CMPT 413 - SPRING 2007 - FINAL EXAM

Answer only seven out of eight questions (5pts each).

You must list seven question numbers on your answer booklet

When you have finished, return your answer booklet along with this question booklet.

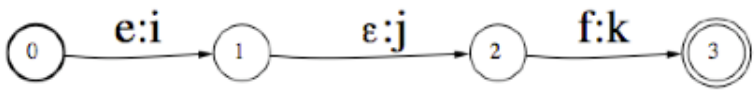
Apr 17, 2007

(1) Finite-state transducers:

a. Provide a finite-state transducer for each of the following regular expressions:

1. `( (a:c) (b:d)* ) | ( (e:g)* (f:h) )`
2. `(g:i) (ϵ:j) (h:k)*`

b. State true or false if the following finite-state transducer is the result of the composition of the two finite-state transducers above.



c. Provide the regular expression for the composed finite-state transducer.

d. For the input language $\mathcal{I} = \{ae^n a \mid n \geq 1\}$, provide the output language produced by the application of the rewrite rule $e \rightarrow aea / ae^*\_a$ on each element of $\mathcal{I}$. Apply the rule from left to right, iteratively.

The application of the rewrite rule should be constrained in the usual way so that it can only produce regular languages.

Provide the output language and also provide a step by step application of the rule on the following input strings (show the lhs and the left and right context match for each step):

1. *aea*
2. *aeeea*

Answer: Output language after application of the rewrite rule is $\{a(aea)^n a \mid n \geq 1\}$

The lhs of the rewrite rule where left and right context matches is written as e:

1. $a\mathsf{e}a \rightarrow aaeaa$
2. $ae^3a = aee\mathsf{e}a \rightarrow aeeaeaa$. Note that at this point e cannot be rewritten even though the left and right context matches, because of the restriction that prohibits rewriting any string that was inserted. This restrictions permits the rule to generate only regular languages. Then $ae\mathsf{e}aeaa \rightarrow a\mathsf{e}aeaaeaa \rightarrow aaeaaeaaeaa = a(aea)^3a$.

(2) Consider grammar G:

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow AC \\ A &\rightarrow aB \\ B &\rightarrow BC \\ B &\rightarrow b \\ C &\rightarrow c \end{aligned}$$

a. Provide $L(G)$, the language recognized by grammar G.

Answer: $L(G) = \{abc^k : k \geq 0\}$

b. Use the Earley algorithm to parse the input string *abcc*.
Provide the contents of the chart produced by the Earley algorithm while parsing this input string. The contents of chart[0] and chart[1] are provided below to get you started. Use the same notation for chart[2] . . . chart[4].

- chart[0]:
  1. $(S \rightarrow \bullet\ A, [0,0])$
  2. $(A \rightarrow \bullet\ A\ C, [0,0])$
  3. $(A \rightarrow \bullet\ a\ B, [0,0])$
- chart[1]:
  1. $(A \rightarrow a\ \bullet\ B, [0,1])$
  2. $(B \rightarrow \bullet\ B\ C, [1,1])$
  3. $(B \rightarrow \bullet\ b, [1,1])$

Answer:

