LING 570: Hw4
Due date: 11:45pm on Oct 28

As usual, the example files are stored under **~/dropbox/09-10/570/hw4/examples/.**

Note: In this assignment, "FSM" (finite state machine) means an FSA or an FST. You can decide which one you want to use.

Q1 (20 points): Write **expand_fsm1.sh**, which builds an expanded FSM given a lexicon and morphotactic rules expressed by an FSA.

- The command line: **expand_fsm1.sh** lexicon morph_rules output_fsm

- "lexicon" is of the format "word classLabel". A sample file is **examples/lexicon_ex**.

- "morph_rules" is an FSA (in the Carmel format) that encodes the morphotactic rules; that is, the input labels in the FSA are class labels (e.g., regular_verb_stem). An example is **examples/morph_rules_ex**, which represents the same FSA as the one on Slide #23 of 10_14_morph.pdf except that the hyphens in class labels are replaced with underlines.

- "output_fsm" is the expanded FSM (in the Carmel format), where the labels of the arcs in the morph_rule FSA are replaced by the words in the lexicon.

Q2 (10 points): Write **morph_acceptor1.sh**, which checks whether the input words are accepted by the FSA created in Q1.

- The command line: **morph_acceptor1.sh** fsm word_list output
- "word_list" is a list of words, one word per line (e.g., **examples/wordlist_ex**)
- "fsm" is the FSM (in the Carmel format) created in Q1
- "output" has the format "word => answer" for each word in the word_list, where "answer" is "yes" if the word is accepted by the morph acceptor, or "no" otherwise (e.g., **examples/q2_result_ex**)

Q3 (30 points): Write **expand_fsm2.sh** and **morph_acceptor2.sh** so that the "output" file produced by morph_acceptor2.sh has the format "word => answer", where "answer" is "morph1/label1 morph2/label2 …" if the word is accepted by the morph acceptor, or "*NONE*" otherwise (e.g., **examples/q3_result_ex**).

- The command line formats of **expand_fsm2.sh** and **morph_acceptor2.sh** are the same as **expand_fsm1.sh** and **morph_acceptor1.sh.**
- In your note file, explain briefly how the fsm produced by expand_fsm1.sh differs from the one produced by expand_fsm2.sh.

Q4 (20 points): Slide #24 of 10_14_morph.pdf shows an FSA for English derivational morphology.

- Create an FSA, **q4_fsa**, that represents the FSA on that slides. The format of the FSA is the same as **examples/morph_rules_ex**.

- Create a lexicon, **q4_lexicon**, so that each arc label in q4_fsa has at least one entry in the lexicon. The format of the lexicon is the same as **examples/lexicon_ex**.

- Create a wordlist, **q4_wordlist**, so that most paths in q4_fsa are tested by at least one word in the wordlist. The format of the wordlist is the same as wordlist_ex.

- Run the following commands from Q3 to generate the expanded FSM and test the words in the wordlist:
    - expand_fsm2.sh q4_lexicon q4_fsa q4_expand_fsm
    - morph_acceptor2.sh q4_expand_fsm q4_wordlist q4_result

- Note: You need to use your knowledge about English suffixes to determine what labels such as adj_al, noun_i mean. You can google "English suffixes" to find pages such as http://www.paulnoll.com/Books/Clear-English/English-suffixes-1.html that provide suffix lists and examples.

Q5 (20 points): Read the porter stemmer algorithm as described at http://www.tartarus.org/~martin/PorterStemmer/def.txt. The perl implementation, porter_stemmer.pl, is stored under ~/dropbox/09-10/570/hw4/. You can test it by running "cat examples/wordlist_ex | ./porter_stemmer.pl". If you are more comfortable with other programming languages, you can download other implementation from http://www.tartarus.org/~martin/PorterStemmer.

Compare the stemmer with the morphological analyzer we discussed in class (the one with three components), and summarize the strengths and weaknesses of each system. The two systems can be compared from various angles. The following are just some examples. Please feel free to add more:

- What kind of knowledge is represented in each system and how are they represented?

- Does each system use rules? If so, are the rules ordered in some way? If so, why do the rules need to be ordered and is it easy to determine the order? Are the rules intuitive and easy to create? Is there any redundancy among the rules?

- In each system, what changes are needed in order to handle a new language? And is it easy to port the system to new languages?

- Suppose the evaluation criterion is to calculate the percentage of words for which the systems can find the correct root forms. Without running some large-scale experiments, which one do you think perform better? Which one runs faster? Explain your answer.

The submission should include:

- The hw4 note file that includes answers to Q3 and Q5.

- The source and shell scripts in Q1, Q2, Q3: **expand_fsm1.sh, morph_acceptor1.sh, expand_fsm2.sh, morph_acceptor2.sh**, and any scripts called by them.

- The files created by running commands in Q1-Q2: **q1_expand_fsm and q1_result**.

```
expand_fsm1.sh  lexicon_ex morph_rules_ex  q1_expand_fsm
morph_acceptor1.sh q1_expand_fsm wordlist_ex q1_result
```

- The files created by running commands in Q3: **q3_expand_fsm and q3_result**.

```
expand_fsm2.sh  lexicon_ex morph_rules_ex  q3_expand_fsm
morph_acceptor2.sh q3_expand_fsm wordlist_ex q3_result
```

- The files created by Q4: **q4_fsa, q4_lexicon, q4_wordlist, q4_expand_fsm, q4_result**.