CMPT 413 - SPRING 2011 - FINAL EXAM

Please write down "FINAL" on the top of the answer booklet.
There are seven questions in this final, each with 10 points. Choose any six to answer.
If you solve more than six, you must list the six question numbers to be graded on the 1st page of the question booklet. Else, you will be graded on the first six questions solved.
When you have finished, return your answer booklet along with this question booklet.
Apr 13, 2011

(1) **Finite-state transducers**:

A rewrite rule has the form $A \rightarrow B/L\_R$ where A is replaced with B if LAR is found in the input string. L and R are the left and right contexts respectively and A, B, L, R are regular expressions.

Consider the following pair of ordered rewrite rules that are left to right and iterative:

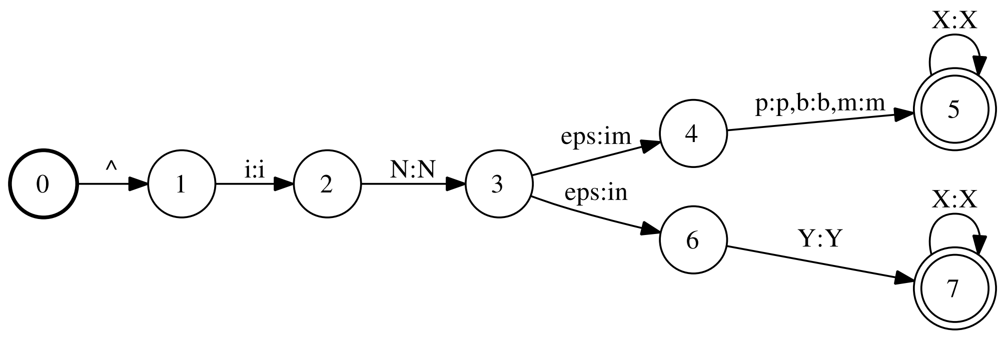
- ˆiN → im / __ (p | b | m)
- ˆiN → in / __

The first rule always applies before the second one, and ˆ stands for the start of the string (matches the start of the word). Notice that the rule is blocked from applying iteratively since it can only apply at the start of the string.

Recall that subsequential transducers are deterministic in the input and may have multiple outputs at the final states. You should use the notion of the alphabet Σ which contains all the valid characters in order to simplify the transitions in your FST. To save time, when writing transitions in your FST, you can output multiple characters when matching a single character in the input, but you must use single characters on the input side of the FST.

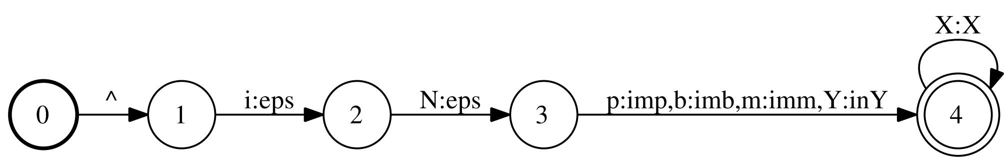
a. Provide a single FST that is not subsequential for the pair of rewrite rules above.

Answer:



b. Provide a subsequential FST for the above pair of rewrite rules.

Answer:



c. Use your FST to provide the output for the input strings: *iNpossible, iNmeasurable, iNtractable, iNelegant*. The solution you provide should match the output of the FST you have provided above.

Answer: impossible, immeasurable, intractable, inelegant

(2) **Rewrite Rules**:

a. Consider the input string *ababb*. Apply the following left to right iterative rewrite rules to this string and show the output as a finite-state machine or regular expression at each step (the output can be ambiguous, which is why we need FSM/regexp output). The output at each step produces the input for the next step. If there are multiple rules in a step, show the output after applying all the rules in the order that the rules are listed.