- chart[2]:
  1. $(B \rightarrow b\ \bullet\ , [1,2])$
  2. $(B \rightarrow B\ \bullet\ C, [1,2])$
  3. $(A \rightarrow a\ B\ \bullet\ , [0,2])$
  4. $(C \rightarrow \bullet\ c, [2,2])$
  5. $(A \rightarrow A\ \bullet\ C, [0,2])$
  6. $(S \rightarrow A\ \bullet\ , [0,2])$
- chart[3]
  1. $(C \rightarrow c\ \bullet\ , [2,3])$
  2. $(B \rightarrow B\ C\ \bullet\ , [1,3])$
  3. $(A \rightarrow A\ C\ \bullet\ , [0,3])$
  4. $(B \rightarrow B\ \bullet\ C, [1,3])$
  5. $(A \rightarrow a\ B\ \bullet\ , [0,3])$
  6. $(A \rightarrow A\ \bullet\ C, [0,3])$
  7. $(S \rightarrow A\ \bullet\ , [0,3])$
  8. $(C \rightarrow \bullet\ c, [3,3])$
- chart[4]
  1. $(C \rightarrow c\ \bullet\ , [3,4])$
  2. $(A \rightarrow A\ C\ \bullet\ , [0,4])$
  3. $(B \rightarrow B\ C\ \bullet\ , [1,4])$
  4. $(A \rightarrow A\ \bullet\ C, [0,4])$
  5. $(S \rightarrow A\ \bullet\ , [0,4])$ (accept state)
  6. $(B \rightarrow B\ \bullet\ C, [1,4])$
  7. $(A \rightarrow a\ B\ \bullet\ , [0,4])$
  8. $(C \rightarrow \bullet\ c, [4,4])$

c. What is the *accept state* and why?

Answer: $(S \rightarrow A\ \bullet\ , [0,4])$
Because, S is the start state. 0, 4 spans the entire input string.

d. The Catalan function is defined to be:

$$\text{Cat}(n) = \frac{1}{n+1}\left(\begin{array}{c} 2n \\ n \end{array} \right)$$

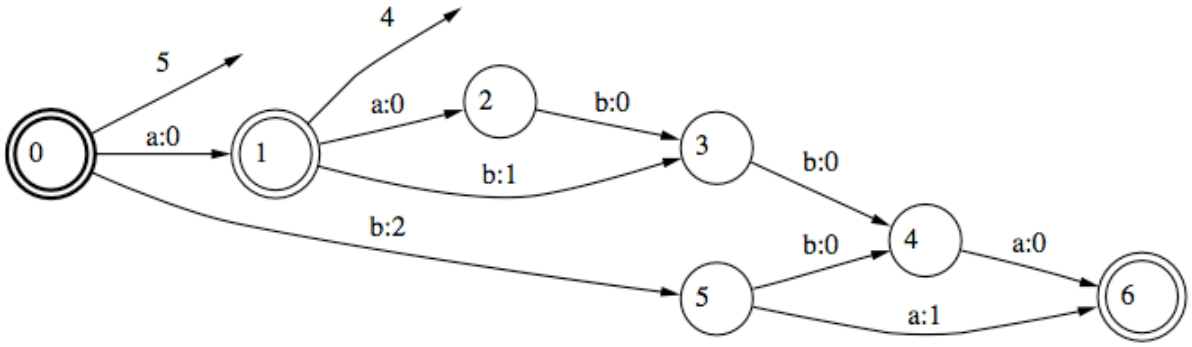
$$\text{where } \left(\begin{array}{c} a \\ b \end{array} \right) = \frac{a!}{(b!(a-b)!)}$$

Explain how Cat(n) can be used to calculate the total number of parses for every string $w \in L(G)$.

Answer: Note that $L(G) = \{abc^k : k \geq 0\}$. The total number of parses for each string $w \in L(G)$ is given by Cat($k + 1$).

(3) *n*-grams:

Consider the finite-state transducer (FST) below. The notation for the transitions is the usual `input:output`. The FST is 1-subsequential: it produces output 5 on reaching state `0`, and output 4 on reaching state `1`.



This FST was built for the text t = *aabba* shown below with each index position in the text:

index	0	1	2	3	4	5
t	*a*	*a*	*b*	*b*	*a*	

The input alphabet of the FST consists of the characters in the text t, while the output alphabet consists of the numbers used to index the text. *This FST encodes for every substring in the text t the set of index positions where that substring occurs in t.*

Here is how it works: when applied to the string a, the FST moves from state `0` to state `1` and produces output 0 which corresponds to the first index position of the string a in the text t. Starting from state `1`, if we collect every path to a final state, we get three paths: $\{(1), (1 \to 2 \to 3 \to 4 \to 6), (1 \to 3 \to 4 \to 6)\}$.

