Code Preparations

In [ ]:
```
import nltk
from nltk import grammar, parse
from nltk.parse.generate import generate
from platform import python_version
python_version()
```

Out[ ]:
```
'3.7.16'
```

In [ ]:
```
print(nltk.__version__)
```

```
3.5
```

In [ ]:
```
nltk.data.show_cfg('h2.fcfg')
```

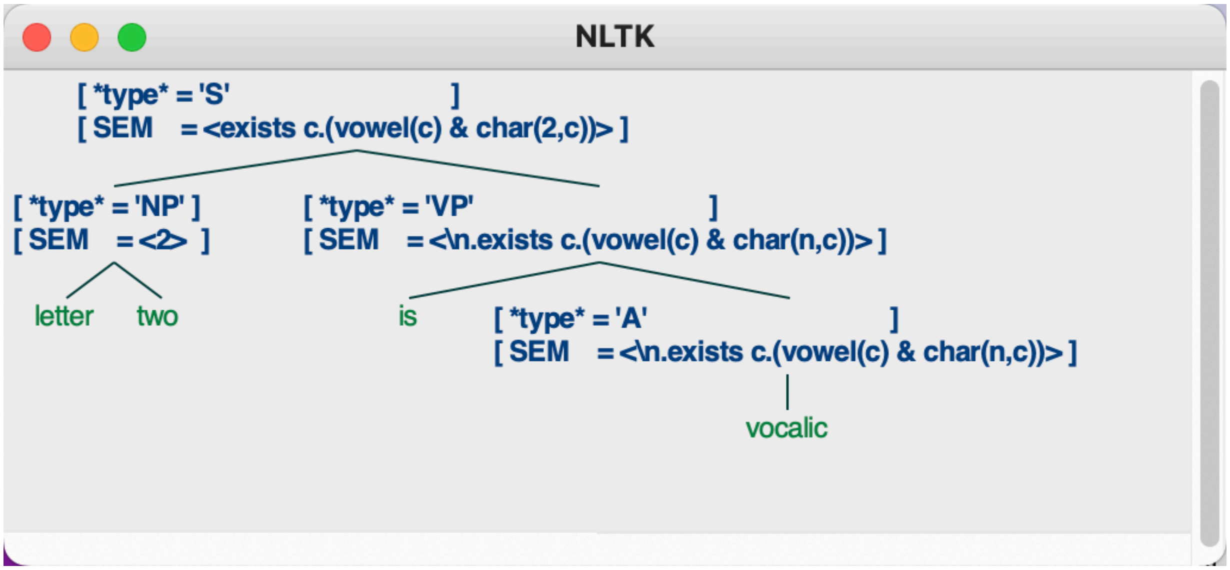
```
% start S
# Grammar Rules
S[SEM=<?subj(?vp)>] -> DP[SEM=?subj] VP[SEM=?vp]
S[SEM=<?vp(?subj)>] -> NP[SEM=?subj] VP[SEM=?vp]
S[SEM=<?vp1(?subj) & ?vp2(?subj)>] -> NP[SEM=?subj] VP[SEM=?vp1] 'and' VP[SE
M=?vp2]
DP[ SEM=<?X(?P)>] -> Det[SEM=?X] N[SEM=?P]
VP[SEM=?Q] -> 'is' A[SEM=?Q]
VP[SEM=?P] -> 'is' 'a' N[SEM=?P]
# This is included for testing.
VP[SEM=<\x.offend(x)>] -> 'offends'
# Transitive verb with individual object.
VP[ SEM=<?R(?n)>] -> TV[SEM=?R] NP[SEM=?n]
# Transitive verb with quantifier object.
# The object is given minimal scope.
VP[ SEM=<\m.?X(\n.(?R(n)(m)))>] -> TV[SEM=?R] DP[SEM=?X]
# Lexical Rules
A[SEM=<\n.exists c.(vowel(c) & char(n,c))>] -> 'vocalic'
A[SEM=<\n.exists c.((-vowel(c)) & char(n,c))>] -> 'consonantal'
A[SEM=<\n.exists c.(capital(n) & char(n,c))>] -> 'capitalized'
A[SEM=<\n. (n = 1) >] -> 'initial'
Det[SEM=<\P Q.all n.(P(n) -> Q(n))>] -> 'every'
Det[SEM=<\P Q.exists n.(P(n) & Q(n))>] -> 'a'
Det[SEM=<\P Q.exists n.(P(n) & Q(n))>] -> 'some'
Det[SEM=<\P Q.all n.(P(n) -> -Q(n))>] -> 'no'
N[SEM=<\n.char(n,leti)>] -> 'i'
N[SEM=<\n.char(n,lete)>] -> 'e'
N[SEM=<\n.char(n,letu)>] -> 'u'
N[SEM=<\n.char(n,letp)>] -> 'p'
N[SEM=<\n.char(n,lett)>] -> 't'
N[SEM=<\n.char(n,letk)>] -> 'k'
N[SEM=<\n.exists c.(glide(c) & char(n,c))>] -> 'glide'
N[SEM=<\n.exists c.char(n,c)>] -> 'letter'
N[SEM=<\n.exists c.(vowel(c) & char(n,c))>] -> 'vowel'
N[SEM=<\n.exists c.(-vowel(c) & char(n,c))>] -> 'consonant'
NP[SEM=<1>] -> 'letter' 'one'
NP[SEM=<2>] -> 'letter' 'two'
NP[SEM=<3>] -> 'letter' 'three'
NP[SEM=<4>] -> 'letter' 'four'
NP[SEM=<5>] -> 'letter' 'five'
TV[SEM=<\m n.le(m,n)>] -> 'follows'
TV[SEM=<\m n.le(n,m)>] -> 'precedes'
```