1. $\epsilon \rightarrow > /$ __ b

   Answer: a > b a > b > b

2. $\epsilon \rightarrow (<_1 \mid <_2) /$ __ (a | b) >

   Answer: $(<_1 \mid <_2)$ a > b $(<_1 \mid <_2)$ a > $(<_1 \mid <_2)$ b > b

3. a > $\rightarrow$ b / $<_1$ __
   b > $\rightarrow$ a / $<_1$ __
   > $\rightarrow \epsilon$ / __

   Answer: $(<_1$ b $\mid <_2$ a) b $(<_1$ b $\mid <_2$ a) $(<_1$ a $\mid <_2$ b) b

4. $<_1 \rightarrow \epsilon$ / b __
   (note that any input string where *b* $<_1$ does not occur is rejected by this rule)

   Answer: $<_2$ a b (b $\mid <_2$ a) (a $\mid <_2$ b) b

5. $<_2 \rightarrow \epsilon$ / !b __
   (! is the negation of the regular expression, it matches if the regexp does not match, including the start of the string. As before, any input string where *b* $<_2$ occurs is rejected by this rule)

   Answer: a b b a b

b. Provide two left to right iterative rewrite rules that concisely capture all the five steps given in Q 2.

Answer: a $\rightarrow$ b / b __ b
b $\rightarrow$ a / b __ b

c. Briefly describe the constraint on rewrite rules that ensure that the rules can be compiled to a single finite-state transducer. Explain what would happen if you apply your left to right iterative rewrite rules from Q 2b to input string *ababb* where the rules were not constrained.

Answer: ab[b]bb ab[a]bb ab[b]bb ...

(3) **Probabilistic Language Models**:

You are given a text of words: $w_1, \dots, w_N$ 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, bigrams w_i, w_{i-1} may have zero counts.

a. Show how we can compute the probability of a sentence or text $P(w_1, \dots, w_N)$ using bigram probabilities $\hat{P}(w_i \mid w_{i-1})$.

Answer:

$$P(w_1, \dots, w_N) = P(w_1) \cdot \prod_{i=2}^{n} \hat{P}(w_i \mid w_{i-1})$$

b. 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 $0 < \delta \leq 1$. 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? Provide a correction and briefly justify why your new formula is correct.

Answer:

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

where $|\mathcal{V}|$ is the size of the vocabulary.

c. Consider the following *simplified* form of the Katz backoff smoothing model where instead of $\hat{P}(w_i \mid w_{i-1})$ we compute $P_{katz}(w_i \mid w_{i-1})$ defined as follows:

$$P_{katz}(w_i \mid w_{i-1}) = \frac{c_{katz}(w_i, w_{i-1})}{\sum_{w_i} c_{katz}(w_i, w_{i-1})}$$

$$c_{katz}(w_i, w_{i-1}) = \begin{cases} r^* & \text{if } r > 0 \text{, where } r = c(w_i, w_{i-1}) \\ \alpha(w_{i-1})\hat{P}(w_i) & \text{if } r = 0 \end{cases}$$

$$\hat{P}(w_i) = \frac{c(w_i)}{N}$$

The bigram frequencies for $r > 0$ are discounted and this discounted probability is used towards $\alpha(w_{i-1})$ which is used to weight the backoff to the unigram probability. The discounting of the bigram frequencies $r = c(w_i, w_{i-1})$ is defined as the Good-Turing estimate r^* based on the number of observations of a particular frequency r denoted as n_r. For each r count, we replace it with a Good-Turing estimate r^* of this count defined as:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

Provide a definition for $\alpha(w_{i-1})$ so that $P_{katz}(w_i \mid w_{i-1})$ remains a well-formed probability function. This means that $\alpha(w_{i-1})$ has to be chosen such that the total number of counts remains unchanged:

$$\sum_{w_i} c_{katz}(w_i, w_{i-1}) = \sum_{w_i} c(w_i, w_{i-1})$$

Answer:

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : r = c(w_i, w_{i-1}) > 0} r^*}{\sum_{w_i : c(w_i, w_{i-1}) = 0} \hat{P}(w_i)} = \frac{1 - \sum_{w_i : r = c(w_i, w_{i-1}) > 0} r^*}{1 - \sum_{w_i : c(w_i, w_{i-1}) > 0} \hat{P}(w_i)}$$

d. TrueCasing is the process of taking text with missing or unreliable case information, e.g. if the input looks like:

```
as previously reported , target letters were issued last month to
michael milken , drexel 's chief of junk-bond operations ; mr. milken 's
brother lowell ; cary maultasch , a drexel trader ; james dahl , a
drexel bond salesman ; and bruce newberg , a former drexel trader .
```

Then the output of the TrueCasing program should be:

```
As previously reported , target letters were issued last month to
Michael Milken , Drexel 's chief of junk-bond operations ; Mr. Milken 's
brother Lowell ; Cary Maultasch , a Drexel trader ; James Dahl , a
Drexel bond salesman ; and Bruce Newberg , a former Drexel trader .
```

Our task is to find the most likely TrueCase output $W_1^*, \ldots, W_n^*$ given the input lowercase input $w_1, \ldots, w_n$:

$$W_1^*, \ldots, W_n^* \approx \operatorname*{argmax}_{W_1,\ldots,W_n} \prod_{i=1}^{n} \underbrace{P_t(w_i \mid W_i)}_{\textit{translation probability}} \cdot \underbrace{P_b(W_i \mid W_{i-1})}_{\textit{bigram language model}}$$

Briefly explain how you can use Hidden Markov Models (HMM) to solve this problem. You should indicate how the above problem can map into the various components of an HMM.

Answer: The observations will be the lowercase words w_i since that is what is observed. The most likely TrueCase word W_i is chosen to match the observed w_i using the translation probability. Therefore, the translation probability $P_t(w_i \mid W_i)$ becomes the emission probability for the HMM. Each succeeding TrueCase word W_{i+1} is also chosen based on the previous TrueCase word W_i and so the bigram language model probability $P_b(W_i \mid W_{i-1})$ becomes the transition probability for the HMM.

e. The output $W_1^*, \ldots, W_n^*$ can be obtained using the Viterbi algorithm which computes the score for the best sequence of TrueCase words:

$$\text{Viterbi}[i+1, W_{i+1}] = \max_{W_i \in \mathcal{V}} (\text{Viterbi}[i, W_i] \times P_t(w_{i+1} \mid W_{i+1}) \times P_b(W_{i+1} \mid W_i))$$

where the max W_i is picked from out of the vocabulary $\mathcal{V}$ of all TrueCase words for each W_{i+1} which is also taken from the vocabulary $\mathcal{V}$ of TrueCase words. We initialize the recursion with the value for Viterbi$[1, W_i]$ for all $W_i \in \mathcal{V}$.

If the number of lowercase words is $|v|$ and the number of TrueCase words is $|\mathcal{V}|$ then the time complexity is $O(|v| \cdot |\mathcal{V}|^2)$. Describe briefly how to improve the average case time complexity of this algorithm so that it does not *always* take time proportional to the square of the vocabulary size!

Answer: Instead of ranging over the entire vocabulary, only use the W_i and W_{i+1} words that actually have a non-zero probability for $P_t(w_{i+1} \mid W_{i+1})$ and $P_b(W_{i+1} \mid W_i)$. We observe very few W_{i+1} for any given W_i so this will speed up the average case considerably.

(4) **Tagging and Chunking**

Tagging is the process of providing a sequence of tags $T_1, \ldots, T_n$ to a sequence of observations $O_1, \ldots, O_n$. We can describe the tagging task using the equation:

$$T_1^*, \ldots, T_n^* = \operatorname*{argmax}_{T_1,\ldots,T_n} P(T_1, \ldots, T_n \mid O_1, \ldots, O_n)$$

a. Define a noisy channel model for tagging by applying Bayes rule to $P(T_1, \ldots, T_n \mid O_1, \ldots, O_n)$ and providing the resulting equation.

