Ps4a

Ej Zhou (yz2876) and Jiawei Sun (js2869)

In [1]:
```
import nltk
from nltk import grammar, parse
from nltk.parse.generate import generate
from IPython.display import display
```

In [2]:
```
import copy
```

In [3]:
```
from typing import Callable, List, Set

def to_model_str(word: str, special_rels: List[Callable[[str], str]]=[]) ->
    """
    Creates the string form of the model for the input word. This string is
    By default, the function will only add the relations mapping i => i for
    mapping char => the set of tuples (i, word[i]). The `special_rels` funct
    be added to the valuation string.

    :param word: The word to create a model string for.
    :param special_rels: A list of functions that when called return a strin
    :returns: a string representing the model for word
    """
    n = len(word)
    model_str = []
    char = []
    for i in range(1, n+1):
        model_str.append(f'{i} => {i}')
        char.append((i, word[i-1]))
    model_str.append(f'char => {set(char)}'.lower())
    return '\n'.join(model_str + [rel(word) for rel in special_rels]).replac
# Angela Liu

import re
get_vowel = lambda w: f'vowel => {set(re.findall(r"[AEIOUaeiou]", w))}'.lowe
get_cons = lambda w: 'cons => {}'.format(set(re.findall(r"[^AEIOUaeiou\W0-9]
follows = lambda w: f'le => {set([(i+1,j+1) for i in range(len(w)) for j in
get_capital = lambda w: f'capital => {set([m.span()[0] + 1 for m in re.findi
# Angela Liu

def emptysets(val:nltk.sem.evaluate.Valuation):
  val.update([(k,set()) for (k,v) in val.items() if v == 'set()'])
```

In [4]:
```
#for w, m in zip(words, models):
    #print(f'{w}\n----------------\n{m}\n')
```

1. Number feature. The following sentences illustrate number agreement between subjects, verb phrases, and within DPs.

No vowels are capitalized.

- No vowels is capitalized.

No vowel is capitalized.

- No vowel are capitalized.

Exactly two vowels are capitalized. *Exactly two vowels is capitalized.

*Exactly two vowel Exactly two vowels

Exactly one vowel *Exactly one vowels

Create a feature grammar including semantics that has singular and plural determiners, singular and plural DPs, and singular and plural VPs, and enforces agreement correctly.

In [5]:
```
parser_str = '''
% start S
#Grammar Rules
S[NUM='sg',SEM=<?X(?P)>] -> DP[NUM='sg', SEM=?X] VP[NUM='sg', SEM=?P]
S[NUM='pl',SEM=<?X(?P)>] -> DP[NUM='pl', SEM=?X] VP[NUM='pl', SEM=?P]

DP[NUM='sg',SEM=<?X(?P)>] -> Det[NUM='sg',SEM=?X] N[NUM='sg', SEM=?P]
DP[NUM='pl',SEM=<?X(?P)>] -> Det[NUM='pl',SEM=?X] N[NUM='pl', SEM=?P]

VP[NUM='sg',SEM=?Q] -> 'is' JJ[SEM=?Q]
VP[NUM='pl',SEM=?Q] -> 'are' JJ[SEM=?Q]
Det[SEM=<?X(?P)>] -> Det[SEM=?X] Det[SEM=?P]
#Lexical Rules
Det[SEM=<\P Q.all n.-(P(n) & Q(n))>] -> 'no'

Det[NUM='sg', SEM=<\P Q.exists n.((P(n) & Q(n)) & all m.((n != m) -> -(P(m)&

Det[NUM='pl',SEM=<\P Q.exists n.(((P(n)&Q(n))& exists m.((n!=m)&P(m)&Q(m)& a

JJ[SEM=<\c.(capital(c))>] -> 'capitalized'
N[NUM='sg',SEM=<\m.exists c.(vowel(c) & char(m,c))>] -> 'vowel'
N[NUM='pl',SEM=<\m.exists c.(vowel(c) & char(m,c))>] -> 'vowels'
'''
```

