Humza Faraz and Joshua Philip Santos

December 4, 2021

CMSC 451

Seabald

Course Project Part 2 Report

Our task for the second part of the course project was to implement a Deterministic Pushdown Automata (DPDA). This automaton contains a list of rules which determines whether an inputted string is accepted or rejected.

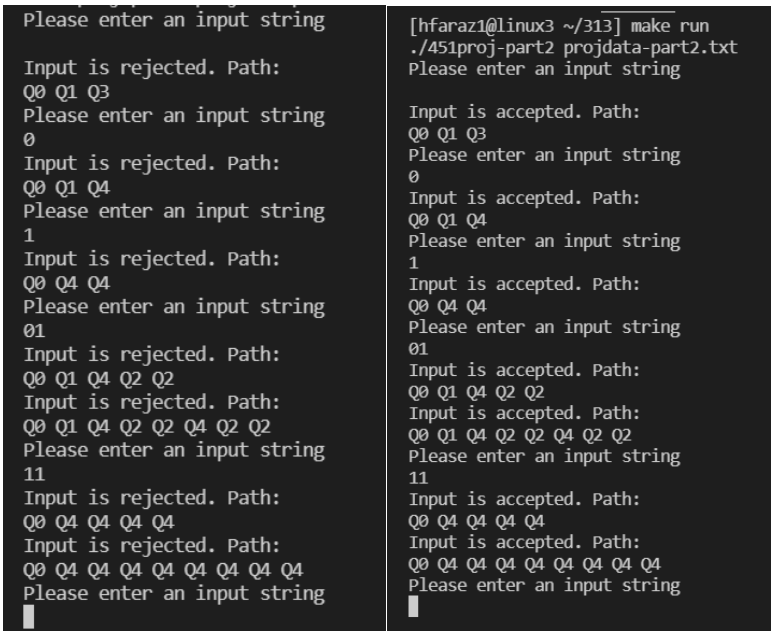
The rules are listed in a text file passed in, which the program the reads. This text files contains the type, input alphabet, stack alphabet, number of states, and accept states. It then lists the rules for the DPDA. Each rule contains the start state, character to read from input, character to read from the stack, transition state, and finally the character to push onto the stack. Unlike part 1 of the project, this has the possibility of containing epsilon (empty) values.

The user's input is then read character by character to determine if it meets the accept state. The rules are based on both the current character and the character at the top of the stack. If the rule met contains a character that must be pushed to the top of the stack. If it is same value as the current top, it is then pushed on. If it is different, then the current top is popped off and the new value is pushed on.

In order to keep track of all 5 values in each rule, I decided to store these in a tuple. Each rule would then be stored in a vector. This way I could iterate through all rules and each of its values when checking if the current state meets any rule.

After validating the text file and creating all rules for the DPDA, I would then prompt the user for a string. I would then validate and then iterate through my vector of rules and check if the current state and top of the stack are equal to any of the rules. When this is found, I push the current state into my vector of states to keep track of the path taken and then change to the transition state. I would then determine if the state needed to be pushed or popped onto my stack. Once the loop is finished, I would check if the current state is the same as any of the accept states. If it does, then it is accepted, otherwise rejected. This process would repeat until the user decided to exit the program.

To test my code, I ran it with the default text file. Next, I would plug invalid values into the text file to make sure the program exits when this occurs. I then tested if all the states were accept states or if there were no accept states to see if it always accepts or rejects.



After the testing completed, I was confident my code was implemented properly. A guide on how to compile and run the code is given in a README along with a makefile.

Sources:

https://www.cplusplus.com/reference/vector/vector/pop_back/

https://www.cplusplus.com/doc/tutorial/files/

https://stackoverflow.com/questions/15734136/any-stl-data-structure-like-pair-that-gives-three-itemstypes-instead-of-two

https://www.geeksforgeeks.org/tuples-in-c/

https://stackoverflow.com/questions/40188779/vector-of-tuples-in-c/40188871

https://www.cplusplus.com/reference/vector/vector/front/

https://stackoverflow.com/questions/5838711/stdcin-input-with-spaces