Analytical Component

problem 1.

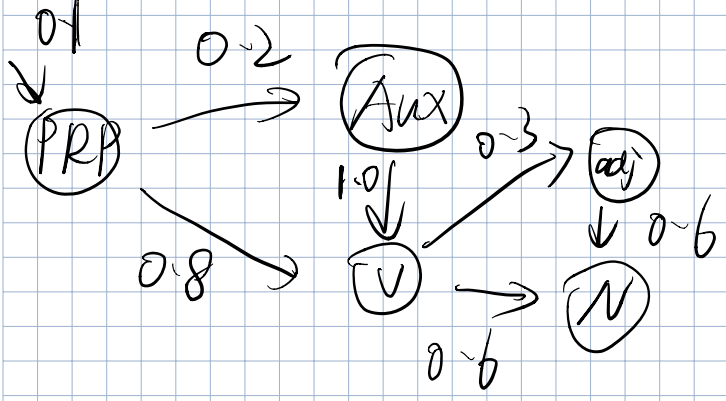
(a)

$P(\text{they} \mid \text{PRP}) = 1$ $P(\text{baking} \mid \text{adj}) = 1.0$

$P(\text{are} \mid \text{V}) = 0.5$ $P(\text{baking} \mid \text{V}) = 0.5$

$P(\text{are} \mid \text{Aux}) = 1$ $P(\text{potatoes} \mid \text{N}) = 1$

(b)



(c)

No, PCFG can model recursion but HMM can't

problem 2.

(a)

chart [0]

S → · NP VP [0,0]

NP → · Adj NP [0,0]

NP → · PRP [0,0]

NP → · N [0,0]

Adj → · baking [0,0]

PRP → · they [0,0]

N → · potatoes [0,0]

chart [1]

PRP → they · [0,1]

NP → PRP · [0,1]

S → NP · VP [0,1]

VP → · V NP [1,1]

VP → · Aux V NP [1,1]

V → · baking [1,1]

V → · are [1,1]

Aux → · are [1,1]

chart [2]

V → are · [1,2]

Aux → are [1,2]

VP → Aux · V NP [1,2]

VP → V · NP [1,2]

V → · baking [2,2]

V → · are [2,2]

NP → · adj NP [2,2]

NP → · PRP [2,2]

NP → · N [2,2]

adj → · baking [2,2]

PRP → · they [2,2]

N → · potatoes [2,2]

chart [3]

V → baking · [2,3]

adj → baking [2,3]

VP → Aux V · NP [1,3]

NP → adj · NP [2,3]

NP → · PRP [3,3]

NP → · N [3,3]

adj → · baking [3,3]

N → · potatoes [3,3]

PRP → · they [3,3]

chart [4]

N → potatoes · [3,4]

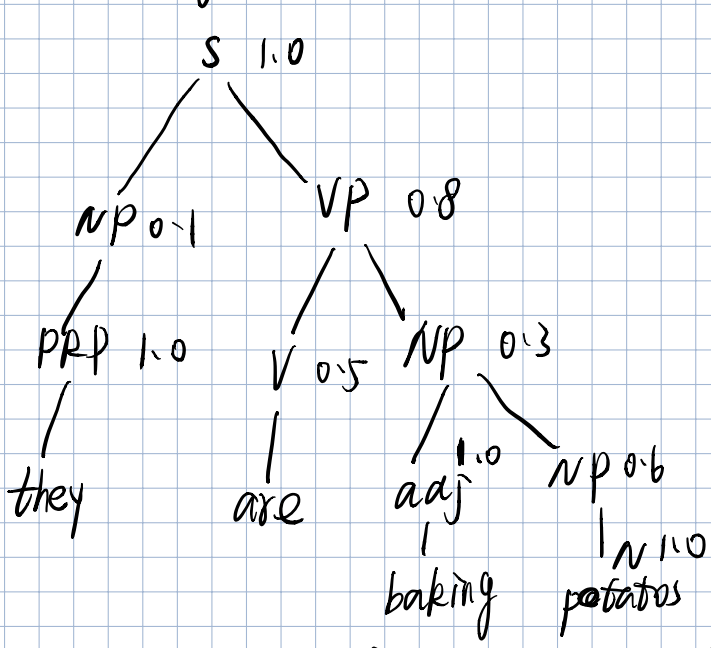
NP → N · [3,4]