In [6]:
```
grammar = nltk.grammar.FeatureGrammar.fromstring(parser_str)
```

In [7]:
```
p = nltk.parse.FeatureChartParser(grammar)
```

In [8]:
```
sens2 = []
sens2.append('no vowels is capitalized'.split())
sens2.append('no vowel are capitalized'.split())
sens2.append('exactly two vowel are capitalized'.split())
sens2.append('exactly one vowels are capitalized'.split())
sens2.append('exactly two vowels is capitalized'.split())
```

In [9]:
```
for sen in sens2:
    try:
        t = next(p.parse(sen))
    except:
        print('"',' '.join(sen),'"', ' is not a valid sentence')
```

```
" no vowels is capitalized "  is not a valid sentence
" no vowel are capitalized "  is not a valid sentence
" exactly two vowel are capitalized "  is not a valid sentence
" exactly one vowels are capitalized "  is not a valid sentence
" exactly two vowels is capitalized "  is not a valid sentence
```

In [10]:
```
wordslist2 = [("jack",True), # no vowel capitalized
          ("klAus",False), # one vowel capitalized
          ("cOAmp",False), # two vowel capitalized
          ("cOAEmp",False)] # three vowel capitalized
```

In [11]:
```
sens3 = []
sens3.append('no vowel is capitalized'.split())
sens3.append('exactly one vowel is capitalized'.split())
sens3.append('exactly two vowels are capitalized'.split())
```

In [12]:
```
for sen in sens3:
    #Setting up Sentence Semantics
    t = next(p.parse(sen))
    fn = t.label()['SEM']

    #Setting up Truth Values
    words = [e[0] for e in wordslist2]
    truths = [e[1] for e in wordslist2]
    vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, get_capita
    assignments = [nltk.Assignment(val.domain) for val in vals]
    for val in vals: emptysets(val)
    models = [nltk.Model(val.domain, val) for val in vals]

    #Print Results
    print(f'{sen}\n---------------')
    for w, a, m in zip(words, assignments, models):
        print(f'{w}\n{m.evaluate(str(fn),a)}\n-----------------')
    print('\n')
```

```
['no', 'vowel', 'is', 'capitalized']
---------------
jack
True
----------------
klAus
False
----------------
cOAmp
False
----------------
cOAEmp
False
----------------


['exactly', 'one', 'vowel', 'is', 'capitalized']
---------------
jack
False
----------------
klAus
True
----------------
cOAmp
False
----------------
cOAEmp
False
----------------


['exactly', 'two', 'vowels', 'are', 'capitalized']
---------------
jack
False
----------------
klAus
False
----------------
cOAmp
True
----------------
cOAEmp
False
----------------
```

1. There insertion

There are at least two vowels. There are exactly two vowels. There are no vowels vowels. There are some vowels vowels.

- There is every vowel.
- There are most vowels.

There is exactly one consonant. There is no consonant.

Notice these points.

(i) The set of determiners that can occur in there-insertion is restricted. These are called weak determiners. Use a feature to distinguish weak determ2iners from strong ones.

(ii) The verb is/are agrees in number with the phrase that follows.

Write a grammar (based on the one from Problem 1) that enforces the restrictions.

