Experiment 8

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
SUBJECT	**NLP Lab**

AIM: To capture linguistic patterns and grammatical constructions with feature based grammars.

THEORY:

In order to gain more flexibility, we change our treatment of grammatical categories like S, NP and V. In place of atomic labels, we decompose them into structures like dictionaries, where features can take on a range of values. In this experiment, we will investigate the role of features in building rule-based grammars. In contrast to feature extractors, which record features that have been automatically detected, we are now going to declare the features of words and phrases. We start off with a very simple example, using dictionaries to store features and their values.

```
>>> kim = {'CAT': 'NP', 'ORTH': 'Kim', 'REF': 'k'}
>>> chase = {'CAT': 'V', 'ORTH': 'chased', 'REL': 'chase'}
```

The objects kim and chase both have a couple of shared features, CAT (grammatical category) and ORTH (orthography, i.e., spelling). In addition, each has a more semantically-oriented feature: kim['REF'] is intended to give the referent of kim, while chase['REL'] gives the relation expressed by chase. In the context of rule-based grammars, such pairings of features and values are known as feature structures.

Feature structures contain various kinds of information about grammatical entities. The information need not be exhaustive, and we might want to add further properties.

For example, in the case of a verb, it is often useful to know what "semantic role" is played by the arguments of the verb. In the case of chase, the subject plays the role of "agent", while the object has the role of "patient". Let's add this information, using 'sbj' and 'obj' as placeholders which will get filled once the verb combines with its grammatical arguments:

```
>>> chase['AGT'] = 'sbj'
>>> chase['PAT'] = 'obj'
```

If we now process a sentence Kim chased Lee, we want to "bind" the verb's agent role to the

subject and the patient role to the object. We do this by linking to the REF feature of the relevant NP. In the following example, we make the simple-minded assumption that the NPs

immediately to the left and right of the verb are the subject and object respectively. We also add a feature structure for Lee to complete the example

```
>>> sent = "Kim chased Lee"
>>> tokens = sent.split()
>>> lee = {'CAT': 'NP', 'ORTH': 'Lee', 'REF': '1'}
>>> def lex2fs(word):
...     for fs in [kim, lee, chase]:
...         if fs['ORTH'] == word:
...             return fs
>>> subj, verb, obj = lex2fs(tokens[0]), lex2fs(tokens[1]), lex2fs(tokens[2])
>>> verb['AGT'] = subj['REF']
>>> verb['PAT'] = obj['REF']
>>> for k in ['ORTH', 'REL', 'AGT', 'PAT']:
...     print("%-5s => %s" % (k, verb[k]))
ORTH  => chased
REL   => chase
AGT   => k
PAT   => 1
```

Processing Feature Structures:

Feature structures in NLTK are declared with the FeatStruct() constructor. Atomic feature values can be strings or integers.

```
>>> fs1 = nltk.FeatStruct(TENSE='past', NUM='sg')
>>> print(fs1)
[ NUM   = 'sg'   ]
[ TENSE = 'past' ]
```

A feature structure is actually just a kind of dictionary, and so we access its values by indexing in the usual way. We can use our familiar syntax to assign values to features:

```
>>> fs1 = nltk.FeatStruct(PER=3, NUM='pl', GND='fem')
>>> print(fs1['GND'])
fem
>>> fs1['CASE'] = 'acc'
```

Overview :

- The traditional categories of context-free grammar are atomic symbols. An important motivation for feature structures is to capture fine-grained distinctions that would otherwise require a massive multiplication of atomic categories.
- By using variables over feature values, we can express constraints in grammar productions that allow the realization of different feature specifications to be interdependent.
- Typically we specify fixed values of features at the lexical level and constrain the values of features in phrases to unify with the corresponding values in their children.

