Sentence :
"A villager went to city to sell his property along with his wife."

```
sent = "A villager went to city to sell his property along with his
wife."
tokens = sent.split()
```

Grammer :

```
grammar = nltk.CFG.fromstring("""
  S  -> NP VP
  NP -> Det N | N | Det N VP
  VP -> V Adj | V NP | V S | V NP PP | Adv V  PP | V Adv PP | V PP | V
PP Adv | PP NP | VP PP |V NP PP | P VP PP | P NP | V Adv PP
  PP -> P NP | PP NP PP | P
  Det -> 'A'
  N -> 'villager' | 'city' | 'property' | 'wife.'
  Adv  -> 'along'
  V ->  'went' | 'sell'
  P -> 'his' | 'with' | 'to'
  """)
```

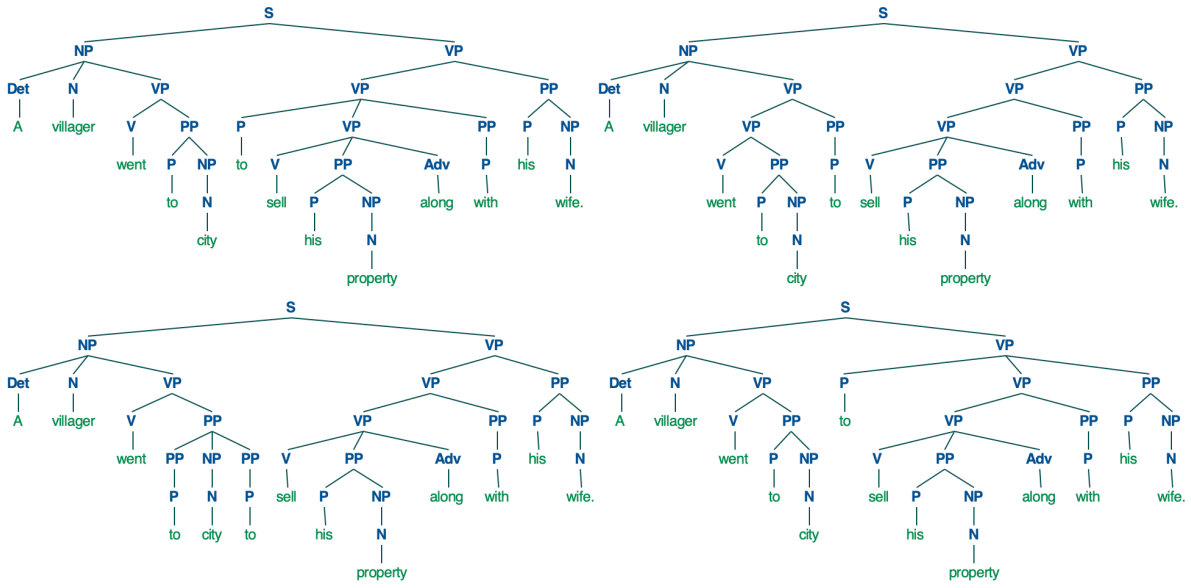
Parsing :

constituency parsing using a BottomUpChartParser

```
#Parsing algorithm

parser = nltk.parse.BottomUpChartParser(grammar)

trees = []
for tree in parser.parse(tokens):
    print(tree)
    trees.append(tree)

from nltk.draw.tree import draw_trees
x  = len(trees)
print(x)
draw_trees(trees[0], trees[1],trees[2], trees[3] )
```

```
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (VP
      (P to)
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
```

```
      (PP (P with)))
    (PP (P his) (NP (N wife.))))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (VP (V went) (PP (P to) (NP (N city)))) (PP (P to))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.))))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (V went) (PP (PP (P to)) (NP (N city)) (PP (P to)))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.))))))
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (P to)
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
```