In [13]:
```
ps2 = '''
% start S
#Grammar Rules
S[NUM='sg',STR='weak',SEM=<?X>] -> PP[NUM='sg',STR='weak'] DP[NUM='sg',STR='
S[NUM='pl',STR='weak',SEM=<?X>] -> PP[NUM='pl',STR='weak'] DP[NUM='pl',STR='

DP[NUM='sg',STR='weak',SEM=<?X(?P)>] -> Det[NUM='sg',STR='weak',SEM=?X] N[NU
DP[NUM='pl',STR='weak',SEM=<?X(?P)>] -> Det[NUM='pl',STR='weak',SEM=?X] N[NU

DP[NUM='sg',STR='strong',SEM=<?X(?P)>] -> Det[NUM='sg', STR='strong',SEM=?X]
DP[NUM='pl',STR='strong',SEM=<?X(?P)>] -> Det[NUM='pl', STR='strong',SEM=?X]

Det[SEM=<?X(?P)>] -> Det[SEM=?X] Det[SEM=?P]

PP[NUM='sg', STR='weak'] -> P[STR='weak'] 'is'
PP[NUM='pl', STR='weak'] -> P[STR='weak'] 'are'

#Lexical Rules
Det[STR='weak', SEM=<\P.all n.-(P(n))>] -> 'no'

Det[NUM='sg', STR='strong', SEM=<\P.all n.(P(n))>] -> 'every'

Det[NUM='pl', STR='weak', SEM=<\P.exists n.(P(n))>] -> 'some'
Det[NUM='pl', STR='strong', SEM=<\P.exists n.(P(n))>] -> 'most'


Det[NUM='sg', STR='weak', SEM=<\P.exists n.((P(n)) & all m.((n != m) -> -(P(

Det[NUM='pl',STR='weak',SEM=<\P.exists n.((P(n)& exists m.((n!=m)&P(m)& all

Det[NUM='pl',STR='weak',SEM=<\P.exists n.(P(n)& exists m.((n!=m)&P(m)))>] ->

P[STR='weak'] -> 'there'

N[NUM='sg',SEM=<\m.exists c.(vowel(c) & char(m,c))>] -> 'vowel'
N[NUM='pl',SEM=<\m.exists c.(vowel(c) & char(m,c))>] -> 'vowels'

N[NUM='sg',SEM=<\m.exists c.(cons(c) & char(m,c))>] -> 'consonant'
N[NUM='pl',SEM=<\m.exists c.(cons(c) & char(m,c))>] -> 'consonants'

'''
```

In [14]:
```
g2 = nltk.grammar.FeatureGrammar.fromstring(ps2)
```

In [15]:
```
p2 = nltk.parse.FeatureChartParser(g2)
```

In [16]:
```
sens2 = []
sens2.append('there is every vowel'.split())
sens2.append('there are most vowels'.split())
sens2.append('there is no vowels'.split())
sens2.append('there is at least two vowels'.split())
sens2.append('there are exactly one consonant'.split())
```

In [17]:
```
for sen in sens2:
    try:
        t = next(p2.parse(sen))
    except:
        print('"',' '.join(sen),'"', ' is not a valid sentence')
```

```
" there is every vowel "  is not a valid sentence
" there are most vowels "  is not a valid sentence
" there is no vowels "  is not a valid sentence
" there is at least two vowels "  is not a valid sentence
" there are exactly one consonant "  is not a valid sentence
```

In [18]:
```
wordslist = [("jck",True), # no vowel, three consonants
             ("jac",False), # one vowel, two consonants
             ("cOA",False), # two vowel, one consonant
             ("OAE",False)] # three vowels, no consonants
```

In [19]:
```
sens = []
sens.append('there are at least two vowels'.split())
sens.append('there are exactly two vowels'.split())
sens.append('there are no vowels'.split())
sens.append('there are some vowels'.split())
sens.append('there are at least two consonants'.split())
sens.append('there is exactly one consonant'.split())
sens.append('there is no consonant'.split())
sens.append('there are some consonants'.split())
```

In [20]:
```
for sen in sens:
    #Setting up Sentence Semantics
    t = next(p2.parse(sen))
    fn = t.label()['SEM']

    #Setting up Truth Values
    words = [e[0] for e in wordslist]
    truths = [e[1] for e in wordslist]
    vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, get_cons])
    assignments = [nltk.Assignment(val.domain) for val in vals]
    for val in vals: emptysets(val)
    models = [nltk.Model(val.domain, val) for val in vals]

    #Print Results
    print(f'{sen}\n---------------')
    for w, a, m in zip(words, assignments, models):
        print(f'{w}\n{m.evaluate(str(fn),a)}\n----------------')
    print('\n')
```

```
['there', 'are', 'at', 'least', 'two', 'vowels']
---------------
jck
False
----------------
jac
False
----------------
cOA
True
----------------
OAE
True
----------------


['there', 'are', 'exactly', 'two', 'vowels']
---------------
jck
False
----------------
jac
False
----------------
cOA
True
----------------
OAE
False
----------------


['there', 'are', 'no', 'vowels']
---------------
```