Now sum the outputs along each of these paths: $\{(4), (0+0+0+0), (1+0+0)\} = \{4, 0, 1\}$.

Notice that we had a zero length path to state `1` which happens to be a final state, and which produces the output 4. Then we add the initial output 0 that we had obtained going from state `0` to state `1` on input a to each element in this set: $\{ 0+4 , 0+0 , 0+1 \} = \{ 4 , 0 , 1 \}$.

The set we obtain using this procedure is $\{ 4 , 0 , 1 \}$ which contains every index position where we observe the string a in the text t.

a. Use the procedure described above to find all index positions for the string b. Provide each step in the procedure.

Answer: $0 \to 5$ on b, output 2

$$\{5 \to 4 \to 6, 5 \to 6\} = \{0+0, 1\} = \{0, 1\}$$

$$\{2+0, 2+1\} = \{2, 3\}$$

b. Provide a technique that uses the procedure described above in order to find the frequency of every substring (of any size) in the input text. For example, the substring a occurs 3 times in text t and the

substring *aabba* occurs once. Explain the technique in one or two sentences.

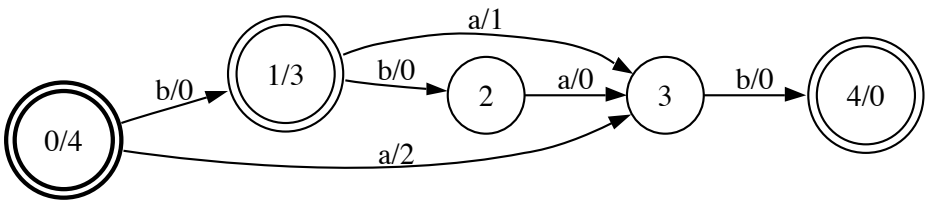
Answer: For every substring of the text t, find the set of positions. The n-gram count for the substring is the number of elements in this set. Counts for substring of length 1 and length 5 are:

$$\begin{aligned} c(a) &= len(\{4,0,1\}) = 3 \\ c(b) &= len(\{2,3\}) = 2 \\ c(aabba) &= len(\{0\}) = 1 \end{aligned}$$

c. For the input text *bbab*, provide a FST that can be used to compute all index positions for every substring. Make sure that your FST computes the following index positions correctly:

$$\begin{aligned} a &= \{2\} \\ b &= \{3,0,1\} \\ bab &= \{1\} \\ bb &= \{0\} \end{aligned}$$

Answer:



(4) Smoothing:

a. You are given the following training data for the prepositional phrase (PP) attachment task.

v	n1	p	n2	Attachment
join	board	as	director	V
is	chairman	of	N.V.	N
using	crocidolite	in	filters	V
bring	attention	to	problem	V
is	asbestos	in	products	N
making	paper	for	filters	N
including	three	with	cancer	N
⋮	⋮	⋮	⋮	⋮

In order to resolve PP attachment ambiguity we can train a probability model: $P(A = N \mid v, n1, p, n2)$ which predicts the attachment A as N if $P > 0.5$ and V otherwise. Since we are unlikely to see the same four words $v, n1, p, n2$ in novel unseen data, in order for this probability model to be useful we need to take care of zero counts.

Provide a Jelinek-Mercer *interpolation* smoothing model $\hat{P}(A = N \mid v, n1, p, n2)$ for this PP attachment probability model. Assume that our training data is large enough to contain all the prepositions we might observe in unseen data.