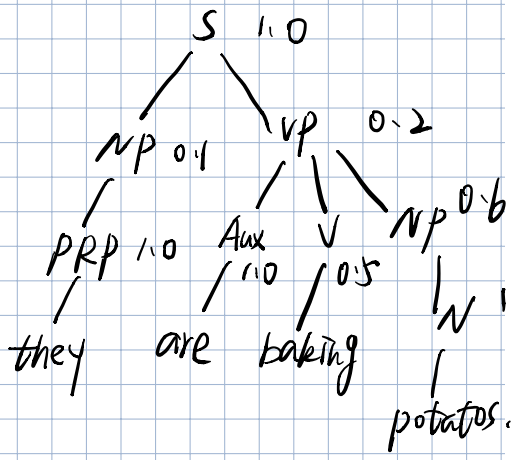
VP → adj NP [2,4]

VP → Aux V NP [1,4]

S → NP VP · [0,4]

(b) according to a, there are two parse trees.





$P_1 = 1.0 \times 0.1 \times 1.0 \times 0.8 \times 0.5 \times 0.3 \times 1.0 \times 0.6 \times 1.0$

$= 0.72\%$

$P_2 = 1.0 \times 0.1 \times 1.0 \times 0.2 \times 1.0 \times 0.5 \times 0.6 \times 1$

$= 0.6\%$

problem 3.

add AV → Aux V

S → NP VP

NP → adj NP

VP → V NP

VP → AV NP

AV → Aux V

NP → they

NP → potatoes

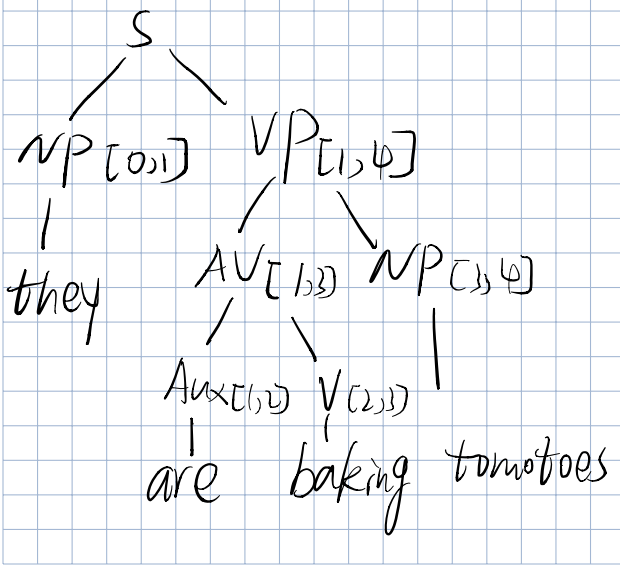
Aux → are

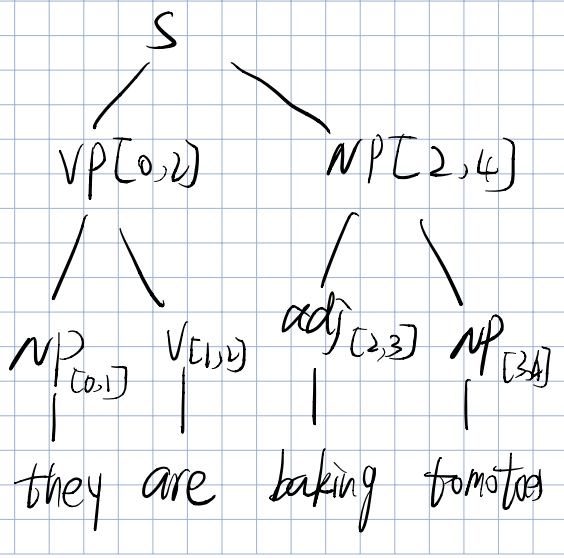
adj → baking

V → are

V → baking.

	they 0	are 1	baking 2	potatoes 3	4
0	NP	VP	VP	S	
1		Aux V	AV	VP	
2			adj V	NP VP	
3				NP	
4					





problem 6

Step	Stack	Word list	Action	Relation added
0	[root]	[he sent her a funny meme today]	shift	
1	[root he]	[sent her a funny meme today]	Left-arc	he ←sent
2	[root]	[sent her a funny meme today]	shift	
3	[root sent]	[her a funny meme today]	shift	
4	[root sent her]	[a funny meme today]	Right-arc	Sent→ her
5	[root sent]	[a funny meme today]	shift	
6	[root sent a]	[funny meme today]	shift	
7	[root sent a funny]	[meme today]	Left arc	meme → funny
8	[root sent a]	[meme today]	Left arc	meme→a
9	[root sent]	[meme today]	shift	
10	[root sent meme]	[today]	Right-arc	sent→meme
11	[root sent]	[today]	shift	
12	[root sent today]	[]	Right-arc	Sent→today
13	[root sent]	[]	Right-arc	root→sent
14	[root]	[]	done	

