COMS 4705 Natural Language Processing (2020 Spring)
Homework 2

Shusen Xu - sx2261@columbia.edu

March 2, 2020

Problem 1 PCFGs and HMMs

1. For sequences: they are baking potatoes, we get two possible sequence of tags, according to the calculation of Problem 2. They are separately:
   Parse 1: PRP V Adj N
   Parse 2: PRP Aux V N
   Since $P(tags, words) = P(words|tags) \cdot P(tags)$:
   The probability of Parse 1 is:
   $P_1 = P(they|PRP) \cdot P(are|V) \cdot P(baking|Adj) \cdot P(potatoes|N) \cdot P(PRP, V, Adj, N)$
   $= 1.0 * 0.5 * 1.0 * 1.0 * 1.0 * 0.1 * 0.8 * 0.3 * 0.6 = 0.0072$

   The probability of Parse 2 is:
   $P_2 = P(they|PRP) \cdot P(are|Aux) \cdot P(baking|V) \cdot P(potatoes|N) \cdot P(PRP, Aux, V, N)$
   $= 1.0 * 0.1 * 1.0 * 0.2 * 1.0 * 0.5 * 0.6 * 1.0 = 0.006$

2. HMM
   the HMM I design is:
   emission probability:

   $P(they|PRP) = 1.0$
   $P(potatoes|N) = 1.0$
   $P(baking|Adj) = 1.0$
   $P(baking|V) = 0.5$
   $P(are|V) = 0.5$
   $P(are|Aux) = 1.0$

   transition probability:

$P(PRP|S) = 0.1$
$P(V|PRP) = 0.8$
$P(Adj|V) = 0.3$
$P(N|Adj) = 0.6$
$P(Aux|PRP) = 0.2$
$P(V|Aux) = 1.0$
$P(N|V) = 0.6$

For Parse 1: the probability calculated by HMMs is:
$P(they|PRP) * P(are|V) * P(baking|Adj) * P(potatoes|N) * P(PRP|S) * P(V|PRP) * P(Adj|V) * P(N|Adj) = 0.0072$

For Parse 2: the probability calculated by HMMs is:
$P(they|PRP) * P(are|Aux) * P(baking|V) * P(potatoes|N) * P(PRP|S) * P(Aux|PRP) * P(V|Aux) * P(N|V) = 0.006$

So, in sum, the HMM I have designed is same as the PCFG.

3. any PCFG can be translated into an HMM

The explanation is as follows: HMMSs is essentially context free grammar because HMM is based on Markov assumption that current state is only depends on the previous state. As the same time, we know PCFG is a form of CNF: $A \rightarrow B\ C$ and $A \rightarrow b$, the probability of $P(A \rightarrow b)$ we could consider as $P(b|A)$. obviously, the PCFG also meets Markov assumption. So HMM and PCFG, essentially, is the same kind of thing.

Problem 2 Earley Parser

1. Parsing with Earley Algorithm

   Notes: Do not need to convert to CNF.
   Let's denote char[i] contains all parser items that end in position i.

Chart[0] · they are baking potatoes

StateID	dotted-rule	position	back pointers	operation
S_0	$S \rightarrow \cdot\ NP\ VP$	$[0,0]$	[]	Predict
S_1	$NP \rightarrow \cdot\ Adj\ NP$	$[0,0]$	[]	Predict
S_2	$NP \rightarrow \cdot\ PRP$	$[0,0]$	[]	Predict
S_3	$NP \rightarrow \cdot\ N$	$[0,0]$	[]	Predict
S_4	$Adj \rightarrow \cdot\ baking$	$[0,0]$	[]	do not scan, baking $\neq S[0] = they$
S_5	$PRP \rightarrow \cdot\ they$	$[0,0]$	[]	scan
S_6	$N \rightarrow \cdot\ potatoes$	$[0,0]$	[]	do not scan, potatoes $\neq S[0] = they$

Chart[1] they · are baking potatoes

StateID	dotted-rule	position	back pointers	operation
S_7	$PRP \rightarrow$ they $\cdot$	$[0,0]$	[]	Complete
S_8	$NP \rightarrow$ PRP $\cdot$	$[0,1]$	[s_7]	Complete
S_9	$S \rightarrow$ NP $\cdot\ VP$	$[0,1]$	[s_8]	Predict
S_{10}	$VP \rightarrow \cdot\ V\ NP$	$[1,1]$	[]	Predict
S_{11}	$VP \rightarrow \cdot\ Aux\ V\ NP$	$[1,1]$	[]	Predict
S_{12}	$V \rightarrow \cdot\ baking$	$[1,1]$	[]	do not scan, baking $\neq S[1] = are$
S_{13}	$V \rightarrow \cdot\ are$	$[1,1]$	[]	Scan
S_{14}	$Aux \rightarrow \cdot\ are$	$[1,1]$	[]	Scan

Chart[2] they are · baking potatoes

StateID	dotted-rule	position	back pointers	operation
S_{15}	$V \rightarrow$ are $\cdot$	$[1,2]$	[]	Complete
S_{16}	$Aux \rightarrow$ are $\cdot$	$[1,2]$	[]	Complete
S_{17}	$VP \rightarrow$ NP $\cdot\ VP$	$[1,2]$	[S_{15}]	Predict
S_{18}	$V \rightarrow$ Aux $\cdot V\ NP$	$[1,2]$	[S_{16}]	Predict
S_{19}	$NP \rightarrow \cdot\ Adj\ NP$	$[2,2]$	[]	Predict
S_{20}	$NP \rightarrow \cdot\ PRP$	$[2,2]$	[]	Predict
S_{21}	$NP \rightarrow \cdot\ N$	$[2,2]$	[]	Predict
S_{22}	$V \rightarrow \cdot\ baking$	$[2,2]$	[]	Scan
S_{23}	$V \rightarrow \cdot\ are$	$[2,2]$	[]	do not Scan,S[2] = baking $\neq are$
S_{24}	$Adj \rightarrow \cdot\ baking$	$[2,2]$	[]	Scan
S_{25}	$PRP \rightarrow \cdot\ they$	$[2,2]$	[]	do not scan, S[2] = baking $\neq they$
S_{26}	$N \rightarrow \cdot\ potatoes$	$[2,2]$	[]	do not scan, S[2] = baking $\neq potatoes$

Chart[3] they are baking · potatoes

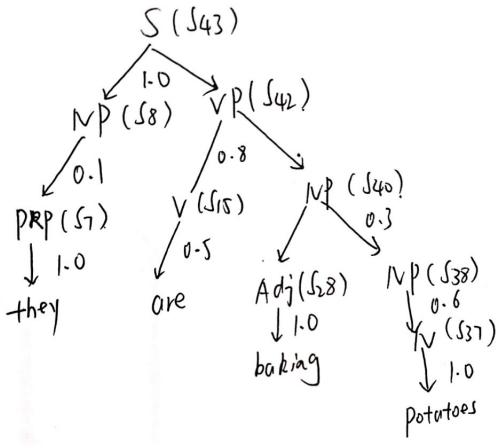
StateID	dotted-rule	position	back pointers	operation
S_{27}	$V \rightarrow$ baking ·	[2, 3]	[]	Complete
S_{28}	$Adj \rightarrow$ baking ·	[2, 3]	[]	Complete
S_{29}	$VP \rightarrow$ Aux V $\cdot NP$	[1, 3]	[S_{16}, S_{18}]	Predict
S_{30}	$NP \rightarrow$ Adj $\cdot NP$	[2, 3]	[S_{19}]	Predict
S_{31}	$NP \rightarrow$ Adj $\cdot NP$	[3, 3]	[]	Predict
S_{32}	$NP \rightarrow \cdot PRP$	[3, 3]	[]	Predict
S_{33}	$NP \rightarrow \cdot N$	[3, 3]	[]	Predict
S_{34}	$Adj \rightarrow \cdot baking$	[3, 3]	[]	do not scan, S[3] = potatoes $\neq$ *baking*
S_{35}	$PRP \rightarrow \cdot they$	[3, 3]	[]	do not Scan,S[3] = potatoes $\neq$ *they*
S_{36}	$N \rightarrow \cdot potatoes$	[3, 3]	[]	Scan

