Homework 2

Congcheng Yan (cy2550)

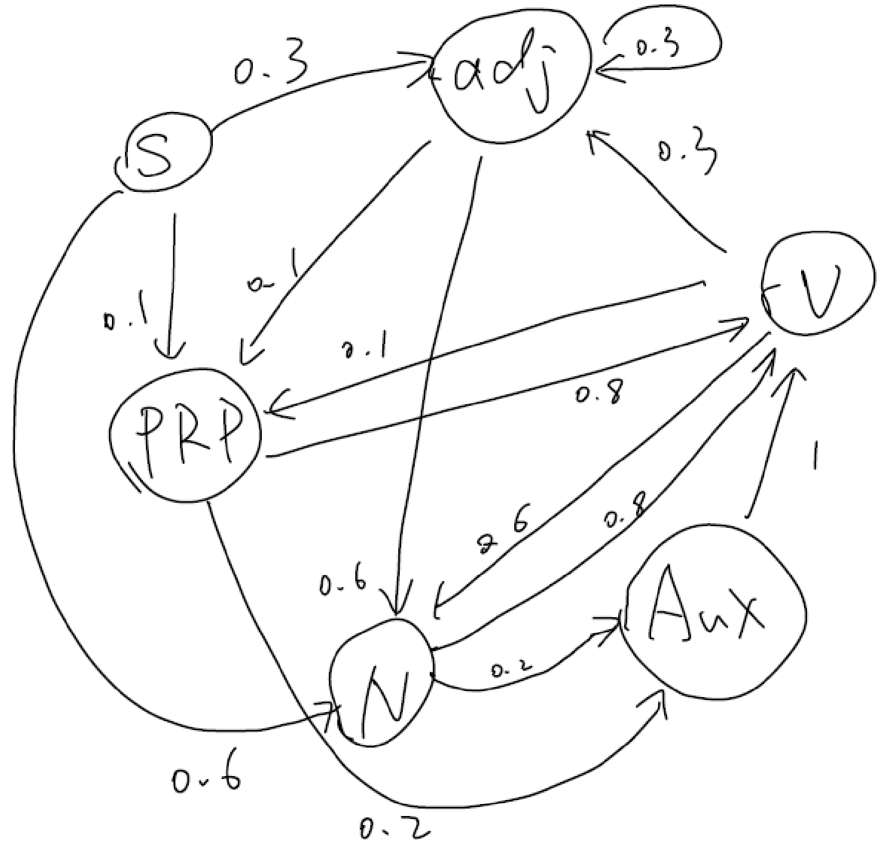
March 3, 2020

Analytical Part

Problem 1

(a)

$$
\begin{aligned}
P(\text{ tags },\ \text{ words }) &= P(\text{ words } \mid \text{ tags })P(\text{ tags }) \\
&= \prod_{\text{not\_}bottom\_layer} P(grammar|\text{p}arent\_grammar) \prod_{\text{b}ottom\_layer} P(\text{t}erminal|grammar)
\end{aligned}
$$


S
adj
PRP
V
N
Aux
0.3
0.3
0.3
0.1
0.1
0.1
0.8
0.6
0.8
1
0.6
0.2
0.6
0.2

$$\begin{tikzcd}
S \arrow[r, "P(PRP|S)=0.1"] & PRP \arrow[r, "P(V|PRP)=0.8"] \arrow[d, "P(they|PRP)=1"'] & V \arrow[r, "P(Adj|V)=0.3"] \arrow[d, "P(are|V)=0.5"'] & Adj \arrow[r, "P(N|Adj)=0.6"] \arrow[d, "P(baking|Adj)=1"'] & N \arrow[d, "P(potatoes|N)=1"'] \\
& \text{they} & \text{are} & \text{baking} & \text{potatoes}
\end{tikzcd}$$

$$0.1 \times 0.8 \times 0.3 \times 0.6 \times 1 \times 0.5 \times 1 \times 1 = 0.0072$$