Answer:

$$\begin{aligned} T_1^*, \ldots, T_n^* &= \operatorname*{argmax}_{T_1,\ldots,T_n} P(T_1, \ldots, T_n \mid O_1, \ldots, O_n) \\ &= \frac{P(T_1, \ldots, T_n) \cdot P(O_1, \ldots, O_n \mid T_1, \ldots, T_n)}{P(O_1, \ldots, O_n)} \\ &\approx P(T_1, \ldots, T_n) \cdot P(O_1, \ldots, O_n \mid T_1, \ldots, T_n) \end{aligned}$$

b. Define a separate unigram, bigram, and trigram model *over tags* by modifying the equation derived in Q (4a) – it should continue to include the probability of emission of observations. Assume suitable padding of extra observations and tags in order to simplify the form of the equation.

Answer:

$$\begin{gathered} P(T_1, \ldots, T_n) \cdot P(O_1, \ldots, O_n \mid T_1, \ldots, T_n) = \\ \prod_{i=1}^{n} P(T_i) \cdot \prod_{i=1}^{n} P(O_i \mid T_i) \text{ (unigram)} \\ \prod_{i=1}^{n} P(T_i \mid T_{i-1}) \cdot \prod_{i=1}^{n} P(O_i \mid T_i) \text{ (bigram)} \\ \prod_{i=1}^{n} P(T_i \mid T_{i-2}, T_{i-1}) \cdot \prod_{i=1}^{n} P(O_i \mid T_i) \text{ (trigram)} \end{gathered}$$

c. Combine the unigram, bigram and trigram models you provide in Q (4b) into a single model for tagging using interpolation. Provide the equation for the interpolated model.

Answer:

$$\begin{aligned} & P(T_1, \ldots, T_n) \cdot P(O_1, \ldots, O_n \mid T_1, \ldots, T_n) = \\ & \quad \prod_{i=1}^{n} P(O_i \mid T_i) \cdot \left(\lambda_1 \prod_{i=1}^{n} P(T_i) + \lambda_2 \prod_{i=1}^{n} P(T_i \mid T_{i-1}) + \lambda_3 \prod_{i=1}^{n} P(T_i \mid T_{i-2}, T_{i-1}) \right) \\ & = \prod_{i=1}^{n} P(O_i \mid T_i) \cdot \prod_{i=1}^{n} (\lambda_1 P(T_i) + \lambda_2 P(T_i \mid T_{i-1}) + \lambda_3 P(T_i \mid T_{i-2}, T_{i-1})) \end{aligned}$$

where, $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$.

d. If the observations O_i are words from a vocabulary $\mathcal{V}$ and the tags T_i are taken from a set of part of speech tags $\mathcal{T} = \{$*Noun, Verb, Prep, Det*$, \ldots\}$ then we can use the tagging model shown above in order to solve the *part of speech tagging* task.

For each of the following tasks show how you can define the set of observations $\mathcal{V}$ and the tag set $\mathcal{T}$ (provide the full set of tags to be used) in order to solve that task using the above model of tagging.

1. Noun phrase chunking.

   Answer: $\mathcal{V}$ are words and $\mathcal{T} = \{B, I, O\}$ for Begin, Inside, or Outside a noun phrase chunk.

2. Word segmentation (for languages like Chinese which typically do not use spaces to separate words).

   Answer: $\mathcal{V}$ are characters in Chinese and $\mathcal{T} = \{B, I\}$ for Begin, or Inside a word.

3. Named entity recognition (find person, organization or location names in text).

   Answer: $\mathcal{V}$ are words and $\mathcal{T}$ = {B-PER, I-PER, B-ORG, I-ORG, B-LOC, I-LOC, O} for Begin or Inside (PERson, ORGanization, or LOCation) or Outside any NER.

