenums`. (In a list (a,b,c), there are 3 elements, 2 commas.) Map each pair of zipped values, calling them the line and the number, to this: is the number modulo 10 equal to 0? If so, this expression is the TEX code for starting horizontal mode (after making the last paragraph) plus the code for a left overlap (spilling out legally into the left margin) of a temporary change to a smaller font plus the line number converted to a string plus two explicit blank spaces. If the number mod 10 is not, the expression returns an empty string. Take whatever we got and append the line to it, and replace the zipped pair by that. Join it all into one string. Now, after all is done, we put back in the ampersand-delimited comments. Globally substitute, for all twenty-number runs calling each run `v`, the TEX control sequence for numbered commentary and braces around the index of this run `v` which equals the string we initially passed in because the index is the string and the value at the index is the run `v`, but take the second character to the next-to-last character of this index string, stripping off the ampersands.

So how is Greek different? First, three $-prefixed global variables change. The language of Perseus' morphology needs to be greek. The regexp for finding lemmata and definitions in an HTML page is different, because of peculiarities in Perseus' system. And, fundamentally, the notion of a word is different. A betacode word is any one of the characters a to z or open or close paren (breathings) or asterisk (capitalization) or forwards or backwards slashes (accents) which because the backslash is the escape character (backslash-n was a newline in `maketex`) and forward slash delimits this regular expression both must be preceded by a backslash or bar (iota subscript) or plus (dieresis) or equals (circumflex), one or more times `+`, and a quote, maybe `?` . Without the quote, Perseus' morphological analysis does not treat it as part of a word. A URL requires all of these special characters to be escaped, and this is what `CGI.escape` does, to a word that has had its backslashes replaced by forwards slashes, because Perseus only permits the latter.

Not only do global variables change, but four functions change as well. (We can just redefine a function, and its definition is updated.) `word_to_tex` returns the word, in a betacode interpretation wrapper. The betacode is slightly buggy, and two consecutive alphas produce *JJ*, so we just shoehorn a *0* in there, which will get ignored. `lem_to_tex` takes a lemma and returns TEX code to start the sans serif family in a footnote plus the TEX conversion of the word after it's betacodified.

`word_to_num` also uses `betacodify`. The lemmas are in a greek transliteration, which betacodify turns into proper betacode. Add six alphas to pad it like in Latin, put it all lower case, strip off anything besides the letters themselves (the backslash accents were problematic at one point), and translate from beta code order to numbers and letters (which `to_i` understands), starting at position 0 take the first 6 characters, and convert it to an integer base 24.

`betacodify` is a very dense function. It tells word to perform an array-based gsub with a zipped array. The array-based gsub uses inject, a fundamental array operator, otherwise known as fold. It takes an initial accumulator value, and a function in braces of two parameters. For every element of the list, call the function with the accumulator value and the element, and replace the accumulator value with the value of the function. For example, 6 factorial is `[2,3,4,5,6].inject(1) {|f,n| f*n}`: 1 times 2, times 3, times 4, times 5, times 6; fold the multiplication function through the list, like chocolate chips in cookie dough. `gsub` can take two parameters, one of an original string, and one of a replacement string, and returns the string after substitution. So `gsubarr` folds this substitution through a array of pairs, accumulating a string that has had these pairwise substitutions. Perseus' *th* becomes a betacode *q*, *ph* an f, *ch* a *7* (because ξ is *c* and η is *h* and we need to save it to something different), *ps* to *y*, *h* to 0 (which will be ignored, and we'll just have a word without breathing), the HTML code for ê to *h*, the HTML code for ô to *w*, *x* to c, and the temporary *7* to x.

And voilà, we are done.

Fun Summer Reading

Autenrieth, Georg, tr R Keep, rev I Flagg. *A Homeric Dictionary.*

Bederson, Benjamin. *Interfaces for Staying in the Flow.*
`hcil.cs.umd.edu/trs/2003-37/2003-37.html`

Butterman, Lee. *Synthesis of Reading Editions for Latin.*
(The aspects that do not pertain to Greek borrow from this.)

Flow.
`wikipedia.org/wiki/Flow_(psychology)`

Garrison, Daniel H. *The Student's Catullus.*

Graham, Paul. *Great Hackers.*
`paulgraham.com/gh.html`

Mental State called Flow.
`http://c2.com/cgi/wiki?MentalStateCalledFlow`

Mihalyi Csikszentmihalyi.
`wikipedia.org/wiki/Mihaly_Csikszentmihalyi`

Perlis, Alan. *Epigrams on Programming.*
`www-pu.informatik.uni-tuebingen.de/users/klaeren/epigrams.html`

Pharr, Clyde. *Vergil's Aeneid.*

Spolsky, Joel. *Human Task Switches Considered Harmful.*
`http://joelonsoftware.com/articles/fog0000000022.html`[sic]

Virgil, R Thomas ed. *Eclogues.* Cambridge University Press, 1988.