$$\begin{tikzcd}
S \arrow[r, "P(PRP|S)=0.1"] & PRP \arrow[r, "P(Aux|PRP)=0.2"] \arrow[d, "P(they|PRP)=1"'] & Aux \arrow[r, "P(V|Aux)=1"] \arrow[d, "P(are|Aux)=1"'] & V \arrow[r, "P(N|V)=0.6"] \arrow[d, "P(baking|V)=0.5"'] & N \arrow[d, "P(potatoes|N)=1"'] \\
& \text{they} & \text{are} & \text{baking} & \text{potatoes}
\end{tikzcd}$$

$$0.1 \times 0.2 \times 1 \times 0.6 \times 1 \times 1 \times 0.5 \times 1 = 0.006$$

(b)

(c) Yes. The two methods are just two perspectives to view the tree. In HMM, we calculate the conditional probabilities of the bottom layer of grammar order by word. In PCFG we calculate the probability by generating the grammar from top to bottom. The probabilities of terminals at the condition of grammar are the same. So, for all the trees (sentences), the joint probability from two models should be same.Thus, one can be transfromed to the other.

Problem 2

(a) 5 charts

chart [0]

S_0 S → · NP VP [0, 0]

S_1 NP → · Adj NP [0, 0]

S_2 NP → · PRP [0, 0]

S_3 NP → · N [0, 0]

S_4 Adj → · baking [0, 0]

S_5 PRP → · they [0, 0]

S_6 N → · potatoes [0, 0]

chart[1]

S_7 PRP → they · [0, 1]

S_8 NP → PRP · [0, 1]

S_9 S → NP · VP [0, 1]

S_{10} VP → · V NP [1, 1]

S_{11} VP → · Aux V NP [1, 1]

S_{12} V → · baking [1, 1]

S_{13} V → · are [1, 1]

S_{14} Aux → · are [1, 1]

chart [2]

S_{15} V → are · [1,2]

S_{16} Aux → are · [1,2]

S_{17} VP → V · NP [1,2]

S_{18} VP → Aux · V NP [1,2]

S_{19} NP → · Adj NP [2,2]

S_{20} NP → · PRP [2,2]

S_{21} NP → · N [2,2]

S_{22} V → · baking [2,2]

S_{23} V → · are [2,2]

S_{24} Adj → · baking [2,2]

S_{25} PRP → · they [2,2]

S_{26} N → · potatoes [2,2]

chart [3]

S_{27} V → baking · [2,3]

S_{28} Adj → baking · [2,3]

S_{29} VP → Aux V · NP [1,3]

S_{30} NP → Adj · NP [2,3]

S_{31} NP → · Adj NP [3,3]

S_{32} NP → · PRP [3,3]

S_{33} NP → · N [3,3]

S_{34} Adj → · baking [3,3]

S_{35} PRP → · they [3,3]

S_{36} N → · potatoes [3,3]

chart [4]

S_{37} N → potatoes · [3,4]

S_{38} NP → N · [3,4]

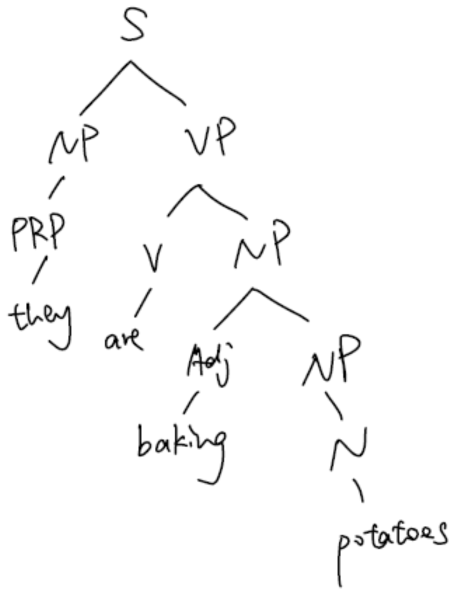
S_{39} VP → Aux V NP · [1,4]