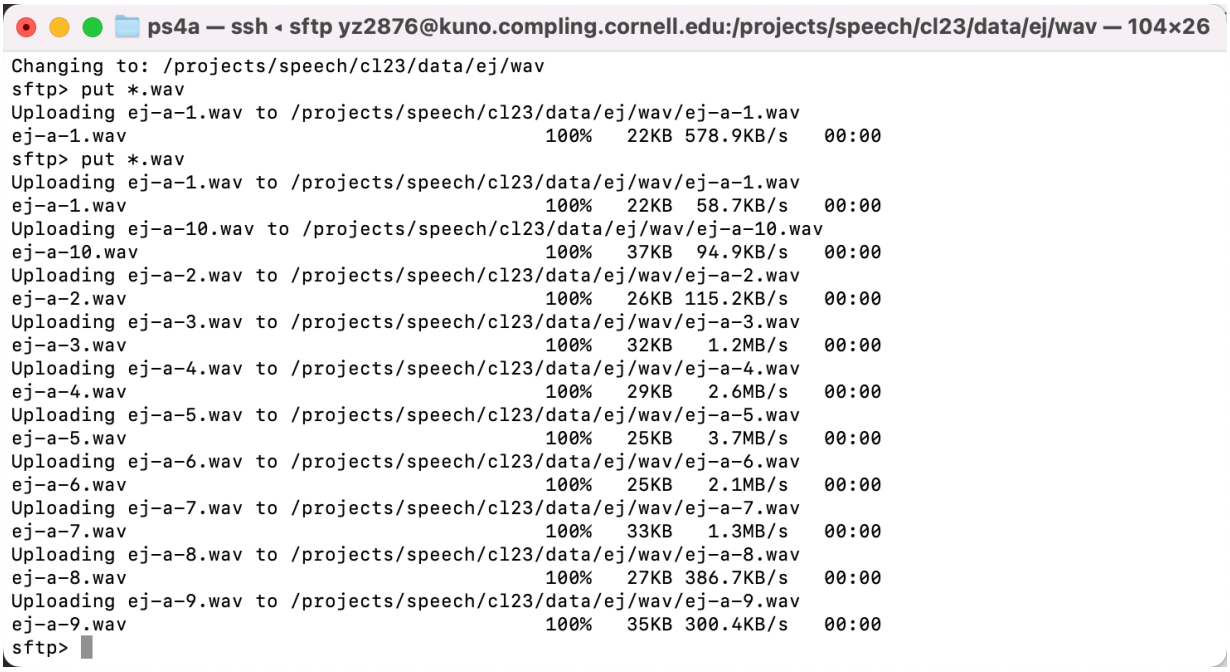
```
jck
True
----------------
jac
False
----------------
cOA
False
----------------
OAE
False
----------------


['there', 'are', 'some', 'vowels']
---------------
jck
False
----------------
jac
True
----------------
cOA
True
----------------
OAE
True
----------------


['there', 'are', 'at', 'least', 'two', 'consonants']
---------------
jck
True
----------------
jac
True
----------------
cOA
False
----------------
OAE
False
----------------


['there', 'is', 'exactly', 'one', 'consonant']
---------------
jck
False
----------------
jac
False
```

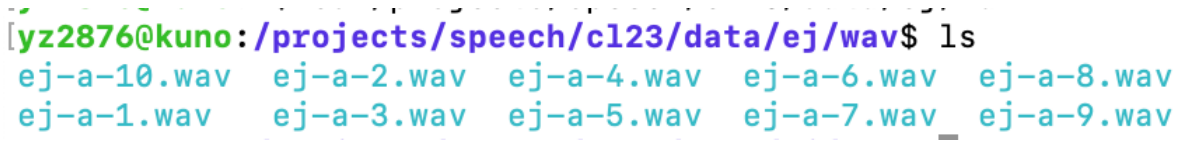
```
----------------
cOA
True
----------------
OAE
False
----------------


['there', 'is', 'no', 'consonant']
---------------
jck
False
----------------
jac
False
----------------
cOA
False
----------------
OAE
True
----------------


['there', 'are', 'some', 'consonants']
---------------
jck
True
----------------
jac
True
----------------
cOA
True
----------------
OAE
False
----------------
```

1. Make a selection of ten sentences about strings. One method is to modify sentences from challenge problems for ps2.

Pick a nickname for yourself, say 'ayako'. Record the sentences as wav files ayako-a-1.wav ... ayako-a-10.wav.