- Feature values are either atomic or complex. A particular sub-case of atomic value is the Boolean value, represented by convention as [+/- f].
- Two features can share a value (either atomic or complex). Structures with shared values are said to be re-entrant. Shared values are represented by numerical indexes (or tags) in AVMs.
- A path in a feature structure is a tuple of features corresponding to the labels on a sequence of arcs from the root of the graph representation.
- Two paths are equivalent if they share a value.
- Feature structures are partially ordered by subsumption. FS0 subsumes FS1 when all the information contained in FS0 is also present in FS1.
- The unification of two structures FS0 and FS1, if successful, is the feature structure FS2 that contains the combined information of both FS0 and FS1.
- If unification adds information to a path π in FS, then it also adds information to every path π' equivalent to π.
- We can use feature structures to build succinct analyses of a wide variety of linguistic phenomena, including verb subcategorization, inversion constructions, unbounded dependency constructions and case government.

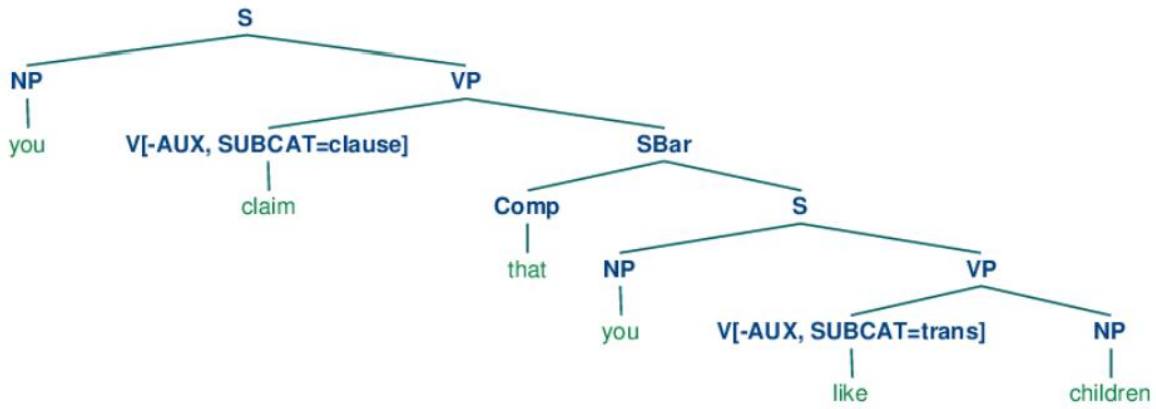
Extending a Feature based Grammar operations:

1. Subcategorization
2. Heads Revisited
3. Auxiliary Verbs and Inversion
4. Unbounded Dependency Constructions
5.

Input: Multiple feature based grammars.

Output: Extend the feature based grammars by performing the operations mentioned above.

Example Input: You claim that you like children



IDE USED: Jupyter Notebook

LIBRARIES USED:

Nltk:

The Natural Language Toolkit (NLTK) is a Python package for natural language processing. NLTK requires Python 3.7, 3.8, 3.9 or 3.10. It can be installed as :

