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):
(1.1.1) \((aba)^+b\)
(1.1.2) \(aba*b\)
(1.1.3) \((a(aa \mid bb)^*)^+\)
In addition to drawing it, give the transition table for 3.

1.2 Problem 2

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

<table>
<thead>
<tr>
<th>State</th>
<th>Final?</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Y</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>N</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>N</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>N</td>
<td>{0, 2}</td>
<td>0</td>
</tr>
</tbody>
</table>

Answer the following questions about this machine:
(1.2.1) Which of the following strings are accepted by the machine?
(a) aaaabb
(b) abaaaaab
(c) abab
(d) abaababab
(1.2.2) Does this machine have a sink state?
(1.2.3) Is this machine non-deterministic?
(1.2.4) Is this machine minimal? Why not? Give the transition table for a deterministic minimal version, if it differs from the table given.
(1.2.5) Give the transition table for the complement language of this machine.
2 Transducers, Spelling and Phonology

Read Sections 3.5.3 and 3.5.5 in *Finite State Morphology* carefully.

Consider the English e-drop spelling rule relating forms like the following:

1. realize + ed = realized
2. realize + ing = realizing
3. realize + er = realizer
4. hope + ing = hoping
5. hop + ing = hopping
6. face + ing = facing
7. flee + ing = fleeing
8. see + ing = seeing
9. die + ing = dying

Using xfst construct a rule that handles this.

You need to handle the suffixes illustrated above, *ing, ed, er*. Use the following definition:

```plaintext
define Suf [ %+Prog+{ing} | %+Pres3PSg:s | %+Past:+{ed} | %+Agentive:+{er} ];
```

You should turn in an xfst script file AND an xfst data file with some words I can do an “apply up” to.

Note: a script file is load with the command source *filename* and should have the extension .xfst. A text file is just a list of forms either from the up language or the down language that the transducer should transduce. It is just a list of words, one to a line. For example, if the file *my_answer.xfst* contained your rules for this problem and the file *words.txt* contained:

```
hoping
hopping
realizer
realizing
```

you could do:
<<Defining Sets>>
defined Vowel: 172 bytes. 2 states, 5 arcs, 5 paths.
defined Cons: 540 bytes. 4 states, 21 arcs, 19 paths.
<<Defining Root>>
defined Root: 500 bytes. 11 states, 11 arcs, 3 paths.
<<Defining Suf>>
defined Suf: 396 bytes. 6 states, 9 arcs, 5 paths.
<<Defining Lexicon>>
defined Lexicon: 780 bytes. 16 states, 25 arcs, 15 paths.
<<Defining Rules>>
defined Rules: 4.7 Kb. 23 states, 332 arcs, Circular.
<<Defining Final Regex>>
848 bytes. 18 states, 28 arcs, 15 paths.
Closing file my_answer.xfst...
xfst[1]: apply up < test.txt
Opening file test.txt...

hopping
hope+Prog

hopping
hop+Prog

realizer
realize+Agentive

realizing
realize+Prog
Closing file test.txt...
xfst[1]:

With some comments that indicate 6 examples (besides those above) that work. The file should be complete and usable. That is, it should work when loaded into the xfst. All of the above examples should also work, of course.

Your transducer MUST rule out:

    hope + ing = hopeing
    realize + ed = realized
    hop + ing = hoping

And
apply up hoping
should give only one answer.

3 Viterbi Algorithm

Use the Viterbi algorithm to find the cheapest path through the weighted FSA in Figure 1 for the following inputs:

1. ababa
2. abaaba

Note:
1. The colon symbol ("\:"") in this case separates the symbol accepted on a transition from its cost (this is not a transducer). "a:4" on a transition means you can accept an "a" on transition at a cost of 4.

2. Cheapest cost for this problem means lowest number.

3. Compute path costs by **adding** transition costs (not multiplying them as we did with probabilities).

   You may solve this problem one of two ways.
   (1) You can produce the Viterbi table by hand showing the computation of the cheapest path for all states and times though the last input observation. You must also show your Viterbi computation for the state you end up on on the cheapest path and one other state at time t=5 on the input ababa.
   (2) You can use this piece of python code at

   http://www-rohan.sdsu.edu/~gawron/compling/new_midterm/additive_viterbi.py

   to produce the table for you. Your task here would be to insert the right HMM into the code. There is an example HMM there now. In fact, the example is the one in Figure 2. You can then run the algorithm to
produce the Viterbi table for you. Turn in the table produced as well as
the modified code. You do NOT need to show the Viterbi computation
for any states if you solve the problem this way.

The Viterbi table solutions for this problem must be arranged as in your
textbook examples; the columns are observations (one per moment in time),
the rows are states in the machine. For the first input above:

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>∅</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Each cell \((i, j)\) is filled with the viterbi value, the cost of the cheapest way
of getting to state \(j\) at time \(i\). For example the values for \(t=0\) when there
is input observation signified by \(∅\), is:

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>∅</td>
<td>a</td>
<td>b</td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

4 A final Thought

Good luck!