Chart[4] they are baking potatoes·

StateID	dotted-rule	position	back pointers	operation
S_{37}	$N \rightarrow$ potatoes ·	[3, 4]	[]	Complete
S_{38}	$NP \rightarrow$ N ·	[3, 4]	[S37]	Complete
S_{39}	$VP \rightarrow$ Aux V NP ·	[1, 4]	[S_{16}, S_{27}, S_{38}]	Complete
S_{40}	$NP \rightarrow$ Adj NP ·	[2, 4]	[S_{28}, S_{38}]	Complete
S_{41}	$S \rightarrow$ NP VP ·	[0, 4]	[S_{39}]	Complete
S_{42}	$VP \rightarrow$ V NP ·	[1, 4]	[S_{15}, S_{40}]	Complete
S_{43}	$S \rightarrow$ NP VP ·	[0, 4]	[S_8, S_{42}]	Complete

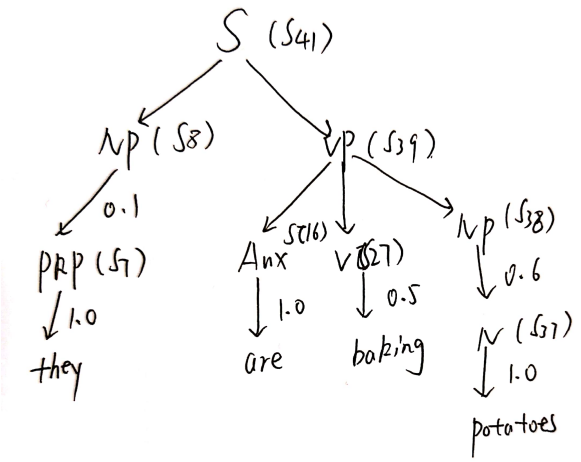
2. Write down all parsing trees

Tree 1:



The probability for PCFG:
1.0 * 0.1 * 1.0 * 0.8 * 0.5 * 0.3 * 1.0 * 0.6 * 1.0 = 0.0072

Tree 2:



The probability for PCFG:
1.0 * 0.1 * 1.0 * 0.2 * 1.0 * 0.5 * 0.6 * 1.0 = 0.006

Problem 3 CKY parsing

1. Convert to CNF

   Converted rules are as follows:
   $S \rightarrow$ NP VP
   $NP \rightarrow$ Adj NP
   $VP \rightarrow$ V NP
   $VP \rightarrow$ Aux VP
   $Adj \rightarrow$ baking
   $V \rightarrow$ baking
   $V \rightarrow$ are
   $Aux \rightarrow$ are
   $NP \rightarrow$ they
   $NP \rightarrow$ potatoes

   (a) For Rules of the form $A \rightarrow B$, we should replace rhs with other rules until the rhs become one terminal or two non terminals. For example, here I combined $NP \rightarrow PRP$ and $PRP \rightarrow they$, I got $NP \rightarrow$ they.

   (b) For rules with three or more nonterminals on the right hand side, we could recursively combine right hand side non terminals into single terminals until there are only two non terminals left. For example, here I utilized $VP \rightarrow Aux\ V\ NP$ and $VP \rightarrow V\ NP$, I got $VP \rightarrow Aux$.

2. CKY Parsing

   CKY Parsing chart:

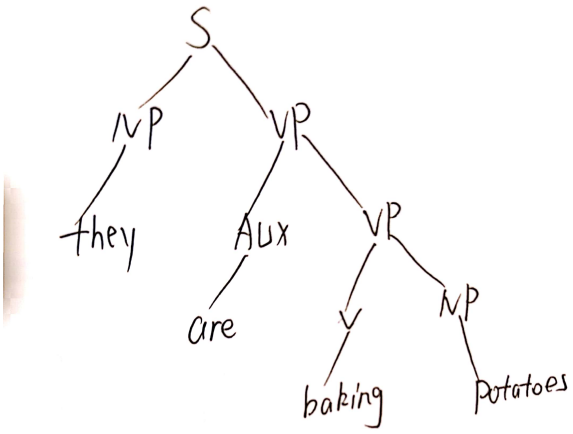
0	1(they)	2(are)	3(baking)	4(potatoes)
0	NP			S
1		V, Aux		VP, VP
2			Adj, V	NP, NP
				NP

   the back pointers:
   when length = 2:
   $NP[2,4] \rightarrow Adj[2,3]\ NP[3,4]$
   $VP[2,4] \rightarrow V[2,3]\ NP[3,4]$

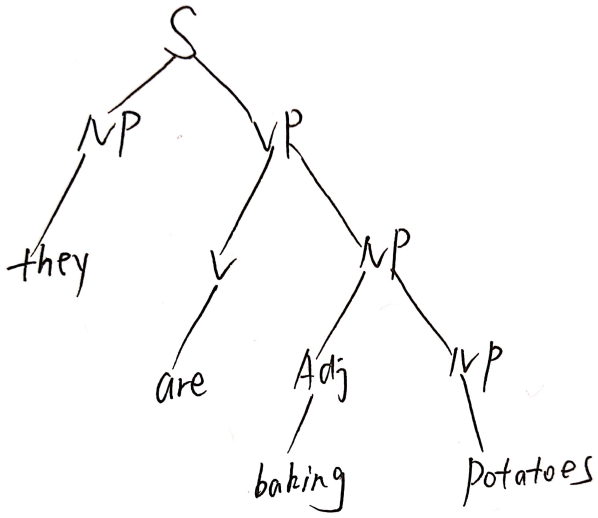
   when length = 3:
   $VP[1,4] \rightarrow V[1,2]\ NP[2,4]$
   $VP[1,4] \rightarrow Aux[1,2]\ VP[2,4]$

when length = 4:
$S[0,4] \rightarrow NP[0,1]\ VP[1,4]$

Based on these above information, we can got two parsing tree:
Parse tree 1:



Parse tree 2:



Problem 4 Transition Based Dependency Parsing

Arc-standard dependency parser:

Initial State:
$([root]_\delta, [he, sent, her, a, funny, meme, today]_\beta, \{\}_A)$

Transition 1: Shift
$([he, root]_\delta, [sent, her, a, funny, meme, today]_\beta, \{\}_A)$

Transition 2: Left-Arc
$([root]_\delta, [sent, her, a, funny, meme, today]_\beta, \{(sent, nsubj, he)\}_A)$

Transition 3: Shift
$([sent, root]_\delta, [her, a, funny, meme, today]_\beta, \{(sent, nsubj, he)\}_A)$

Transition 4: Right-Arc
$([root]_\delta, [sent, a, funny, meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her)\}_A)$

Transition 5: Shift
$([sent, root]_\delta, [a, funny, meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her)\}_A)$

Transition 6: Shift
$([a, sent, root]_\delta, [funny, meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her)\}_A)$

Transition 7: Shift
$([funny, a, sent, root]_\delta, [meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her)\}_A)$

Transition 8: Left-Arc
$([a, sent, root]_\delta, [meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny)\}_A)$

Transition 9: Left-Arc
$([sent, root]_\delta, [meme, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a)\}_A)$

Transition 10: Right-Arc

$([root]_\delta, [sent, today]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a), (sent, dobj, meme)\}_A)$

Transition 11: Shift
$([sent, root]_\delta, [today]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a), (sent, dobj, meme)\}_A)$

Transition 12: Right-Arc
$([root]_\delta, [sent]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a), (sent, dobj, meme), (send, advmod, today)\}_A)$

Transition 13: Right-Arc
$([]_\delta, [root]_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a), (sent, dobj, meme), (send, advmod, today), (root, pred, sent)\}_A)$

Transition 14: Right-Arc
$([root]_\delta, []_\beta, \{(sent, nsubj, he), (sent, jobj, her), (meme, amod, funny), (meme, det, a), (sent, dobj, meme), (send, advmod, today), (root, pred, sent)\}_A)$

Programming part

HW2_NLP

March 3, 2020

1 Programming Part

1.1 Part 1 PCFGs and HMMs

[21]:
```
import sys
from collections import defaultdict
from math import fsum
```