Answer:

$$\begin{aligned}\hat{P}(A = N \mid v, n1, p, n2) \quad = \quad & \lambda_1 P(A = N \mid v, n1, p, n2) \\ & +\lambda_2 P(A = N \mid v, n1, p) \\ & +\lambda_3 P(A = N \mid n1, p) \\ & +\lambda_4 P(A = N \mid v, p) \\ & +\lambda_5 P(A = N \mid p)\end{aligned}$$

Of course, to be a well-formed interpolation model, $\sum_i \lambda_i = 1$. There are many other solutions but the structure of the model has to of the kind shown above.

b. You are given a text of words: $w_1, \ldots, w_N$ and you proceed to estimate bigram probabilities with the maximum likelihood estimate using the bigram frequencies $c(w_i, w_{i-1})$ and unigram frequencies $c(w_i)$:

$$\hat{P}(w_i \mid w_{i-1}) = \frac{c(w_i, w_{i-1})}{c(w_{i-1})}$$

You notice that the frequency for all individual words w_i are non-zero, i.e. $c(w_i) > 0$ for all w_i. However, you do notice that many bigrams w_i, w_{i-1} have zero counts.

In order to avoid zeroes in the numerator of the above equation, you decide to add a small factor δ to each count in the following manner:

$$\hat{P}(w_i \mid w_{i-1}) = \frac{\delta + c(w_i, w_{i-1})}{c(w_{i-1})}$$

Assume $0 < \delta < 1$. Is this formulation of smoothing correct? If not, what is the basic condition on $P(w_i \mid w_{i-1})$ violated in this formula? If incorrect, provide a correction for this formula.

Answer: Since there are N many $c(w_i, w_{i-1})$ observed.

$$\hat{P}(w_i \mid w_{i-1}) = \frac{\delta + c(w_i, w_{i-1})}{\delta N + c(w_i)}$$

c. For bigram probabilities, Katz backoff smoothing is defined as follows:

$$P_{katz}(w_i \mid w_{i-1}) = \begin{cases} \frac{c^*(w_{i-1},w_i)}{c(w_{i-1})} & \text{if } c(w_{i-1},w_i) > 0 \\ \alpha(w_{i-1})P_{katz}(w_i) & \text{otherwise} \end{cases}$$

where $\alpha(w_{i-1})$ is chosen to make sure that $P_{katz}(w_i \mid w_{i-1})$ is a proper probability

$$\alpha(w_{i-1}) = 1 - \sum_{w_i} \frac{c^*(w_{i-1},w_i)}{c(w_{i-1})}$$

where c^* is the Good-Turing estimate. Provide the Good-Turing estimate $c^*(w_{i-1},w_i)$ and $c^*(w_i)$ for the above smoothing equations. Assume linear interpolation has provided you with all the missing values required.

Answer:

$$c^*(w_{i-1},w_i) = (c(w_{i-1},w_i)+1) \times \frac{n_{c(w_{i-1},w_i)+1}}{n_{c(w_{i-1},w_i)}}$$

$$c^*(w_i) = (c(w_i)+1) \times \frac{n_{c(w_i)+1}}{n_{c(w_i)}}$$

where $n_{c(w_{i-1},w_i)}$ and $n_{c(w_i)}$ stands for the number of different w_{i-1}, w_i or w_i types observed for count $c(w_{i-1},w_i)$ and $c(w_i)$ respectively.

(5) Part-of-speech Tagging:

Consider the task of assigning the most likely part of speech tag to each word in an input sentence. We want to get the best (or most likely) tag sequence as defined by the equation:

$$T^* = \underset{t_0,\ldots,t_n}{\text{argmax}}\, P(t_0, \ldots, t_n \mid w_0, \ldots, w_n)$$

a. Write down the equation for computing the probability $P(t_0, \ldots, t_n \mid w_0, \ldots, w_n)$ using Bayes Rule and a trigram probability model over part of speech tags.

Answer:

$$\begin{aligned} P(t_0, \ldots, t_n \mid w_0, \ldots, w_n) &= P(w_0, \ldots, w_n \mid t_0, \ldots, t_n) \times P(t_0, \ldots, t_n) \\ &= \left(\prod_{i=0}^{n} P(w_i \mid t_i)\right) \times \left(P(t_0) \times P(t_1 \mid t_0) \times \prod_{i=2}^{n} P(t_i \mid t_{i-2}, t_{i-1})\right) \end{aligned}$$

