LAB-13 IMPROVING GRAMMAR TO PARSE AMBIGIOUS SENTENCES

NAME : DINESH KUMAR ¶

ROLLNO : 225229108

In [1]:
```
import nltk
from nltk.tree import Tree
from nltk.tokenize import word_tokenize
from IPython.display import display
import nltk,re,pprint
from nltk.tag import pos_tag
from nltk.chunk import ne_chunk
import numpy as npt
```

EXERCISE-1

In [2]:
```
Grammar_1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'
S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
P -> 'from' | 'in'
S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'
S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'
S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'
S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
""")
```

1. Examine the parser output from the previous lab. Is any of the sentences ambiguous, that is, has more than one parse tree? Pick an example and provide an explanation.

Yes! here we have two sentences which has more than one parse tree

1. Homer and his friends from work drank and sang in the bar
2. Lisa told her brother that she liked peanut butter very much

In [4]:
```
sentence5 = word_tokenize("Homer and his friends from work drank and sang in t
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sentence5):
 print(i)
```

```
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (VP (V drank)) (CONJ and) (VP (V sang)))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP
    (NP (NP (N Homer)) (CONJ and) (NP (Det his) (N friends)))
    (PP (P from) (NP (N work))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
(S
  (NP
    (NP (N Homer))
    (CONJ and)
    (NP (NP (Det his) (N friends)) (PP (P from) (NP (N work)))))
  (VP
    (VP (V drank))
    (CONJ and)
    (VP (VP (V sang)) (PP (P in) (NP (Det the) (N bar))))))
```

In [6]:
```
sentence6 = word_tokenize("Lisa told her brother that she liked peanut butter
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sentence6):
    print(i)
```

```
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
      (S
        (NP (PRO she))
        (VP
          (VP (V liked) (NP (N peanut) (N butter)))
          (ADVP (ADV very) (ADV much)))))))
(S
  (NP (N Lisa))
  (VP
    (V told)
    (NP (Det her) (N brother))
    (CP
      (COMP that)
```

2. Have your parser parse this new sentence. It is covered by the grammar, therefore the parser should be able to handle it:

   (s12): Lisa and her friends told Marge that Homer punched the bully in the bar

In [7]:
```
s12 = word_tokenize("Lisa and her friends told Marge that Homer punched the bu
par = nltk.ChartParser(Grammar_1)
for i in par.parse(s12):
 print(i)
```

```
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP (N Marge))
      (CP
        (COMP that)
        (S (NP (N Homer)) (VP (V punched) (NP (Det the) (N bully))))))
    (PP (P in) (NP (Det the) (N bar)))))
(S
  (NP (NP (N Lisa)) (CONJ and) (NP (Det her) (N friends)))
  (VP
    (V told)
    (NP
      (NP
        (NP (N Marge))
        (CP
          (COMP that)
```

3. Come up with a sentence of your own that's covered by grammar1 and have the parser parse it. Are you satisfied with the result?

In [8]:
```
result = word_tokenize("Homer and friends punched the tiny nerdy kid after sch
par = nltk.ChartParser(Grammar_1)
for i in par.parse(result):
 print(i)
```

```
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (VP (V punched) (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid)))
    (PP (P after) (NP (N school)))))
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (V punched)
    (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
    (PP (P after) (NP (N school)))))
(S
  (NP (NP (N Homer)) (CONJ and) (NP (N friends)))
  (VP
    (V punched)
    (NP
      (NP (Det the) (ADJ tiny) (ADJ nerdy) (N kid))
      (PP (P after) (NP (N school))))))
```

4. Let's revisit our first three sentences from the previous lab.

   (s1): Marge will make a ham sandwich

   (s2): will Marge make a ham sandwich

   (s3): Homer ate the donut on the table

As it is, your grammar1 does not cover them. But we can extend it with the CF rules from the three sentences' trees. Follow the steps below.

a. From the three sentence trees, create a list of all production rules in them. Turn it into a set, which removes all duplicates. (Hint: use set().)