[22]:
```
class Pcfg(object):
    """
    Represent a probabilistic context free grammar.
    """

    def __init__(self, grammar_file):
        self.rhs_to_rules = defaultdict(list)
        self.lhs_to_rules = defaultdict(list)
        self.startsymbol = None
        self.read_rules(grammar_file)

    def read_rules(self,grammar_file):

        for line in grammar_file:
            line = line.strip()
            if line and not line.startswith("#"):
                if "->" in line:
                    rule = self.parse_rule(line.strip())
                    lhs, rhs, prob = rule
                    self.rhs_to_rules[rhs].append(rule)
                    self.lhs_to_rules[lhs].append(rule)
                else:
                    startsymbol, prob = line.rsplit(";")
                    self.startsymbol = startsymbol.strip()


    def parse_rule(self,rule_s):
        lhs, other = rule_s.split("->")
        lhs = lhs.strip()
        rhs_s, prob_s = other.rsplit(";",1)
```

```
        prob = float(prob_s)
        rhs = tuple(rhs_s.strip().split())
        return (lhs, rhs, prob)

    def verify_grammar(self):
        """
        Return True if the grammar is a valid PCFG in CNF.
        Otherwise return False.
        """
        # TODO, Part 1
        # value is a list

        for key, value in self.lhs_to_rules.items():
            temp_list = []
            for var in value:
                temp_list.append(var[2])
            if abs(1 - math.fsum(temp_list)) > 0.00001:
                print("invalid PCFG in CNF")
                return False
        return True


'''
if __name__ == "__main__":

    with open(sys.argv[1],'r') as grammar_file:
        grammar = Pcfg(grammar_file)

    with open('./atis3.pcfg','r') as grammar_file:
        grammar = Pcfg(grammar_file)
        print(grammar.verify_grammar())
'''
```

[22]:
```
'\nif __name__ == "__main__":\n    \n    with open(sys.argv[1],\'r\') as
grammar_file:\n        grammar = Pcfg(grammar_file)\n    \n    with
open(\'./atis3.pcfg\',\'r\') as grammar_file:\n        grammar =
Pcfg(grammar_file)\n        print(grammar.verify_grammar())\n'
```

1.1.1 test for Part 1 reading the grammar and getting started

[23]:
```
with open('./atis3.pcfg','r') as grammar_file:
        grammar = Pcfg(grammar_file)
        print(grammar.verify_grammar())
```

```
True
```

1.2 Part 2-5

[9]:
```
import math
import sys
from collections import defaultdict
import itertools
from grammar import Pcfg
import numpy as np
```

[10]:
```
### Use the following two functions to check the format of your data structures␣
↪in part 3 ###
def check_table_format(table):
    """
    Return true if the backpointer table object is formatted correctly.
    Otherwise return False and print an error.
    """
    if not isinstance(table, dict):
        sys.stderr.write("Backpointer table is not a dict.\n")
        return False
    for split in table:
        if not isinstance(split, tuple) and len(split) ==2 and \
          isinstance(split[0], int)  and isinstance(split[1], int):
            sys.stderr.write("Keys of the backpointer table must be tuples␣
↪(i,j) representing spans.\n")
            return False
        if not isinstance(table[split], dict):
            sys.stderr.write("Value of backpointer table (for each span) is not␣
↪a dict.\n")
            return False
        for nt in table[split]:
            if not isinstance(nt, str):
                sys.stderr.write("Keys of the inner dictionary (for each span)␣
↪must be strings representing nonterminals.\n")
                return False
            bps = table[split][nt]
            if isinstance(bps, str): # Leaf nodes may be strings
                continue
            if not isinstance(bps, tuple):
                sys.stderr.write("Values of the inner dictionary (for each span␣
↪and nonterminal) must be a pair ((i,k,A),(k,j,B)) of backpointers. Incorrect␣
↪type: {}\n".format(bps))
                return False
            if len(bps) != 2:
                sys.stderr.write("Values of the inner dictionary (for each span␣
↪and nonterminal) must be a pair ((i,k,A),(k,j,B)) of backpointers. Found␣
↪more than two backpointers: {}\n".format(bps))
                return False
```

```
            for bp in bps:
                if not isinstance(bp, tuple) or len(bp)!=3:
                    sys.stderr.write("Values of the inner dictionary (for each␣
↪span and nonterminal) must be a pair ((i,k,A),(k,j,B)) of backpointers.␣
↪Backpointer has length != 3.\n".format(bp))
                    return False
                if not (isinstance(bp[0], str) and isinstance(bp[1], int) and␣
↪isinstance(bp[2], int)):
                    print(bp)
                    sys.stderr.write("Values of the inner dictionary (for each␣
↪span and nonterminal) must be a pair ((i,k,A),(k,j,B)) of backpointers.␣
↪Backpointer has incorrect type.\n".format(bp))
                    return False
    return True

def check_probs_format(table):
    """
    Return true if the probability table object is formatted correctly.
    Otherwise return False and print an error.
    """
    if not isinstance(table, dict):
        sys.stderr.write("Probability table is not a dict.\n")
        return False
    for split in table:
        if not isinstance(split, tuple) and len(split) ==2 and␣
↪isinstance(split[0], int) and isinstance(split[1], int):
            sys.stderr.write("Keys of the probability must be tuples (i,j)␣
↪representing spans.\n")
            return False
        if not isinstance(table[split], dict):
            sys.stderr.write("Value of probability table (for each span) is not␣
↪a dict.\n")
            return False
        for nt in table[split]:
            if not isinstance(nt, str):
                sys.stderr.write("Keys of the inner dictionary (for each span)␣
↪must be strings representing nonterminals.\n")
                return False
            prob = table[split][nt]
            if not isinstance(prob, float):
                sys.stderr.write("Values of the inner dictionary (for each span␣
↪and nonterminal) must be a float.{}\n".format(prob))
                return False
            if prob > 0:
                sys.stderr.write("Log probability may not be > 0.  {}\n".
↪format(prob))
```

```
                return False
    return True



class CkyParser(object):
    """
    A CKY parser.
    """

    def __init__(self, grammar):
        """
        Initialize a new parser instance from a grammar.
        """
        self.grammar = grammar

    def is_in_language(self,tokens):
        """
        Membership checking. Parse the input tokens and return True if
        the sentence is in the language described by the grammar. Otherwise
        return False
        """
        # TODO, part 2
        # CKY for CFG

        table= None
        n = len(tokens)

        # content: possible nontermial ends in the specified position
        # eg{(i,j): {'NP'}}
        table = defaultdict(list)
        '''
        table = [None]*n
        for i in range(n):
            table[i] = [list()]*(n+1)
        '''

        # initialization
        for i in range(0,n):
            token = tokens[i]
            temp_list = []
            for lhs in self.grammar.rhs_to_rules[tuple([token])]:
                temp_list.append(lhs[0])
            table[(i,i+1)] = temp_list


        # CKY parsing for CFG
```

```
        for length in range(2, n+1):
            for i in range(0, n-length+1):
                j = i + length
                for k in range(i+1, j):
                        temp_B = table[(i,k)]
                        temp_C = table[(k,j)]
                        temp_tuples_list = []
                        for temp1 in temp_B:
                            for temp2 in temp_C:
                                temp_tuples_list.append(tuple([temp1,temp2]))

                        for temp_tuple in temp_tuples_list:
                            if len(self.grammar.rhs_to_rules[temp_tuple]) > 0:
                                for lhs in self.grammar.
↪rhs_to_rules[temp_tuple]:
                                    if lhs[0] not in table[(i,j)]:
                                        table[(i,j)].append(lhs[0])

        # check if true
        if self.grammar.startsymbol in table[(0,n)]:
            return True

        return False


    def parse_with_backpointers(self, tokens):
        """
        Parse the input tokens and return a parse table and a probability table.
        """
        # TODO, part 3
        table= None
        probs = None

        # content: tuple of split array
        table = defaultdict(defaultdict)
        # content: the prob
        probs = defaultdict(lambda: defaultdict(float))
        n = len(tokens)


        # initialization
        for i in range(0,n):
            token = tokens[i]
            for lhs in self.grammar.rhs_to_rules[tuple([token])]:
                probs[(i,i+1)][lhs[0]] = math.log(lhs[2], 2)
                table[(i,i+1)][lhs[0]] = token
```

```
        # parsing
        for length in range(2, n+1):
            for i in range(0, n-length+1):
                j = i + length
                for k in range(i+1, j):
                    temp_B_dict = table[(i,k)]
                    temp_C_dict = table[(k,j)]
                    temp_B = temp_B_dict.keys();
                    temp_C = temp_C_dict.keys();
                    temp_tuples_list = []
                    for temp1 in temp_B:
                        for temp2 in temp_C:
                            temp_tuples_list.append(tuple([temp1,temp2]))
                    # possible LHS
                    for temp_tuple in temp_tuples_list:
                            if len(self.grammar.rhs_to_rules[temp_tuple]) > 0:
                                for lhs in self.grammar.
 ↪rhs_to_rules[temp_tuple]:
                                    # with log: multiply means addition
                                    new_log_prob = math.log(lhs[2], 2) +␣
 ↪probs[(i,k)][temp_tuple[0]] + probs[(k,j)][temp_tuple[1]]
                                    if probs[(i,j)][lhs[0]] != 0.0:
                                        if probs[(i,j)][lhs[0]] < new_log_prob:
                                            probs[(i,j)][lhs[0]] = new_log_prob
                                            table[(i,j)][lhs[0]] =␣
 ↪((temp_tuple[0], i, k), (temp_tuple[1], k, j))
                                    else:
                                        probs[(i,j)][lhs[0]] = new_log_prob
                                        table[(i,j)][lhs[0]] = ((temp_tuple[0],␣
 ↪i, k), (temp_tuple[1], k, j))

        return table, probs


def get_tree(chart, i,j,nt):
    """
    Return the parse-tree rooted in non-terminal nt and covering span i,j.
    """
    # TODO: Part 4
    #Recursively traverse the parse chart to assemble this tree.
    temp_list = []
    temp_list.append(nt)
    # left child
    if type(chart[(i,j)][nt]) is not str:
        for child in chart[(i,j)][nt]:
            temp_list.append(get_tree(chart, child[1], child[2], child[0]))
```

```
    else:
        temp_list.append(chart[(i,j)][nt])
    return tuple(temp_list)
'''
if __name__ == "__main__":

    with open('atis3.pcfg','r') as grammar_file:
        grammar = Pcfg(grammar_file)
        parser = CkyParser(grammar)
        toks =['flights', 'from','miami', 'to', 'cleveland','.']
        #print(parser.is_in_language(toks))
        #table,probs = parser.parse_with_backpointers(toks)
        #assert check_table_format(chart)
        #assert check_probs_format(probs)
'''
```

