Computational Linguistics Midterm
Jean Mark Gawron
San Diego State University
Linguistics 581
March 24, 2003

1 FSAs and Regular Languages

1.1 Problem 1

Draw an FSA corresponding to each of the following regular expressions (assume the alphabet is a,b):
(1.) (aba)+b
(2.) aba*b*
(3.) (a(aa|bb)*)+
In addition to drawing it, give the transition table for state 3.

1.2 Problem 2

Consider the following FSA, defined on the alphabet a,b:

State	Final?	a	b
0	Y	1	
1	N	1	2
2	N	2	0
3	N	{0, 2}	0

Answer the following questions about this machine:
(1.2.1) Which of the following strings are accepted by the machine?
- (a) abab
- (b) aabb
- (c) bbaabbab
- (d) abababbbbb

(1.2.2) Does this machine have a sink state? How do you know?
(1.2.3) Is this machine non-deterministic? What state or states tells you that?
(1.2.4) Draw the FSA for the complement language of this machine. [Don't give transition table; draw the machine.]

1.3 Problem 3

In this problem you are asked to define a language **L** by drawing an FSA that accepts L.

> L includes any string from the alphabet $\{0, 1\}$ that has any number of 0's and such that the number of 1's is exactly divisible bvy 3 (0 1's counts as divisible by 3).

Note that the 1's and 1's should be able to be interspersed. Allow:

> 000000111111

as well as

> 100101010110

2 Viterbi Algorithm

Figure 1 shows another toy HMM to be used for tagging.

1. The numbers on the transition are transition probabilities. The .4 on the transition from state s to state n means you can make that transition with a probability of .4, that is, P(n | s) = .4

2. Best path for this problem means highest probability.

3. The difference between this HMM and the one in the homework problem is that there is both a start and an end state. All observation start at the start state s and end start state e. There are no observations at s and e (or you can think of the observations being the fictional words START and END which always start and end sentences). Tagging the first word in a sentence as a noun means transitioning from the start state "s" to the "n" state.

4. The observation probabilities of our HMM are as follows:

w | v		w | a		w | n	
fat	.1	fat	.8	fat	.2
dog	.1	dog	.1	dog	.6
smiles	.8	smiles	.1	smiles	.2

To make sure we understand how to compute a path probability, let us compute the probability of one tagging of the input