In [ ]:

```
cp = nltk.load_parser('h2.fcfg', trace=0)
sent = 'letter two is vocalic'.split()
for tree in cp.parse(sent): print(tree)
```

```
(S[SEM=<exists c.(vowel(c) & char(2,c))>]
  (NP[SEM=<2>] letter two)
  (VP[SEM=<\n.exists c.(vowel(c) & char(n,c))>]
    is
    (A[SEM=<\n.exists c.(vowel(c) & char(n,c))>] vocalic)))
```

In [ ]:
```
tree.draw()
```



to use Angela Liu's implementation from mapping the word

In [ ]:
```
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

get_glide = lambda w: f'glide => {set(re.findall(r"[YWyw]", w))}'.lower()
# added

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

words = ['cat', 'mAtch', 'peRiLOuSy']
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, get_cons, foll
for v in vals: emptysets(v)
models = [nltk.Model(val.domain, val) for val in vals]
for w, m in zip(words, models):
    print(f'{w}\n----------------\n{m}\n')
# Angela Liu
```

```
cat
----------------
Domain = {'3', 'a', 't', '2', 'c', '1'},
Valuation =
{'1': '1',
 '2': '2',
 '3': '3',
 'capital': set(),
```

```
 'char': {('2', 'a'), ('3', 't'), ('1', 'c')},
 'cons': {('t',), ('c',)},
 'glide': set(),
 'le': {('1', '2'), ('2', '3'), ('1', '3')},
 'vowel': {('a',)}}

mAtch
----------------
Domain = {'h', '3', 'a', '5', 'm', 't', '2', '4', 'c', '1'},
Valuation =
{'1': '1',
 '2': '2',
 '3': '3',
 '4': '4',
 '5': '5',
 'capital': {('2',)},
 'char': {('4', 'c'), ('5', 'h'), ('3', 't'), ('1', 'm'), ('2', 'a')},
 'cons': {('m',), ('h',), ('t',), ('c',)},
 'glide': set(),
 'le': {('1', '2'),
        ('1', '3'),
        ('1', '4'),
        ('1', '5'),
        ('2', '3'),
        ('2', '4'),
        ('2', '5'),
        ('3', '4'),
        ('3', '5'),
        ('4', '5')},
 'vowel': {('a',)}}

peRiLOuSy
----------------
Domain = {'6', 'u', '3', '8', '5', 'r', 'o', 'y', 's', '2', '4', '9', '7', '
1', 'e', 'l', 'i', 'p'},
Valuation =
{'1': '1',
 '2': '2',
 '3': '3',
 '4': '4',
 '5': '5',
 '6': '6',
 '7': '7',
 '8': '8',
 '9': '9',
 'capital': {('8',), ('6',), ('5',), ('3',)},
 'char': {('1', 'p'),
          ('2', 'e'),
          ('3', 'r'),
          ('4', 'i'),
          ('5', 'l'),
          ('6', 'o'),
          ('7', 'u'),
```

```
          ('8', 's'),
          ('9', 'y')},
 'cons': {('r',), ('p',), ('l',), ('y',), ('s',)},
 'glide': {('y',)},
 'le': {('1', '2'),
        ('1', '3'),
        ('1', '4'),
        ('1', '5'),
        ('1', '6'),
        ('1', '7'),
        ('1', '8'),
        ('1', '9'),
        ('2', '3'),
        ('2', '4'),
        ('2', '5'),
        ('2', '6'),
        ('2', '7'),
        ('2', '8'),
        ('2', '9'),
        ('3', '4'),
        ('3', '5'),
        ('3', '6'),