[10]:
```
'\nif __name__ == "__main__":\n    \n    with open(\'atis3.pcfg\',\'r\') as
grammar_file: \n        grammar = Pcfg(grammar_file) \n        parser =
CkyParser(grammar)\n        toks =[\'flights\', \'from\',\'miami\', \'to\',
\'cleveland\',\'.\'] \n        #print(parser.is_in_language(toks))\n
#table,probs = parser.parse_with_backpointers(toks)\n        #assert
check_table_format(chart)\n        #assert check_probs_format(probs)\n'
```

1.2.1 test for Part 2 Membership checking with CKY

[11]:
```
with open('atis3.pcfg','r') as grammar_file:
    grammar = Pcfg(grammar_file)
    parser = CkyParser(grammar)
    toks =['flights', 'from','miami', 'to', 'cleveland','.']
```

[12]:
```
parser.is_in_language(toks)
```

[12]:
```
True
```

[13]:
```
toks =['miami', 'flights','cleveland', 'from', 'to','.']
parser.is_in_language(toks)
```

[13]:
```
False
```

1.2.2 test for part 3

[14]:
```
# test for part 3
with open('atis3.pcfg','r') as grammar_file:
    grammar = Pcfg(grammar_file)
    parser = CkyParser(grammar)
    toks =['flights', 'from', 'miami', 'to', 'cleveland', '.']
```

```
    table, probs = parser.parse_with_backpointers(toks)
```

[507]:
```
table[(0, len(toks))]['TOP']
```

[507]:
```
(('NP', 0, 5), ('PUN', 5, 6))
```

[15]:
```
check_table_format(table)
```

[15]:
```
True
```

[16]:
```
check_probs_format(probs)
```

[16]:
```
True
```

[ ]:
```
# test for generate tree
```

1.2.3 test for Part 4 Retrieving a parse tree

[17]:
```
with open('atis3.pcfg','r') as grammar_file:
    grammar = Pcfg(grammar_file)
    parser = CkyParser(grammar)
    toks =['flights', 'from', 'miami', 'to', 'cleveland', '.']
    table, probs = parser.parse_with_backpointers(toks)
    tree = get_tree(table, 0, len(toks), grammar.startsymbol)
```

[18]:
```
tree
```

[18]:
```
('TOP',
 ('NP',
  ('NP', 'flights'),
  ('NPBAR',
   ('PP', ('FROM', 'from'), ('NP', 'miami')),
   ('PP', ('TO', 'to'), ('NP', 'cleveland')))),
 ('PUN', '.'))
```

[ ]:
```
# evaluate_parser.py
```

1.3 Part 5 Evaluating the Parser

[19]:
```
from cky import Pcfg, CkyParser, get_tree
import sys
```

[20]:
```
from cky import Pcfg, CkyParser, get_tree
import sys

def tokenize(line):
```

```
    tok = ''
    for c in line:
        if c == " ":
            if tok:
                yield tok
                tok = ""
        elif c == "(" or c==")":
            if tok:
                yield tok
            yield c
            tok = ""
        else:
            tok += c
    if tok:
        yield tok
        tok = ""

def parse_tree(line):
    toks = tokenize(line)
    stack = []
    t = next(toks)
    try:
        while t:
            if t=="(":
                stack.append(t)
            elif t==")":
                subtree = []
                s = stack.pop()
                while s[0]!="(":
                    subtree.append(s)
                    s = stack.pop()
                stack.append(tuple(reversed(subtree)))
            else:
                stack.append(t)
            t = next(toks)
    except StopIteration:
        return stack.pop()


def get_leafs(tree):
    if isinstance(tree,str):
        return [tree]
    else:
        result = []
        for x in tree[1:]:
            result.extend(get_leafs(x))
        return result
```

```
def get_constituents(tree,left=0):
    if not tree:
        return [], left
    start = left
    if isinstance(tree,str):
        return [],left+1
    else:
        result = []
        phrase = tree[0]
        for subtree in tree[1:]:
            subspans, right = get_constituents(subtree, left)
            result.extend(subspans)
            left = right
        result.append((phrase,start,left))
        return result, left

def compute_parseval_scores(gold_tree, test_tree):

    gold_const = set(get_constituents(gold_tree)[0])
    test_const = set(get_constituents(test_tree)[0])

    if not test_const:
        return 0.0,0.0,0.0

    correct = len(gold_const.intersection(test_const))
    recall = correct / float(len(gold_const))
    precision = correct / float(len(test_const))
    fscore = (2*precision*recall) / (precision+recall)
    return precision, recall, fscore

def evaluate_parser(parser, treebank_file):

    total = 0
    unparsed = 0
    fscore_sum = 0.0
    for line in treebank_file:
        gold_tree = parse_tree(line.strip())
        tokens = get_leafs(gold_tree)
        print("input: ",tokens)
        chart,probs = parser.parse_with_backpointers(tokens)
        print("target:    ",gold_tree)
        total += 1
        if not chart:
            unparsed += 1
            res = tuple()
```

```
        else:
            try:
                res = get_tree(chart,0,len(tokens),parser.grammar.startsymbol)
            except KeyError:
                unparsed += 1
                res = tuple()
        print("predicted: ",res)
        #print(compute_parseval_scores(gold_tree, res))
        p,r,f = compute_parseval_scores(gold_tree, res)
        fscore_sum += f
        print("P:{} R:{} F:{}".format(p,r,f))
        print()

    parsed = total-unparsed
    if parsed == 0:
        coverage = 0.0
        fscore_parsed = 0.0
        fscore_all = 0.0
    else:
        coverage =  (parsed / total) *100
        fscore_parsed = fscore_sum / parsed
        fscore_all = fscore_sum / total
    print("Coverage: {:.2f}%, Average F-score (parsed sentences): {}, Average␣
↪F-score (all sentences): {}".format(coverage, fscore_parsed, fscore_all))

'''
if __name__ == "__main__":

    if len(sys.argv)!=3:
        print("USAGE: python evaluate_parser.py [grammar_file] [test_file]")
        sys.exit(1)

    with open(sys.argv[1],'r') as grammar_file, open(sys.argv[2],'r') as␣
↪test_file:
        grammar = Pcfg(grammar_file)
        parser = CkyParser(grammar)
        evaluate_parser(parser,test_file)
'''
```

