Kevin Andrade
CS-373
Writing Assignment – Programming Assignment 1

PARTIAL NDFA

The spring 2019 CS 373 class had to simulate a partial non deterministic finite automata as our programming assignment 1. The program was to be written in either Java, C, or C++. There were a few basic steps that we had to follow to ensure guidelines.

We first had to make sure we read the definition of the machine from a file, the first command line argument), with the second command line argument being a string to simulate the machine running on, the input of the automata. We then had to write the output to the standard output. Our output had a strict format; it was either the word “accept” followed by a blank space and then followed by the list of the accept states or the word “reject” followed by a list of states that the automata can end up in after reading the input string. Any of our outputs had to end in a newline character. We also had to take into consideration on the format of the input file. The format of the input file could be a

written in a few different ways and it also was never guaranteed that the first state on the file was the accept state or that the order was contiguous. The format of the transitions in the file would be in the order of "transition p x q" where p and q where states between 0-1000 and x is the symbol to read.

My solution to this assignment was straight forward. First, I checked that the input matched the requirements. If the arguments were correct, then I read in the file and started parsing. I differentiated between state and transitions. If it was state then then I would check if it was followed by either a start, accept, accept start or a start accept and parsed the information properly. I then had a Boolean array and stored the state number that were accept states at that index of the array. Now if I read a transition line, I would push the states that were adjacent to the state currently in. once I looped through the whole file along with the nodes that it can transition to, I started formulating the algorithm

performed in class to see if it would be a valid string. As our third argument, we took an input string. We took the start state and saw where we could go from there, took the next symbol and saw where we could go from there. Basically, we saw if there were any symbols that matched our parsed input and traversed to that input and then we saw if there was any other NON used symbol with our parsed input and traversed there. If the last digit of our input was consumed and we landed at an accept state (we check this by verifying it with our Boolean array that contains accept states) then we know we have accepted and return the path it followed.

I wrote a bunch of self-programs for this assignment. I wrote multiple files that had no valid transitions, files that followed project specifications, files that ended in reject states, and files that had misspellings to them along with creative NDFA'S (a bunch of loops and transitions back to the

front). I felt that this was enough to verify the

solution to my implementation.