9.19: Computational Psycholinguistics, Pset 2
due 4 October 2021

20 September 2021

The Colab notebook for this problem can be found at https://colab.research.google.com/drive/1nxUoxQywa88dAoxe-qSDu9b48Sw03t38?usp=sharing.

1 Log base conversion

Prove that $\frac{\log_a x}{\log_a b} = \log_b x$.

2 Using regular expressions to syllabify English words

Write a Python function that uses regular expressions to syllabify English words. You're not expected to come up with something that performs perfectly, as that is not an easy problem, but you should come up with something that performs well on a wide range of English words, and doesn't have systematic gaps. Although syllabification is really best described on the phonemic representation of words, as available e.g. in English via the CMU Pronouncing Dictionary, for this exercise we will stick with the orthographic representation of English.

Here are some examples of syllabifications of English words:

Word	Syllabification
i	i
air	air
big	big
strength	strength
steal	steal
ideal	i deal
quiet	qui et
enter	en ter
able	a ble
pandas	pan das
intake	in take
capable	ca pa ble
serendipity	se ren di pi ty

Your function should also work for "nonce" English orthographic words—letter sequences that don't happen to be words, but that could be. For example:

```
sneed   sneed
snoded  sno ded
ilskig  il skig
```

In the Colab notebook for this pset, provide your implementation in the section for this problem. Try your implementation on new English words of your choosing, and describe what kinds of words you find hard to handle.

After this pset is over, we will try out everyone's syllabifiers on a set of challenge words, and report whose does the best! (This is just for fun; performance on this challenge set will not affect your grade.)

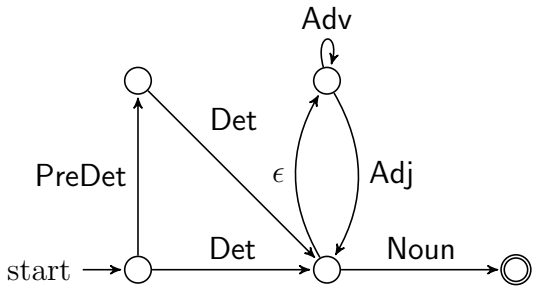
3 Regular expressions and finite-state automata

For drawing finite-state automata, there is a nice web-based tool at http://madebyevan.com/fsm/ that you might want to use for this problem.

Task: write a finite-state automaton that accepts and rejects the same strings as the following regular expression, and list the words that it accepts:

```
((in)(disputabl|conceivabl)|(un)(believabl|trustabl))(e|y)
```

Task: Write a regular expression that accepts and rejects the same strings as the following finite-state automata, and list some sentences of English exemplifying what the automata/regex accepts. (Make sure that each edge in the automaton is traversed by at least one of your example sentences.)



Part of speech	**Example words**
PreDeterminer	*all*, *only*
Determiner	*the*, *a*
Adverb	*very*, *especially*
Adjective	*big*, *green*
Noun	*woman*, *table*

4 A finite-state grammar for a fragment of English syntax

For drawing finite-state automata, there is a nice web-based tool at http://madebyevan.com/fsm/ that you might want to use for this problem.

In English, it is possible to add adverbs and prepositional phrases to the beginning of a sentence:

> Kim saw a movie.
> Recently Kim saw a movie.
> Recently at the theater Kim saw a movie.
> Recently at the theater with her friend Kim saw a movie.
> ⋮

It is also possible to recursively embed a sentence inside a COMPLEMENT CLAUSE using a verb like *say*, *think*, *claim*, and so forth:

> Kim saw a movie.
> Pat said that Kim saw a movie.
> Terry claimed that Pat said that Kim saw a movie.
> ⋮

Adding adverbs and prepositional phrases can be mixed together with embedding inside a complement clause:

> Kim saw a movie.
> Pat said that at the theater Kim saw a movie.
> Terry claimed that recently Pat said that at the theater Kim saw a movie.
> ⋮

Task: write a finite-state automaton that accepts sentences of English involving simple transitive and intransitive sentences, recursive complement clauses, and adverbs and prepositional phrases at the beginning of clauses. Use $\Sigma = \{\mathsf{Name}, \mathsf{Noun}, \mathsf{Det}, \mathsf{Verb}, \mathsf{Prep}, \mathsf{Adv}, \mathsf{that}\}$ as your input alphabet.

Hint: the automaton below accepts simple transitive and intransitive sentences, and you can get to a solution by adding states and edges to it.