[20]:
```
'\nif __name__ == "__main__":\n\n    if len(sys.argv)!=3:\n        print("USAGE:
python evaluate_parser.py [grammar_file] [test_file]")\n        sys.exit(1)\n\n
with open(sys.argv[1],\'r\') as grammar_file, open(sys.argv[2],\'r\') as
test_file: \n        grammar = Pcfg(grammar_file) \n        parser =
CkyParser(grammar)\n        evaluate_parser(parser,test_file)\n'
```

[15]:
```
with open('./atis3.pcfg','r') as grammar_file, open('./atis3_test.ptb','r') as␣
↪test_file:
```

```
        grammar = Pcfg(grammar_file)
        parser = CkyParser(grammar)
        evaluate_parser(parser,test_file)
```

```
input:  ['flights', 'from', 'los', 'angeles', 'to', 'pittsburgh', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', ('LOS', 'los'), ('ANGELES', 'angeles'))), ('PP', ('TO', 'to'), ('NP',
'pittsburgh')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', ('LOS', 'los'), ('ANGELES', 'angeles'))), ('PP', ('TO', 'to'), ('NP',
'pittsburgh')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['with', 'the', 'least', 'expensive', 'fare', '.']
target:     ('TOP', ('PP', ('WITH', 'with'), ('NP', ('THE', 'the'), ('NPBAR',
('ADJP', ('LEAST', 'least'), ('EXPENSIVE', 'expensive')), ('FARE', 'fare')))),
('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['flights', 'between', 'tampa', 'and', 'saint', 'louis', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('PP', ('BETWEEN', 'between'),
('NP', ('NP', 'tampa'), ('NPBAR', ('AND', 'and'), ('NP', ('SAINT', 'saint'),
('LOUIS', 'louis')))))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('BETWEEN',
'between'), ('NP', 'tampa')), ('NPBAR', ('AND', 'and'), ('NP', ('SAINT',
'saint'), ('LOUIS', 'louis'))))), ('PUN', '.'))
P:0.8461538461538461 R:0.8461538461538461 F:0.8461538461538461

input:  ['i', "'d", 'like', 'a', 'flight', 'tomorrow', 'from', 'columbus', 'to',
'houston', 'with', 'a', 'stopover', 'in', 'nashville', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ("'D", "'d"), ('VP', ('LIKE',
'like'), ('NP', ('NP', ('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('NP',
'tomorrow'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'columbus')), ('NPBAR',
('PP', ('TO', 'to'), ('NP', 'houston')), ('PP', ('WITH', 'with'), ('NP', ('NP',
('A', 'a'), ('STOPOVER', 'stopover')), ('PP', ('IN', 'in'), ('NP',
'nashville'))))))))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['display', 'the', 'fare', 'codes', '.']
target:     ('TOP', ('VP', ('DISPLAY', 'display'), ('NP', ('THE', 'the'),
('NPBAR', ('FARE', 'fare'), ('CODES', 'codes')))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['what', 'flights', 'from', 'kansas', 'city', 'to', 'denver', '.']
```

```
target:     ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', ('KANSAS', 'kansas'), ('CITY',
'city'))), ('PP', ('TO', 'to'), ('NP', 'denver')))), ('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', ('KANSAS', 'kansas'), ('CITY',
'city'))), ('PP', ('TO', 'to'), ('NP', 'denver')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'flights', 'from', 'minneapolis', 'to', 'pittsburgh', '.']
target:     ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'minneapolis')), ('PP', ('TO',
'to'), ('NP', 'pittsburgh')))), ('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'minneapolis')), ('PP', ('TO',
'to'), ('NP', 'pittsburgh')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'flights', 'from', 'tampa', 'to', 'cincinnati', '.']
target:     ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'tampa')), ('PP', ('TO', 'to'),
('NP', 'cincinnati')))), ('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'tampa')), ('PP', ('TO', 'to'),
('NP', 'cincinnati')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['which', 'of', 'these', 'leave', 'after', 'noon', 'and', 'stop', 'in',
'phoenix', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHNP', 'which'), ('PP', ('OF', 'of'),
('NP', 'these'))), ('VP', ('VP', ('LEAVE', 'leave'), ('PP', ('AFTER', 'after'),
('NP', 'noon'))), ('VPBAR', ('AND', 'and'), ('VP', ('STOP', 'stop'), ('PP',
('IN', 'in'), ('NP', 'phoenix')))))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', ('WHNP', 'which'), ('PP', ('OF', 'of'),
('NP', 'these'))), ('VP', ('LEAVE', 'leave'), ('VPBAR', ('PP', ('AFTER',
'after'), ('NP', 'noon')), ('VPBAR', ('AND', 'and'), ('VP', ('STOP', 'stop'),
('PP', ('IN', 'in'), ('NP', 'phoenix')))))), ('PUN', '.'))
P:0.9523809523809523 R:0.9523809523809523 F:0.9523809523809523

input:  ['flights', 'from', 'boston', 'to', 'pittsburgh', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'boston')), ('PP', ('TO', 'to'), ('NP', 'pittsburgh')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'boston')), ('PP', ('TO', 'to'), ('NP', 'pittsburgh')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['i', 'prefer', 'a', 'morning', 'flight', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ('PREFER', 'prefer'), ('NP', ('A',
'a'), ('NPBAR', ('MORNING', 'morning'), ('FLIGHT', 'flight'))))), ('PUN', '.'))
```

```
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['show', 'me', 'the', 'first', 'flight', 'that', 'arrives', 'in',
'toronto', 'from', 'cincinnati', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('NPBAR', ('FIRST', 'first'), ('FLIGHT', 'flight'))),
('NPBAR', ('SBAR', ('WHNP', 'that'), ('VP', ('ARRIVES', 'arrives'), ('PP',
('IN', 'in'), ('NP', 'toronto')))), ('PP', ('FROM', 'from'), ('NP',
'cincinnati')))))), ('PUN', '.'))
predicted:  ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('THE', 'the'), ('NPBAR', ('ADVP', 'first'), ('NP', ('NP', 'flight'), ('SBAR',
('WHNP', 'that'), ('VP', ('ARRIVES', 'arrives'), ('VPBAR', ('PP', ('IN', 'in'),
('NP', 'toronto')), ('PP', ('FROM', 'from'), ('NP', 'cincinnati'))))))))),
('PUN', '.'))
P:0.6956521739130435 R:0.6956521739130435 F:0.6956521739130435

input:  ['which', 'of', 'these', 'is', 'last', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHNP', 'which'), ('PP', ('OF', 'of'),
('NP', 'these'))), ('VP', ('IS', 'is'), ('ADJP', 'last'))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', ('WHNP', 'which'), ('PP', ('OF', 'of'),
('NP', 'these'))), ('VP', ('IS', 'is'), ('ADJP', 'last'))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['i', "'d", 'like', 'a', 'flight', 'tomorrow', 'from', 'san', 'diego',
'to', 'toronto', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ("'D", "'d"), ('VP', ('LIKE',
'like'), ('NP', ('NP', ('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('NP',
'tomorrow'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', ('SAN', 'san'), ('DIEGO',
'diego'))), ('PP', ('TO', 'to'), ('NP', 'toronto')))))))), ('PUN', '.'))
predicted:  ('TOP', ('S', ('NP', 'i'), ('VP', ("'D", "'d"), ('VP', ('LIKE',
'like'), ('NP', ('NP', ('A', 'a'), ('FLIGHT', 'flight')), ('NP', ('NP',
'tomorrow'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', ('SAN', 'san'), ('DIEGO',
'diego'))), ('PP', ('TO', 'to'), ('NP', 'toronto')))))))), ('PUN', '.'))
P:0.9565217391304348 R:0.9565217391304348 F:0.9565217391304348