pip install nltk

CODE & OUTPUT:

```
In [1]: import nltk
        import re
        from collections import OrderedDict
        from nltk.probability import FreqDist
        from nltk.tokenize import sent_tokenize,word_tokenize

In [2]: nltk.download('punkt')

        [nltk_data] Downloading package punkt to
        [nltk_data]     C:\Users\AIM\AppData\Roaming\nltk_data...
        [nltk_data]   Package punkt is already up-to-date!


        True

In [3]: nltk.download('averaged_perceptron_tagger')

        [nltk_data] Downloading package averaged_perceptron_tagger to
        [nltk_data]     C:\Users\AIM\AppData\Roaming\nltk_data...
        [nltk_data]   Package averaged_perceptron_tagger is already up-to-
        [nltk_data]       date!


        True
```

```
kim = {'CAT': 'NP', 'ORTH': 'Kim', 'REF': 'k'}
chase = {'CAT': 'V', 'ORTH': 'chased', 'REL': 'chase'}

chase['AGT'] = 'sbj'
chase['PAT'] = 'obj'
```

```
sent = "Kim chased Lee"
tokens = sent.split()
lee = {'CAT': 'NP', 'ORTH': 'Lee', 'REF': 'l'}
def lex2fs(word):
  for fs in [kim, lee, chase]:
      if fs['ORTH'] == word:
          return fs
subj, verb, obj = lex2fs(tokens[0]), lex2fs(tokens[1]), lex2fs(tokens[2])
verb['AGT'] = subj['REF']
verb['PAT'] = obj['REF']
for k in ['ORTH', 'REL', 'AGT', 'PAT']:
  print("%-5s => %s" % (k, verb[k]))
```

```
ORTH  => chased
REL   => chase
AGT   => k
PAT   => l
```

```
surprise = {'CAT': 'V', 'ORTH': 'surprised', 'REL': 'surprise','SRC': 'sbj', 'EXP': 'obj'}
```

```
nltk.data.show_cfg('content/feat0.fcfg')
```

```
% start S
# ###################
# Grammar Productions
# ###################
# S expansion productions
S -> NP[NUM=?n] VP[NUM=?n]
# NP expansion productions
NP[NUM=?n] -> N[NUM=?n]
NP[NUM=?n] -> PropN[NUM=?n]
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
NP[NUM=pl] -> N[NUM=pl]
# VP expansion productions
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t, NUM=?n]
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP
# ###################
# Lexical Productions
# ###################
Det[NUM=sg] -> 'this' | 'every'
Det[NUM=pl] -> 'these' | 'all'
Det -> 'the' | 'some' | 'several'
PropN[NUM=sg]-> 'Kim' | 'Jody'
N[NUM=sg] -> 'dog' | 'girl' | 'car' | 'child'
N[NUM=pl] -> 'dogs' | 'girls' | 'cars' | 'children'
IV[TENSE=pres,  NUM=sg] -> 'disappears' | 'walks'
TV[TENSE=pres, NUM=sg] -> 'sees' | 'likes'
IV[TENSE=pres,  NUM=pl] -> 'disappear' | 'walk'
```

```
nltk.data.show_cfg('content/feat0.fcfg')
```

```
% start S
# ###################
# Grammar Productions
# ###################
# S expansion productions
S -> NP[NUM=?n] VP[NUM=?n]
# NP expansion productions
NP[NUM=?n] -> N[NUM=?n]
NP[NUM=?n] -> PropN[NUM=?n]
NP[NUM=?n] -> Det[NUM=?n] N[NUM=?n]
NP[NUM=pl] -> N[NUM=pl]
# VP expansion productions
VP[TENSE=?t, NUM=?n] -> IV[TENSE=?t, NUM=?n]
VP[TENSE=?t, NUM=?n] -> TV[TENSE=?t, NUM=?n] NP
# ###################
# Lexical Productions
# ###################
Det[NUM=sg] -> 'this' | 'every'
Det[NUM=pl] -> 'these' | 'all'
Det -> 'the' | 'some' | 'several'
PropN[NUM=sg]-> 'Kim' | 'Jody'
N[NUM=sg] -> 'dog' | 'girl' | 'car' | 'child'
N[NUM=pl] -> 'dogs' | 'girls' | 'cars' | 'children'