        ('3', '7'),
        ('3', '8'),
        ('3', '9'),
        ('4', '5'),
        ('4', '6'),
        ('4', '7'),
        ('4', '8'),
        ('4', '9'),
        ('5', '6'),
        ('5', '7'),
        ('5', '8'),
        ('5', '9'),
        ('6', '7'),
        ('6', '8'),
        ('6', '9'),
        ('7', '8'),
        ('7', '9'),
        ('8', '9')},
 'vowel': {('u',), ('o',), ('i',), ('e',)}}
```

Start of the work:

```
Semantics of sentences about strings
Computational Linguistics Spring 2023
Problems Set 2
```

The text for this module is the NLTK book Chapter 9. Building Feature Based Grammars and Chapter 10. Analyzing the Meaning of Sentences

See also lecture8_2023.ipynb and string_2023.ipynb

The purpose of the assingnment is to develop feature-based grammars that include logical semantics, and to evaluate the adequacy of the semantics by computing truth in logically constructed models. For instance, we want to be able to evaluate whether the sentence

every consonant is capitalized

is true or false as description of the word

CINEMA

or

Cinema

or

CINEmA.

In each problem n do these steps. The problem statement gives sentence sn. See Chapters 9 and 10 for the methodology.

(i) Define a feature based grammar gn that includes all the words in sentence sn its lexicon. The feature grammars will usually add a word and/or construction to a base grammar which will be similar to simple-sem.fcfg. This base grammar will be distributed. (It will be helpful to figure out how to add a lexical item or production to a grammar in Python. Discuss the method for this on the forum. Or you can define the grammar from scratch.)

(ii) Parse the sentenced and display the tree.

(iii) Map sn to a logical formula fn by parsing with gn and extracting the semantics that annotates the root.

(iv) Define a combination four words (serving as models) and the intuitive truth values of sentence sn as a description of the word.

(v) Transform the four words into four models or valuations in the sense of Chapter 10. This can be done as in Lecture 8, or by using a function. Code for this may be shared and discussed on the forum.

(vi) Evaluate formula fn in the four models to obtain four truth values. Compare them to the target truth values.

Work individually, except that code for mapping a word to a valuation may be shared. Post techinical questions and requests for hints on the forum.

Notes The problems are selected so that quantifiers are used only in subject position. So it is not necessary (except perhaps in challenge problems) to apply the strategy from simple-sem.fcfg to fit quantified NPs into object positions.