input:  ['which', 'of', 'those', 'leave', 'before', 'eight', 'a.m', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHNP', 'which'), ('PP', ('OF', 'of'),
('NP', 'those'))), ('VP', ('LEAVE', 'leave'), ('PP', ('BEFORE', 'before'),
('NP', ('EIGHT', 'eight'), ('A.M', 'a.m'))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['what', 'airlines', 'from', 'washington', 'd', 'c', 'to', 'columbus',
'.']
target:     ('TOP', ('FRAG', ('WHNP', ('WHAT', 'what'), ('AIRLINES',
'airlines')), ('FRAGBAR', ('PP', ('FROM', 'from'), ('NP', ('NP', 'washington'),
('NP', ('D', 'd'), ('C', 'c')))), ('PP', ('TO', 'to'), ('NP', 'columbus')))),
```

```
('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('AIRLINES',
'airlines')), ('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', ('NP', 'washington'),
('NP', ('D', 'd'), ('C', 'c')))), ('PP', ('TO', 'to'), ('NP', 'columbus')))),
('PUN', '.'))
P:0.8823529411764706 R:0.8823529411764706 F:0.8823529411764706

input:  ['what', 'flights', 'from', 'chicago', 'to', 'kansas', 'city', 'in',
'the', 'morning', '.']
target:     ('TOP', ('WHNP', ('NP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'chicago')), ('WHNPBAR', ('PP',
('TO', 'to'), ('NP', ('KANSAS', 'kansas'), ('CITY', 'city'))), ('PP', ('IN',
'in'), ('NP', ('THE', 'the'), ('MORNING', 'morning')))))), ('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', 'chicago')), ('WHNPBAR', ('PP',
('TO', 'to'), ('NP', ('KANSAS', 'kansas'), ('CITY', 'city'))), ('PP', ('IN',
'in'), ('NP', ('THE', 'the'), ('MORNING', 'morning')))))), ('PUN', '.'))
P:0.9523809523809523 R:0.9523809523809523 F:0.9523809523809523

input:  ['what', 'type', 'of', 'aircraft', 'is', 'used', 'on', 'those',
'flights', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHNP', ('WHAT', 'what'), ('TYPE',
'type')), ('PP', ('OF', 'of'), ('NP', 'aircraft'))), ('VP', ('IS', 'is'), ('VP',
('USED', 'used'), ('PP', ('ON', 'on'), ('NP', ('THOSE', 'those'), ('FLIGHTS',
'flights')))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['price', 'of', 'flight', 'a', 'a', 'one', 'thousand', 'three',
'hundred', 'nineteen', '.']
target:     ('TOP', ('NP', ('NP', 'price'), ('PP', ('OF', 'of'), ('NP',
('FLIGHT', 'flight'), ('NPBAR', ('A', 'a'), ('NPBAR', ('A', 'a'), ('NPBAR',
('ONE', 'one'), ('NPBAR', ('THOUSAND', 'thousand'), ('NPBAR', ('THREE',
'three'), ('NPBAR', ('HUNDRED', 'hundred'), ('NINETEEN', 'nineteen'))))))))),
('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['show', 'me', 'the', 'ground', 'transportation', 'available', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('NPBAR', ('GROUND', 'ground'), ('TRANSPORTATION',
'transportation'))), ('ADJP', 'available')))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['which', 'is', 'the', 'latest', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'which'), ('VP', ('IS', 'is'), ('NP',
('THE', 'the'), ('LATEST', 'latest')))), ('PUN', '.'))
```

```
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['what', 'about', 'after', 'seven', 'p.m', '.']
target:     ('TOP', ('FRAG', ('X', ('WHAT', 'what'), ('ABOUT', 'about')), ('PP',
('AFTER', 'after'), ('NP', ('SEVEN', 'seven'), ('P.M', 'p.m')))), ('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('X', ('WHAT', 'what'), ('ABOUT', 'about')), ('PP',
('AFTER', 'after'), ('NP', ('SEVEN', 'seven'), ('P.M', 'p.m')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'flights', 'from', 'salt', 'lake', 'city', 'to', 'las',
'vegas', '.']
target:     ('TOP', ('FRAG', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS',
'flights')), ('PP', ('FROM', 'from'), ('NP', ('SALT', 'salt'), ('NPBAR',
('LAKE', 'lake'), ('CITY', 'city'))))), ('PP', ('TO', 'to'), ('NP', ('LAS',
'las'), ('VEGAS', 'vegas')))), ('PUN', '.'))
predicted:  ('TOP', ('WHNP', ('WHNP', ('WHAT', 'what'), ('FLIGHTS', 'flights')),
('WHNPBAR', ('PP', ('FROM', 'from'), ('NP', ('SALT', 'salt'), ('NPBAR', ('LAKE',
'lake'), ('CITY', 'city')))), ('PP', ('TO', 'to'), ('NP', ('LAS', 'las'),
('VEGAS', 'vegas'))))), ('PUN', '.'))
P:0.8947368421052632 R:0.8947368421052632 F:0.8947368421052632

input:  ['flights', 'after', 'twelve', 'hundred', 'hours', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('PP', ('AFTER', 'after'), ('NP',
('TWELVE', 'twelve'), ('NPBAR', ('HUNDRED', 'hundred'), ('HOURS', 'hours'))))),
('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('NP', 'flights'), ('PP', ('AFTER', 'after'),
('NP', ('TWELVE', 'twelve'), ('NPBAR', ('HUNDRED', 'hundred'), ('HOURS',
'hours'))))), ('PUN', '.'))
P:0.9090909090909091 R:0.9090909090909091 F:0.9090909090909091

input:  ['what', 'is', 'the', 'price', 'of', 'united', 'airlines', 'flight',
'nine', 'seven', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('NP', ('THE', 'the'), ('PRICE', 'price')), ('PP', ('OF', 'of'), ('NP',
('UNITED', 'united'), ('NPBAR', ('AIRLINES', 'airlines'), ('NPBAR', ('FLIGHT',
'flight'), ('NPBAR', ('NINE', 'nine'), ('SEVEN', 'seven'))))))))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('NP', ('THE', 'the'), ('PRICE', 'price')), ('NPBAR', ('PP', ('OF', 'of'),
('NP', ('UNITED', 'united'), ('AIRLINES', 'airlines'))), ('NPBAR', ('FLIGHT',
'flight'), ('NPBAR', ('NINE', 'nine'), ('SEVEN', 'seven')))))))), ('PUN', '.'))
P:0.8571428571428571 R:0.8571428571428571 F:0.8571428571428571

input:  ['cheapest', 'airfare', 'from', 'orlando', 'to', 'tacoma', '.']
target:     ('TOP', ('NP', ('NP', ('CHEAPEST', 'cheapest'), ('AIRFARE',
'airfare')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'orlando')), ('PP',
('TO', 'to'), ('NP', 'tacoma')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', ('CHEAPEST', 'cheapest'), ('AIRFARE',
```

```
'airfare')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'orlando')), ('PP',
('TO', 'to'), ('NP', 'tacoma')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['price', 'of', 'flight', 'from', 'cleveland', 'to', 'nashville', '.']
target:     ('TOP', ('NP', ('NP', 'price'), ('PP', ('OF', 'of'), ('NP', ('NP',
'flight'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'cleveland')), ('PP',
('TO', 'to'), ('NP', 'nashville')))))), ('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('NP', 'price'), ('PP', ('OF', 'of'), ('NP',
('FLIGHT', 'flight'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'cleveland')),
('PP', ('TO', 'to'), ('NP', 'nashville')))))), ('PUN', '.'))
P:0.8666666666666667 R:0.8666666666666667 F:0.8666666666666667

input:  ['what', 'is', 'the', 'price', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('THE', 'the'), ('PRICE', 'price')))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('THE', 'the'), ('PRICE', 'price')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'is', 'the', 'price', 'of', 'flights', 'from', 'indianapolis',
'to', 'memphis', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('NP', ('THE', 'the'), ('PRICE', 'price')), ('PP', ('OF', 'of'), ('NP', ('NP',
'flights'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'indianapolis')), ('PP',
