Homework 3: Pynini

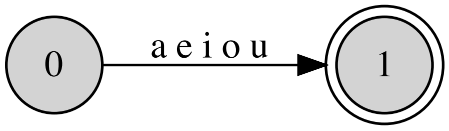
1 Pynini

Pynini is a python-based version of Thrax (which uses regular expressions and context-dependent rewrite rules to form weighted finite-state transducers) developed by Kyle Gorman. You can read Pynini's documentation here and see additional Pynini examples here.

1.1 Union

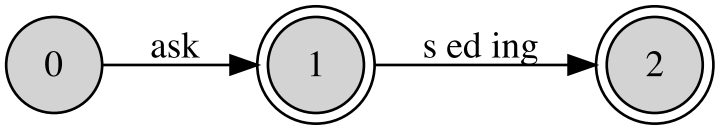
The regular expression `[aeiou]` represents the **union** of the characters 'a', 'e', 'i', 'o', and 'u' (i.e. orthographic vowels).

Pynini's union's function is `pynini.union()`, also represented by `|`. The regular expression above, `[aeiou]`, would be represented in Pynini as `pynini.union("a", "e", "i", "o", "u")`, and this is shown in the finite-state automaton below.



1.2 Concatenation

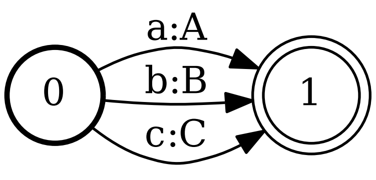
The regular expression `ask(s|ed|ing)?` represents the **concatenation** of "ask" with its inflectional suffixes and this is shown in the finite-state automaton below.



Pynini's concatenation function is `pynini.concat()`, also represented by `+`. The regular expression above, `ask(s|ed|ing)?`, would be represented in Pynini as `pynini.concat("ask", pynini.union("s", "ed", "ing").ques)`. This notation, `ques`, indicates the optionality of the inflectional suffixes, and will be discussed in subsection 1.5.

1.3 Transducers

finite-state **transducers** are finite-state automata with input and output. For example, below is an example of a finite-state transducer that transduces lowercase letters "a", "b", and "c" to uppercase "A", "B", and "C".



1.4 Composition

Composition produces the composition of two finite-state transducers. For example, if A transduces the string “cat” to “Katze” and B transduces “Katze” to “chat”, then their composition transduces string “cat” to “chat”.

Pynini’s composition function is `pynini.compose()`, also represented by `*`

The power of finite-state transducers is their ability to be composed.

Additionally, when we define finite-state transducers, we have to define Σ^*, the set of all strings over a given alphabet. As such, we can define an alphabet, and use Pynini’s `closure()` function, which is equivalent to `*` in regular expression notation, in order to specific that each element of the alphabet can occur “zero or more times”. The `closure()` function is discussed in more detail in subsection 1.5.

```
import pynini

# use all valid bytes, and escape reserved characters
chars = [chr(i) for i in range(1, 91)] + [r"\[", r"\\", r"\]"] + [chr(i) for i in range(94, 256)]
sigma_star = pynini.union(*chars).closure()
```

1.5 Closure

Repetition operators are represented in Pynini with the `pynini.closure()` function. See the table below for regular expressions and Pynini function equivalents repeated here from the Pynini documentation.

Regex:	Pynini method:	Shortcuts:
/x?/	x.closure(0, 1)	x.ques
/x*/	x.closure()	x.star
/x+/	x.closure(1)	x.plus
/x{N}/	x.closure(N, N)	
/x{M,N}/	x.closure(M, N)	
/x{N,}/	x.closure(N)	
/x{,N}/	x.closure(0, N)	