4. FAQ segmentation (for a given document of sentences, label each sentence as a question or answer sentence assuming the typical structure of a Frequently Asked Questions document).

   Answer: $\mathcal{V}$ are sentences and $\mathcal{T} = \{Q, A\}$ for Question, or Answer sentence.

(5) **Context-free Grammars**:

For the CFG G given below:

$$\begin{aligned} S &\rightarrow A \mid c \\ A &\rightarrow B\, a \\ B &\rightarrow b\, S \end{aligned}$$

a. What is the language $L(G)$?

Answer: $b^n c a^n : n \geq 0$

b. What is the tree set $T(G)$? You can use any convenient short-hand notation to represent an infinite set of trees.

Answer:

```
T(G) = {
(S (A (B b
         (S (A (B b
                  (S ... c))
               a))
      a)))
}
```

or:

```
T(G) =
{ (S c) ,
  (S (A (B b (S c)) a)),
  (S (A (B b (S (A (B b (S c)) a)) a))),
  ...
}
```

c. Assign probabilities to each rule in the CFG above so that for each string $w \in L(G)$:

$$P(w) = exp\left(\frac{|w| - 1}{2} * ln(0.3) + ln(0.7)\right)$$

where, $|w|$ is the length of string w, exp is exponentiation, and ln is log base e.

Answer:

```
0.3 S -> A
0.7 S -> c
1.0 A -> B a
1.0 B -> b S
```

d. Convert the PCFG from the answer to Q (5c) into Chomsky Normal Form (CNF). The CNF grammar must also have the right probabilities.

Answer:

```
0.7 S -> c
0.3 S -> B A'
1.0 B -> B' S
1.0 B' -> b
1.0 A' -> a
```

e. Provide a leftmost derivation using your CNF grammar for the input string *bbcaa*. The derivation should include probabilities for each step – you can keep them as multiples of individual probabilities if you wish.

Answer:

```
S => BA' (0.3)
=> B'SA' (0.3 * 1.0)
=> bBA'A' (0.3)
=> bB'SA'A' (0.3 * 0.3)
=> bbSA'A' (0.09)
=> bbcA'A' (0.09 * 0.7)
=> bbcaA' (0.027)
=> bbcaa (0.027)
```

f. Briefly explain why conversion into CNF implies that we can parse any CFG in time $O(G^2 n^3)$ where G is the number of non-terminals in the CFG, and n is the length of the input string. (A short 1–2 line answer will suffice).

Answer: The rules of type $A \rightarrow a$ can be used to fill in each span of length one, and to find a start symbol spanning the entire string we use rules of type $A \rightarrow BC$ to recursively find a span i, j if we have previously found a span from i, k for B and k, j for C. Since B and C span over all non-terminals we need G^2 time, and since we need to find all spans i, j with every k in between we need n^3 time.

(6) **Parsing Context-Free Grammars**:

The following set of states were computed using the Earley parsing algorithm for context-free grammars (CFGs). The states shown are an arbitrary subset of a larger set of states stored in the chart data structure while parsing an input string. The actual context-free grammar and the input string are not relevant for answering this question.

