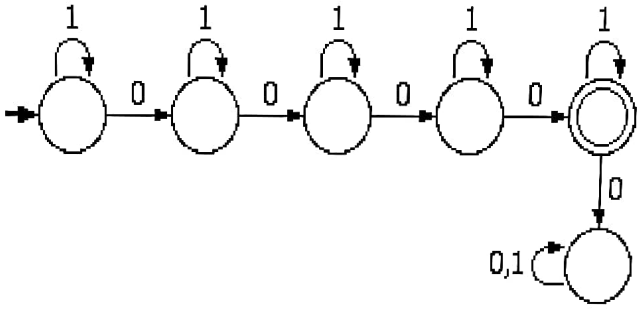
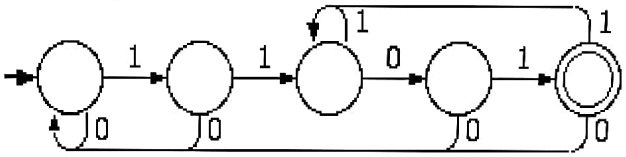
Automata Theory

1. DFAs

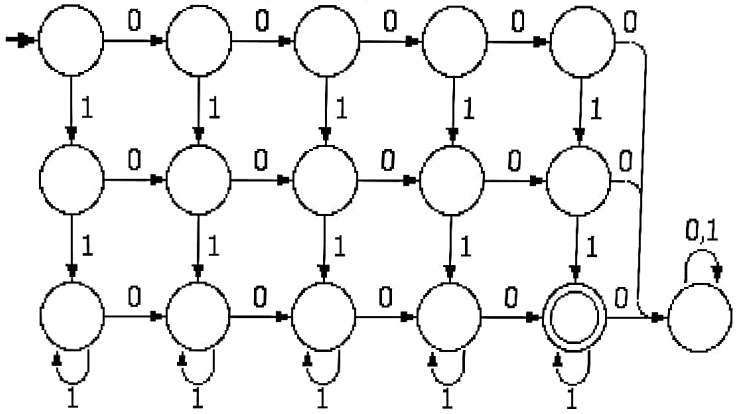
a. All strings that contain exactly 4 0s.



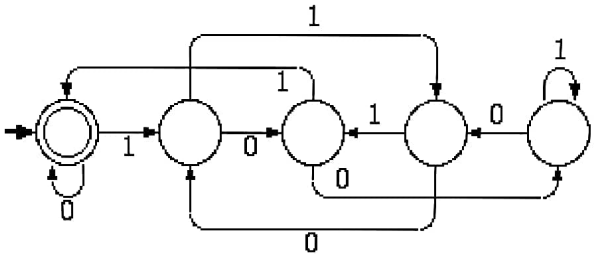
b. All strings ending in 1101.



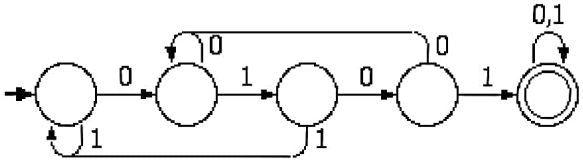
c. All strings containing exactly 4 0s and at least 2 1s.



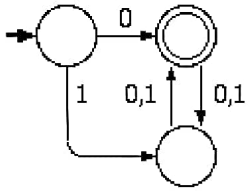
d. All strings whose binary interpretation is divisible by 5.



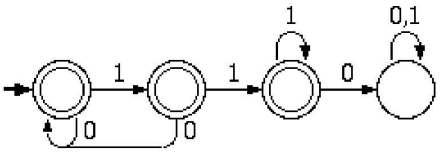
e. All strings that contain the substring 0101.



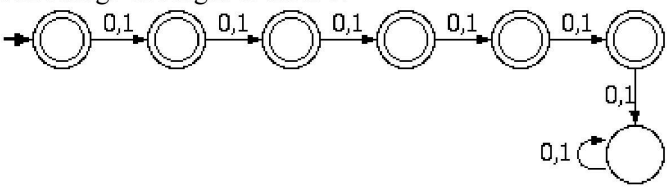
f. All strings that start with 0 and has odd length or start with 1 and has even length.



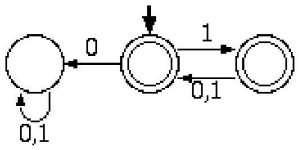
g. All strings that don't contain the substring 110.



h. All strings of length at most 5.

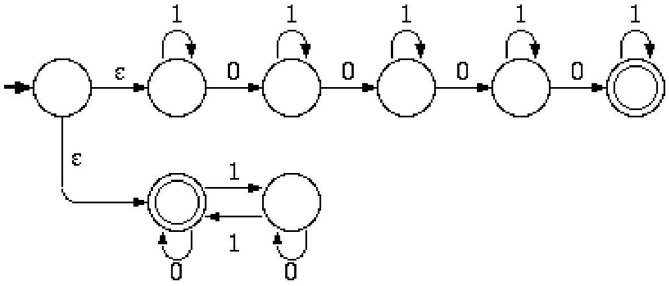


i. All strings where every odd position is a 1.

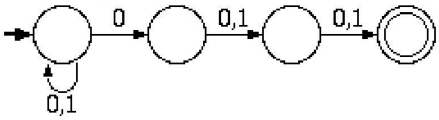


2. NFAs

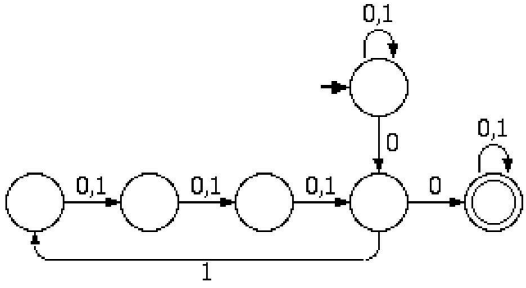
a. All strings containing exactly 4 0s or an even number of 1s. (8 states)



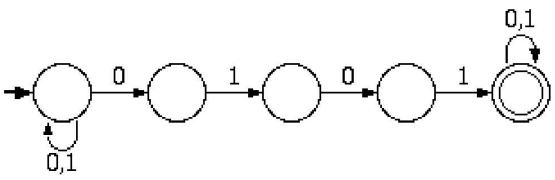
b. All strings such that the third symbol from the right end is a 0. (4 states)



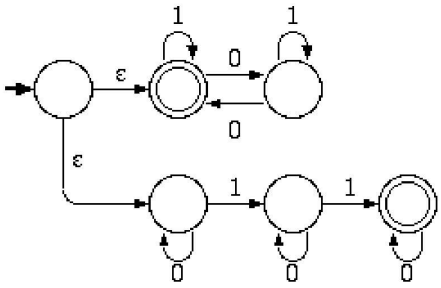
c. All strings such that some two zeros are separated by a string whose length is 4i for some i>=0. (6 states)



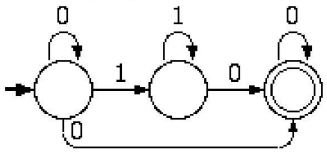
d. All strings that contain the substring 0101. (5 states)



e. All strings that contains an even number of 0s or exactly two 1s. (6 states)

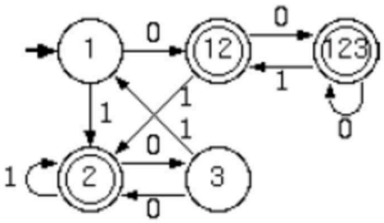


f. The language 0*1*0*0 (3 states)



3. Converting NFAs to DFAs

a. Convert the NFA into a minimal DFA.



b. Write a regular expression for the set the machine accepts.

```
[0+(0+1)(1+00)*01]*(0+1)(1+00)*
```

4. Discrete Math Review - Proofs

L1: The set of strings where each string w has an equal number of zeros and ones; and any prefix of w has at least as many zeros as ones.
L2: The set of strings defined inductively as follows: if w is in the set then 0w1 is also in the set; if u and v are in the set then so is uv; and the empty string is in the set.

a. Prove that every string in L2 is contained in L1

We can analyze L2 inductively to see that it maintains the property of L1 for each case:

1. The empty set. This is a member of L1, since it satisfies the properties vacuously.
2. 0w1. Assuming that w is in L1, we maintain the equal number of 0s and 1s because we add one of each. We also maintain the prefix condition, since the 0 is added before the 1.
3. uv. Assuming that u and v are both in L1, simply concatenating them together will maintain the equal number of 0s and 1s. The prefix condition is slightly more difficult. We consider the following prefixes:
    a. PREFIX(u). Since u is in L1, this must be in L1.
    b. u. Again, since u is in L1, this must be in L1.
    c. uPREFIX(v). Since u has an equal number of 0s and 1s, and v is in L1, this must maintain the prefix property.

b. For those of you who are paying attention, this problem is extemely similar to

the stream-crossing ghostbusters problem from algorithms. The proof is by induction on the length of strings in L1:

1. The base case is the empty string. This is in L2 by definition.
2. For the inductive step, suppose that all strings in L1 of length <= n are in L2. Let w be a string in L1 of lenght n+1 and suppose it is of the form $A_1A_2...A_{n+1}$, where A_i is either 0 or 1. Let j be the first index with the property that $A_1A_2...A_j$ has the same number of zeros and ones. There are two cases to analyze.
    a. $j < n+1$. Then not only does $u = A_1A_2...A_j$ have the same number of zeros and ones, any prefix of u will have at least as many zeros as ones since it is also a prefix of w. So u is in L1. Let $v = A_{j+1}A_{j+2}...A_{n+1}$. Then v must have the same number of zeros and ones since both u and w=uv do. Also, any prefix x of v cannot have more ones than zeros in it since then ux would be a prefix of w that had more ones than zeros. Therefore v must be in L1. Since both u and v are of length <= n, by the induction hypothesis they are in L2. Therefore w = uv must be in L2, by the definition of L2.
    b. j = n+1. Then w = 0u1 for some string u, and u has the same number of zeros and ones, since w does. Also, no prefix x of u can have more ones than zeros, since then 0x would either have more ones than zeros which is impossible by hypothesis, or 0x would have the same number of ones as zeros, which is also impossible by since j = n+1. Therefore we can conclude that u is in L1, and since it is of length <=n it is in L2 by the induction hypothesis.

This completes the inductive step, and therefore L1 is contained in L2.

5. Closure Problems

a. Prove that if L1 is regular and L2 is regular then so is L1-L2 (the set of all strings in L1 but not in L2).

L1-L2 is the same as the intersection of L1 and the complement of L2. Since the set of regular languages is closed under each of these operations, L1-L2 must be regular.

b. Prove that if L is regular then Prefix(L) is regular. Prefix(L) is the set of all strings which are a proper prefix of a string in L.

We can construct a DFA to decide Prefix(L) by taking the DFA for L and marking all states from which an accept state is reachable as accept states. So, Prefix(L) must be regular.

c. Prove that Regular Sets are closed under MIN. MIN(R), where R is a regular set, is the set of all strings w in R where every proper prefix of w is in not in R. (Note that this is not simply the complement of PREFIX).

We can construct a DFA to decide MIN(R) by taking the DFA for R and redirecting all outgoing arrows from all the accept states to a dead state. So, MIN(R) must be regular.

d. Prove that Regular Sets are NOT closed under infinite union. (A counterexample suffices).

Consider the sets {0}, {01}, {0011}, etc. Each one is regular because it only contains one string. But the infinite union is the set $\{0^i1^i \mid i>=0\}$ which we know is not regular. So the infinite union cannot be closed for regular languages.

e. What about infinite intersection?

We know that

$\{0^i1^i \mid i>=0\} = \{0\} \cup \{01\} \cup \{0011\} \cup ...,$

Taking complements and applying DeMorgan's law gives us

$\{0^i1^i \mid i>=0\}^c = \{0\}^c \wedge \{01\}^c \wedge \{0011\}^c \wedge ...,$

Where we are using U to deonte union and ^ to denote intersection. Recall the complement of a regular language is regular, and hence the complement of a not-regular language is not regular. So we can conclude that the left hand side of the equation is not-regular, and each term in the intersection is regular. Therefore infinite intersection does not preserve regularity.

