Final Project Report

CS 3110.01 Formal Languages and Automata

Christopher Koepke

Dr. Tingting Chen

May 19, 2021

1) The language is any string of length 3n where n is a positive integer. Its alphabet is made up of the set of a, b, and c. For a DFA, each element must have a transition. From the start state, you must have three transitions before you reach the final state. Each transition can either be an a, b, or c. A loop occurs at the final state with three transitions for the condition that n is greater than one.

2) For this language, there is two options to transition from the start state. First is the (aa + b)* set and the other is the ab*a. From these two options (aa + b) has a loop. Once looping is done, transition to a final state and the final state has a transition to the start state to loop the entire graph. For using JFAP, I first started a new FA graph. I than selected the State Creator key and created four states, start state and the three states of the ab*a set. After these states have been created, I then selected the Transition Creator key and created lambda transition, a, b in a loop, and lastly a. Next I created four states to for the (aa + b)* set. For this set, there is two paths, a, then a as one path and b as the other. They meet up at a single state using lambda transitions. This state than has two paths, one to a final state and the other to the beginning of the (aa + b)* set again using lambda. The final state than lastly transitions to the start state to loop through the language again using a lambda transition. For conversion to DFA, I highlighted the convert tab at the top of the window and selected convert to DFA option. A new window opens with just the start state. I than selected the complete key on the toolbar area to have the program complete the graph. Once finished I clicked on the Done? key to see if the DFA is fully built.

3) This PDA was difficult to figure out. The easiest part was realizing that an accepted string could have zero a's, zero b's and any number of c's, so I did not need to push nor pop anything when reading a c. Next was figuring out how to count the number of a's and b's. After many attempts on paper, I came to the conclusion that when reading an b, I need to push two elements onto the stack and only pop once when reading an a. As long as the stack was empty at the end of input, then the string is accepted. The example string cbaabbaaaa is as follows for each element read. Read the input c and do nothing. Push two zeros onto the stack. Pop the zero from the stack. Pop the other zero from the stack. Push two zeros onto the stack. Push two zeros onto the stack. Pop a zero from the stack for each a. After all input is read, pop an empty stack symbol, and push the empty stack symbol back onto the stack and enter the final state. Once I felt confident in my PDA on paper, I then proceeded to create the PDA using JFAP and multiple character input. Once my PDA was created, I then tested a number of strings of varying lengths to see if my design is correct.

4) For this language I first attempted to understand the rules of the topmost transitions. I came to the conclusion that the language is $\{ a^n b^n : n \geq 1 \}$. For the bottom transition I came to the conclusion that the language is $\{a^n b^m : n \leq m\}$. My reasoning for the second language is the loop between the states q_2 and q_3, which is just counting the number of b's in the string after all a's have been read their elements have been popped from the stack. As long as a string has more b's than a's, than the string is accepted.