IV[TENSE=pres,  NUM=sg] -> 'disappears' | 'walks'
TV[TENSE=pres, NUM=sg] -> 'sees' | 'likes'
IV[TENSE=pres,  NUM=pl] -> 'disappear' | 'walk'
TV[TENSE=pres, NUM=pl] -> 'see' | 'like'
IV[TENSE=past] -> 'disappeared' | 'walked'
TV[TENSE=past] -> 'saw' | 'liked'
```

```
tokens = 'Kim likes children'.split()
from nltk import load_parser
```

```
|.Kim .like.chil.|
Leaf Init Rule:
|[----]    .    .| [0:1] 'Kim'
|.    [----]    .| [1:2] 'likes'
|.    .    [----]| [2:3] 'children'
Feature Bottom Up Predict Combine Rule:
|[----]    .    .| [0:1] PropN[NUM='sg'] -> 'Kim' *
Feature Bottom Up Predict Combine Rule:
|[----]    .    .| [0:1] NP[NUM='sg'] -> PropN[NUM='sg'] *
Feature Bottom Up Predict Combine Rule:
|[---->    .    .| [0:1] S[] -> NP[NUM=?n] * VP[NUM=?n] {?n: 'sg'}
Feature Bottom Up Predict Combine Rule:
|.    [----]    .| [1:2] TV[NUM='sg', TENSE='pres'] -> 'likes' *
Feature Bottom Up Predict Combine Rule:
|.    [---->    .| [1:2] VP[NUM=?n, TENSE=?t] -> TV[NUM=?n, TENSE=?t] * NP[] {?n: 'sg', ?t: 'pres'}
Feature Bottom Up Predict Combine Rule:
|.    .    [----]| [2:3] N[NUM='pl'] -> 'children' *
Feature Bottom Up Predict Combine Rule:
|.    .    [----]| [2:3] NP[NUM='pl'] -> N[NUM='pl'] *
Feature Bottom Up Predict Combine Rule:
|.    .    [---->| [2:3] S[] -> NP[NUM=?n] * VP[NUM=?n] {?n: 'pl'}
Feature Single Edge Fundamental Rule:
|.    [---------]| [1:3] VP[NUM='sg', TENSE='pres'] -> TV[NUM='sg', TENSE='pres'] NP[] *
Feature Single Edge Fundamental Rule:
|[==============]| [0:3] S[] -> NP[NUM='sg'] VP[NUM='sg'] *
(S[]
  (NP[NUM='sg'] (PropN[NUM='sg'] Kim))
  (VP[NUM='sg', TENSE='pres']
    (TV[NUM='sg', TENSE='pres'] likes)
    (NP[NUM='pl'] (N[NUM='pl'] children))))
```

```
fs1 = nltk.FeatStruct(TENSE='past', NUM='sg')
print(fs1)
```

```
[ NUM   = 'sg'   ]
[ TENSE = 'past' ]
```

```
fs1 = nltk.FeatStruct(PER=3, NUM='pl', GND='fem')
print(fs1['GND'])
fs1['CASE'] = 'acc'
```

```
fem
```

```
fs2 = nltk.FeatStruct(POS='N', AGR=fs1)
print(fs2)
```

```
[       [ CASE = 'acc' ] ]
[ AGR = [ GND  = 'fem' ] ]
[       [ NUM  = 'pl'  ] ]
[       [ PER  = 3     ] ]
[                        ]
[ POS = 'N'              ]
```

```
print(fs2['AGR'])
```

```
[ CASE = 'acc' ]
[ GND  = 'fem' ]
[ NUM  = 'pl'  ]
[ PER  = 3     ]
```

```
print(fs2['AGR']['PER'])
```

```
3
```

```
print(nltk.FeatStruct("[POS='N', AGR=[PER=3, NUM='pl', GND='fem']]"))
```

```
[       [ GND = 'fem' ] ]
[ AGR = [ NUM = 'pl'  ] ]
[       [ PER = 3     ] ]
[                       ]
[ POS = 'N'             ]
```

```
print(nltk.FeatStruct(NAME='Lee', TELNO='01 27 86 42 96', AGE=33))
```

```
[ AGE   = 33              ]
[ NAME  = 'Lee'           ]
[ TELNO = '01 27 86 42 96' ]
```

```
print(nltk.FeatStruct("""[NAME='Lee', ADDRESS=(1)[NUMBER=74, STREET='rue Pascal'], SPOUSE=[NAME='Kim', ADDRI
```

```
  [ ADDRESS = (1) [ NUMBER = 74          ] ]
  [               [ STREET = 'rue Pascal' ] ]
  [                                         ]
  [ NAME    = 'Lee'                         ]
  [                                         ]
  [ SPOUSE  = [ ADDRESS -> (1)  ]           ]
  [           [ NAME    = 'Kim' ]           ]
```

```
print(nltk.FeatStruct("[A='a', B=(1)[C='c'], D->(1), E->(1)]"))
