Lab Session VI

Analysing Sentence Structure

(Parsing Algorithms)

Defining CFG

In NLTK, CFG can be defined using CFG package as follows:

```
grammar = CFG.fromstring("""
S -> NP VP
NP -> Det Nom | PropN
Nom -> Adj Nom | N
VP -> V Adj | V NP | V S | V NP PP
PP -> P NP
PropN -> 'Buster' | 'Chatterer' | 'Joe'
Det -> 'the' | 'a'
N -> 'bear' | 'squirrel' | 'tree' | 'fish' | 'log'
Adj -> 'angry' | 'frightened' | 'little' | 'tall'
V -> 'chased' | 'saw' | 'said' | 'thought' | 'was' | 'put'
P -> 'on'
""")
```

We can check the start symbol of the grammar as:

grammar.start()

The productions of the grammar can be checked as:

grammar.productions()

We can list productions by matching their left hand side or right hand side

For example,

a=grammar.productions(rhs='said')

a[0].lhs()

Recursive Decent Parser (Top down, Depth-first parser)

```
>>>from nltk.parse import RecursiveDescentParser
>>> rd = RecursiveDescentParser(grammar)
>>> sentence1 = 'the cat chased the dog'.split()
>>> sentence2 = 'the cat chased the dog on the rug'.split()
>>> for t in rd.parse(sentence1):
...     print(t)
(S (NP the (N cat)) (VP (V chased) (NP the (N dog))))
>>> for t in rd.parse(sentence2):
...     print(t)
(S
  (NP the (N cat))
  (VP (V chased) (NP the (N dog) (PP (P on) (NP the (N rug))))))
(S
  (NP the (N cat))
```

```
  (VP (V chased) (NP the (N dog)) (PP (P on) (NP the (N rug)))))
```

We can also trace the working of algorithm by setting trace=2 i.e.

rd = RecursiveDescentParser(grammar,trace=2)

Shift-Reduce Parser (Simple Bottom up Parser)

Create and run a shift reduce parser over both a syntactically ambiguous and unambiguous sentence. Note that unlike the recursive descent parser, one and only one parse is ever returned.

```
>>> from nltk.parse import ShiftReduceParser
>>> sr = ShiftReduceParser(grammar)
>>> sentence1 = 'the cat chased the dog'.split()
>>> sentence2 = 'the cat chased the dog on the rug'.split()
>>> for t in sr.parse(sentence1):
...     print(t)
(S (NP the (N cat)) (VP (V chased) (NP the (N dog))))
```

The shift reduce parser uses heuristics to decide what to do when there are multiple possible shift or reduce operations available - for the supplied grammar clearly the wrong operation is selected.

```
>>> for t in sr.parse(sentence2):
...     print(t)
```