Colin Quinn
Prof. Huggins
CS 312 Exam 1

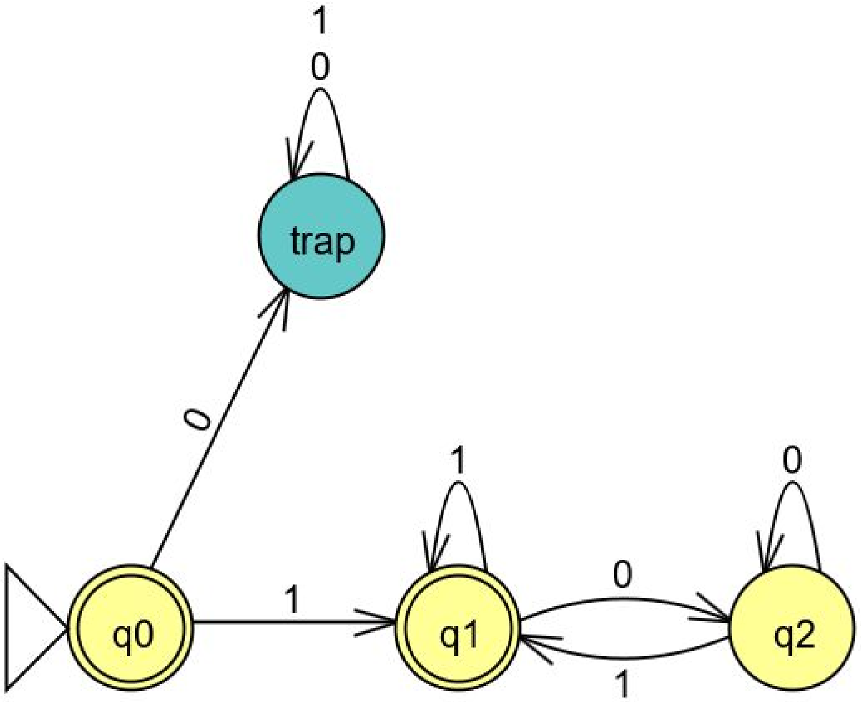
Note: PDF did not want to let me edit it, so this was the closest I could think of.

Note: In all problems on this exam, let Σ = {0, 1} be the alphabet from which strings are produced.

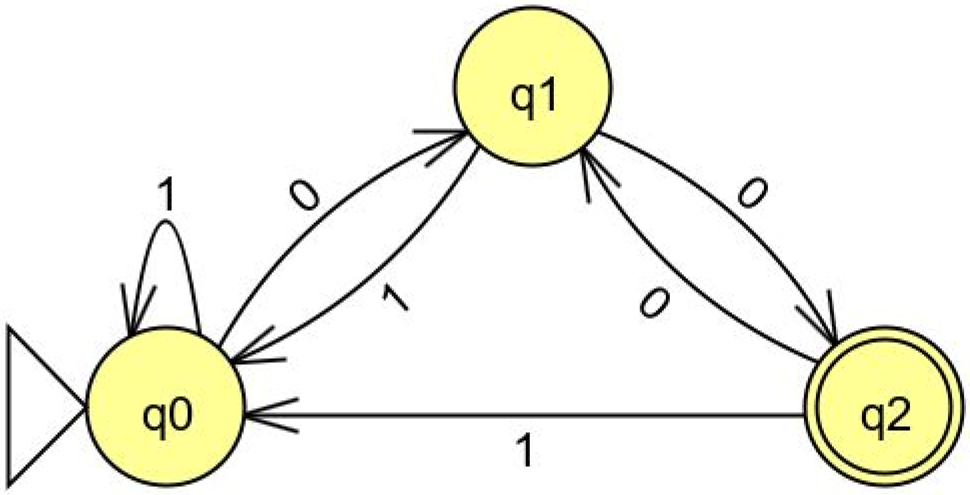
1. 10 points each. Give deterministic finite automata which accept the following languages:

(a) Strings in which every 0 is immediately preceded and followed by a 1. (Note: strings without any 0s, including epsilon, are in this language.)

(b) Strings which end in 00.



1. a)



b)

2. 10 points each. Give regular expressions for the following languages:

(a) Strings where the number of 1s is divisible by 3.

(b) Odd-length strings that start with 1, or even-length strings that start with 0.

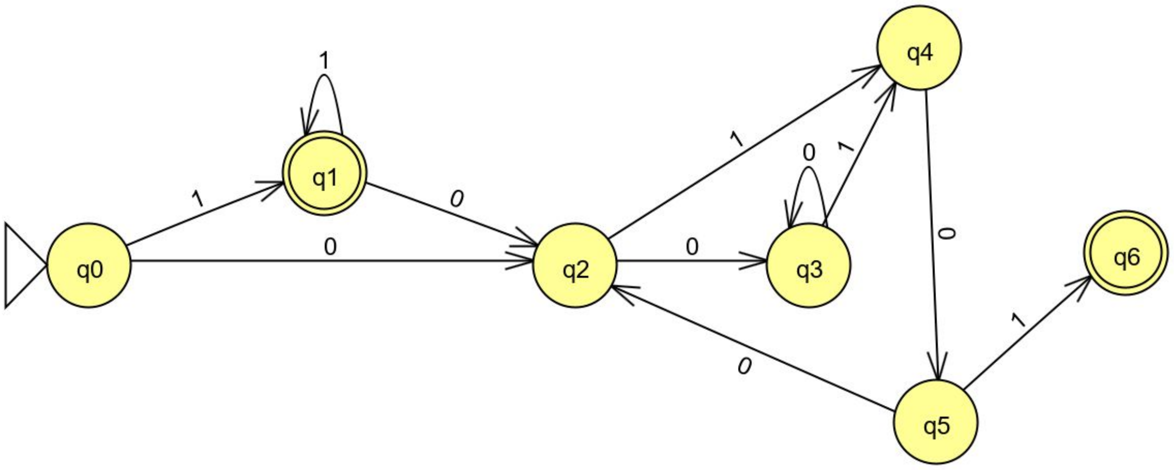
2. a) 0*(10*10*10*)*0*

 b) 1(00 + 01 + 10 + 11)* + (00 + 01(00 + 01 + 10 + 11)*)

3. 10 points. Convert the following regular expression to a nondeterministic finite automaton:

1*(0 + 10)*1

(Note: Do not use the automated regular expression conversion tool in JFLAP. You may use the JFLAP editor to draw and test your solution.)



3.

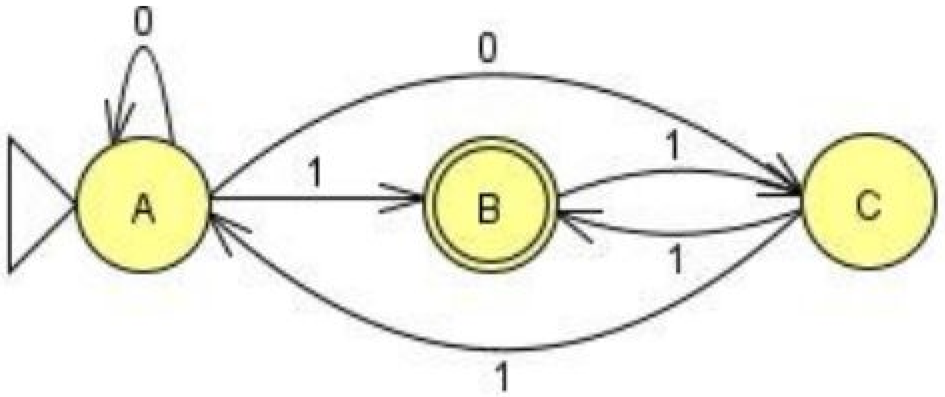
4. 10 points. The POSIX standard for regular expressions includes many extensions to regular expressions. One of these is the numerical quantifier.
Let r be a regular expression and n be a non-negative integer. The expression r{n, } matches *n or more consecutive occurrences of r*. For example, (01){3, } matches 010101 and 0101010101, but not 0101.
Show that the numerical quantifier operator does not change the class of languages accepted by regular expressions. That is: show how to transform any regular expression containing a numerical quantifier operator into an equivalent regular expression without a numerical quantifier operator.