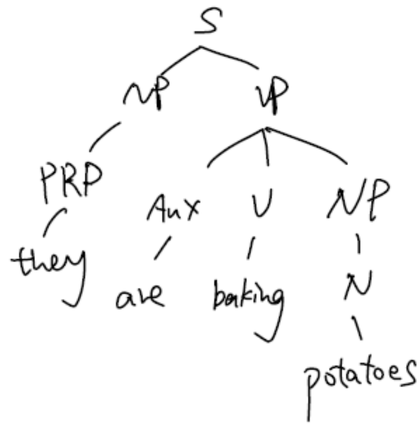
S_{40} NP → Adj NP · [2,4]

S_{41} S → NP VP · [0,4]

S_{42} VP → V NP · [1,4]

(b) $0.1 * 0.8 * 0.5 * 0.3 * 0.6 = 0.0072$
$0.1 * 0.2 * 0.5 * 0.6 = 0.006$


S
NP
VP
PRP
V
NP
they
are
Adj
NP
baking
N
potatoes


S
NP
VP
PRP
Aux
V
NP
they
are
baking
N
potatoes

0.0072

0.006

Problem 3

(a) 1. Replace the single arrow $A \to B$ and $B \to CD$ with $A \to CD$

2. Add a new nonterminal N, recursively make the transformation: combine the first two as N, and replace it. Loop ends when there are two terms on the right.