```

```
  [ A = 'a'            ]
  [                    ]
  [ B = (1) [ C = 'c' ] ]
  [                    ]
  [ D -> (1)           ]
  [ E -> (1)           ]
```

```
fs1 = nltk.FeatStruct(NUMBER=74, STREET='rue Pascal')
```

```
fs2 = nltk.FeatStruct(CITY='Paris')
```

```
print(fs1.unify(fs2))
```

```
  [ CITY   = 'Paris'      ]
  [ NUMBER = 74           ]
  [ STREET = 'rue Pascal' ]
```

```
print(fs2.unify(fs1))
```

```
print(fs2.unify(fs1))
```

```
 [ CITY   = 'Paris'      ]
 [ NUMBER = 74           ]
 [ STREET = 'rue Pascal' ]
```

```
fs0 = nltk.FeatStruct(A='a')
fs1 = nltk.FeatStruct(A='b')
fs2 = fs0.unify(fs1)
print(fs2)
```

```
None
```

```
fs0 = nltk.FeatStruct("""[NAME=Lee,
                          ADDRESS=[NUMBER=74,
                                   STREET='rue Pascal'],
                           SPOUSE= [NAME=Kim,
                                   ADDRESS=[NUMBER=74,
                                            STREET='rue Pascal']]]""")

print(fs0)
```

```
 [ ADDRESS = [ NUMBER = 74          ]             ]
 [           [ STREET = 'rue Pascal' ]             ]
 [                                                 ]
 [ NAME    = 'Lee'                                 ]
 [                                                 ]
 [           [ ADDRESS = [ NUMBER = 74          ] ] ]
```

```
[                                       ]
[          [ ADDRESS = [ NUMBER = 74          ] ] ]
[ SPOUSE  = [           [ STREET = 'rue Pascal' ] ] ]
[          [                                  ] ] ]
[          [ NAME    = 'Kim'                  ] ] ]
```

```
fs1 = nltk.FeatStruct("[SPOUSE = [ADDRESS = [CITY = Paris]]]")
print(fs1.unify(fs0))
```

```
[ ADDRESS = [ NUMBER = 74          ]              ]
[           [ STREET = 'rue Pascal' ]              ]
[                                                  ]
[ NAME    = 'Lee'                                  ]
[                                                  ]
[           [           [ CITY   = 'Paris'      ] ] ]
[ ADDRESS = [ NUMBER = 74           ] ] ]
[ SPOUSE  = [           [ STREET = 'rue Pascal' ] ] ]
[           [                                  ] ] ]
[           [ NAME    = 'Kim'                  ] ] ]
```

```
fs2 = nltk.FeatStruct("""[NAME=Lee, ADDRESS=(1)[NUMBER=74, STREET='rue Pascal'],
                          SPOUSE=[NAME=Kim, ADDRESS->(1)]]""")
print(fs1.unify(fs2))
```

```
[              [ CITY   = 'Paris'      ] ]
[ ADDRESS = (1) [ NUMBER = 74           ] ]
[              [ STREET = 'rue Pascal' ] ]
[                                        ]
[ NAME    = 'Lee'                        ]
[                                        ]
[ SPOUSE  = [ ADDRESS -> (1)  ]          ]
[           [ NAME    = 'Kim' ]          ]
```

```
fs1 = nltk.FeatStruct("[ADDRESS1=[NUMBER=74, STREET='rue Pascal']]")
fs2 = nltk.FeatStruct("[ADDRESS1=?x, ADDRESS2=?x]")
print(fs2)

print(fs2.unify(fs1))
```

```
[ ADDRESS1 = ?x ]
[ ADDRESS2 = ?x ]
[ ADDRESS1 = (1) [ NUMBER = 74           ] ]
[                [ STREET = 'rue Pascal' ] ]
[                                          ]
[ ADDRESS2 -> (1)                          ]
```

```