Check on the forum for specifications regarding sample rate and the like. Install the sentences on kuno in a location that will be specied on the forum.





In [ ]:
```
sftp> put *.wav
```

1. Kaldi setup Set up jupyter and bash kernel on kuno. In a jupyter bash notebook, do these things.

a. Run the demo run.sh in egs/yesno in the notebook, using your own copy of egs/yesno.

b. Using kaldi tools, print a matrix with the first ten rows and first five columns of one of the utterances. Methodology will be covered in class. The relevant programs are in featbin.

```
yz2876@kuno:/projects/speech/cl23/data/ej/wav$ cd /projects/speech/ASR/kaldi/egs/yesno
yz2876@kuno:/projects/speech/ASR/kaldi/egs/yesno$ ls
README.txt  s5-amw369  s5-fz227  s5-jg2369  s5-jz642   s5-nac86   s5-sc2359
s5          s5-apf65   s5-gcc68  s5-jk2298  s5-lad279  s5-rk658   s5-ya79
s5-aaa385   s5-clean   s5-ib262  s5-jrs673  s5-mb2342  s5-sc23    s5-yc2544
s5-abc123   s5-cyz22   s5-jf675  s5-js2869  s5-mr249   s5-sc2343
yz2876@kuno:/projects/speech/ASR/kaldi/egs/yesno$ cp -R s5-clean s5-yz2876
yz2876@kuno:/projects/speech/ASR/kaldi/egs/yesno$ ls
README.txt  s5-amw369  s5-fz227  s5-jg2369  s5-jz642   s5-nac86   s5-sc2359
s5          s5-apf65   s5-gcc68  s5-jk2298  s5-lad279  s5-rk658   s5-ya79
s5-aaa385   s5-clean   s5-ib262  s5-jrs673  s5-mb2342  s5-sc23    s5-yc2544
s5-abc123   s5-cyz22   s5-jf675  s5-js2869  s5-mr249   s5-sc2343  s5-yz2876
```

In [ ]:
```
cd /projects/speech/ASR/kaldi/egs/yesno
cp -R s5-clean s5-yz2876
cd s5-yz2876/
source hardpath.sh
source run.sh
```

```
yz2876@kuno:/projects/speech/ASR/kaldi/egs/yesno/s5-yz2876$ source hardpath.sh
yz2876@kuno:/projects/speech/ASR/kaldi/egs/yesno/s5-yz2876$ source run.sh
--2023-04-19 16:43:21--  http://www.openslr.org/resources/1/waves_yesno.tar.gz
Resolving www.openslr.org (www.openslr.org)... 46.101.158.64
Connecting to www.openslr.org (www.openslr.org)|46.101.158.64|:80... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://us.openslr.org/resources/1/waves_yesno.tar.gz [following]
--2023-04-19 16:43:21--  https://us.openslr.org/resources/1/waves_yesno.tar.gz
Resolving us.openslr.org (us.openslr.org)... 46.101.158.64
Connecting to us.openslr.org (us.openslr.org)|46.101.158.64|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4703754 (4.5M) [application/x-gzip]
Saving to: 'waves_yesno.tar.gz'

waves_yesno.tar.gz  100%[===================>]   4.49M  5.31MB/s    in 0.8s

2023-04-19 16:43:22 (5.31 MB/s) - 'waves_yesno.tar.gz' saved [4703754/4703754]

waves_yesno/
waves_yesno/1_0_0_0_0_0_1_1.wav
waves_yesno/1_1_0_0_1_0_1_0.wav
waves_yesno/1_0_1_1_1_1_0_1.wav
waves_yesno/1_1_1_1_0_1_0_0.wav
waves_yesno/0_0_1_1_1_0_0_0.wav
waves_yesno/0_1_1_1_1_1_1_1.wav
waves_yesno/0_1_0_1_1_1_0_0.wav
waves_yesno/1_0_1_1_1_0_1_0.wav
waves_yesno/1_0_0_1_0_1_1_1.wav
waves_yesno/0_0_1_0_1_0_0_0.wav
waves_yesno/0_1_0_1_1_0_1_0.wav
waves_yesno/0_0_1_1_0_1_1_0.wav
waves_yesno/1_0_0_0_1_0_0_1.wav
waves_yesno/1_1_0_1_1_1_1_0.wav
waves_yesno/0_0_1_1_1_1_0_0.wav
waves_yesno/1_1_0_0_1_1_1_0.wav
waves_yesno/0_0_1_1_0_1_1_1.wav
waves_yesno/1_1_0_1_0_1_1_0.wav
waves_yesno/0_1_0_0_0_1_1_0.wav
waves_yesno/0_0_0_1_0_0_0_1.wav
waves_yesno/0_0_1_0_1_0_1_1.wav
waves_yesno/0_0_1_0_0_0_1_0.wav
```