Programming component

Part 1 - reading the grammar and getting started

```
def verify_grammar(self):
    """
    Return True if the grammar is a valid PCFG in CNF.
    Otherwise return False.
    """
    # TODO, Part 1
    for key in self.lhs_to_rules:
        lhs_rule=self.lhs_to_rules[key]
        all_pro=[]
        for lhs in lhs_rule:
            all_pro.append(lhs[2])
            if len(lhs[1])==2 and (lhs[1][0].islower() or lhs[1][1].islower()):
                return False
            elif len(lhs[1])==1 and (lhs[1][0].isupper() or lhs[1][0].isupper()):
                return False
            else:
                continue
        if round(fsum(all_pro),2)!=1:
            return False
    return True
```

Part 2 - Membership checking with CKY

```
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
    n=len(tokens)
    chart=[[[] for i in range(n)]for j in range(n)]
    for i in range(n):
        rhs=self.grammar.rhs_to_rules[(tokens[i],)]
        for rh in rhs:
            chart[i][i].append(rh[0])
    for length in range(2,n+1):
        for i in range(n-length+1):
            j=i+length
            for k in range(i+1,j):
                M=itertools.product(chart[i][k - 1], chart[k][j - 1])
                for m in M:
```

```
                if m in self.grammar.rhs_to_rules:
                    rhs=self.grammar.rhs_to_rules[m]
                    for rh in rhs:
                        chart[i][j-1].append(rh[0])
    if self.grammar.startsymbol in chart[0][n-1]:
        return True
    else:
        return False
```

Part 3 - Parsing with backpointers

```
def parse_with_backpointers(self, tokens):
    """
    Parse the input tokens and return a parse table and a probability table.
    """
    # TODO, part 3
    table= defaultdict(dict)
    probs = defaultdict(dict)
    n=len(tokens)
    for i in range(n):
        this_range=(i,i+1)
        if (tokens[i],) in self.grammar.rhs_to_rules:
            rhs=self.grammar.rhs_to_rules[(tokens[i],)]
            for rh in rhs:
                if rh[0] not in table[this_range]:
                    table[this_range][rh[0]]=tokens[i]
                    probs[this_range][rh[0]]=math.log2(rh[2])
                elif math.log2(rh[2])>probs[this_range][rh[0]]:
                    table[this_range][rh[0]] = tokens[i]
                    probs[this_range][rh[0]] = math.log2(rh[2])
    for length in range(2,n+1):
        for i in range(n-length+1):
            j=i+length
            this_range = (i, j)
            table[this_range]
            probs[this_range]
            for k in range(i+1,j):
                M=itertools.product(table[(i,k)].keys(),table[(k,j)].keys())
                for m in M:
                    rhs=self.grammar.rhs_to_rules[m]
                    for rh in rhs:
                        if rh[0] not in table[this_range]:
                            table[this_range][rh[0]]=((m[0],i,k),(m[1],k,j))
                            probs[this_range][rh[0]]=math.log(rh[2])+probs[(i,k)][m[0]]+probs[(k,j)][m[1]]
                        else:
```

```
                        if
probs[this_range][rh[0]]<math.log2(rh[2])+probs[(i,k)][m[0]]+probs[(k,j)][m[1]]:
                            table[this_range][rh[0]]=((m[0],i,k),(m[1],k,j))
                            probs[this_range][rh[0]]=math.log2(rh[2])+probs[(i,k)][m[0]]+probs[(k,j)][m[1]]


    return table, probs
```

Part 4 - Retrieving a parse tree

```
def get_tree(chart, i,j,nt):
    """
    Return the parse-tree rooted in non-terminal nt and covering span i,j.
    """
    # TODO: Part 4
    if j==i+1:
        return (nt,chart[i,j][nt])
    left=get_tree(chart,chart[(i,j)][nt][0][1],chart[(i,j)][nt][0][2],chart[(i,j)][nt][0][0])
    right=get_tree(chart,chart[(i,j)][nt][1][1],chart[(i,j)][nt][1][2],chart[(i,j)][nt][1][0])
    return (nt,left,right)
```