('TO', 'to'), ('NP', 'memphis')))))))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('NP', ('THE', 'the'), ('PRICE', 'price')), ('NPBAR', ('PP', ('OF', 'of'),
('NP', 'flights')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', 'indianapolis')),
('PP', ('TO', 'to'), ('NP', 'memphis')))))), ('PUN', '.'))
P:0.9047619047619048 R:0.9047619047619048 F:0.9047619047619048

input:  ['show', 'me', 'the', 'flights', 'from', 'newark', 'new', 'jersey',
'to', 'ontario', 'international', 'next', 'saturday', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('FLIGHTS', 'flights')), ('NPBAR', ('PP', ('FROM',
'from'), ('NP', ('NP', 'newark'), ('NP', ('NEW', 'new'), ('JERSEY',
'jersey')))), ('NPBAR', ('PP', ('TO', 'to'), ('NP', ('ONTARIO', 'ontario'),
('INTERNATIONAL', 'international'))), ('NP', ('NEXT', 'next'), ('SATURDAY',
'saturday')))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['what', 'flights', 'leave', 'phoenix', 'on', 'wednesday', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHAT', 'what'), ('FLIGHTS',
'flights')), ('VP', ('LEAVE', 'leave'), ('VPBAR', ('NP', 'phoenix'), ('PP',
('ON', 'on'), ('NP', 'wednesday'))))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', ('WHAT', 'what'), ('FLIGHTS',
```

```
'flights')), ('VP', ('LEAVE', 'leave'), ('VPBAR', ('NP', 'phoenix'), ('PP',
('ON', 'on'), ('NP', 'wednesday'))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['show', 'me', 'the', 'meal', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('THE', 'the'), ('MEAL', 'meal')))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['show', 'business', 'class', 'fares', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('NP', ('BUSINESS', 'business'),
('NPBAR', ('CLASS', 'class'), ('FARES', 'fares')))), ('PUN', '.'))
predicted:  ('TOP', ('VP', ('SHOW', 'show'), ('NP', ('BUSINESS', 'business'),
('NPBAR', ('CLASS', 'class'), ('FARES', 'fares')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['i', 'have', 'a', 'friend', 'living', 'in', 'denver', 'that', 'would',
'like', 'to', 'visit', 'me', 'here', 'in', 'washington', 'd', 'c', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ('HAVE', 'have'), ('NP', ('NP',
('A', 'a'), ('FRIEND', 'friend')), ('NPBAR', ('VP', ('LIVING', 'living'), ('PP',
('IN', 'in'), ('NP', 'denver'))), ('SBAR', ('WHNP', 'that'), ('VP', ('WOULD',
'would'), ('VP', ('LIKE', 'like'), ('VP', ('TO', 'to'), ('VP', ('VISIT',
'visit'), ('VPBAR', ('NP', 'me'), ('ADVP', ('ADVP', 'here'), ('PP', ('IN',
'in'), ('NP', ('NP', 'washington'), ('NP', ('D', 'd'), ('C', 'c'))))))))))))),
('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['cheapest', '.']
target:     ('TOP', ('ADJP', 'cheapest'), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['list', 'nonstop', 'flights', 'from', 'burbank', 'to', 'denver',
'arriving', 'by', 'six', 'p.m', '.']
target:     ('TOP', ('VP', ('LIST', 'list'), ('NP', ('NP', ('NONSTOP',
'nonstop'), ('FLIGHTS', 'flights')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'burbank')), ('NPBAR', ('PP', ('TO', 'to'), ('NP', 'denver')), ('VP',
('ARRIVING', 'arriving'), ('PP', ('BY', 'by'), ('NP', ('SIX', 'six'), ('P.M',
'p.m')))))))), ('PUN', '.'))
predicted:  ('TOP', ('VP', ('LIST', 'list'), ('NP', ('NP', ('NONSTOP',
'nonstop'), ('FLIGHTS', 'flights')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'burbank')), ('NPBAR', ('PP', ('TO', 'to'), ('NP', 'denver')), ('VP',
('ARRIVING', 'arriving'), ('PP', ('BY', 'by'), ('NP', ('SIX', 'six'), ('P.M',
'p.m')))))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0
```

```
input:  ['shortest', 'morning', 'flights', 'from', 'cincinnati', 'to', 'tampa',
'.']
target:     ('TOP', ('NP', ('NP', ('SHORTEST', 'shortest'), ('NPBAR',
('MORNING', 'morning'), ('FLIGHTS', 'flights'))), ('NPBAR', ('PP', ('FROM',
'from'), ('NP', 'cincinnati')), ('PP', ('TO', 'to'), ('NP', 'tampa')))), ('PUN',
'.'))
predicted:  ('TOP', ('NP', ('NP', ('SHORTEST', 'shortest'), ('NPBAR',
('MORNING', 'morning'), ('FLIGHTS', 'flights'))), ('NPBAR', ('PP', ('FROM',
'from'), ('NP', 'cincinnati')), ('PP', ('TO', 'to'), ('NP', 'tampa')))), ('PUN',
'.'))
P:1.0 R:1.0 F:1.0

input:  ['flights', 'from', 'kansas', 'city', 'to', 'cleveland', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', ('KANSAS', 'kansas'), ('CITY', 'city'))), ('PP', ('TO', 'to'), ('NP',
'cleveland')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', ('KANSAS', 'kansas'), ('CITY', 'city'))), ('PP', ('TO', 'to'), ('NP',
'cleveland')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['from', 'toronto', 'to', 'atlanta', 'in', 'the', 'afternoon', '.']
target:     ('TOP', ('FRAG', ('PP', ('FROM', 'from'), ('NP', 'toronto')),
('FRAGBAR', ('PP', ('TO', 'to'), ('NP', 'atlanta')), ('PP', ('IN', 'in'), ('NP',
('THE', 'the'), ('AFTERNOON', 'afternoon'))))), ('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('PP', ('FROM', 'from'), ('NP', 'toronto')),
('FRAGBAR', ('PP', ('TO', 'to'), ('NP', 'atlanta')), ('PP', ('IN', 'in'), ('NP',
('THE', 'the'), ('AFTERNOON', 'afternoon'))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['now', 'i', 'need', 'a', 'flight', 'on', 'tuesday', 'from', 'phoenix',
'to', 'detroit', '.']
target:     ('TOP', ('S', ('ADVP', 'now'), ('SBAR', ('NP', 'i'), ('VP', ('NEED',
'need'), ('NP', ('NP', ('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('PP',
('ON', 'on'), ('NP', 'tuesday')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'phoenix')), ('PP', ('TO', 'to'), ('NP', 'detroit')))))))), ('PUN', '.'))
predicted:  ('TOP', ('S', ('ADVP', 'now'), ('SBAR', ('NP', 'i'), ('VP', ('NEED',
'need'), ('NP', ('NP', ('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('PP',
('ON', 'on'), ('NP', 'tuesday')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'phoenix')), ('PP', ('TO', 'to'), ('NP', 'detroit')))))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['i', 'need', 'a', 'flight', 'the', 'next', 'day', 'from', 'newark',
'to', 'orlando', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ('NEED', 'need'), ('NP', ('NP',
('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('NP', ('THE', 'the'), ('NPBAR',
('NEXT', 'next'), ('DAY', 'day'))), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'newark')), ('PP', ('TO', 'to'), ('NP', 'orlando')))))))), ('PUN', '.'))
```

```
predicted:  ('TOP', ('S', ('NP', 'i'), ('VP', ('NEED', 'need'), ('NP', ('NP',
('A', 'a'), ('FLIGHT', 'flight')), ('NP', ('NP', ('THE', 'the'), ('NPBAR',
('NEXT', 'next'), ('DAY', 'day'))), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
'newark')), ('PP', ('TO', 'to'), ('NP', 'orlando')))))), ('PUN', '.'))
P:0.9565217391304348 R:0.9565217391304348 F:0.9565217391304348

input:  ['flights', 'from', 'pittsburgh', 'to', 'newark', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'pittsburgh')), ('PP', ('TO', 'to'), ('NP', 'newark')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'pittsburgh')), ('PP', ('TO', 'to'), ('NP', 'newark')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'is', 'airline', 'f', 'f', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('AIRLINE', 'airline'), ('NPBAR', ('F', 'f'), ('F', 'f'))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['julysixteenth', 'please', '.']