- chart[4]:
  1. $(NP \rightarrow Det\ N\ \bullet\ , [2,4])$
  2. $(NP \rightarrow NP\ \bullet\ PP, [2,4])$
  3. $(S \rightarrow SBJ\ VP\ \bullet\ , [0,4])$
  4. $(VP \rightarrow VP\ \bullet\ PP, [1,4])$
- chart[7]:
  1. $(NP \rightarrow Det\ N\ \bullet\ , [5,7])$
  2. $(PP \rightarrow P\ NP\ \bullet\ , [4,7])$
  3. $(NP \rightarrow NP\ \bullet\ PP, [5,7])$
  4. $(NP \rightarrow NP\ PP\ \bullet\ , [2,7])$
  5. $(VP \rightarrow VP\ PP\ \bullet\ , [1,7])$
  6. $(PP \rightarrow\ \bullet\ P\ NP, [7,7])$
  7. $(VP \rightarrow V\ NP\ \bullet\ , [1,7])$
  8. $(NP \rightarrow NP\ \bullet\ PP, [2,7])$
  9. $(S \rightarrow SBJ\ VP\ \bullet\ , [0,7])$
  10. $(VP \rightarrow VP\ \bullet\ PP, [1,7])$
- chart[8]:
  1. $(PP \rightarrow P\ \bullet\ NP, [7,8])$
  2. $(NP \rightarrow\ \bullet\ NP\ PP, [8,8])$
  3. $(NP \rightarrow\ \bullet\ Det\ N, [8,8])$
- chart[10]:
  1. $(NP \rightarrow Det\ N\ \bullet\ , [8,10])$
  2. † $(PP \rightarrow P\ NP\ \bullet\ , [7,10])$

Consider state number (2) from the list of states in chart[10] (marked with †).

a. Provide a list of states that the **completer** step would attempt to **enqueue** into the chart taking state number (2) from chart[10] as input.

Answer:
1. $(NP \rightarrow NP\ PP\ \bullet\ , [5,10])$
2. $(NP \rightarrow NP\ PP\ \bullet\ , [2,10])$
3. $(VP \rightarrow VP\ PP\ \bullet\ , [1,10])$

b. For each state provided in your answer, provide the index i indicating that the **completer** step will attempt to **enqueue** the state into chart[i].

Answer: $i = 10$ for all states inserted

c. Briefly list two reasons why the Earley parsing algorithm for CFGs is superior to the CKY algorithm.

Answer: (1) No need to convert to CNF, (2) best case ≠ worst case time complexity, (3) top-down prediction combined with bottom-up completion leads to fewer edges in the chart.

d. You are assigned the task of updating a working Earley parser which takes a regular context-free grammar and making it work with a probabilistic CFG (PCFG). Every rule in the PCFG $A \rightarrow \alpha$ also comes with a probability p. For this question we will denote the PCFG rule as $p : A \rightarrow \alpha$. We will write the Earley state which is a dotted rule $A \rightarrow \alpha \bullet \beta$ with span i, j with probability p using the notation $p : (A \rightarrow \alpha \bullet \beta, [i, j])$. The Greek symbols $\alpha, \beta, \ldots$ refer to a (possibly empty) sequence of terminal and non-terminal symbols.

Your job is to modify the Earley parser in order to compute the most likely derivation. The main functions of the Earley parser are listed below as pseudo-code. Provide the necessary values (marked with **?** in the code) that will ensure that the Earley parser will recover the most likely derivation.

For each function write down the name of the function and then write down the complete single line that contains the correct value for **?** in that line. You do not need to copy the entire pseudo-code, only write down the line with the completed value for **?**.

- enqueue(p_1 : state, j):

```
input: state = $(A \rightarrow \alpha \bullet \beta, [i, j])$ with probability $p_1$
input: $j$ (insert state into chart[$j$])
if state not in chart[$j$] with any probability then
    chart[$j$].add($p_1$ : state)
else if $p_2$ : state is in chart[$j$] then
    chart[$j$].remove($p_2$ : state)
    chart[$j$].add(? : state)
end if
```

Answer: chart[j].add(max(p_1, p_2) : state)

- predictor(p_1 : state):

```
input: state = $(A \rightarrow \alpha \bullet C\beta, [i, j])$ with probability $p_1$
for all rules $p_2 : C \rightarrow \gamma$ in the grammar do
    newstate = ? : $(C \rightarrow \bullet \gamma, [j, j])$
    enqueue(newstate, j)
end for
```