$S \to NP \quad VP$

$NP \to Adj \quad NP$

$NP \to$ they

$NP \to$ potatoes

$VP \to V \quad NP$

$VP \to New \quad NP$

$New \to Aux \quad V$

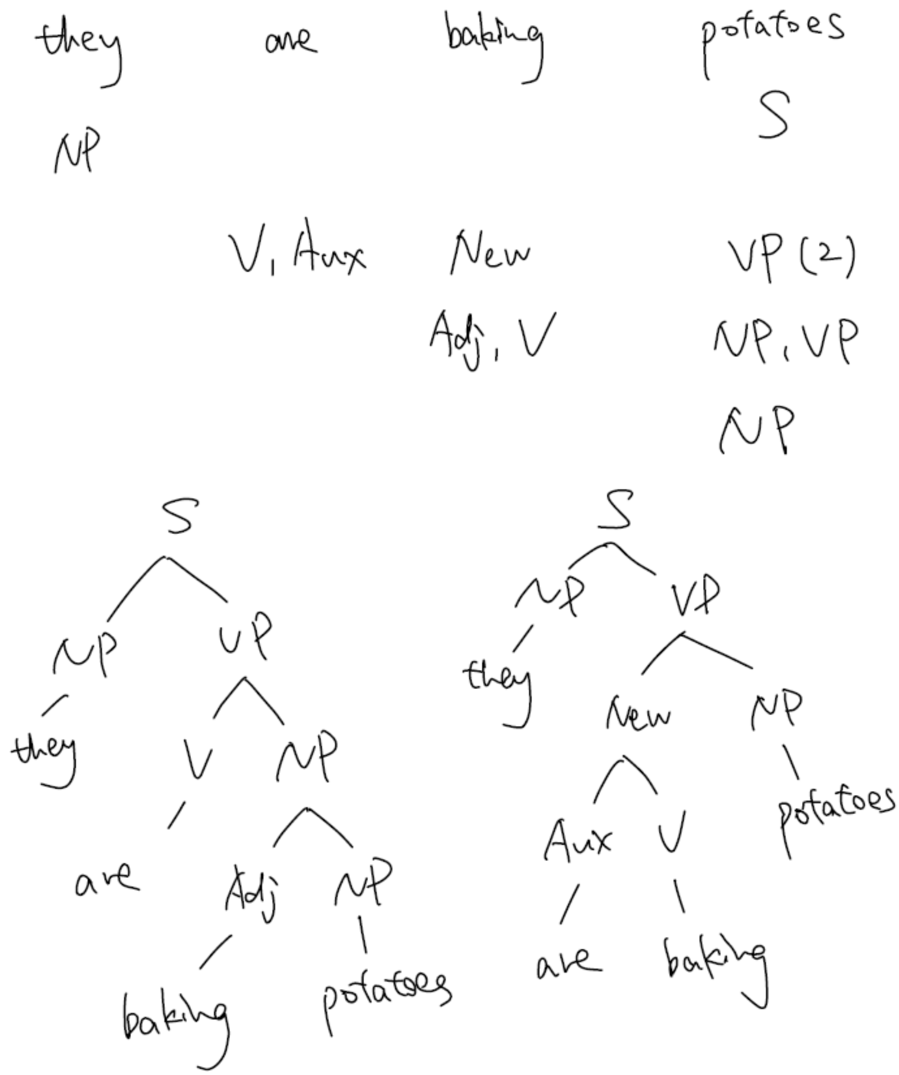
$Adj \to$ baking

$V \to$ baking

$V \to$ are

$Aux \to$ are

(b) .



Problem 4

1. initial, next: shift
   ([root $]_\sigma$, [he, sent, her, a, funny, meme, today $]_\beta, \{\}_A$)
2. next: left-arc
   ([root, he $]_\sigma$, [sent, her, a, funny, meme, today $]_\beta, \{\}_A$)

3. next: shift

   $([\text{ root}]_\sigma, [\text{ sent, her, a, funny, meme, today }]_\beta, \{\text{(sent, nsubj, he)}\}_A)$

4. next: right-arc

   $([\text{ root, sent}]_\sigma, [\text{ her, a, funny, meme, today }]_\beta, \{\text{(sent, nsubj, he)}\}_A)$

5. next: shift

   $([\text{ root}]_\sigma, [\text{ sent, a, funny, meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her)}\}_A)$

6. next: shift

   $([\text{ root, sent}]_\sigma, [\text{ a, funny, meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her)}\}_A)$

7. next: shift

   $([\text{ root, sent, a}]_\sigma, [\text{ funny, meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her)}\}_A)$

8. next: left-arc

   $([\text{ root, sent, a, funny}]_\sigma, [\text{ meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her)}\}_A)$

9. next: left-arc

   $([\text{ root, sent, a}]_\sigma, [\text{ meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her), (meme, amod, funny)}\}_A)$

10. next: right-arc

    $([\text{ root, sent}]_\sigma, [\text{ meme, today }]_\beta, \{\text{(sent, nsubj, he), (sent, iobj, her) ,}$
    $\text{(meme, amod, funny), (meme, det, a) }\}_A\ )$

11. next: shift

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

12. next: right-arc

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

13. next: right-arc

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

14. next: shift

    $([\ ]_\sigma, [\text{ root }]_\beta, \{($ sent, nsubj, he$), ($sent, iobj, her $)$,
    (meme, amod, funny), (meme, det, a), (sent, dobj, meme),
    (sent, advmod, today), (root, pred, sent) $\}_A$)

15. terminal state

    $([\text{ root}]_\sigma, [\ ]_\beta, \{($ sent, nsubj, he$), ($sent, iobj, her $)$,
    (meme, amod, funny), (meme, det, a), (sent, dobj, meme),
    (sent, advmod, today), (root, pred, sent) $\}_A$)

Programming Part

Part 1

```python
def verify_grammar(self):
    """
    Return True if the grammar is a valid PCFG in CNF.
    Otherwise return False.
    """
    # TODO, Part 1
    result = True
    for key in self.lhs_to_rules.keys():
        if not key.isupper():
            result = False
        #print('testhere',[x[1] for x in self.lhs_to_rules[key]])
        for item in [x[1] for x in self.lhs_to_rules[key]]:
            if len(item) == 2 and item[0].isupper() and item[1].isupper():
                pass
            elif len(item) == 1:
                for j in item[0]:
                    if j.isupper():
                        result = False
        prob_list = [x[-1] for x in self.lhs_to_rules[key]]
        if abs(fsum(prob_list))< 0.00001:
            #print(prob_list,fsum(prob_list))
            result = False
    return result
```

