Mike Roylance
Homework 2
roylance@uw.edu

Q1

Under sample/ directory, run the following commands:
carmel -k 1 fsa7 wfst1
cat ~/dropbox/13-14/570/hw2/examples/wfst1_test | carmel -k 1 -sli wfst1

Do they yield the same results? What do these commands do?

The commands:

produce the same results.

The first command is reading in an FSA and pushing it through the WFST.

The second command is reading a raw string from the input (sli, s is for “standard input”, “l” means to append to the left side of the transduction sequence, and “i” means that the string is in string-form rather than FST-form.

WFSA and WFST files have an advantage that “-k” can be used to output the most probable strings. Both commands are using -k to do this.

Q4

In your note file, briefly explain what data structure you use to store the input FSA and how your code handles NFA.

Approach:
If you look in the fsa.js file, I create an object that parses a string input from the user. I assume that I am given a correct formatted string.

The fsa object first parses the end state, then it parses each transition. I hold each transition in an object in the file “transitionState.js”. A transition has the following properties: from, to, val and props (an array to be used in a later assignment, perhaps).

In the parseInput function on fsa.js, I start my way from the end state and at the end of the user input string. I calculate the previous states and see if there is a valid transition. If there is, I push the next item to check in the stack. I keep processing until I have none left.

If I reach a state where there isn’t any more user input, I then check to see if I am at a beginning

state. I currently define a beginning state as a state that is not referenced by other states or a state that is recursive.

This object handles both nfas and fsa because it calculates all possibilities and determines if a correct path can be reached. It ends on the first correct path found.