target:     ('TOP', ('FRAG', ('NP', 'julysixteenth'), ('INTJ', 'please')),
('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['airports', 'in', 'new', 'york', '.']
target:     ('TOP', ('NP', ('NP', 'airports'), ('PP', ('IN', 'in'), ('NP',
('NEW', 'new'), ('YORK', 'york')))), ('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('NP', 'airports'), ('PP', ('IN', 'in'), ('NP',
('NEW', 'new'), ('YORK', 'york')))), ('PUN', '.'))
P:0.8888888888888888 R:0.8888888888888888 F:0.8888888888888888

input:  ['show', 'me', 'the', 'flights', 'from', 'baltimore', 'to', 'seattle',
'.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('FLIGHTS', 'flights')), ('NPBAR', ('PP', ('FROM',
'from'), ('NP', 'baltimore')), ('PP', ('TO', 'to'), ('NP', 'seattle')))))),
('PUN', '.'))
predicted:  ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('FLIGHTS', 'flights')), ('NPBAR', ('PP', ('FROM',
'from'), ('NP', 'baltimore')), ('PP', ('TO', 'to'), ('NP', 'seattle')))))),
('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['friday', 'afternoon', '.']
target:     ('TOP', ('NP', ('FRIDAY', 'friday'), ('AFTERNOON', 'afternoon')),
('PUN', '.'))
predicted:  ('TOP', ('NP', ('FRIDAY', 'friday'), ('AFTERNOON', 'afternoon')),
```

```
('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['how', 'many', 'stops', 'are', 'there', '.']
target:     ('TOP', ('SBAR', ('WHNP', ('WHADJP', ('HOW', 'how'), ('MANY',
'many')), ('STOPS', 'stops')), ('SQ', ('ARE', 'are'), ('NP', 'there'))), ('PUN',
'.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['flights', 'from', 'pittsburgh', 'to', 'los', 'angeles', 'thursday',
'evening', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'pittsburgh')), ('NPBAR', ('PP', ('TO', 'to'), ('NP', ('LOS', 'los'),
('ANGELES', 'angeles'))), ('NP', ('THURSDAY', 'thursday'), ('EVENING',
'evening'))))), ('PUN', '.'))
predicted:  ('TOP', ('FRAG', ('NP', 'flights'), ('FRAGBAR', ('PP', ('FROM',
'from'), ('NP', 'pittsburgh')), ('FRAGBAR', ('PP', ('TO', 'to'), ('NP', ('LOS',
'los'), ('ANGELES', 'angeles'))), ('NP', ('THURSDAY', 'thursday'), ('EVENING',
'evening'))))), ('PUN', '.'))
P:0.8235294117647058 R:0.8235294117647058 F:0.8235294117647058

input:  ['show', 'me', 'thelatestflight', 'from', 'salt', 'lake', 'city', 'to',
'phoenix', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', 'thelatestflight'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', ('SALT',
'salt'), ('NPBAR', ('LAKE', 'lake'), ('CITY', 'city')))), ('PP', ('TO', 'to'),
('NP', 'phoenix')))))), ('PUN', '.'))
predicted:  ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', 'thelatestflight'), ('NPBAR', ('PP', ('FROM', 'from'), ('NP', ('SALT',
'salt'), ('NPBAR', ('LAKE', 'lake'), ('CITY', 'city')))), ('PP', ('TO', 'to'),
('NP', 'phoenix')))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['of', 'those', 'flights', 'which', 'ones', 'stop', 'in', 'minneapolis',
'.']
target:     ('TOP', ('SBARQ', ('PP', ('OF', 'of'), ('NP', ('THOSE', 'those'),
('FLIGHTS', 'flights'))), ('SBARQBAR', ('WHNP', ('WHICH', 'which'), ('ONES',
'ones')), ('VP', ('STOP', 'stop'), ('PP', ('IN', 'in'), ('NP',
'minneapolis'))))), ('PUN', '.'))
predicted:  ('TOP', ('S', ('PP', ('OF', 'of'), ('NP', ('NP', 'those'),
('FLIGHTS', 'flights'))), ('SBAR', ('WHNP', ('WHICH', 'which'), ('ONES',
'ones')), ('VP', ('STOP', 'stop'), ('PP', ('IN', 'in'), ('NP',
'minneapolis'))))), ('PUN', '.'))
P:0.8235294117647058 R:0.8235294117647058 F:0.8235294117647058

input:  ['tuesday', '.']
target:     ('TOP', ('NP', 'tuesday'), ('PUN', '.'))
```

```
predicted:  ('TOP', ('NP', 'tuesday'), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['flights', 'from', 'miami', 'to', 'cleveland', '.']
target:     ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'miami')), ('PP', ('TO', 'to'), ('NP', 'cleveland')))), ('PUN', '.'))
predicted:  ('TOP', ('NP', ('NP', 'flights'), ('NPBAR', ('PP', ('FROM', 'from'),
('NP', 'miami')), ('PP', ('TO', 'to'), ('NP', 'cleveland')))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

input:  ['what', 'is', 'b', 'n', 'a', '.']
target:     ('TOP', ('SBARQ', ('WHNP', 'what'), ('SQ', ('IS', 'is'), ('NP',
('B', 'b'), ('NPBAR', ('N', 'n'), ('A', 'a'))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['show', 'me', 'the', 'flights', 'that', 'accept', 'frequent', 'flyer',
'tickets', '.']
target:     ('TOP', ('VP', ('SHOW', 'show'), ('VPBAR', ('NP', 'me'), ('NP',
('NP', ('THE', 'the'), ('FLIGHTS', 'flights')), ('SBAR', ('WHNP', 'that'),
('VP', ('ACCEPT', 'accept'), ('NP', ('FREQUENT', 'frequent'), ('NPBAR',
('FLYER', 'flyer'), ('TICKETS', 'tickets')))))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0

input:  ['which', 'flights', 'depart', 'burbank', 'between', 'twelve', 'noon',
'and', 'six', 'p.m', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHICH', 'which'), ('FLIGHTS',
'flights')), ('VP', ('DEPART', 'depart'), ('VPBAR', ('NP', 'burbank'), ('PP',
('BETWEEN', 'between'), ('NP', ('NP', ('TWELVE', 'twelve'), ('NOON', 'noon')),
('NPBAR', ('AND', 'and'), ('NP', ('SIX', 'six'), ('P.M', 'p.m')))))))), ('PUN',
'.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', ('WHICH', 'which'), ('FLIGHTS',
'flights')), ('VP', ('DEPART', 'depart'), ('NP', ('NP', 'burbank'), ('NPBAR',
('PP', ('BETWEEN', 'between'), ('NP', ('TWELVE', 'twelve'), ('NOON', 'noon'))),
('NPBAR', ('AND', 'and'), ('NP', ('SIX', 'six'), ('P.M', 'p.m')))))), ('PUN',
'.'))
P:0.8571428571428571 R:0.8571428571428571 F:0.8571428571428571

input:  ['i', 'need', 'a', 'flight', 'from', 'kansas', 'city', 'to', 'newark',
'on', 'the', 'first', 'of', 'july', '.']
target:     ('TOP', ('S', ('NP', 'i'), ('VP', ('NEED', 'need'), ('NP', ('NP',
('A', 'a'), ('FLIGHT', 'flight')), ('NPBAR', ('PP', ('FROM', 'from'), ('NP',
('KANSAS', 'kansas'), ('CITY', 'city'))), ('NPBAR', ('PP', ('TO', 'to'), ('NP',
'newark')), ('PP', ('ON', 'on'), ('NP', ('NP', ('THE', 'the'), ('FIRST',
'first')), ('PP', ('OF', 'of'), ('NP', 'july')))))))), ('PUN', '.'))
predicted:  ()
P:0.0 R:0.0 F:0.0
```

```
input:  ['what', 'flights', 'are', 'there', 'from', 'new', 'york', 'to', 'las',
'vegas', '.']
target:     ('TOP', ('SBARQ', ('WHNP', ('WHAT', 'what'), ('FLIGHTS',
'flights')), ('SQ', ('ARE', 'are'), ('SQBAR', ('NP', 'there'), ('SQBAR', ('PP',
('FROM', 'from'), ('NP', ('NEW', 'new'), ('YORK', 'york'))), ('PP', ('TO',
'to'), ('NP', ('LAS', 'las'), ('VEGAS', 'vegas')))))), ('PUN', '.'))
predicted:  ('TOP', ('SBARQ', ('WHNP', ('WHAT', 'what'), ('FLIGHTS',
'flights')), ('SQ', ('ARE', 'are'), ('SQBAR', ('NP', 'there'), ('SQBAR', ('PP',
('FROM', 'from'), ('NP', ('NEW', 'new'), ('YORK', 'york'))), ('PP', ('TO',
'to'), ('NP', ('LAS', 'las'), ('VEGAS', 'vegas')))))), ('PUN', '.'))
P:1.0 R:1.0 F:1.0

Coverage: 67.24%, Average F-score (parsed sentences): 0.9504475408614075,
Average F-score (all sentences): 0.6390940360964636
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: