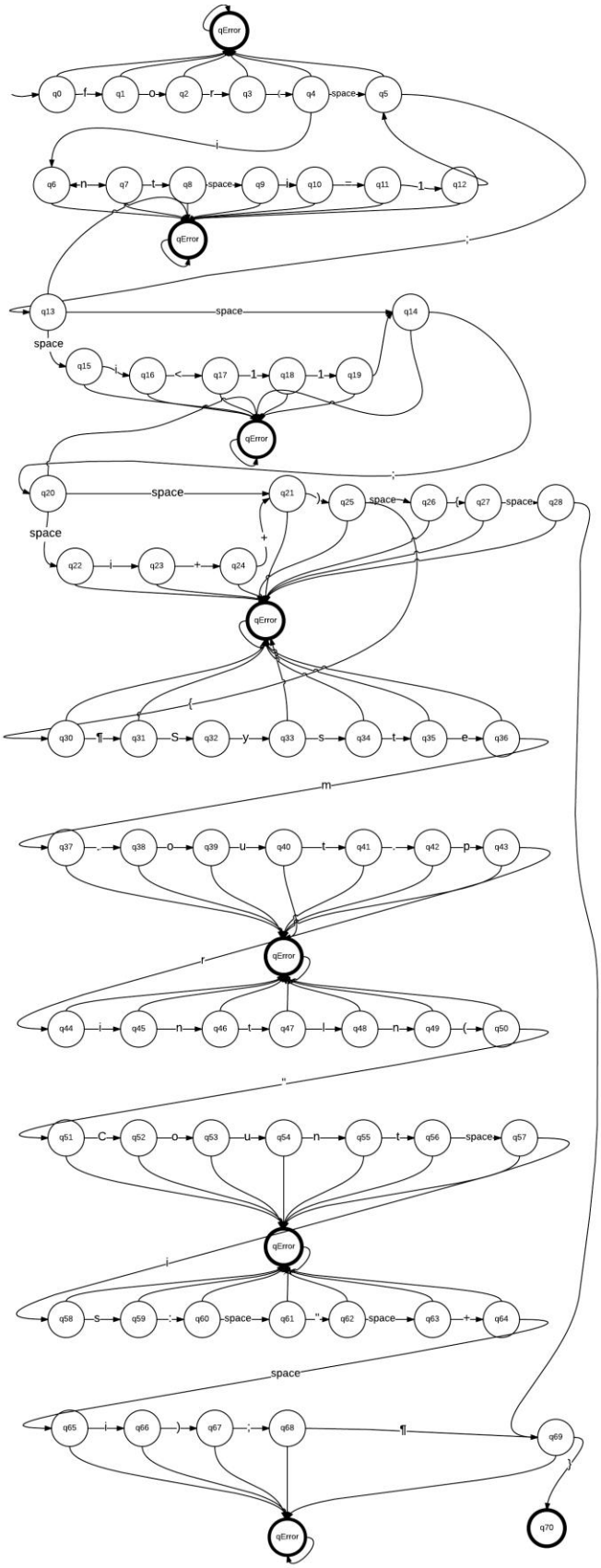
Alex Hochberg

CMPT 440 – Rivas

02/01/2016

Lab 1: Create a DFA

(1) See next page.


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

(2)

When making the DFA diagram of the “for” loop, I very quickly learned a lot about good and bad practice of diagram making in LucidChart. First, I learned that it is very important to map out how you diagram will flow prior to making the diagram, by maybe making a quick sketch on paper of all the possible paths your DFA chart will go into. This way, your DFA diagram will be more organized and less confusing for people to follow, because DFA diagrams can very quickly get complicated. When making the above DFA diagram, I made a decision to sacrifice some neatness and beauty for the sake of accuracy and best logic. When possible, I choose to have the DFA path flow through the same nodes. Although in theory, I could have created two separate paths that only intersect at the beginning and end for the two different “for” loops that have to be validated, I chose instead to share as many nodes as possible. I made this decision, knowing it would be more of a head-ache for myself and more complicated looking for others, because I think this would be the logic a computer would use if I were to code this.

(3)

If I were to implement this in Java, a major concern I would have is expandability. In the above diagram, I made a conscious effort to only validate for two examples of “for” loops. Though, if I wanted to create a way to validate any “for” loop string I would go about it differently. I would first make solid checks for the mandatory parts of a “for” loop. I would do this by dividing the string into an array of characters, and then I would start checking each character one by one. For example, I would start by making sure the string starts with an ‘f’, and then goes to an ‘o’, or I would throw an error if this was not the case. Then I would allow a range of options for the fields in a “for” loop that are not mandatory and up to the user. For example after checking “for (“ I would have to have a check for all the possible ways of initialization. This would get quite complicated because Java has multiple ways of declaring initializations. I would continue by checking all the mandatory characters and optional ones in order. The statement body might be the most problematic because there are many different combinations of characters that could be valid or could lead to an error. I would look for the final '}' in the end and then I would return either an error or a valid "for" loop.