1.6 Context-dependent rewrite rules

Pynini can compile context-dependent rewrite rules into finite-state transducers using the `cdrewrite()` function. This function requires four arguments: 1. a transducer, 2, the left context, 3. the right context, and 4. Σ^*. The documentation (try `help(pynini.cdrewrite)` refers to the left context as “lamba” and the right context as “rho”; the rewrite itself is a transducer defining a regular relation between upper language Ψ and lower language φ. For example, if you wanted to specify that the plural morpheme /Z/ becomes voiceless [S] after a voiceless sound, you would specify this in Pynini like this:

```
pynini.cdrewrite(pynini.transducer("Z", "S"), pynini.union("P", "T", "K", "F", "TH"), "", sigma_star)
```

```
<vector Fst at 0x7faa3c3d0308>
```

1.7 An example: Phonological Finite-State Transducer

The following is an example that that takes phonetic input and returns phones based on finite-state transducers made using Pynini. In order to retrieve pronunciation, we can use the Carnegie Mellon University Pronouncing Dictionary (hereafter, CMUDict)[1]. Previously, we have been searching for orthographic (written) symbols

[1]See Chapter 2 in the NLTK book for more information on retrieving pronunciations

in a text, but there is not always a one-to-one correspondence between orthography and sound[2]. Because there is not a precise one-to-one mapping of sounds to symbols, linguists use the International Phonetic Alphabet (hereafter, IPA) which consists of one symbol for every sound. However, some of these symbols are not easily read by a computer, for example, [θ], the first sound in "thigh". For that reason, ARPAbet was created to be a machine-readable version of the IPA. The APRAbet phones used by CMUdict are featured here on their website.

The CMUdict gives the pronunciation of the word *writer* as 'R AY1 T ER0', but do you *really* pronounce the T in that word? If you speak American English, you pronounce *writer* with a flap in the middle: 'R AY1 DX ER0'. Using finite-state transducers, we can get from the phonemes of 'writer': 'R AY1 T ER0', to the phonological realization of 'writer': 'R AY1 DX ER0'.

Phonological rules are very similar to context-dependent rewrite rules; they both take the shape of A $\rightarrow$ B / C_D, making finite-state phonology a natural application.

This transducer applies the phonological rule **palatalization**: Alveolar[3] consonants become palatalized (i.e. more like a palatal) before a palatal sound.

```
# Here, we'll define some transducers, and put them all together using union().
# These transducers take the alveolar consonants S, T, D, and Z and change them
# to their palatalized counterparts, SH, CH, JH, and ZH, respectively.

palatalization_map = pynini.union(
        pynini.transducer("S", "SH"),
        pynini.transducer("T", "CH"),
        pynini.transducer("D", "JH"),
        pynini.transducer("Z", "ZH")
)

phonologize = pynini.cdrewrite(palatalization_map, "", "Y", sigma_star)

# Define a function called 'make_phonology' that takes the input 'string'
# and returns the composition of the string (with the function strip()
# to remove any leading/trailing white spaces) and the transducers we defined above.
# The stringify() function returns the result as a string (instead of a transducer).

def make_phonology(string):
        return(pynini.compose(string.strip(), phonologize).stringify())

# Call the function and assign the result to the variable 'phones'.
phones = make_phonology("MIHSYUW")
phones
```

```
'MIHSHYUW'
```

2 Text normalization and Pynini

In addition to finite-state phonology, finite-state transducers have applications in text normalization as well. In Homework 2, we saw that we could normalize the archaic past tense affix "-'d" could be normalized to the modern affix "-ed". We could do the same thing in Pynini.

[2]For example, *read* (present tense) is pronounced differently than *read* (past tense) even though they are spelled the same. There is a famous example illustrating the difference in orthography and pronunciation: one could spell spell fish as "ghoti" using the"gh" from *tough*, the "o" from *women*, and the "ti" from *celebration*.