The words 'precedes' and 'follows' are interpreted in the sense of 'not necessarily immediately'.

Problems

1. letter two is consonantal
   Base your analysis on 'letter two is vocalic', which is covered by the base grammar.

I added a grammar rule

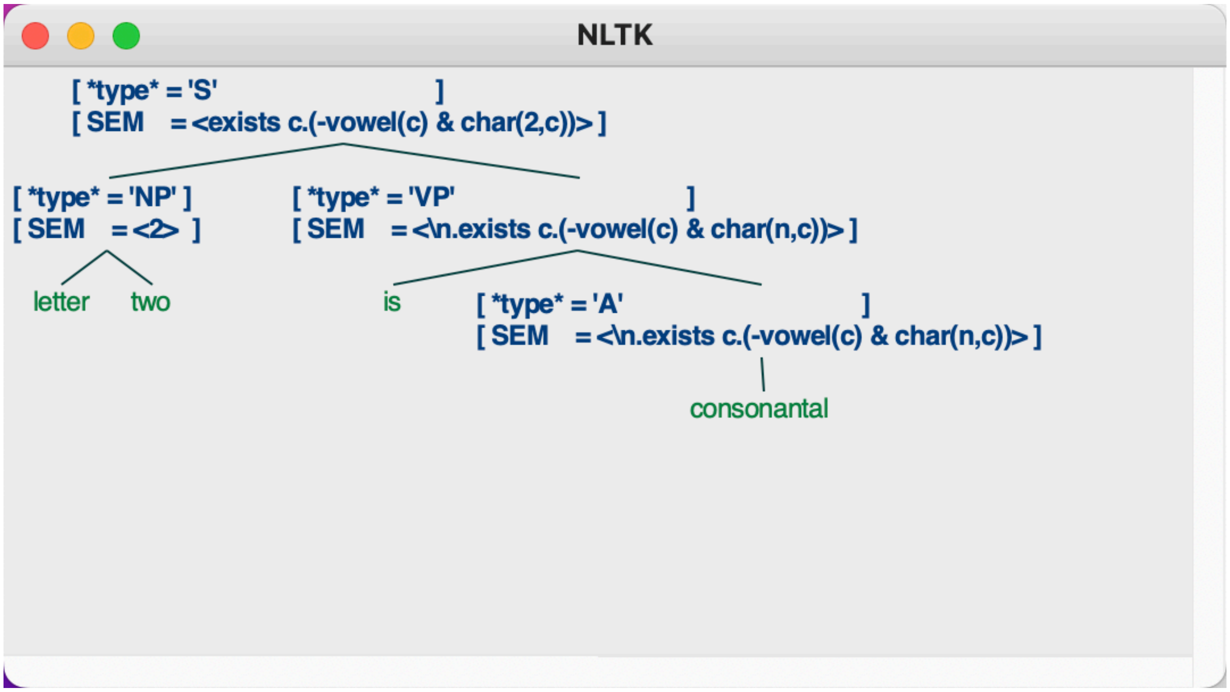
```
A[SEM=<\n.exists c.(consonant(c) & char(n,c))>] ->
'consonantal'
```

parse and display

In [ ]:
```
g1 = nltk.load_parser('h2.fcfg', trace=0)
s1 = 'letter two is consonantal'
s1_split = s1.split()
for tree in g1.parse(s1_split): print(tree)
```

```
(S[SEM=<exists c.(-vowel(c) & char(2,c))>]
  (NP[SEM=<2>] letter two)
  (VP[SEM=<\n.exists c.(-vowel(c) & char(n,c))>]
    is
    (A[SEM=<\n.exists c.(-vowel(c) & char(n,c))>] consonantal)))
```

