Christopher Ray
Professor Blanton
CSE 30151
30 March 2015

Project 2 Report

My submission for this project was written in Python and tested on student00.cse.nd.edu.

The program works by first reading in inputs from a file describing the states, input alphabet, stack alphabet, transitions, start state, and final states for a pushdown automaton. The program maintains a list for each of those categories and adds elements to those lists as it parses each line of input. After reading in the final states, the program then iterates through each state in the automaton in order to ensure that there are no ambiguous transitions out of a state. If it finds such an ambiguous transitions for a state, the program will immediately stop and inform the user that the automaton that has been built is nondeterministic. If the program fails to find any ambiguous transitions, it assumes that the machine is deterministic and proceeds to wait for tape inputs from the user.

A user can submit tape inputs to the DPDA is two ways: either by manually entering each input or by putting the inputs into a text file and using the '<' character when executing the program. Either way, inputting tape inputs involves first inputting an integer indicating how many tape inputs to expect. The program then proceeds to loop for that specified amount, reading in a tape input with each iteration. Each tape input is parsed and its individual characters sent through the DPDA's transition function. The transition function works by taking the input that has been read in and the top element of the stack and iterating through the list of transitions to see if there is a transition that

utilizes those two inputs (or some combination of those two inputs and empty strings inputs). If a match is found, the input character from the tape input is set to ‘e’, the current state is updated, and a new element is pushed on to the stack (if applicable) before the process repeats itself. The transition function attempts to make as many transitions as possible with a given input and stack. Once it reaches a point where no more transitions are possible, the main driver program resumes control in order to feed in another input character, check if the DPDA is currently in an accept state, or terminate due to some error with the transition function.

The stack implemented for this program was actually a simple Python list, with the last element of the list serving as the “top” of the stack. Pushing on to the stack meant appending a value to the end of the list and popping the top element of the stack simply involved calling the pop() method, which removes and returns the last element of a list.

I tested the program using the test files provided on the project website. I first tested whether or not the program could successfully construct a PDA from an input file. Once I confirmed that running the program with **dpda1.txt** successfully created a PDA with the parameters specified in the file, I then tested whether or not the program was capable of recognizing nondeterministic PDAs by using the files **pda2.txt** and **pda3.txt** as input files. Once the program was successfully printing out error messages for those two input files, I then tested whether or not the program could recognize DPDAs by using **dpda1.txt** and **dpda4.txt** as input files. Once the program was properly recognizing DPDAs and reached the point where it was waiting for user input, I tested whether or not a user could manually input tape inputs, and by extension feed in input

files using the '<' character, to the machine without causing any issues. Once I confirmed that the program could read and interpret user inputs, I ran the program with **dpda1.txt** and **dpda4.txt** as input files along with **dpda-input1.txt** and **dpda-input4.txt** containing tape arguments for their respective automata. Once I confirmed that the outputs from the above tests matched the outputs given on the course website in **dpda-output1.txt** and **dpda-ouput4.txt**, I concluded that the program was working as intended.

The major difficulty I encountered with this project was implementing an algorithm to check whether or not the automaton was deterministic. My first implementation involved running through every possible combination of current state, user input, stack input, next state, and item to push on to the stack and checking whether or not that would result in the current state being able to transition to multiple states. This method proved to not only be slow on the student machines, but was also ineffective at catching every transition that violated the rules for DPDAs. My second and current implementation is similar in that it iterates through every transition out of every state, but now determines whether or not the machine should read from user input, stack input, neither, or both depending on the current state. Attempting any other transition when the machine is not reading either input, attempting to switch between reading from only user input to only reading stack input, or attempting a transition which is more or less specific compared to a previously seen transition result in an error.