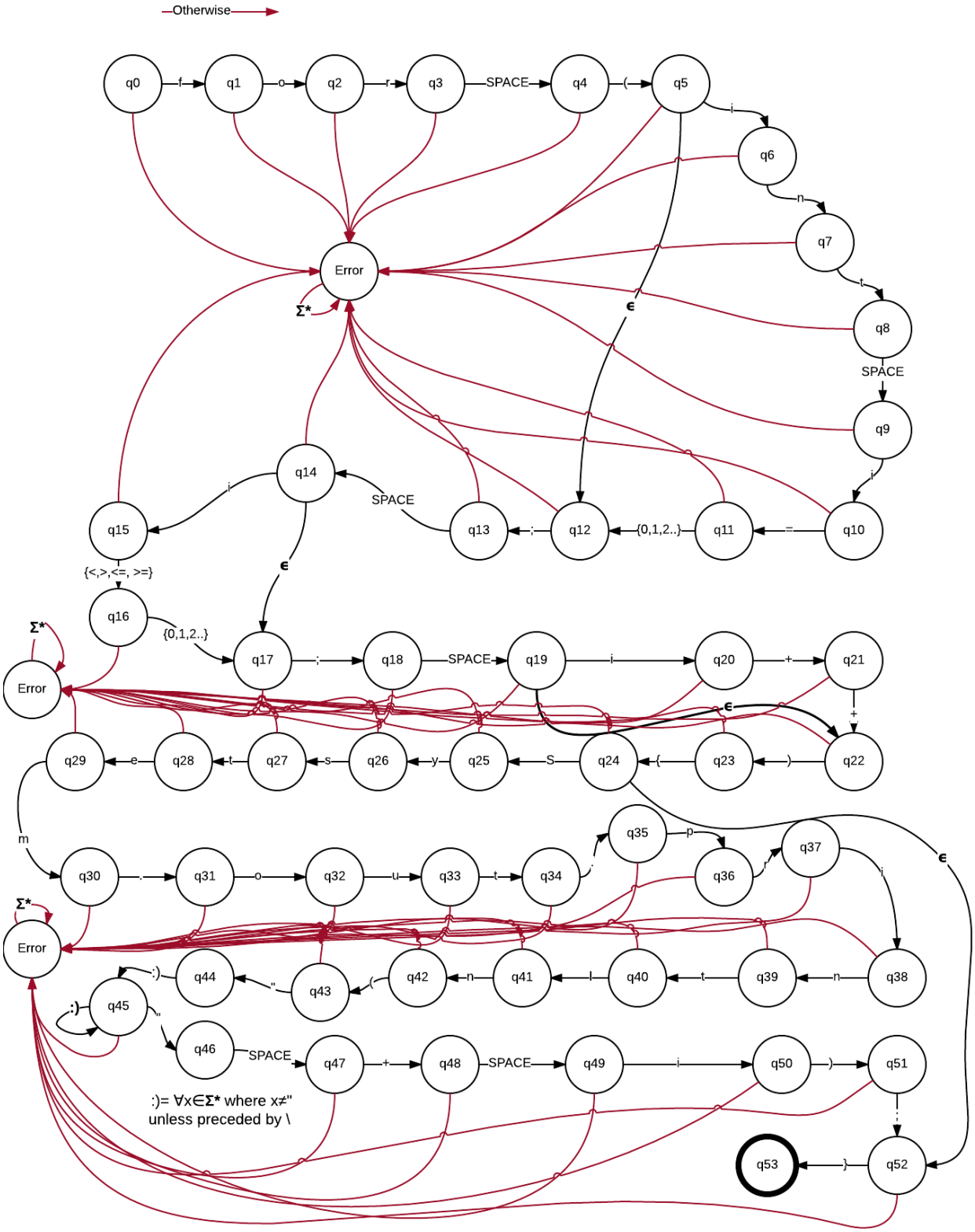
Graham Harrison Lab 1 For Loop DFA



2.

There were many problems that I faced while creating my for loop DFA diagram. Originally my q9-q10, q14-q15, q19-q20, and q49-q50 valid transitions were from the set {a,b…z,A,B…Z}. Though this set seemed to allow my DFA to accept a greater range of for loops, I decided to restrict these transitions to the character i. If I were to permit any single lower or upper case letters in each of these transitions, the following for loop could be considered valid even though it is not: for(int i=0; j<10; K++){}. Working with a diagram of this size was also a challenge. Therefore I took any opportunity that I could to simplify my diagram. For example, my transition for typing text inside of the System.out.println command is simplified to a predefined set ☺ where ☺ is all characters x that are elements of the set of all strings over the language except the " character unless it is preceded by \. Q45 also has a loopback ☺ transition which allows for multiple characters to be typed. Thus, instead of restricting myself to one single message, I am able to print any combination of characters as long as it is closed with an exiting " character.

3.

I would define my for loop DFA as a java object and compare strings against it for validation. My for loop DFA object would have a tree structure where the nodes would be all of the q states. Each node's data would that q state's valid transitions and the child nodes of that q state would be all of the possible successor q states. Q5 for example would be a node with three children: Error, q6 and q12. In addition its valid transitions would be i-> q6 or the empty character -> q12. Anything else would lead to an error.

In order to validate a string as a for loop I would need to create a function to compare the string against my DFA tree. This function would recursively iterate through the string and compare each character against the DFA tree and traverse the tree according to each character input. This function would return true if the string ends up at the valid end state (q53) and false if it lands on an error. At each recursive step the function would retrieve the current q state's validation rules and compare a character against it. After determining which q state should be the successor, the function would call itself using the next node of the DFA and the next character of the string.