Answer: newstate = $p_2 : (C \rightarrow \bullet \gamma, [j, j])$

- scanner(p_1 : state, tokens):

```
input: state = $(A \rightarrow \alpha \bullet a\beta, [i, j])$ with probability $p_1$
input: tokens (list of input tokens to the parser)
if tokens[$j$] == $a$ then
    newstate = ? : $(A \rightarrow \alpha\, a \bullet \beta, [i, j + 1])$
    enqueue(newstate, j+1)
end if
```

Answer: newstate = $p_1 : (A \rightarrow \alpha\, a \bullet \beta, [i, j + 1])$

- completer(p_1 : state):

```
input: state = $(A \rightarrow \gamma \bullet, [j, k])$ with probability $p_1$
for all rules $p_2 : X \rightarrow \alpha \bullet A\beta, [i, j]$ in chart[$j$] do
    newstate = ? : $(X \rightarrow \alpha A \bullet \beta, [i, k])$
    enqueue(newstate, k)
end for
```

Answer: newstate = $p_1 \times p_2 : (X \rightarrow \alpha A \bullet \beta, [i, k])$

(7) **Feature unification**:

a. Provide all pairwise subsumption relations $\sqsubseteq$ for the following feature structures.

1. $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \end{bmatrix} \end{bmatrix}$

2. $\begin{bmatrix} \text{g:} \begin{bmatrix} \text{i: b} \end{bmatrix} \end{bmatrix}$

3. $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \end{bmatrix} \\ \text{g:} \begin{bmatrix} \text{i: b} \end{bmatrix} \end{bmatrix}$

4. $\begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

5. $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g:} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \end{bmatrix}$

6. $\begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

Answer:

1. $1 \sqsubseteq 3, 4, 5, 6$
2. $2 \sqsubseteq 3, 4, 5, 6$
3. $3 \sqsubseteq 4, 5, 6$
4. $4 \sqsubseteq 6$
5. $5 \sqsubseteq 6$

b. Define unification of two feature structures $A \sqcup B$ by using only the subsumption relation $\sqsubseteq$.

Answer: $A \sqcup B = C$ where $A \sqsubseteq C$ and $B \sqsubseteq C$ and there is no D such that $D \sqsubseteq C$ and $C \not\sqsubseteq D$.

c. Provide the output of unifying the following pairs of feature structures:

1. $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{g:} \begin{bmatrix} \text{i: b} \end{bmatrix} \end{bmatrix}$

Answer: $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \end{bmatrix} \\ \text{g:} \begin{bmatrix} \text{i: b} \end{bmatrix} \end{bmatrix}$

2. $\begin{bmatrix} \text{g:} \begin{bmatrix} \text{i: b} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

Answer: $\begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

3. $\begin{bmatrix} \text{f:} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g:} \begin{bmatrix} \text{h: a} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

> *Answer:* $\begin{bmatrix} \text{f: } \boxed{1} \begin{bmatrix} \text{h: a} \\ \text{i: b} \end{bmatrix} \\ \text{g: } \boxed{1} \end{bmatrix}$

d. Provide all the valid strings that can be recognized by the following feature-based CFG:

$$\begin{array}{lcl}
S & \rightarrow & NP_{\left[\text{case: nominative}\right]}\ VP \\
VP & \rightarrow & V\ NP_{\left[\text{case: accusative}\right]} \\
V & \rightarrow & \textit{saw} \\
NP_{\left[\text{case: } \boxed{1}\right]} & \rightarrow & \textit{he}_{\left[\text{case: } \boxed{1} \text{ nominative}\right]} \\
NP_{\left[\text{case: } \boxed{1}\right]} & \rightarrow & \textit{him}_{\left[\text{case: } \boxed{1} \text{ accusative}\right]} \\
NP_{\left[\text{case: } \boxed{1}\right]} & \rightarrow & \textit{John}_{\left[\text{case: } \boxed{1} \text{ nominative} \mid \text{accusative}\right]}
\end{array}$$

> *Answer:*
> - *he saw him*
> - *John saw him*
> - *he saw John*
> - *John saw John*