Lisa Gray
Project 1
CS311 – Computational Structures
Mark Morissey
4/24/15

So I decided to use Haskell as my language for this project because I have not used it before and wanted to solve the problem in a short amount lines. I started first by just hard coding a simple DFA into the main file and tested how it all worked using GHCi. I decided to use lists for the set of states, Sigma and the set of accept states. This allowed me to easily use the `elem` function to find out if the current state was an accept state or not. I used Data.Map to hold all the transitions and used Data.Map.lookup to find out which state I was transitioning to based on the character passed to my delta function. The Map I used held a tuple of the following signature ((State, Char), State) where State is just a String. So my delta function took a tuple of the current state and a character and returns another state. This made it fairly easy to use pattern matching and recursion (Haskell only allows recursion) to evaluate the DFA.

The hard part about using Haskell was the IO. I chose to store DFAs in a JSON file to read each part of a DFA because Haskell had a library and JSON structured everything nicely. I had to create an alias for the tuple because it did not like converting a String to ((String, Char), String). I made DfaJSON record and derived generic so that I could create instances of FromJSON and ToJSON without having to explicitly come up with the instance myself. I then started to create the main IO loop which was quite the challenge. I originally wanted to be able to have it loop where one could enter strings and go until they want to stop. However since Haskell is Lazy, it would print while typing the next string.

I make the assumption that the user will type the correct characters available to the defined alphabet, I am not fluent enough in Haskell to handle that situation and started to run out of time near the end. However, it does not return an “accepted” it simply returns the exception that it did not find key containing that Char in the Map. Another assumption, I make is that they name the file dfa.json of

the DFA they want to evaluate. I was going to have it as a command-line argument, but I ran out of time before I could get it working. Another restriction I make is that members of the alphabet are not strings in themselves, it did not occur to me until I was doing the writeup that it might be nice to have a language in which its alphabet contains a list of strings.

For testing, like I said I started with a hard coded DFA inside the Haskell file. I wanted to make sure that the evaluate function was working and then get what kind of types and data structures I would be using. From there, I went on to loading the DFA via JSON. I tested the original one that hard coded and then created two more DFAs. I used the multiples of five from assignment two and gave it several input strings like “101” to strings like “10001000111”. The other DFA I tested accepted strings in which contained the substrings “aba” or “bab”. The other thing I tested for was if given an empty alphabet and one state that is an accept state, that it would accepted because the empty set is also a language. I gave several non-accepting strings to each DFA such as “(a)*(b)*” and “10011100001110”.