b. We realize that we can get better tagging accuracy if we can condition the current tag on the previous tag and the next tag, i.e. if we can use $P(t_i \mid t_{i-1}, t_{i+1})$. Thus, we define the best (or most likely) tag sequence as follows:

$$\begin{aligned} T^* &= \underset{t_0,\ldots,t_n}{\text{argmax}}\, P(t_0, \ldots, t_n \mid w_0, \ldots, w_n) \\ &\approx \underset{t_0,\ldots,t_n}{\text{argmax}} \prod_{i=0}^{n+1} P(w_i \mid t_i) \times P(t_i \mid t_{i-1}, t_{i+1}) \text{ where } t_{-1} = t_{n+1} = \text{none} \end{aligned}$$

Explain why the Viterbi algorithm cannot be directly used to find T^* for the above equation.

Answer: The standard way of using Viterbi would be to have states in the HMM as pairs of tags and then we store the best path upto each tag pair $\langle t_{i-1}, t_i \rangle$ for time step i and then we can efficiently compute the best score upto state t_{i+1} recursively as follows:

$$\text{Viterbi}[i+1, \langle t_{i+1}, t_i \rangle] = \max_{\langle t_{i-1}, t_i \rangle} (\text{Viterbi}[i, \langle t_{i-1}, t_i \rangle] \times P(w_{i+1} \mid t_{i+1}) \times P(t_{i+1} \mid \langle t_{i-1}, t_i \rangle))$$

For t_i we can only condition on the previous tag t_{i-1} in the Viterbi algorithm since we have entered the score for each possible t_{i-1} in the Viterbi matrix. Crucially Viterbi proceeds from left to right, and so it cannot condition on a tag t_{i+1} which occurs on the right (and is not still part of the Viterbi table). Running Viterbi from right to left also does not work, due to tag t_{i-1} which occurs on the left. As a result, we cannot simultaneously consider t_{i-1} and t_{i+1} for Viterbi computation for the trigram probability $P(t_i \mid t_{i-1}, t_{i+1})$.

c. BestScore is the score for the maximum probability tag sequence for a given input word sequence.

$$\text{BestScore} = \max_{t_0,\ldots,t_n} P(t_0, \ldots, t_n \mid w_0, \ldots, w_n)$$

It is a bit simpler to compute than Viterbi since it does not compute the best sequence of tags (no back pointer is required). For the standard trigram model $P(t_i \mid t_{i-2}, t_{i-1})$:

$$\text{BestScore} = \max_{t_0,\ldots,t_n} \prod_{i=0}^{n+1} P(w_i \mid t_i) \times P(t_i \mid t_{i-2}, t_{i-1})$$

Assuming that $t_{-1} = t_{-2} = t_{n+1} = \text{none}$, we can compute BestScore recursively from left to right as follows:

$$\begin{aligned}\text{BestScore}[i+1, t_{i+1}, t_i] &= \max_{t_{i-1}, t_i} (\ \text{BestScore}[i, t_{i-1}, t_i] \times P(w_{i+1} \mid t_{i+1}) \times P(t_{i+1} \mid t_{i-1}, t_i)\ ) \\ &\quad \text{for all } -1 \leq i \leq n \\ \text{BestScore} &= \max_{\langle t, \text{none} \rangle} \text{BestScore}[n+1, \langle t, \text{none} \rangle]\end{aligned}$$

This algorithm for computing BestScore is simply the recursive forward algorithm for HMMs but with the sum replaced by max.

Provide an algorithm in order to compute BestScore for the improved trigram model $P(t_i \mid t_{i-1}, t_{i+1})$:

$$\text{BestScore} = \max_{t_0, \ldots, t_n} \prod_{i=0}^{n+1} P(w_i \mid t_i) \times P(t_i \mid t_{i-1}, t_{i+1}) \text{ where } t_{-1} = t_{n+1} = \text{none}$$

As before assume that: $P(t_0 \mid t_{-1} = \text{none}, t_1) \approx P(t_0 \mid t_1)$ and $P(t_n \mid t_{n-1}, t_{n+1} = \text{none}) \approx P(t_n \mid t_{n-1})$

You can provide either pseudo code, a recursive definition of the algorithm, or a recurrence relation.