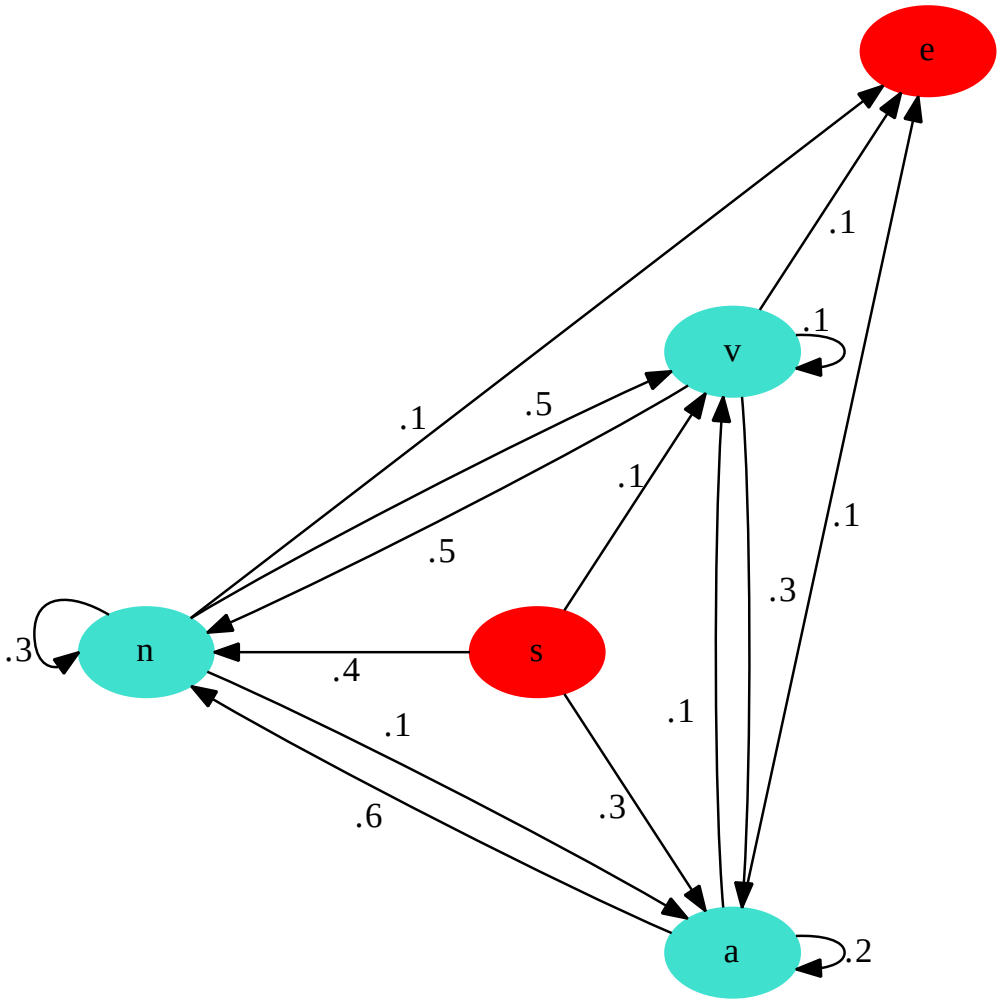

Figure 1: FSA for Viterbi Problem

dog smiles fat

The tagging we will use is start-v-n-a-end. That is:

$$\underset{\text{p(v | s)}}{.1} * \underset{\text{s} \rightarrow \text{v}}{.1} * \underset{\text{v} \rightarrow \text{n}}{.5} * \underset{\text{n} \rightarrow \text{a}}{.2} * \underset{\text{a} \rightarrow \text{e}}{.1} * .8 * .1$$
$$\text{p(dog | v)} * \text{p(n | v)} * \text{p(smiles | n)} * \text{p(a | n)} * \text{p(fat | a)} * \text{p(e | a)}$$

To help you complete the problem, here are Viterbi and backtrace table templates appropriate for the given input. The Viterbi values for t=0 the backtrace values for t=1 have generously been filled in.

Viterbi

e	0.0				
v	0.0				
n	0.0				
a	0.0				
s	1.0				
	0 start	1 fat	2 dog	3 smiles	4 end

Backtrace

e					
v		s			
n		s			
a		s			
s					
	0 start	1 fat	2 dog	3 smiles	4 end

Your task in this problem is to use the Viterbi algorithm to find the **best** path through the HMM in Figure 1 for the following input:

fat dog smiles

You need to fill out the Viterbi table with appropriate Viterbi numbers. You also need to fill out the backtrace table with the states transitioned from on best paths. Finally, you will need to interpret your final input by stating what the best path through the HMM is, given this input.

3 CKY parsing

1. Modify the pseudocode for the CKY algorithm given in Figure 13.10 to make it a parser. This means you need to add a "backtrace" table

like the one you used for the Viterbi algorithm. Thus every time you add a category to the CKY table, you need to add a corresponding record in the backtrace table recording the critical information of how that category was built.

2. Modify the pseudocode for the CKY algorithm given in Figure 13.10 to allow unit productions.

3. Finally you can do EITHER of the following two things:

   (a) You can implement the CKY algorithm in Python (use the CNF grammar given in the chapter to test; use NLTK grammars as your data structure for storing grammars).

   (b) Or you can create a backtrace table for the completed CKY chart in Figure 13.9. Use the following record format:

   (mother, dtr1, dtr2, midpoint)

   So, when adding an np to the CKY table, that was built with a det that ended at index 1 and a nom that started at index 1, here is what you add to the corresponding cell in the backtrace table:

   (np, det, nom, 1)

   For this task you hand in the backtrace table and answers to the following questions:

   (a) How many parses does the example in 13.9 have?

   (b) What computation do you have to perform on the CKY table and the backtrace table to retrieve all the parse trees? Your answer can be in pseudocode or in the form of a Python program.