In [ ]:
```
tree.draw()
```



now the logical formula

In [ ]:
```
t1=next(g1.parse(s1_split))
f1 = t1.label()['SEM']
print(f1)
```

```
exists c.(-vowel(c) & char(2,c))
```

define a sample four word

In [ ]:
```
e1 = [('emu',True),('bat',False),('cat',False),('aka',True)]
```

In [ ]:
```
words = [e[0] for e in e1]
truths = [e[1] for e in e1]
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, follows])) for
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]
```

In [ ]:
```
print(f'{s1}\n---------------')
for w, a, m in zip(words, assignments, models):
    print(f'{w}\n{m.evaluate(str(f1),a)}\n----------------')
```

```
letter two is consonantal
---------------
emu
True
----------------
bat
False
----------------
cat
False
----------------
aka
True
----------------
```

```
+ The answer is correct! emu & aka has the second letter being
consonants
```

1. every vowel precedes letter three Add a production for 'precedes' to the grammar. Don't add it to the valuations. Instead, define the semantics of 'precedes' in terms of the primitive used for 'follows'.

Add grammar

```
TV[SEM=<\m n.le(n,m)>] -> 'precedes'
N[SEM=<\n.exists c.(vowel(c) & char(n,c))>] -> 'vowel'
```

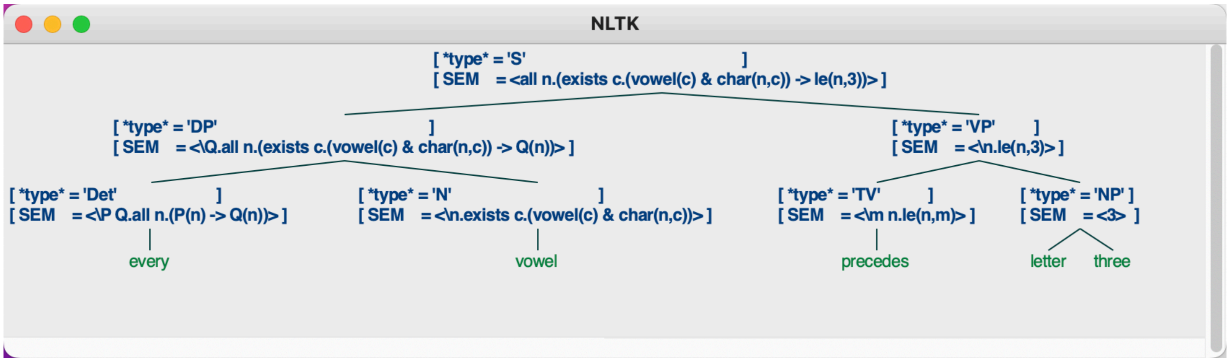
In [ ]:

```
g2 = nltk.load_parser('h2.fcfg', trace=0,cache=False)
s2 = 'every vowel precedes letter three'
s2_split = s2.split()
for tree in g2.parse(s2_split): print(tree)
```

```
(S[SEM=<all n.(exists c.(vowel(c) & char(n,c)) -> le(n,3))>]
  (DP[SEM=<\Q.all n.(exists c.(vowel(c) & char(n,c)) -> Q(n))>]
    (Det[SEM=<\P Q.all n.(P(n) -> Q(n))>] every)
    (N[SEM=<\n.exists c.(vowel(c) & char(n,c))>] vowel))
  (VP[SEM=<\n.le(n,3)>]
    (TV[SEM=<\m n.le(n,m)>] precedes)
    (NP[SEM=<3>] letter three)))
```

In [ ]:

```
tree.draw()
```



In [ ]:
```python
t2=next(g2.parse(s2_split))
f2 = t2.label()['SEM']
print(f2)
```

```
all n.(exists c.(vowel(c) & char(n,c)) -> le(n,3))
```

In [ ]:
```python
e2 = [('emmmu',False),('aekjd',True),('maqwr',True),('aeiou',False)]
words = [e[0] for e in e2]
truths = [e[1] for e in e2]
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, follows])) for
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]
```

In [ ]:
```python
print(f'{s2}\n---------------')
for w, a, m in zip(words, assignments, models):
    print(f'{w}\n{m.evaluate(str(f2),a)}\n----------------')
```

```
every vowel precedes letter three
---------------
emmmu
False
----------------
aekjd
True
----------------
maqwr
True
----------------
aeiou
False
----------------
```

+ The answer is correct! emmmu & aeiou is not correct

1. some vowel is capitalized

   You need to add "capitalized" to the grammar, and to the valuations.

