9.19: Computational Psycholinguistics, Pset 3
due 18 October 2023

5 October 2023

The Colab notebook for this problem can be found at https://colab.research.google.com/drive/11bbLYa3Nu1wkZ6PKu-59R2j2oyAVjoMX

1 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.

This problem will also give you a brief introduction to the International Phonetic Alphabet (IPA). For convenience, the `eng_to_ipa` Python package does IPA lookup for a large set of English words, using the Carnegie Mellon University Pronouncing Dictionary. You don't need to have a comprehensive knowledge of IPA to do this problem, but these things will help:

A. Every syllable has at most one vowel:

   - Some vowels are MONOPHTHONGS—involving a simple, or "pure", sound—ɑ (as in *palm*), æ (as in *hand*), ə (as in *hut*), ɪ (as in *hit*), i (as in *high*), ɔ (as in *thought*), ʊ (as in *look*), and u (as in *hoot*).
   - Other vowels are DIPHTHONGS—involving a transition between two simple vowel sounds—including aɪ (as in *hide*), aʊ (as in *mouth*), eɪ (as in *face*), oʊ (as in *row*), and ɔɪ (as in *choice*).

B. A syllable can also have zero or more consonants *before* the vowel, and zero or more consonants *after* the vowel.

C. One syllable per word receives PRIMARY STRESS. This syllable is conventionally marked with a ˈ mark before it, unless the word is monosyllabic in which case no stress mark is needed. For example, *open* is written as ˈoʊ pən, and *anaconda* is written as æ nə ˈkɑn də. **In this exercise, however, you don't need to worry about placing stress.**

D. There are some distinctions made that we won't worry about here. This includes writing the "schwa" (ə) as a "wedge" (ʌ) when it is stressed. Also, some consonants including r, l, m, n and ŋ can behave as SYLLABIC, meaning they play the role of vowels. We will instead write them in these cases as being preceded by a schwa—e.g., *letter* is lɛtər.

Here are some examples of syllabifications of English words:

Word	IPA representation	Syllabification
i	aɪ	aɪ
air	ɛr	ɛr
big	bɪg	bɪg
strength	strɛŋθ	strɛŋθ
steal	stil	stil
ideal	aɪdil	aɪ dil
quiet	kwaɪət	kwaɪ ət
enter	ɛntər	ɛn tər
able	eɪbəl	eɪ bəl
pandas	pændəz	pæn dəz
intake	ɪnteɪk	ɪn teɪk
capable	keɪpəbəl keɪ pə bəl	
serendipity	sɛrɛndɪpɪti	sɛ rɛn dɪ pɪ ti

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:

Word	IPA representation	Syllabification
sneed	snid	snid
snoded	snoʊdɛd	snoʊ dɛd
ilskig	ɪlskɪg	ɪl skɪg

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.)

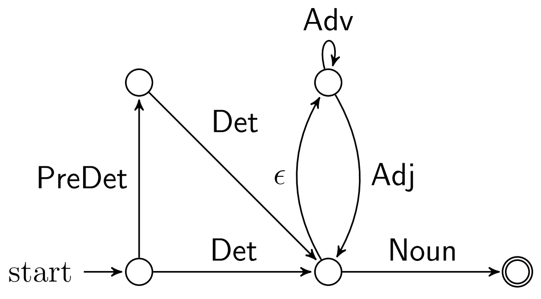
2 Regular expressions and finite-state automata

We'll use the automata-lib and visual-automata libraries for this problem—see the Colab notebook.

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*

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

We'll use the automata-lib and visual-automata libraries for this problem—see the Colab notebook.

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:

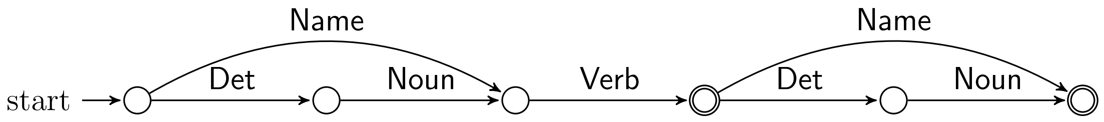
Kim saw a movie.
Pat said that Kim saw a movie.
Terry claimed that Pat said that Kim saw a movie.
⋮

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.
$\vdots$

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.