[3]If these terms are unfamilar to you, please review the following definitions: **Palatal**: a sound pronounced by the tongue and hard palate, like English [j], the first sound in *yes*, **Alveolar**: a sound pronounced using the tongue and alveolar ridge (the area behind the front teeth that feels like corrugated cardboard), like English [t, d, s, z, n].

```
past_tense_map = pynini.transducer("'d", "ed")

normalize = pynini.cdrewrite(past_tense_map, "", "", sigma_star)

def make_normal(string):
        return(pynini.compose(string.strip(), normalize).stringify())

make_normal("As I ponder'd")
```

```
'As I pondered'
```

```
make_normal("the shore and dark-color'd sea-rocks")
```

```
'the shore and dark-colored sea-rocks'
```

We can also try some additional text normalization in Pynini. You may want to normalize times, for example:

- 2:30 → "two thirty"
- 9:00 → "nine o'clock"
- 4:02 → "four oh three"

```
# transduce from :00 to "o'clock"
oclock_map = pynini.transducer(":00", " o'clock")

# replace the colon with a space
colon_del = pynini.transducer(":", " ")

# transducer in order to get the "oh" in 2:03
oh_map = pynini.transducer("0", "oh ")

# delete a zero at the end of a time, as in 2:30
zero_del = pynini.transducer("0", "")

ones_map = pynini.union(
        pynini.transducer("1", "one"),
        pynini.transducer("2", "two"),
        pynini.transducer("3", "three"),
        pynini.transducer("4", "four"),
        pynini.transducer("5", "five"),
        pynini.transducer("6", "six"),
        pynini.transducer("7", "seven"),
        pynini.transducer("8", "eight"),
        pynini.transducer("9", "nine"),
 )

teens_map = pynini.union(
        pynini.transducer("10", "ten"),
        pynini.transducer("11", "eleven"),
        pynini.transducer("12", "twelve"),
        pynini.transducer("13", "thirteen"),
        pynini.transducer("14", "fourteen"),
        pynini.transducer("15", "fifteen"),
        pynini.transducer("16", "sixteen"),
        pynini.transducer("17", "seventeen"),
        pynini.transducer("18", "eighteen"),
        pynini.transducer("19", "nineteen")
)
```

```
tens_map = pynini.union(
         pynini.transducer("2", "twenty "),
         pynini.transducer("3", "thirty "),
         pynini.transducer("4", "forty "),
         pynini.transducer("5", "fifty ")
)

number = pynini.union("1", "2", "3", "4", "5", "6", "7", "8", "9", "0")

time_normalize = (
        pynini.cdrewrite(oclock_map, number, "", sigma_star) *
        pynini.cdrewrite(oh_map, ":", number, sigma_star) *
        pynini.cdrewrite(colon_del, "", "", sigma_star) *
        pynini.cdrewrite(tens_map, "", number, sigma_star) *
        pynini.cdrewrite(teens_map, "", "", sigma_star) *
        pynini.cdrewrite(ones_map, "", "", sigma_star) *
        pynini.cdrewrite(zero_del, "", "", sigma_star)
)

def normalize(string):
        return(pynini.compose(string.strip(), time_normalize).stringify())

normalize("2:03")
```

```
'two oh three'
```

```
normalize("4:00")
```

```
"four o'clock"
```

```
normalize("9:30")
```

```
'nine thirty '
```

Q1 Your task in this homework is to normalize numbers (up to 9999) and quantities using Pynini. Example input and output are listed below. In your write up, provide: your python code, your input, your output, and at least a paragraph reflecting on what you did in order to define transducers, cdrewrite rules, etc.

- 3 c. → three cups
- 40 mi. → forty miles
- 1 lb. → one pound
- 1000 lbs. → one thousand pounds

Here are some standard abbreviations for quantities:

bbl.	barrel	cm	centimeter
cu.	cubic	GB	gigabyte
doz.	dozen	G	gigabyte
fl. oz.	fluid ounce	g	gram
ft.	foot	ha	hectare
gal.	gallon	KB	kilobyte
in.	inch	kg	kilogram
kt.	knot	kl	kiloliter
lb.	pound	km	kilometer
mi.	mile	l	liter
mph	miles per hour	m	meter
oz.	ounce	MB	megabyte
qt.	quart	mg	milligram
rpm	revolutions per minute	ml	milliliter
tbsp.	tablespoon	mm	millimeter
tsp.	teaspoon	W	watt
yd.	yard	kW	kilowatt
B	byte	kWh	kilowatt-hour
cc	cubic centimeter		