In [9]:
```
a_set = set()
```

In [12]:
```
s1 = Tree.fromstring('(S(NP (N Marge))(AUX will)(VP (V make) (NP (Det a) (N ha
s1_r = s1.productions()
s1_r
```

Out[12]:
```
[S -> NP AUX VP,
 NP -> N,
 N -> 'Marge',
 AUX -> 'will',
 VP -> V NP,
 V -> 'make',
 NP -> Det N N,
 Det -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

In [14]:
```
s2 = Tree.fromstring('(S(AUX will)(NP (N Marge))(VP (V make) (NP (Det a) (N ha
s2_r = s2.productions()
s2_r
```

Out[14]:
```
[S -> AUX NP VP,
 AUX -> 'will',
 NP -> N,
 N -> 'Marge',
 VP -> V NP,
 V -> 'make',
 NP -> Det N N,
 Det -> 'a',
 N -> 'ham',
 N -> 'sandwich']
```

In [16]:
```
s3 = Tree.fromstring('(S(NP (N Homer))(VP(V ate)(NP(NP (Det the) (N donut))(PP
s3_r = s3.productions()
s3_r
```

Out[16]:
```
[S -> NP VP,
 NP -> N,
 N -> 'Homer',
 VP -> V NP,
 V -> 'ate',
 NP -> NP PP,
 NP -> Det N,
 Det -> 'the',
 N -> 'donut',
 PP -> 'on' NP,
 NP -> DET N,
 DET -> 'the',
 N -> 'table']
```

In [17]:
```
s_1r = []
s_1r = s1_r.copy()
```

In [18]:
```
s_2r = []
s_2r = s2_r.copy()
```

In [19]:
```
s_3r = []
s_3r = s3_r.copy()
```

In [21]:
```
sr = []
for i in s_1r:
    sr.append(i)
for i in s_2r:
    sr.append(i)
for i in s_3r:
    sr.append(i)
```

In [24]:
```
for i in sr:
    a_set.add(i)
```

In [25]:
```
a_set
```

Out[25]:
```
{AUX -> 'will',
 DET -> 'the',
 Det -> 'a',
 Det -> 'the',
 N -> 'Homer',
 N -> 'Marge',
 N -> 'donut',
 N -> 'ham',
 N -> 'sandwich',
 N -> 'table',
 NP -> DET N,
 NP -> Det N,
 NP -> Det N N,
 NP -> N,
 NP -> NP PP,
 PP -> 'on' NP,
 S -> AUX NP VP,
 S -> NP AUX VP,
 S -> NP VP,
 V -> 'ate',
 V -> 'make',
 VP -> V NP}
```

In [26]:
```
more_r = []
more_r = list(a_set)
```

In [27]:
```
more_r
```

Out[27]:
```
[S -> AUX NP VP,
 AUX -> 'will',
 N -> 'table',
 N -> 'donut',
 NP -> DET N,
 NP -> NP PP,
 DET -> 'the',
 N -> 'ham',
 N -> 'Marge',
 S -> NP VP,
 PP -> 'on' NP,
 N -> 'Homer',
 NP -> Det N N,
 NP -> Det N,
 NP -> N,
 S -> NP AUX VP,
 Det -> 'the',
 V -> 'ate',
 N -> 'sandwich',
 VP -> V NP,
 V -> 'make',
 Det -> 'a']
```

c. Add the additional rules to your grammar1's production rules, using the .extend() method.

In [28]:
```
Grammar_1.productions().extend(list(more_r))
```

Marge will make a ham sandwich

In [29]:
```
results = word_tokenize("")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(results):
    print(i)
```

d. And then, you have to re-initialize the grammar using the extended production rules (highlighted part). An illustration:

In [30]:
```
grammar3 = nltk.CFG.fromstring("""
S -> NP VP
NP -> N
VP -> V
N -> 'Homer'
V -> 'sleeps'
""")
```

In [31]:
```
print(grammar3)
```

```
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
```

```
In [32]: more_r
```

```
Out[32]: [S -> AUX NP VP,
          AUX -> 'will',
          N -> 'table',
          N -> 'donut',
          NP -> DET N,
          NP -> NP PP,
          DET -> 'the',
          N -> 'ham',
          N -> 'Marge',
          S -> NP VP,
          PP -> 'on' NP,
          N -> 'Homer',
          NP -> Det N N,
          NP -> Det N,
          NP -> N,
          S -> NP AUX VP,
          Det -> 'the',
          V -> 'ate',
          N -> 'sandwich',
          VP -> V NP,
          V -> 'make',
          Det -> 'a']
```

e. Now, rebuild your chart parser with the updated grammar1. And try parsing the three sentences. It should successfully parse them.

In [33]:
```
grammar3.productions().extend(more_r)
grammer3 = nltk.grammar.CFG(grammar3.start(), grammar3.productions())
print(grammer3)
```

