Homework #1: Finite State Techniques for Natural Language Processing

**Assigned:** January 24, 2012  
**Due:** To be determined

This assignment is mostly about teaching you how to use finite state techniques such as acceptors/generators and transducers in several important NLP tasks such as morphological analysis and information extraction. The `lextools`, `fsmtools`, and `dot` packages are described in their respective home pages. See the course notes for the URLs. Make sure that you have all appropriate tutorials and man pages handy.

**IMPORTANT:** Start early as this is a long assignment.

---

1 (10 points) Go to `/data0/class/nlpw12/hw1` and look at the files in it. You will need to copy some of these files to your `hw1` directory (e.g., `/data2/users/class/smith/hw1`). Make sure that the following environment variables are exported:

```bash
export PATH=$PATH:/data0/tools/dot/bin
export PATH=$PATH:/data0/tools/fsmtools/fsm-4.0/bin
export PATH=$PATH:/data0/tools/lextools/bin
export LEXFRAGDIR=/data0/tools/lextools/lib/Fragments
```

Submit the output of the following commands:

- `cat NPL.isyms`
- `cat NPL.osyms`
- `cat NPL.txt`
- `cat input.txt`
- `fsmcompile -t NPL.txt > NPL.fst`
- `fsmprint -i NPL.isyms -o NPL.osyms NPL.fst`
- `fsmdraw -i NPL.isyms -o NPL.osyms NPL.fst > NPL.dot`
- `dot -Tps NPL.dot > NPL.ps`
- `fsmcompile input.txt > input.fsm`
- `fsmprint -i NPL.isyms input.fsm`
- `fsmcompose input.fsm NPL.fst > output.fsm`
- `fsmprint -i NPL.isyms -o NPL.osyms output.fsm`
- `fsmdraw -i NPL.isyms -o NPL.osyms input.fsm > input.dot`

---

1
2 (10 points) Create a new set of acceptors/generators and transducer(s) to convert arbitrary nouns from singular to plural. Make sure to cover at least the following examples: regular nouns (e.g., cat/cats), semi-regular nouns (e.g., fox/foxes, boss/bosses) but also child/children, sheep/sheep, mouse/mice, and goose/geese.

In this exercise, don’t hard code the input nouns. Write your automata in a way to handle arbitrary input strings. Obviously, if your input is bad (e.g., ”zzuk”), your output will also be bad (e.g., ”zzuks”).

Submit a printout indicating that composition was successful and show the plurals for all of the nouns above plus five more examples, including other regular and semi-regular nouns.

3 (10 points) Write a finite-state acceptor/generator \( A \) for the regular language \((CV)^*\) where \( C \) is any consonant in English (including ”y”) and \( V \) is any English vowel. An example of a string produced by \( A \) is ”kubigalame”.

Using \texttt{fsmrandgen}, generate 100 strings using this automaton. Compute the average length of each string.

You can use a command like this:

\[
\texttt{fsmrandgen A.fst | fsmprint -i A.isyms -o A.osyms | fsmstring}
\]

to produce random strings. If you don’t get an output or you always get the same output, somethings must be wrong with your automaton.

4 (10 points) Write a weighted automaton for the language in the previous exercise. This automaton must have the following property - the length of the strings that it produces must be approximately 8 (that is, CVCVCCCV) on average. What weights do you need to put on the different transitions to achieve this?

5 (10 points) Go to /data0/class/nlp/lextools. You may have to copy some of these files to your directory. Look at the Makefile. It contains a large number of examples of running the \texttt{lextools} package. Study the Makefile carefully as it will be helpful to do the next few exercises.

Submit the output of the following command:

\[
\texttt{lexcompre -l example.lab -s "1,100" | fsmcompose - numbers.fst | lexfsmstrings -l example.lab}
\]

Now try different input strings (e.g., -s "4,321.39"). Show the output of the command above for five (interesting) inputs.
6 (10 points) Using fsmtools (and, optionally lextools) write an automaton that performs the opposite of the one in the previous exercise. For example, given the input "three hundred and twenty-nine" it should produce 329.

For full credit, make sure to cover all integers from 0 to 1,000,000. Of course, your grammar should not consist of one million and one rules, one for each possible input (smile).

For extra credit (up to 5 points) allow your input to include also real numbers in the same range (0 to 1,000,000) in intervals of .01 (that is 0, 0.01, 0.02, ...). Your inputs should either be integers (e.g., "twelve") or real numbers with exactly two digits after the decimal point (e.g., "ten point zero five" or "ten point oh five"). For example, the following input "ten point five" should be rejected (only one digit after the decimal point).

7 (15 points) Complete exercise 3.6 in the textbook and implement one of the steps of the Porter stemmer as a transducer using lextools and fsmtools (particularly, "fsmcompose"). Indicate what step it is that you’ve implemented and provide a few examples of input and output.