nltk.data.show_cfg('/content/feat1.fcfg')
```

```
% start S
# ###################
# Grammar Productions
# ###################
S[-INV] -> NP VP
S[-INV]/?x -> NP VP/?x
S[-INV] -> NP S/NP
S[-INV] -> Adv[+NEG] S[+INV]
S[+INV] -> V[+AUX] NP VP
S[+INV]/?x -> V[+AUX] NP VP/?x
SBar -> Comp S[-INV]
SBar/?x -> Comp S[-INV]/?x
VP -> V[SUBCAT=intrans, -AUX]
VP -> V[SUBCAT=trans, -AUX] NP
VP/?x -> V[SUBCAT=trans, -AUX] NP/?x
VP -> V[SUBCAT=clause, -AUX] SBar
```

```
tokens = 'who do you claim that you like'.split()
from nltk import load_parser
cp = load_parser('/content/feat1.fcfg')
for tree in cp.parse(tokens):
    print(tree)
```

```
(S[-INV]
  (NP[+WH] who)
  (S[+INV]/NP[]
    (V[+AUX] do)
    (NP[-WH] you)
    (VP[]/NP[]
      (V[-AUX, SUBCAT='clause'] claim)
      (SBar[]/NP[]
        (Comp[] that)
        (S[-INV]/NP[]
          (NP[-WH] you)
          (VP[]/NP[] (V[-AUX, SUBCAT='trans'] like) (NP[]/NP[] )))))))
```

```
tokens = 'you claim that you like cats'.split()
for tree in cp.parse(tokens):
     print(tree)
```

```
(S[-INV]
  (NP[-WH] you)
  (VP[]
    (V[-AUX, SUBCAT='clause'] claim)
    (SBar[]
      (Comp[] that)
      (S[-INV]
        (NP[-WH] you)
   (NP[-WH] you)
   (VP[]
     (V[-AUX, SUBCAT='clause'] claim)
     (SBar[]
       (Comp[] that)
       (S[-INV]
         (NP[-WH] you)
         (VP[] (V[-AUX, SUBCAT='trans'] like) (NP[-WH] cats))))))
```

```
tokens = 'rarely do you sing'.split()
for tree in cp.parse(tokens):
     print(tree)
```

```
(S[-INV]
  (Adv[+NEG] rarely)
  (S[+INV]
    (V[+AUX] do)
    (NP[-WH] you)
    (VP[] (V[-AUX, SUBCAT='intrans'] sing))))
```

```
nltk.data.show_cfg('content/german.fcfg')
```

```
% start S
# Grammar Productions
S -> NP[CASE=nom, AGR=?a] VP[AGR=?a]
NP[CASE=?c, AGR=?a] -> PRO[CASE=?c, AGR=?a]
NP[CASE=?c, AGR=?a] -> Det[CASE=?c, AGR=?a] N[CASE=?c, AGR=?a]
VP[AGR=?a] -> IV[AGR=?a]
VP[AGR=?a] -> TV[OBJCASE=?c, AGR=?a] NP[CASE=?c]
# Lexical Productions
# Singular determiners
# masc
```

```
tokens = 'ich folge den Katzen'.split()
cp = load_parser('/content/german.fcfg')
for tree in cp.parse(tokens):
     print(tree)
```

```
(S[]
  (NP[AGR=[NUM='sg', PER=1], CASE='nom']
    (PRO[AGR=[NUM='sg', PER=1], CASE='nom'] ich))
  (VP[AGR=[NUM='sg', PER=1]]
    (TV[AGR=[NUM='sg', PER=1], OBJCASE='dat'] folge)
    (NP[AGR=[GND='fem', NUM='pl', PER=3], CASE='dat']
      (Det[AGR=[NUM='pl', PER=3], CASE='dat'] den)
      (N[AGR=[GND='fem', NUM='pl', PER=3]] Katzen))))
```

```
tokens = 'ich folge den Katze'.split()
cp = load_parser('/content/german.fcfg', trace=2)
for tree in cp.parse(tokens):
    print(tree)
```

```
|.ich.fol.den.Kat.|
Leaf Init Rule:
|[---]   .   .   .| [0:1] 'ich'
|.   [---]   .   .| [1:2] 'folge'
|.   .   [---]   .| [2:3] 'den'
|.   .   .   [---]| [3:4] 'Katze'
Feature Bottom Up Predict Combine Rule:
|[---]   .   .   .| [0:1] PRO[AGR=[NUM='sg', PER=1], CASE='nom'] -> 'ich' *
Feature Bottom Up Predict Combine Rule:
|[---]   .   .   .| [0:1] NP[AGR=[NUM='sg', PER=1], CASE='nom'] -> PRO[AGR=[NUM='sg', PER=1], CASE='nom'] *
```

REFERENCES:

1. https://www.nltk.org/book/ch09.html
2. http://courses.washington.edu/ling571/ling571_WIN2016/slides/ling571_class9_featgram_flat.pdf