John Lake
Kyle Kozak
April 21, 2015
Project 3 Report

For this project, the executable code was written in python and tested on personal laptops as well as the student00-03.cse.nd.edu machines. All code was made using the basic python library and the imported sys and re libraries.

The executable program functions by taking the Turing machine input file and reading each line of the file in a loop into the appropriately designated list or dictionary. The line that starts with "A" goes in the alphabet list, "Z" in the tape alphabet list, "Q" in the states list, "T" in the transitions dictionary, "S" in the start state list, and "F" in the end states list. Each line is stripped of delimiters and the individual parts placed in the lists, in addition to some error checking that occurs to ensure that the Turing machine follows the rules for it to be deterministic. These data structures are placed in a dictionary labeled "tm" that describes the Turing machine. Next the machine processes the input, from the command line or a file fed into the executable. After verifying that the number of input tapes to be received is a positive number, the program then stores the input characters in a list. For each line of input that it was told to expect, the program verifies that every character in the line is also in the input alphabet. Having verified this, the program saves the designated start state, sets it as the current state, and begins the process of determining the effects of the input. Each time it goes through the loop to interpret the effects, it checks to make sure that it has not made over 1000 steps without reaching a finishing state, and that we have not reached a finishing state in the current step. In either case, the loop exits and the correct outputs are printed. Otherwise, we continue on and pass the information about our current input and location in the Turing machine into a function that interprets the outcome. If our current state and input match a rule in the transitions list of the tm list, then we interpret the applicable transition rule, print the output, and return the information to the loop that called the function. The loop stores the information and proceeds again, until we hit 1000 steps or a finish state is reached.

The first Turing Machine we created performs basic addition on two 4 bit numbers. When provided a string in the form of x + y = 00000, where x and y are both 4 bit binary

integers, it will add the two and display the results in the third segment. For example, the input "0001+0010=0000" would output the completed operation in the form "xxxx+xxxx=00011". It performs this operation by first copying in the first string into the area where the sum is displayed, from left to right, placing an x in spaces that have already been read. However, it only copies the 1's, since those values carry meaning. Once it has reached the "+" sign, it does the same for the second string of numbers. This time however, if it tries to copy a 1 into a space already occupied by a 1, it changes it to a 0 and tries to place a 1 in the first space to the left, the same way carrying works in binary addition. Once it has successfully placed a 1, it returns to the place it left off, which is the first place without an x. Once it reaches the "=" sign, it has completed the operation, and it goes to the finish state. Each string in the file tm1-input.txt shows the machine's ability to successfully carry and add from any spot in those 4 bit numbers. The result from any correctly formatted input is the correctly added numbers displayed at the end of the string in binary form. This machine satisfies the requirement for a Turing Machine that performs an arithmetic operation.

The second Turing machine inverts the string it is given. When given a string in the form of %x#y, where x is {a,b,c}* and y is $^n where n = |x|. The machine will take x and replace the symbols in y with x^R (x reversed). It does this by first going to the middle point, taking the closest character in x, replacing it with a # symbol, and then finding the first $ it can reach and replacing it with the appropriate character, then moving back into x until it finds the next character. It proceeds until it reaches the % symbol, which designates the start. The different input strings show that it works regardless of the order of a, b, and c, and regardless of the length, so long as it follows the format. The result of this operation will be a % symbol preceding a series of # symbols before the inverted string. This machine satisfies the requirement for a Turing Machine that performs an operation on a string.

The third Turing Machine recognizes any binary string that has a length of an odd number of digits. That is to say, if the string is w, and |w| = 2n+1 where n is greater than or equal to 0, then the machine accepts the string. It is able to recognize this by moving to right through the string and alternating between two states such that if it reaches the end of the string in one of them, then it must be odd and it transitions to the accept state, or it reaches the end in the other and it must be even, and transitions to the reject state. The inputs show that it works on strings of length 1 up to some arbitrarily large number. The output of this machine will either

accept or reject depending on if the string is odd or even. This machine satisfies the requirement for a Turing Machine of our choice.

No major difficulties were experiences aside from making our Turing Machines work correctly when converted into the machine description. The only notable implementation decision was our use of a dictionary in our code.