```
Grammar with 27 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'
    S -> AUX NP VP
    AUX -> 'will'
    N -> 'table'
    N -> 'donut'
    NP -> DET N
    NP -> NP PP
    DET -> 'the'
    N -> 'ham'
    N -> 'Marge'
    S -> NP VP
    PP -> 'on' NP
    N -> 'Homer'
    NP -> Det N N
    NP -> Det N
    NP -> N
    S -> NP AUX VP
    Det -> 'the'
    V -> 'ate'
    N -> 'sandwich'
    VP -> V NP
    V -> 'make'
    Det -> 'a'
```

5. Try parsing another sentence of your own that is covered by the newly extended grammar1. Are you satisfied with the result?. Also, compare the result with other parsers – Recursive Descent Parser and Shift Reduce Parser.

In [34]:
```
Grammar_1 = nltk.CFG.fromstring("""
S -> NP VP | NP VP
NP -> N | Det N | PRO | N N
VP -> V NP CP | VP ADVP | V NP
ADVP -> ADV ADV
CP -> COMP S
N -> 'Lisa' | 'brother' | 'peanut' | 'butter'
V -> 'told' | 'liked'
COMP -> 'that'
Det -> 'her'
PRO -> 'she'
ADV -> 'very' | 'much'
S -> NP VP
NP -> NP CONJ NP | N | NP PP | Det N | N | Det N
VP -> VP PP | VP CONJ VP | V | V
PP -> P NP | P NP
N -> 'Homer' | 'friends' | 'work' | 'bar'
V -> 'drank' | 'sang'
CONJ -> 'and' | 'and'
Det -> 'his' | 'the'
P -> 'from' | 'in'
S -> NP VP
NP -> NP CONJ NP | N | N
VP -> V ADJP
ADJP -> ADJP CONJ ADJP | ADJ | ADV ADJ
N -> 'Homer' | 'Marge'
V -> 'are'
CONJ -> 'and' | 'but'
ADJ -> 'poor' | 'happy'
ADV -> 'very'
S -> NP VP | NP AUX VP
NP -> PRO | NP CP | Det N | PRO | PRO | PRO | N |Det N
VP -> V NP PP | V NP NP
CP -> COMP S
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he' | 'I' | 'him'
N -> 'book' | 't' | 'sister'
V -> 'gave' | 'given'
COMP -> 'that'
AUX -> 'had'
P -> 'to'
S -> NP VP
NP -> PRO | Det N | Det N
VP -> V NP PP
PP -> P NP
Det -> 'the' | 'his'
PRO -> 'he'
N -> 'book' | 'sister'
V -> 'gave'
P -> 'to'
S -> NP VP
NP -> Det ADJ N | Det ADJ ADJ N | N
VP -> V NP|VP PP
PP -> P NP
Det -> 'the' | 'the'
ADJ -> 'big' | 'tiny' | 'nerdy'
N -> 'bully' | 'kid' | 'school'
V -> 'punched'
P -> 'after'
S -> NP AUX VP
NP -> N | Det N N
VP -> V NP
N -> 'Marge' | 'ham' | 'sandwich'
AUX -> 'will'
V -> 'make'
Det -> 'a'
S -> AUX NP VP
NP -> N | Det N N
VP -> V NP
N -> 'Marge'
V -> 'make'
AUX -> 'will'
Det -> 'a'
N -> 'Marge' | 'ham' | 'sandwich'
S -> NP VP
NP -> N | NP PP | Det N | Det N
PP -> P NP
VP -> V NP
N -> 'Homer' | 'donut' | 'table'
V -> 'ate'
Det -> 'the' | 'the'
P -> 'on'
""")
```

In [35]:
```
sent = word_tokenize("will Marge make a ham sandwich")
par = nltk.ChartParser(Grammar_1)
for i in par.parse(sent):
    print(i)
```

```
(S
  (AUX will)
  (NP (N Marge))
  (VP (V make) (NP (Det a) (N ham) (N sandwich))))
(S
  (AUX will)
  (NP (N Marge))
  (VP (V make) (NP (Det a) (N ham)) (NP (N sandwich))))
```

5. Try parsing another sentence of your own that is covered by the newly extended grammar1. Are you satisfied with the result?. Also,compare the result with other parsers – Recursive Descent Parser and Shift Reduce Parser.

In [37]:
```
#sen = word_tokenize("will Marge make a ham sandwich")
#rd_par = nltk.RecursiveDescentParser(Grammar_1)
#for tree in rd_par.parse(sen):
#print(tree)
```

In [38]:
```
#sent = word_tokenize("will Marge make a ham sandwich")
#sr_par = nltk.ShiftReduceParser(Grammar_1)
#for tree in rd_par.parse(sent):
 #print(tree)
```

6. As the final step, pickle your grammar1 as lab12_grammar.pkl.

In [39]:
```
import pickle
with open('lab12_grammar.pkl', 'wb') as f:
    pickle.dump(Grammar_1, f)
```

In [ ]:
```
```