Hint: The first step would be to extend the recursive BestScore algorithm given above to read the input from right to left.

Answer: We can define a dynamic programming algorithm that can be used to efficiently compute the best sequence of tags for the given equation. The algorithm is shown below as a recursive function (each call to this function can be stored in a matrix to give a polynomial time algorithm). We assume the input is padded with none's so that we can get probabilities $P(t_0 \mid t_{-1} = \text{none}, t_1)$ and $P(t_n \mid t_{n-1}, t_{n+1} = \text{none})$.

```
function bestScore()
    return NotViterbi(n + 2, none, none, none)

function NotViterbi(i, t_{i-2}, t_{i-1}, t_i)
    if (i − 1 == −1)
        if (t_{i-2}, t_{i-1}, t_i == none, none, none) return 1
        else return 0
    return max_{t_{i-1}} NotViterbi(i − 1, t_{i-3}, t_{i-2}, t_{i-1}) × P(w_{i-1} | t_{i-1}) × P(t_{i-1} | t_{i-2}, t_i)
```

(6) Feature structures:

a. Draw the directed acyclic graph representation for the following feature structure:

$$\begin{bmatrix} \text{cat: VP} \\ \text{agreement: } \boxed{1} \\ \text{subject: } \begin{bmatrix} \text{agreement: } \boxed{1} \begin{bmatrix} \text{person: 3} \\ \text{number: sg} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

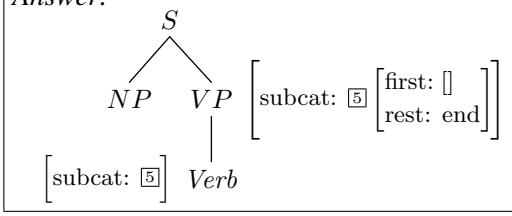
b. The following CFG has been augmented with feature structures. The *subcat* feature captures subcategorization frames for the *Verb* and *cat* represents the category of each non-terminal in the subcat frame. It is important to note that after we obtain a full derivation with this CFG, the *Verb* node in the tree will contain the subcat information for the verb.

$$\begin{array}{lcl} S & \rightarrow & NP \quad VP \begin{bmatrix} \text{subcat: } \boxed{1} \begin{bmatrix} \text{first: []} \\ \text{rest: end} \end{bmatrix} \end{bmatrix} \\ VP \begin{bmatrix} \text{subcat: } \boxed{1} \end{bmatrix} & \rightarrow & Verb \begin{bmatrix} \text{subcat: } \boxed{1} \end{bmatrix} \\ VP \begin{bmatrix} \text{subcat: } \boxed{1} \end{bmatrix} & \rightarrow & VP \begin{bmatrix} \text{subcat: } \begin{bmatrix} \text{first: } \boxed{2} \\ \text{rest: } \boxed{1} \end{bmatrix} \end{bmatrix} \quad X \begin{bmatrix} \text{cat: } \boxed{2} \text{ NP} \end{bmatrix} \end{array}$$

Assume that *X* is a terminal symbol. So the CFG recognizes strings like {*Verb*, *Verb X*, *Verb X X*, . . .}. Provide the parse trees with fully unified feature structures for the following subcategorization frames as represented by the *subcat* feature:

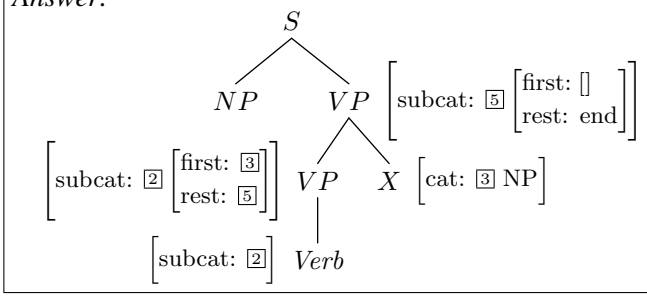
1. no arguments

Answer:



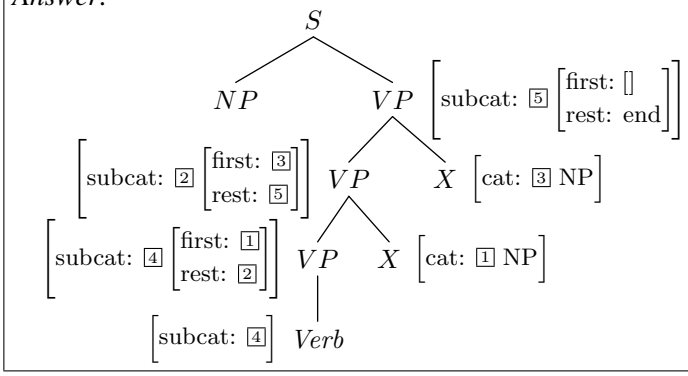
2. NP

Answer:



3. NP NP

Answer:



c. The following CFG rules are added to the grammar above. Describe all the subcategorization frames that can be recognized by this augmented CFG.

$$VP\left[\text{subcat: } \boxed{1}\right] \rightarrow VP\left[\text{subcat: } \left[\begin{array}{l}\text{first: } \boxed{2}\\ \text{rest: } \boxed{1}\end{array}\right]\right] \quad X\left[\text{cat: } \boxed{2}\ \text{PP}\right]$$

$$VP\left[\text{subcat: } \boxed{1}\right] \rightarrow VP\left[\text{subcat: } \left[\begin{array}{l}\text{first: } \boxed{2}\\ \text{rest: } \boxed{1}\end{array}\right]\right] \quad X\left[\text{cat: } \boxed{2}\ \text{S}\right]$$

Answer: All the subcat frames recognized by this CFG can be described by the following regular expression:

Verb $(NP \mid PP \mid S)^*$

(7) (5pts) You are provided with a treebank which consists of three tree *types*: T_1, T_2, T_3. Each tree type occurs with the following frequency:

$$\begin{aligned} 2 \times \quad & T_1 = \text{(S (B a) (C a a))} \\ 1 \times \quad & T_2 = \text{(S (B a a))} \\ 7 \times \quad & T_3 = \text{(S (C a a a))} \end{aligned}$$

a. From the treebank shown above, extract a probabilistic CFG (PCFG) G.
Hint: note that the rule S $\rightarrow$ BC appears in G with probability p_1.

Answer:

$p_1 = \frac{2}{10}$ S $\rightarrow$ BC
$p_2 = \frac{1}{10}$ S $\rightarrow$ B
$p_3 = \frac{7}{10}$ S $\rightarrow$ C
$\frac{p_1}{p_1+p_2} = \frac{2}{3}$ B $\rightarrow$ a
$\frac{p_2}{p_1+p_2} = \frac{1}{3}$ B $\rightarrow$ aa
$\frac{p_1}{p_1+p_3} = \frac{2}{9}$ C $\rightarrow$ aa
$\frac{p_3}{p_1+p_3} = \frac{7}{9}$ C $\rightarrow$ aaa

b. A PCFG is proper if every rule $A \rightarrow \alpha$ in the CFG has probability $P(A \rightarrow \alpha \mid A)$. What is the condition on p_1, p_2 and p_3 that will ensure that PCFG G is proper.

Answer:

$$\sum_i p_i = 1$$

c. Provide the tree set $\mathcal{T}$ for the CFG G.

Answer:

$T_4 =$ (S (B a) (C a a a))
$T_5 =$ (S (B a a) (C a a))
$T_6 =$ (S (B a a) (C a a a))
$T_7 =$ (S (B a))
$T_8 =$ (S (C a a))

$$\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8\}$$

d. Provide the language $\mathcal{L}$ (the set of strings) for the CFG G.