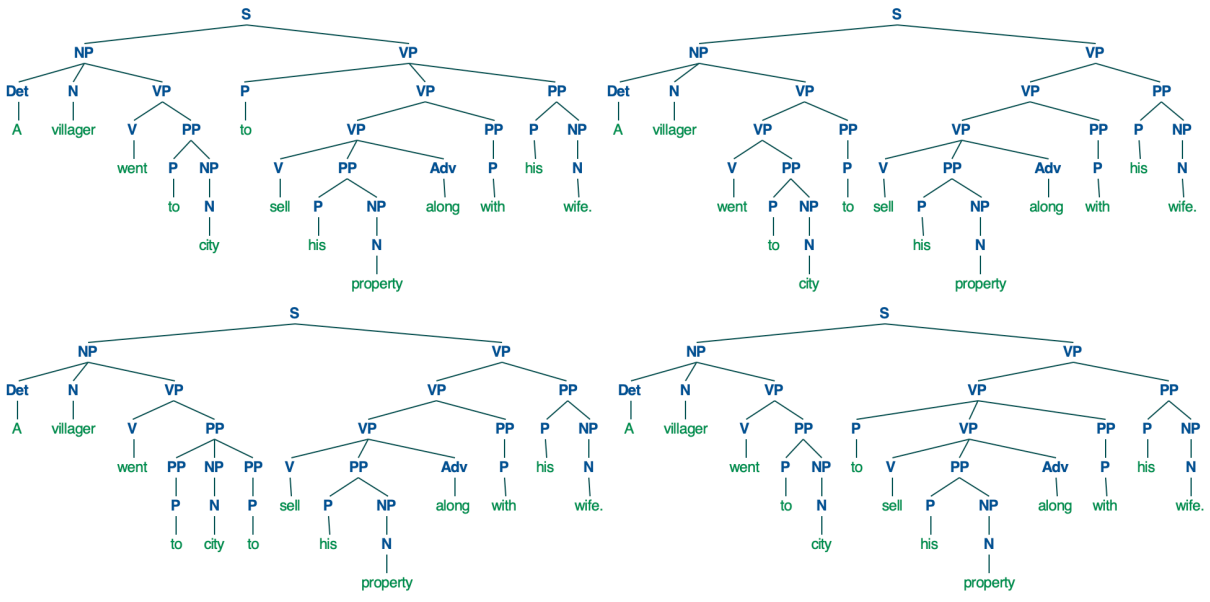


constituency parsing using a BottomUpLeftCornerChartParser

```
#Parsing algorithm

parser = nltk.parse.BottomUpLeftCornerChartParser(grammar)

trees = []
for tree in parser.parse(tokens):
    print(tree)
```

```
    trees.append(tree)

from nltk.draw.tree import draw_trees
x  = len(trees)
print(x)
draw_trees(trees[0], trees[1],trees[2], trees[3] )
```

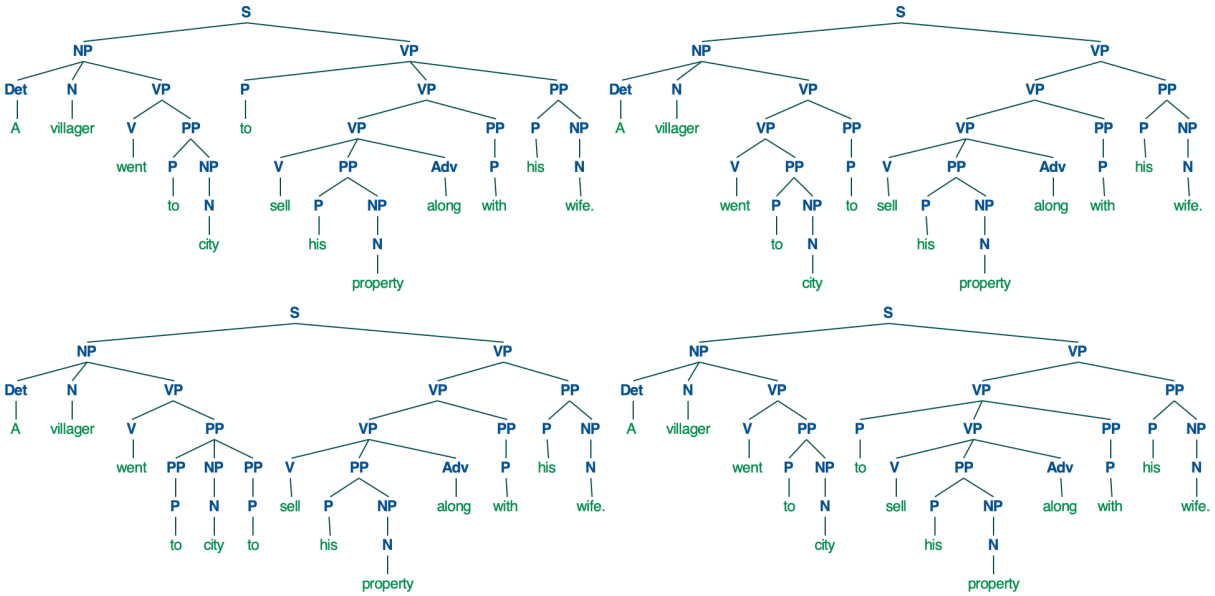
```
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (P to)
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (VP (V went) (PP (P to) (NP (N city)))) (PP (P to))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (V went) (PP (PP (P to)) (NP (N city)) (PP (P to)))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (VP
      (P to)
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
```



constituency parsing using a LeftCornerChartParser.

```
#Parsing algorithm

parser = nltk.parse.LeftCornerChartParser(grammar)

trees = []
for tree in parser.parse(tokens):
    print(tree)
    trees.append(tree)

from nltk.draw.tree import draw_trees
x  = len(trees)
print(x)
draw_trees(trees[0], trees[1],trees[2], trees[3] )
```

```
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (P to)
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (VP (V went) (PP (P to) (NP (N city)))) (PP (P to))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP
    (Det A)
    (N villager)
    (VP (V went) (PP (PP (P to)) (NP (N city)) (PP (P to)))))
  (VP
    (VP
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
(S
  (NP (Det A) (N villager) (VP (V went) (PP (P to) (NP (N city)))))
  (VP
    (VP
      (P to)
      (VP (V sell) (PP (P his) (NP (N property))) (Adv along))
      (PP (P with)))
    (PP (P his) (NP (N wife.)))))
```



2. All parser are same giving same results for given set of grammar, but if you different set of grammar them some parser don't give result. So efficiency of prayer must be depend on the grammar we choose.

3.all parser giving same results so all are doing same job. As a result we are getting four different kinds of parser tree filtering different edges for all parser since given sentence have four meanings.

1. Villager went to sell his property and also his wife.
2. Villager and his wife went to city to sell his property.
3. Villager went to city to sell his(third person) property and also his(third person) wife.
4. Villager and his wife went to city to sell his(third person) property.