Add grammar

```
A[SEM=<\n.exists c.(capital(n) & char(n,c))>] ->
'capitalized'
```

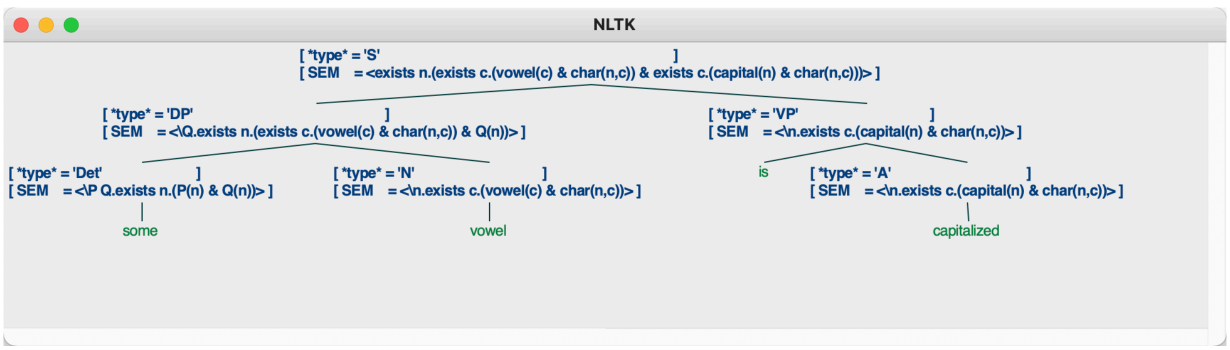
In [ ]:

```python
g3 = nltk.load_parser('h2.fcfg', trace=0,cache=False)
s3 = 'some vowel is capitalized'
s3_split = s3.split()
for tree in g3.parse(s3_split): print(tree)
```

```
(S[SEM=<exists n.(exists c.(vowel(c) & char(n,c)) & exists c.(capital(n) & c
har(n,c)))>]
  (DP[SEM=<\Q.exists n.(exists c.(vowel(c) & char(n,c)) & Q(n))>]
    (Det[SEM=<\P Q.exists n.(P(n) & Q(n))>] some)
    (N[SEM=<\n.exists c.(vowel(c) & char(n,c))>] vowel))
  (VP[SEM=<\n.exists c.(capital(n) & char(n,c))>]
    is
    (A[SEM=<\n.exists c.(capital(n) & char(n,c))>] capitalized)))
```

In [ ]:

```python
tree.draw()
```



In [ ]:

```python
t3=next(g3.parse(s3_split))
f3 = t3.label()['SEM']
print(f3)
```

```
exists n.(exists c.(vowel(c) & char(n,c)) & exists c.(capital(n) & char(n,c)
))
```

In [ ]:

```python
e3 = [('emmmu',False),('aEkjd',True),('mAEOwr',True),('aeiou',False)]
words = [e[0] for e in e3]
truths = [e[1] for e in e3]
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, get_capital]))
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]
```

In [ ]:

```python
print(f'{s3}\n----------------')
for w, a, m in zip(words, assignments, models):
    print(f'{w}\n{m.evaluate(str(f3),a,trace=None)}\n----------------')
```

```
some vowel is capitalized
---------------
emmmu
False
----------------
aEkjd
True
----------------
mAEOwr
True
----------------
aeiou
False
----------------
```

```
+ The answer is correct! aEkjd & mAEOwr has capitalized vowel(s)
```

1. no glide is capitalized

   The glides are "y" and "w". Define "no", with the same semantic type as "every" and "some". Include a constant 'glide' in the valuations.