Part 2

```python
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
    #initialization
    tdic = defaultdict(list)
    for i in range(len(tokens)):
        j = tokens[i]
        if tdic == []:
            return False
        tdic[i,i+1] = [x[0] for x in self.grammar.rhs_to_rules[j,]]
        # print('123',tdic[i, i + 1])
    #main loop
    for length in range(2,len(tokens)+1):
        for i in range(len(tokens)+1-length):
            j = i + length
            for k in range(i+1,j):
                # print(i,k,j,tdic[i, k],tdic[k, j])
                for a in tdic[i, k]:
                    for b in tdic[k, j]:
                        tdic[i, j] = list(set(tdic[i, j]).union(set([x[0] for x in self.grammar.rhs_to_rules[(a,b)]])))
                        # tdic[i, j] = tdic[i, j] + [x[0] for x in grammar.rhs_to_rules[(a,b)]]


                # print((tdic[i, k],tdic[k, j]))
                # print([x[0] for x in grammar.rhs_to_rules[(tdic[i, k],tdic[k, j])]])
                # tdic[i,j] = tdic[i,j] + [x[0] for x in grammar.rhs_to_rules[(tdic[i, k],tdic[k, j])]]
    return tdic[0,len(tokens)] == [self.grammar.startsymbol]
```

Part 3

```python
def parse_with_backpointers(self, tokens):
    """
    Parse the input tokens and return a parse table and a probability table.
    """

    # if self.is_in_language(tokens) == False:
    #     print("the grammar can NOT parse this sentence")
    #     return 0
    # initialization
    table = defaultdict(dict)
    probs = defaultdict(dict)

    for i in range(len(tokens)):
        j = tokens[i]
        probs[(i, i + 1)] = defaultdict(int)
        table[(i, i + 1)] = defaultdict(str)
        for x in self.grammar.rhs_to_rules[j,]:
            # print('123',x)
            probs[(i, i + 1)][x[0]] = x[2]
            table[(i, i + 1)][x[0]] = x[1][0]
    # print('testhere',table)
```

```python
    # main loop
    for length in range(2, len(tokens) + 1):
        for i in range(len(tokens) + 1 - length):
            j = i + length
            #print('i','j',i,j)
            probs[(i, j)] = defaultdict(int)
            for k in range(i + 1, j):
                # print(i,k,j,table[(i, k)].keys(),table[(k, j)].keys())
                for a in table[(i, k)].keys():
                    for b in table[(k, j)].keys():
                        for q in self.grammar.rhs_to_rules[(a,b)]:
                            # print(table)
                            # print(probs)
                            # print(q,i,k,j)
                            if probs[(i, j)][q[0]] < q[2] * probs[(i, k)][a] * probs[(k, j)][b]:
                                probs[(i, j)][q[0]] = q[2]*probs[(i,k)][a]*probs[(k,j)][b]
                                table[(i, j)][q[0]] = ((a,i,k),(b,k,j))
                            #print(probs[(i, j)][q[0]], q[2] * probs[(i, k)][a] * probs[(k, j)][b],i,k,j)
    for i in probs.keys():
        for j in probs[i].keys():
            if probs[i][j]>0:
                probs[i][j] = math.log(probs[i][j])


    return table, probs
```

Part 4

```python
def get_tree(chart, i,j,nt):
    """
    Return the parse-tree rooted in non-terminal nt and covering span i,j.
    """
    # TODO: Part 4
    tree = (nt,chart[i,j][nt][0],chart[i,j][nt][1])
    print('tree',tree)
    def sub_tree(otree):
        #print('0',otree)

        if type(chart[otree[1],otree[2]][otree[0]]) != str:
            # print('1', chart[otree[1][1],otree[1][2]][otree[1][0]])
            # print('2',chart[otree[2][1],otree[2][2]][otree[2][0]])
            return (otree[0], sub_tree(chart[otree[1],otree[2]][otree[0]][0]),sub_tree(chart[otree[1],otree[2]][otree[0]][1]))
        else:
            return (otree[0], chart[otree[1],otree[2]][otree[0]])



    tree = (tree[0],sub_tree(tree[1]),sub_tree(tree[2]))

    return tree
```