4. The bounds of the language do not change. Assuming *r* is in the language, simply repeating *r* for a minimum amount of times does not break the bounds of the language. For example: let *r = 101* and *n = 5.* Thus (101){5, } is the expression to convert.
   A valid regular expression for this numerical quantifier could be simply to repeat *r*, *n* amount of times and include *r** at the end. Such as this regular expression:
   101101101101101(101)*

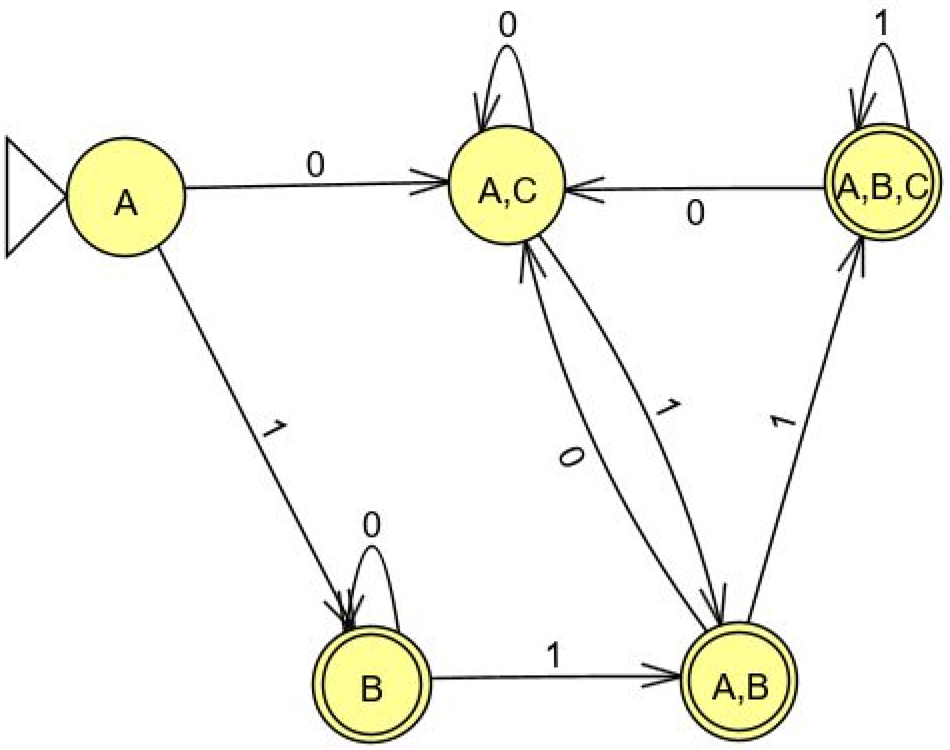
5. 10 points. Let L1 and L2 be regular languages. Describe an algorithm to decide if L1 ∩ L2 != ∅. That is, is there any string w such that w ∈ L1 and w ∈ L2?

5. An algorithm that would determine if L1 ∩ L2 != ∅ would be one such that includes links going from a start state to a final state when creating a DFA to fit L1 ∩ L2. The path(s) that lead to the final state would be the strings accepted by this DFA. If this link exists, there must be some string such that fits in L1 ∩ L2, meaning it is not empty. If there is no link between start and final states then this set would be an empty set.

6. 15 points. Convert the nondeterministic finite automaton shown below to a deterministic finite automaton, using the “powerset” construction presented in class (and the textbook). Label the states of your constructed DFA with the sets of states of the original NFA.



(Note: Do not use the automated DFA conversion tool in JFLAP. You may use the JFLAP editor to draw and test your solution.)



6.

7. 15 points. Let L = {0^a1^b0^c : a = b + c}. Prove that L is not regular.

7. Assume that L is a regular language and is by extension accepted by some DFA. Using the pumping lemma, a string z ∈ L where |z| >= a, there must be substrings u, v, and w such that z = uvw where v != ∅, |uv| <= a, and uv^iw ∈ L for any i >= 0.

Let z = $0^a1^b0^c$ where a = b + c. Applying the pumping lemma to the first half of the string, or the first a characters, where we would now have the string z = uv^2w. By the pumping lemma, uv^2w ∈ L, but this no longer fits in L due to z = $0^{2a}1^b0^c$!∈ L. There are now double the amount of 0's at the beginning of the string, thus no longer existing in L.