Mark Franceschini
CMPT 440
1/30/16

Essay 1

During the process of creating the DFA diagram on lucid charts I ran into a few issues. The first issue right off the bat was that the free account on lucid chart only allows you to use 60 objects in a document. This is obviously a problem since the DFA has over 70 nodes and also an arrow and textbox that goes with each node. However I was able to overcome this by signing up for a student account. The second and biggest problem I had while creating the DFA diagram was readability. A diagram of this size while easy to make is hard to make readable. When I finished my first attempt I had trouble figuring out which arrow came from where and where it went to. I solved this by moving every node and textbox farther away from each other so that there would be less overlap and it would be easy to see each individual node and textbox. If I had to do this lab again I would spend more time on planning how I wanted the diagram to look like and how to handle different branches that arise from accommodating two different variants of the for loop. Another problem I had was the error nodes. The error nodes were difficult to implement because they effectively double the amount of lines in the diagram making it even harder to read. Also I was not sure if there was a set symbol for a blank space so I just wrote SPACE whenever I needed a bland space, I had the same issue with tab so I used TAB as the tab symbol.

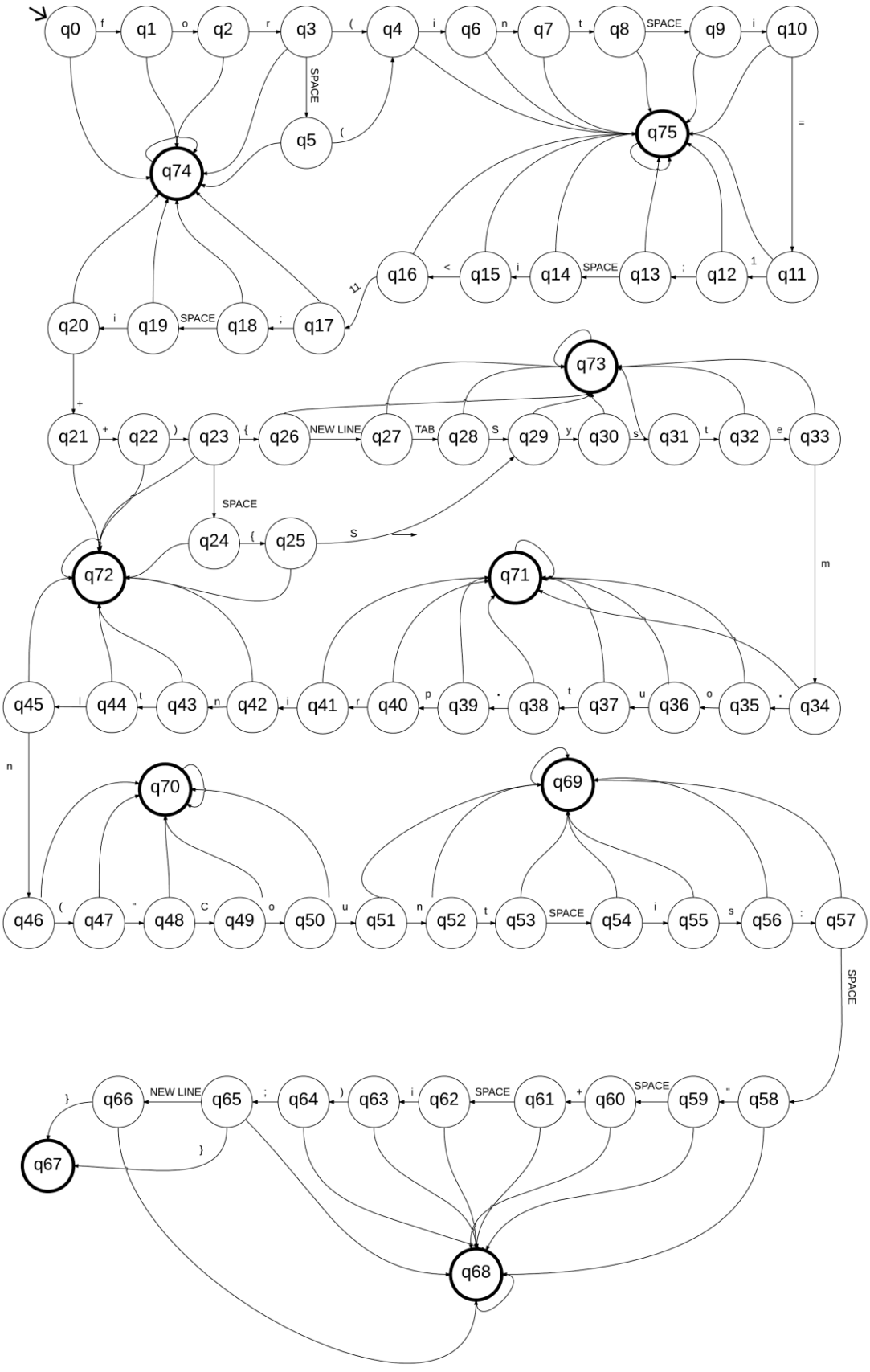
Essay 2

To implement this in java I would use 3 arrays, a for loop and if statements. Assuming the users input is submitted as a string of characters, the first array would contain the user input each element being a char. The second array would be the first possible variation of the for loop and the third would be the second possible variation of the for loop as provided by the lab. I would use a for loop to compare the elements of the user input array to the two acceptable variant arrays. For example the for loop would start by checking if the first element of the user array was equal to the first element of the first variant array if they were equal then it would move on to element two. If they were not equal then it would check the first element of the user array against the first element of the second variant array, again if these values were equal it would move on to the next element in the user array. However if these values were not equal it would exit the loop and print an error message saying that the user input is invalid.
This is a sudo-code example of the algorithm

```
char[] userInput;
char[] variatnOne;
char[] variantTwo;

for loop i = 0 , i to userInput size
        if userInput[i] == variantOne[i] OR userInput[i] == variantTwo[i]
                Continue with loop
        Else
                Exit loop print “error invalid user input”
        END IF
END LOOP
```

The picture on the following page is the DFA. It starts on node q0 and ends on node q67. All nodes above q67 are error nodes and all nodes with a bold border are exit nodes. Also each error node circles back to itself to show that any input from that point on is an error. To avoid more clutter I did not write the word other for every line that points to an error node, but it should be understood that if the user input is anything other than a specified accepted input then the user is moved to an error node. For example in q0 if the user inputs anything other than 'f' the user is moved to the error node q74. As mentioned before I was unsure of what symbol to user for a blank space, a tab, and a new line so I used SPACE, TAB, and NEW LINE respectively.


q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
q14
q15
q16
q17
q18
q19
q20
q21
q22
q23
q24
q25
q26
q27
q28
q29
q30
q31
q32
q33
q34
q35
q36
q37
q38
q39
q40
q41
q42
q43
q44
q45
q46
q47
q48
q49
q50
q51
q52
q53
q54
q55
q56
q57
q58
q59
q60
q61
q62
q63
q64
q65
q66
q67
q68
q69
q70
q71
q72
q73
q74
q75
f
o
r
(
i
n
t
SPACE
=
;
1
<
+
)
{
NEW LINE
TAB
S
y
s
e
m
C
"
}