Answer:

$$\mathcal{L} = \{aaa, aa, aaaa, aaaaa, a\}$$

e. Find the parse tree with highest probability according to PCFG G for the input string *aa*

Answer:

0.2 S $\rightarrow$ BC
0.1 S $\rightarrow$ B
0.7 S $\rightarrow$ C
$\frac{2}{3}$ B $\rightarrow$ a
$\frac{1}{3}$ B $\rightarrow$ aa
$\frac{2}{9}$ C $\rightarrow$ aa
$\frac{7}{9}$ C $\rightarrow$ aaa

$P(T_2) = 0.0333$ and $P(T_8) = 0.1555$, so T_8 is the parse tree with highest probability for input string *aa*.

f. Consider a case where for some input string accepted by the PCFG G, the parse tree with highest probability is *not* the one that was observed in the treebank used to create the PCFG in the first place. Briefly (in one sentence) point out the relevant property of PCFGs which explains why such a situation can occur.

Answer: Since a PCFG is "context-free" every context into which a PCFG rule is equivalent. So if a rule $r = B \rightarrow \alpha$ is expanded from a rule $r_1 = A \rightarrow BC$ with a different frequency in the treebank as opposed to a rule $r_2 = X \rightarrow BY$, then the PCFG cannot capture the fact that trees in the treebank with r_1 and r might be more frequent than a tree with r_2 and r.

(8) Probability models:

a. The noisy channel model for spelling correction is shown below, where t is the typo word and c is the correct word:

$$P(c \mid t) = P(t \mid c) \times P(c)$$

Assume that this model is used only for spelling correction with a single error per word for the following two types:

1. a single insertion error where $(x)_c$ is typed as $(xy)_t$, and
2. a single transposition/reversal error where $(xy)_c$ is typed as $(yx)_t$.

Provide an appropriate definition for $P(t \mid c)$ and $P(c)$. Clearly explain the symbols used in your equations.

Answer: We define $P(t \mid c)$ as follows:

$$P(t \mid c) = \begin{cases} P(xy_t \mid x_c) & = & \frac{ins_{[x,y]}}{chars_{[x]}} \\ P(yx_t \mid xy_c) & = & \frac{rev_{[y,x]}}{chars_{[x,y]}} \end{cases}$$

In the above equations, *ins* contains the frequency of a pair of chars x, y in the typo words t which was mistyped by inserting a y when compared to the correct word c. *chars* contains the total frequency of the characters across all typos (including no typos). Similarly, *rev* contains the frequency of a pair of chars y, x in the typo words t which was mistyped by reversing the chars x, y from the correct word c.

Note that in the above equation we assume only one typo per word, and that typo is in addition assumed to be of the two types listed in the question: insertion or reversal.

We define $P(c)$ to be equal to $\frac{chars_{[c]}}{\sum_c chars_{[c]}}$.

b. A verb like *pour* can be used in two different ways:

1. Custom demands that cognac be *poured* from a freshly opened bottle.
2. Salsa and rap music *pour* out of the windows.

Liquid being poured from a bottle is the literal meaning of the verb *pour*, while music pouring out of windows is the non-literal, metaphorical or figurative usage of the verb *pour*.

Explain how this problem can be viewed as a word-sense disambiguation task. Assume you have sufficient training data in which each verb is marked as being either literal or non-literal. Provide a probability model that can be trained using this training data to automatically disambiguate between unseen cases of literal vs. non-literal usages of verbs.

You can assume that all verbs you will observe in test data have been observed at least once in the training data.

Answer:

Let $f_0, \ldots, f_N$ be the features extracted from the sentence. They can be simply all the words in the sentence except for the target verb.

$$\begin{aligned} P(\text{sense}, \text{verb} \mid f_0, \ldots, f_N) &\approx P(\text{sense}, \text{verb}) \times P(f_0, \ldots, f_N \mid \text{sense}, \text{verb}) \\ &= P(\text{sense}, \text{verb}) \times \prod_{i=0}^{N} P(f_i \mid \text{sense}, \text{verb}) \end{aligned}$$