4 Writing context-free grammars

This is an exercise in writing context-free grammars (CFGs) to capture generalizations about natural language syntax. You can use an automatic parser available in `NLTK` to check whether your grammar accounts for the key generalizations. To get you going, consider the following grammar that we covered in the CFG lecture notes:

NP $\rightarrow$ Det N	N $\rightarrow$ woman
NP $\rightarrow$ NP PP	N $\rightarrow$ umbrella
PP $\rightarrow$ P NP	N $\rightarrow$ street
Det $\rightarrow$ a	P $\rightarrow$ about
Det $\rightarrow$ an	P $\rightarrow$ with
N $\rightarrow$ joke	P $\rightarrow$ on

Running the following code will parse the string *a joke about the woman with an umbrella on the street* with start symbol (i.e., goal category) NP, generating the five parses that we saw in the CFG lecture notes. Note that `NLTK`'s `CFG.fromstring()` function takes the left-hand-side of the first rule listed as the goal category, and allows multiple rewrites for a single category to be expressed with a disjunction on the right-hand side of a single rule, so that the rule `X -> Y1 ... Ym | Z1 ... Zn` is shorthand for the two rules $X \rightarrow Y_1 \ldots Y_m$ and $X \rightarrow Z_1 \ldots Z_n$.

```
import nltk
from nltk import Nonterminal, nonterminals, Production, CFG

grammar = CFG.fromstring("""
NP -> Det N | NP PP
PP -> P NP
Det -> 'the' | 'a' | 'an'
N -> 'joke' |'woman' | 'umbrella' | 'street'
P -> 'about' | 'with' | 'on'
""")

parser = nltk.parse.BottomUpChartParser(grammar)
sentence = ['a', 'joke', 'about', 'the', 'woman',
    'with', 'the', 'umbrella', 'on', 'the', 'street']
for tree in parser.parse(sentence):
    tree.pretty_print()
```

A. Write a context-free grammar that will capture the structural ambiguity in the sentence *They are flying planes.* Your grammar should respect the facts that (i) an NP should be substitutable with a pronoun given the right context; and (ii) a verb and an immediately following NP can combine to form a VP. You can check your work with the NLTK parser to make sure that your grammar behaves the way you think it will behave.

B. Extend your grammar to capture the structural ambiguity in the sentence *Flying planes can be dangerous.* (**Hint:** a non-finite VP can serve as the subject of an English sentence, such as in the sentences *To err is human* or *Defeated by the Miami Heat is not how I expected the Milwaukee Bucks to finish in the NBA playoffs*, and it is OK to use a unary rewrite rule with a right-hand-side element that is a phrasal category. See SLP3 Section 12.3.1 for an example of this, though it is a different unary rewrite than you would use for this problem.)

C. The ambiguity of the preceding sentence is eliminated if you change *can be* to either *is* or *are*. Why? Modify your grammar so that it captures this disambiguation effect.

5 Argument structure and unbounded dependencies in context-free grammars

Let's start with the following grammar in NLTK style (i.e., `X -> Y1 ... Ym | Z1 ... Zn` means that there are two separate rules, `X -> Y1 ... Ym` and `X -> Z1 ... Zn`):

S	→ NP VP
NP	→ Det N | Pronoun
VP	→ V
VP	→ V NP
VP	→ V SBAR
SBAR	→ WHNP S
SBAR	→ COMP S
COMP	→ 'that' |
WHNP	→ 'who' | 'what'
Det	→ 'the' | 'a' | 'an' | 'my' | 'your' | 'her' | 'his' | 'their'
N	→ 'joke' | 'women' | 'umbrella' | 'street' | 'apple'
Pronoun	→ 'I' | 'you' | 'she' | 'he' | 'they'
V	→ 'slept' | 'devoured' | 'know' | 'said' | 'know'

(**Note:** the rule `COMP -> 'that' |` expands out to `COMP -> 'that'` and `COMP ->` , the latter of which is `NLTK`'s way of expressing an empty rewrite (i.e. it's equivalent to `COMP` $\rightarrow \epsilon$).

This grammar will give correct parses for sentences like:

(1) I devoured the apple

(2) I said you slept

(3) you know what I devoured

However, it will incorrectly accept sentences like:

(4) *I slept that you said

(5) *the women devoured

(6) *I know what you said the joke

and incorrectly reject sentences like:

(7) you know who slept

(8) the women know who I said devoured the apple

A. Revise the grammar so that it correctly accounts for the different **argument structures** of the different verbs. Your grammar should now correctly reject (5) and (6) but incorrectly reject (3).

B. **Want a challenge?** Examples like (8) involve UNBOUNDED DEPENDENCY constructions as covered in class. Implement meta-rules, using S, NP, VP, and SBAR as your BASIC CATEGORIES, and the non-terminal rewrites in the grammar as your BASIC RULES, to add a set of new DERIVED CATEGORIES and DERIVED RULES to your grammar. Your grammar should now correctly accept and reject all the above examples. (This challenge is not worth any points, but we will give you feedback on any solution you offer.)