Add grammar

```
Det[SEM=<\P Q.all n.(P(n) -> -Q(n))>] -> 'no'
N[SEM=<\n.exists c.(glide(c) & char(n,c))>] -> 'glide'
```

Add in function

```
get_glide = lambda w: f'glide => {set(re.findall(r"[YWyw]",
w))}'.lower()
```

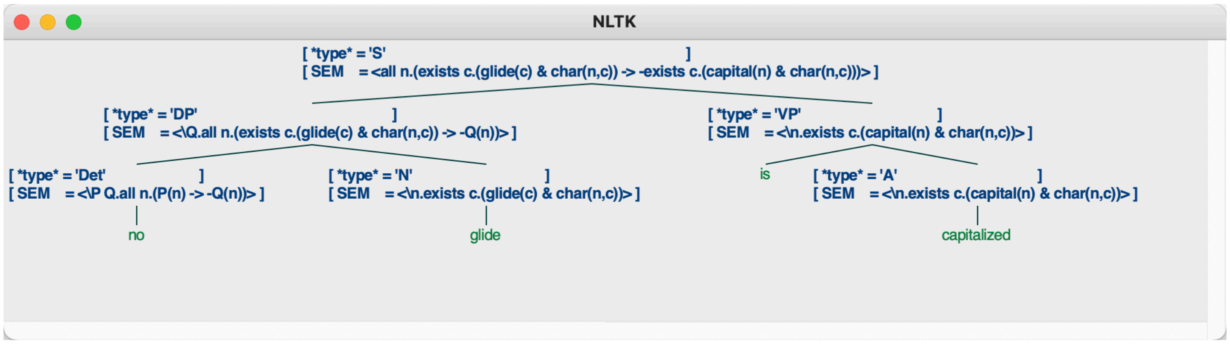
In [ ]:

```
g4 = nltk.load_parser('h2.fcfg', trace=0,cache=False)
s4 = 'no glide is capitalized'
s4_split = s4.split()
for tree in g4.parse(s4_split): print(tree)
```

```
(S[SEM=<all n.(exists c.(glide(c) & char(n,c)) -> -exists c.(capital(n) & ch
ar(n,c)))>]
  (DP[SEM=<\Q.all n.(exists c.(glide(c) & char(n,c)) -> -Q(n))>]
    (Det[SEM=<\P Q.all n.(P(n) -> -Q(n))>] no)
    (N[SEM=<\n.exists c.(glide(c) & char(n,c))>] glide))
  (VP[SEM=<\n.exists c.(capital(n) & char(n,c))>]
    is
    (A[SEM=<\n.exists c.(capital(n) & char(n,c))>] capitalized)))
```

In [ ]:

```
tree.draw()
```



In [ ]:
```python
t4=next(g4.parse(s4_split))
f4 = t4.label()['SEM']
print(f4)
```

```
all n.(exists c.(glide(c) & char(n,c)) -> -exists c.(capital(n) & char(n,c))
)
```

In [ ]:
```python
e4 = [('emwywu',True),('aEWWWjd',False),('mAEOwr',True),('aeYWou',False)]
words = [e[0] for e in e4]
truths = [e[1] for e in e4]
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel, get_capital,ge
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]
```

In [ ]:
```python
print(f'{s4}\n---------------')
for w, a, m in zip(words, assignments, models):
    print(f'{w}\n{m.evaluate(str(f4),a,trace=None)}\n----------------')
```

```
no glide is capitalized
---------------
emwywu
True
----------------
aEWWWjd
False
----------------
mAEOwr
True
----------------
aeYWou
False
----------------
```

+ Correct!

