CSCI 5832 Spring 2010 Exam 1

Name: _____

On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work. _____.

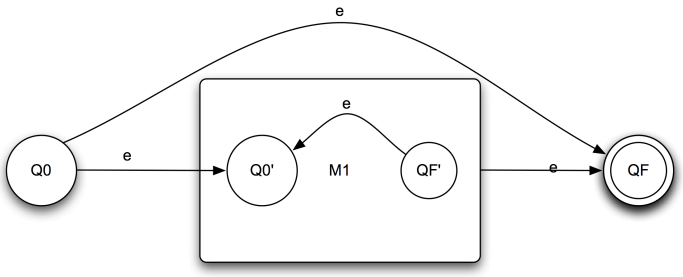
1. **(5 points) True/False:** The monitor in this classroom is too small.

True. At least given all the squinting I'm seeing.

2. **(5 points) True/False:** Any language accepted by a non-deterministic finite-state automaton can be captured by an equivalent regular expression.

True.

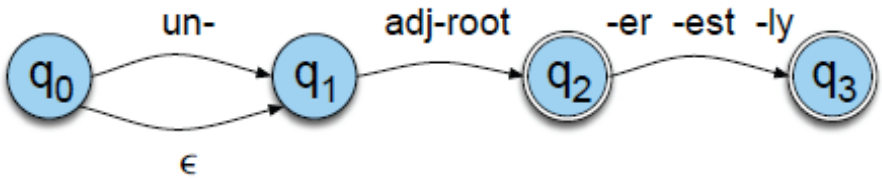
3. **(5 Points)** The following figure shows an FSA M1 with a single start and accept state (assume there are other states and transitions unseen). Assume M1 accepts the language L1. Construct a new machine (just draw on this one) that accepts the language L1*.



4. **(10 Points)** Consider the following FSA designed to capture derivational facts about English adjectives (think about a word like *happy*). What's wrong with this machine? And what would be the first step in designing a fix? Hint: Don't worry about orthographics/spelling changes (like the change of "y" to "i").

It works ok for happy. But it overgenerates for most English adjectives. That is, it accepts forms like ungreen, or greenly.

The usual fix is to divide the base class (adj-root) up into sub-classes and replace the adj-root with the subclass that works correctly for the given FSA and then create new FSAs that correspond to the other sub-classes.



5. **(5 points)** Assume in the context of word segmentation (think of your HW), that you have access to correct answers to each of the examples in a test set. That is, for each input in the test set you have a correct segmentation. Given the output for your system, suggest a metric for evaluating your system (hint: you might think in terms of an algorithm you've seen in the text).

There are any number of good answers here. One reasonable one is to use the minimum edit distance between the system and the reference answer. You'd want to average and length normalize the values over the test set.

Another simpler answer is just accuracy at predicting the segmentation points, but that's pretty crude.

Another answer that I gave credit for is to use perplexity. That's kind of a reach and very indirect given that we know the right answers.

6. **(10 Points)** Assuming you had access to bigram counts (say from the Google corpus) describe a way to use a probabilistic language model to improve the performance of your HW. (Don't worry about implementation details, just describe what the computation is). You might use two segmentations of #fortherecord as an example.

You'd want to argmax **P(segmentation|input)** using a bigram language model to get the P() for each possible segmentation. To make this work you might need to do some smoothing to get rid of zero count bigrams.

7. **(5 points)** Describe a problem for the kind of probabilistic model aproach that is specifically posed by examples like #thereason. (Hint: write out the possible segmentations of this example for inspiration).

The segmentations generated in this approach won't all have the same length. For example "the reason" will involve the calculation

P(the | #)*P(reason | the)

while the segmentation "there as on" or "there a son" involve 3 bigrams

P(there | #)*P(a | there)P(son | a)

Multiplying three probabilities (numbers between 0 and 1) will normally yield a smaller number than multiplying two numbers (in roughly the same range of values). Short story is that language models have a problem comparing hypotheses of differing lengths.

8. **(5 points) True or False:** The *Viterbi* algorithm computes the maximum probability state sequence through an HMM given an input sequence.

True.