relevant program is in : /projects/speech/ASR/kaldi/src/featbin/copy-feats.cc

In [ ]:
```
which copy-feats
/projects/speech/ASR/kaldi/src/featbin/copy-feats
```

every function inside

```
yz2876@kuno:/projects/speech/ASR/kaldi/src/featbin$ ls *.cc
add-deltas.cc                           compute-cmvn-stats-two-channel.cc  extend-transform-dim.cc      get-full-lda-mat.cc          shift-feats.cc
add-deltas-sdc.cc                       compute-fbank-feats.cc             extract-feature-segments.cc  interpolate-pitch.cc         splice-feats.cc
append-post-to-feats.cc                 compute-kaldi-pitch-feats.cc       extract-segments.cc          modify-cmvn-stats.cc         subsample-feats.cc
append-vector-to-feats.cc               compute-mfcc-feats.cc              feat-to-dim.cc               multiply-vectors.cc          subset-feats.cc
apply-cmvn.cc                           compute-plp-feats.cc               feat-to-len.cc               paste-feats.cc               transform-feats.cc
apply-cmvn-sliding.cc                   compute-spectrogram-feats.cc       fmpe-acc-stats.cc            paste-vectors.cc             wav-copy.cc
compare-feats.cc                        concat-feats.cc                    fmpe-apply-transform.cc      post-to-feats.cc             wav-reverberate.cc
compose-transforms.cc                   copy-feats.cc                      fmpe-est.cc                  process-kaldi-pitch-feats.cc wav-to-duration.cc
compute-and-process-kaldi-pitch-feats.cc copy-feats-to-htk.cc              fmpe-init.cc                 process-pitch-feats.cc
compute-cmvn-stats.cc                   copy-feats-to-sphinx.cc            fmpe-sum-accs.cc             select-feats.cc
```

There might be several ways to do this, for example using the copy-feats, subset-feats or select-feats. But the easiest way might be simply using the head and cut method in shell script

In [ ]:
```
copy-feats scp:data/train_yesno/feats.scp  ark,t:- | head -n 10 | cut -d " "
```

0_0_0_0_1_1_1_1 [48.9744 -14.08839 -0.1344485 4.717914 21.6918 53.68612 -10.14594 -1.394663 -2.119221 13.08845 55.30577 -10.31021 2.78328 6.130801 18.00465 56.4837 -16.38815 -2.418089 8.250131 5.304466 59.04967 -6.238433 -3.889265 -4.782257 1.989483 61.00519 -5.754362 -2.929802 -1.887652 9.401299 61.16815 -6.39979 -4.081158 -1.308731 0.9340096 61.98295 -7.206575 -1.714484 2.512146 2.200577 60.51631 -6.722504 -2.482053 -1.656084 4.4851

```
copy-feats scp:data/train_yesno/feats.scp ark,t:-
0_0_0_0_1_1_1_1  [
  48.9744 -14.08839 -0.1344485 4.717914 21.6918
  53.68612 -10.14594 -1.394663 -2.119221 13.08845
  55.30577 -10.31021 2.78328 6.130801 18.00465
  56.4837 -16.38815 -2.418089 8.250131 5.304466
  59.04967 -6.238433 -3.889265 -4.782257 1.989483
  61.00519 -5.754362 -2.929802 -1.887652 9.401299
  61.16815 -6.39979 -4.081158 -1.308731 0.9340096
  61.98295 -7.206575 -1.714484 2.512146 2.200577
  60.51631 -6.722504 -2.482053 -1.656084 4.4851
```