1. letter one is initial and is a consonant
   Define "initial", meaning 'is at the start of the word' in terms of the available primitives.

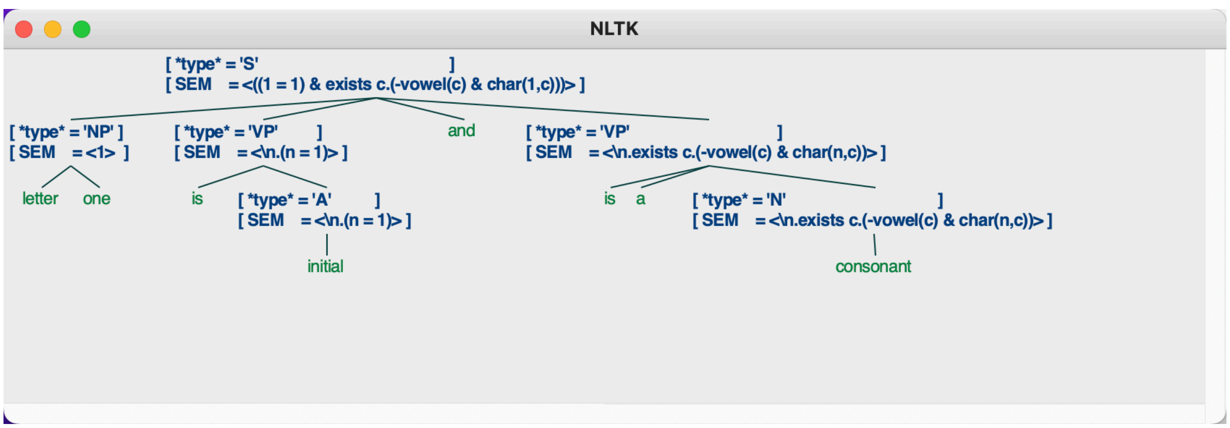
Add grammar

```
S[SEM=<?vp1(?subj) & ?vp2(?subj)>] -> NP[SEM=?subj] VP[SEM=?
vp1] 'and' VP[SEM=?vp2]
VP[SEM=?P] -> 'is' 'a' N[SEM=?P]
A[SEM=<\n. (n = 1) >] -> 'initial'
```

In [ ]:
```python
g5 = nltk.load_parser('h2.fcfg', trace=0,cache=False)
s5 = 'letter one is initial and is a consonant  '
s5_split = s5.split()
for tree in g5.parse(s5_split): print(tree)
```

```
(S[SEM=<((1 = 1) & exists c.(-vowel(c) & char(1,c)))>]
  (NP[SEM=<1>] letter one)
  (VP[SEM=<\n.(n = 1)>] is (A[SEM=<\n.(n = 1)>] initial))
  and
  (VP[SEM=<\n.exists c.(-vowel(c) & char(n,c))>]
    is
    a
    (N[SEM=<\n.exists c.(-vowel(c) & char(n,c))>] consonant)))
```

In [ ]:
```python
tree.draw()
```



In [ ]:
```python
t5=next(g5.parse(s5_split))
f5 = t5.label()['SEM']
print(f5)
```

```
((1 = 1) & exists c.(-vowel(c) & char(1,c)))
```

In [ ]:
```python
e5 = [('emwywu',True),('aEWWWjd',False),('mAEOwr',True),('aeYWou',False)]
words = [e[0] for e in e5]
truths = [e[1] for e in e5]
vals = [nltk.Valuation.fromstring(to_model_str(w, [get_vowel])) for w in wor
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]
```

In [ ]:
```
print(f'{s5}\n---------------')
for w, a, m in zip(words, assignments, models):
    print(f'{w}\n{m.evaluate(str(f5),a,trace=0)}\n----------------')
```

```
letter one is initial and is a consonant
---------------
emwywu
False
----------------
aEWWWjd
False
----------------
mAEOwr
True
----------------
aeYWou
False
----------------
```

+ Correct!