```
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.

Grammar_1 = nltk.CFG.fromstring("""
    S → NP VP | NP NP
    NP → N | Det N | PRO | N N.
    VP → V NP CP | VP ADVP | V NP.
    ADVP → ADV ADV.
    CP → COMPS.
    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' | 'bad'
    V → 'drank' | 'sang'
    CONJ → 'and' | 'and'
    Det → 'his' | 'the'
```

Natural Language Processing Lab
Lab13. Improving Grammar to Parse Ambiguous Sentences

In this lab, you will refine the grammar you have built in the previous lab. Because, the grammar does not parse some sentences that are ambiguous.

EXERCISE-1

In this part, you will be updating the grammar and the parser you built in the previous lab.

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.

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

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?

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()`.)

   b. From it, create a new list called `more_rules`, which consists of CF rules from the three trees *that are not already in `grammar1`*.

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

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

```
>>> print(grammar3)
Grammar with 5 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
    N -> 'Homer'
    V -> 'sleeps'

>>> more_rules
[V -> 'sings', V -> 'drinks']

>>> grammar3.productions().extend(more_rules)
>>> grammar3 = nltk.grammar.CFG(grammar3.start(), grammar3.productions())
>>> print(grammar3)
Grammar with 7 productions (start state = S)
    S -> NP VP
    NP -> N
    VP -> V
```

P → 'from' | 'in'

S → NP VP

NP → NP CONJ NP | N | N

VP → V ADJP

ADJP → ADJP CONJ ADJP | ADJ | ADV ADJ

N → 'book' | 'it' | 'sister'

V → 'gave' | 'given'

COMP → 'that'

AUX → 'had'

P → 'to'

S → NP VP

NP → PRO | Det N | Det N

VP → V NP PP

PP → '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'

)))))

```
sentence65 = word_tokenize
    ("Homer and his
     friends from
     coors drank and sang in
     the bar").
par = nltk.ChartParser(Grammar1)
for p in par.parse
    (sentence65):
    print(p).

sentence6 = word_tokenize
    ("Lisa told her brother
     that she liked peanut
     butter very much")
par = nltk.ChartParser
    (Grammar_1)
for p in par.parse(sentence6):
    print(p)

sb = word_tokenize("Lisa and
     her friends told Marge
     that Homer punched Nick
     bully in the bar")
par = nltk.ChartParser(Grammar1)
for p in par.parse(sb):
    print(p)
```

```
    N -> 'Homer'
    V -> 'sleeps'
    V -> 'sings'
    V -> 'drinks'
>>>
```

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

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.

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

result = Word_tokenize("Homer and friends punched the tiny nerdy kid after_school!")

par = nltk.ChartParser(Grammar1)

for i in par.parse(result):
    print(i)

(a) a_set = set()

S1 = Tree.fromstring('(S(NP (N Marge)) (Aux will) (v make) (Det a) (N_ham) (N sandwich)))')

S1_r = S1.productions()

S1_r

S2 = Tree.fromstring('(S(Aux will) (NP(N Marge)) (VP(v make) (NP(Det a) (N_ham (N sandwich)))))')

S2_r = S2.productions()

S2_r

S3 = Tree.fromstring('(S(NP (N Homer)) (VP(V ate) (NP(NP(Det the) (N donut)) (PP_(P on) (NP(Det the) (N table))))))')

S3_r = S3.productions()

S3_r

```
s_1r = []
s_1r = s1.r.copy()

s_2r = []
s_2r = s2.r.copy()

s_3r = []
s_3r = s3.r.copy()

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)

for i in sr:
    a_set.add(i)

a_set

more_r = []
more_r = list(a_set)

more_r
```

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

print(grammar3)

more_r

(e)
grammar3.productions().extend(more_r)
grammar3 = nltk.grammar.CFG
    (grammar3.start(),
     grammar3.productions())

print(grammar3)
```

```
(C) Grammar1.productions().extend(list(more_r))

Marge will make a ham sandwich.

results = word_tokenize("")
par = nltk.chartparser(Grammar_1)
for i in par.parse(results):
    print(i)
```

NOTES

```
Grammar1 = nltk.CFG.fromstring("""
    S -> NP VP | NP VP
    NP -> N | Det N | PRO NN
    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 | PNP.
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 ADVP.
```

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' | 'f' | 'sisters'

V → 'gave' | 'given'

COMP → 'that'

AUX → 'had'

P → 'to'

S → NP VP

NP → Det ADJ N | Det ADJ ADJ N | N

VP → V NP | NP 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 → 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'

// // //

NOTES

```
sent = word_tokenize ("will Marge make a ham sandwich")
par = nltk.ChartParser (Grammar_1)
for t in par.parse (sent):
    print(t)
```

5)

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

```
import pickle
with open ('lab12-grammar.pkl', 'wb') as f:
    pickle.dump (Grammar_1, f)
```