LING 570: Hw3
Due date: 11pm on Oct 19
Total points: 100

Goal: Become familiar with FST.
All the example files mentioned below are under **hw3/examples**/.

Q1 (6 points): Manually create FSTs for the following regular relations and save the FSTs in Carmel format as files "fst1", "fst2", "fst3" under **q1**/.

- fst1 for $\{(a^{2n}, b^n) \mid n >= 0\}$
- fst2 for $\{(a^n, b^{2n}c) \mid n >= 0\}$
- fst3 for $\{(a^n d^*, (bc)^n g) \mid n >= 0\}$

Q2 (14 points): Use Carmel to build a FST acceptor, **fst_acceptor.sh**.

- The format of the command line is: fst_acceptor.sh fst_file input_file > output_file
- fst_file is an FST in the Carmel format (e.g., "**examples/fst0**", "**examples/wfst1", "examples/wfst2"**)
- Each line in the input file is a string (e.g., "**examples/ex**", "**examples/ex2**")
- Each line in the output_file has the format "x => y prob" (e.g., "**examples/ex.fst0**"), where
  - x is the string from the input file.
  - y is the output string if x is accepted by the FST, or *none* if x is not accepted by the FST.
  - prob is the probability of the path whose yield is x.
  - The probability of a path is the product of the probabilities of the edges in the path.
  - If there are multiple paths for an input string x, y is the output string of the path with the highest probability (for paths with the same probabilities, Carmel breaks the tie somehow)

- Run your fst_acceptor.sh with the FSTs in Q1 and hw3/examples/ex as input file, save the output files in ex.fst[1-3], respectively, under **q2/.**

```
fst_acceptor.sh  q1/fst1  hw3/examples/ex  >  q2/ex.fst1
fst_acceptor.sh  q1/fst2  hw3/examples/ex  >  q2/ex.fst2
fst_acceptor.sh q1/fst3 hw3/examples/ex >  q2/ex.fst3
```

- Run the following commands and save the output files under **q2/.**

fst_acceptor.sh hw3/examples/**wfst1** hw3/examples/ex**2** > q2/ex2.wfst1
fst_acceptor.sh hw3/examples/**wfst2** hw3/examples/ex**2** > q2/ex2.wfst2

Q3 (25 points): Build **fst_acceptor2.sh** WITHOUT using Carmel, which has the same command line format and functionality as fst_acceptor.sh except:

- fst_acceptor2.sh CANNOT use Carmel
- Since we have not covered Viterbi algorithm, fst_acceptor2.sh will handle only non-ambiguous FST; that is, if you ignore the output symbols on the arcs, the resulting FSA is a DFA, not an NFA.
- If the FST is ambiguous, your code should print to stderr "The input FST is ambiguous", and print out nothing to stdout.

a) Run the following commands and save the output files under **q3/.** Remember to use fst1 and fst2, not wfst1 and wfst2 for the following:

fst_acceptor**2**.sh hw3/examples/**fst1** hw3/examples/ex2 > q3/ex2.fst1

fst_acceptor**2**.sh hw3/examples/**fst2** hw3/examples/ex2 > q3/ex2.fst2

Q4 (30 points): Build **nfa_to_dfa.sh** WITHOUT using Carmel, which converts an input NFA to an equivalent DFA (with the caveat below).

- The format of the command line is:
  nfa_to_dfa.sh input_nfa_file > output_dfa_file

- Both input_nfa_file and output_dfa_file are FSA files in the Carmel format: the former is an NFA, the latter is a DFA. If the former is a DFA, the script will simply output the same FSA.

- To learn how to convert an NFA to an DFA, please look at the slides attached to this assignment called NFA-to-DFA-conversion.pdf. You can also find videos on youtube such as https://www.youtube.com/watch?v=taClnxU-nao

- The main idea is that each state in the output DFA will correspond to a set of states in the input NFA. The algorithm will start from the start state of the NFA and determine for each input symbol, what states will be reached and make those states into a single state in the DFA. Then for each new DFA state, determine what states can be reached for each input symbol. Repeat the process until no new DFA states will be created. A DFA state is a final state if and only if one of the corresponding NFA states is a final state in the NFA.

- One caveat about the DFA: If the resulting DFA may have more than one final state, because the Carmel format allows only one final state, you have to create a new final state (let's call it **FinalState**), and add arcs that go from each of these

DFA final states to FinalStatewith with the empty string as the label. So in that sense, the output-dfa-file is not a real DFA, but this is due to the limitation of the Carmel format. If the resulting DFA has only one final state, do NOT create a new final state.

- For the resulting "DFA", let's use the following convention for naming the states in the DFA:
    - Suppose a DFA state corresponds to the set S, say {q1, q3, q5}, the state should be called q1-q3-q5 (the states are sorted alphabetically), not q1-q5-q3, etc.
        - If S has only one member, say q1, then the DFA state should be called q1 as well.
    - If the DFA has multiple final states, and we have to create a new final state, let's call that final state **FinalState**. You can assume that the input NFA does not have a state with the same name.

- The conversion algorithm in the NFA-to-DFA-conversion.pdf file defines a few functions:
    - $Move_{NFA}(s, a)$ is the set of states in the input NFA that can be reached from a state *s* when the input symbol is *a*. Here, *s* is a single state in the NFA.
    - $Move_{NFA}(S, a)$ is the set of states in the input NFA that can be reached from any state *s* in S when the input symbol is *a*. Here, *S* is a set of states in the NFA.
    - ε-Closure(*s*) is the set of states in the input NFA that can be reached from a state *s* by going over zero or more arcs with empty string as the label.
    - ε-Closure(*S*) is the set of states in the input NFA that can be reached from any state in *S* by going over zero or more arcs with empty string as the label.
    - $Move_{DFA}(A, a)=B$, where *B* is the set of states in the input NFA that can be reached from a state *A* in the DFA when the input symbol is a. Here, *A* is a state in the DFA which means that *A* actually corresponds to a set of states in the NFA. If *B* is not already a state in the current DFA, we then make it as a new state in the DFA.

- Run the following commands and save the output files under **q4/.**

```
nfa_to_dfa.sh hw3/examples/nfa1 > q4/dfa1

nfa_to_dfa.sh hw3/examples/nfa2 > q4/dfa2
```

The submission should include:

- The readme.(pdf | txt) file as always.
- Hw.tar.gz that includes all the files specified in submit-file-list