6. Regular Expressions

a. **(10+0)*(1+10)***
(10+0)* will generate all strings that do not contain a pair of 1s, and (1+10)*, the strings that do not contain a pair of 0s. So, the concatenation will generate all strings in which every occurrence of 00 precedes every occurrence of 11.

b. **0*(1+000*)*0***
We can generate a string which does not contain the occurrence 101 by making sure that any 0 in the middle (between two 1s) of the string must be paired with another 0.

c. **(e+0)(10)*(e+0)(10)*(e+1)(10)*(e+1) +**
(e+0)(10)*(e+1)(10)*(e+0)(10)*(e+0)
The both terms are just alternating 1's and 0s , eg (e+0)(10)*(e+1) where you are allowed to inert at most one extra 1 or 0 in between. We need two terms, depending on whether the double 1 or the double 0 comes first.

7. Converting Finite Automata to Regular Expressions

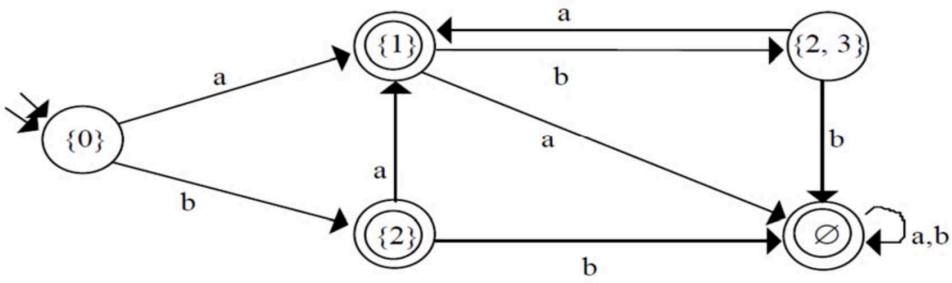
a. Write a regular expression for the language recognized by the following FSM:

Without using the algorithm for finding a regular expression from an FSM, we can note in this case that the lower right state is a dead state, i.e., an absorbing, non-accepting state. We can leave and return to the initial state, the only accepting state, by reading ab along the upper path or by reading ba along the lower path. These can be read any number of times, in any order, so the regular expression is (ab ∪ ba)*. Note that ε is included, as it should be.

b. Consider the following FSM M:
(i) Write a regular expression for the language accepted by M.

$$\varepsilon \cup ((a \cup ba)(ba)^*b)$$

(ii) Give a deterministic FSM that accepts the complement of the language accepted by M.



8. Regular Expression Identities

   a. r(s + t) and rs + rt are equivalent because the first describes a string from r followed by either a string from s or a string from t, and the second describes a string from r followed by a string from s or a string from r followed by a string from t and these two are clearly the same thing.

   b. (r*)*and r* are equivalent because the first describes the concatenation of an arbitrary number of terms that themselves are concatenations of arbitrary numbers of terms in r. This is the same as r* which is the concatenation of an arbitrary number of terms in r.

   c. (r + s)* and r*s* are not equivalent because if s_1 is a string in s and r_1 is a word in r then s_1r_1 is in (r+s)* but not r*s* but not the latter.

9. Final States

   a. Every NFA can be converted into an equivalent NFA with only a single accept state by creating a new accept state with epsilon moves from each of the old accept states.

   b. This does not work for DFAs. The DFAs of problems 1g, 1h, and 1i are all good counterexamples.

      In general if the minimum DFA for a regular language has more than one final state, then the language cannot be generated by a DFA with one final state. This is because minimization cannot increase the number of final states.

   c. *Claim:* The regular languages that can be represented by a DFA with one final state are of the form RS*, where R and s are regular prefix-free languages.

      Proof: We need the following lemma first: A prefix free regular language M can generated by a machine with one final state. Suppose we have DFA representation of M that has multiple final states. Then all outgoing transitions from those final states must go to dead states since M is prefix free. But when we mimize the DFA, all the dead states will become equivalent, and therefore all the final states will become equivalent too.
      We also need the following lemma: The Kleene star, M*, of prefix free regular language M can be generated by a machine with one final state. From the previous lemma we know there is a DFA that generates M that has one final state. We can make M* by taking the minimal DFA that accepts M and removing the transitions from the final state and collapsing it together with the initial state (while keeping it a final state).

      From these to lemmas it is clear that RS* can be generated by a machine with one final state if R and S are prefix free, because we can just concatenate the machines for R and S*.

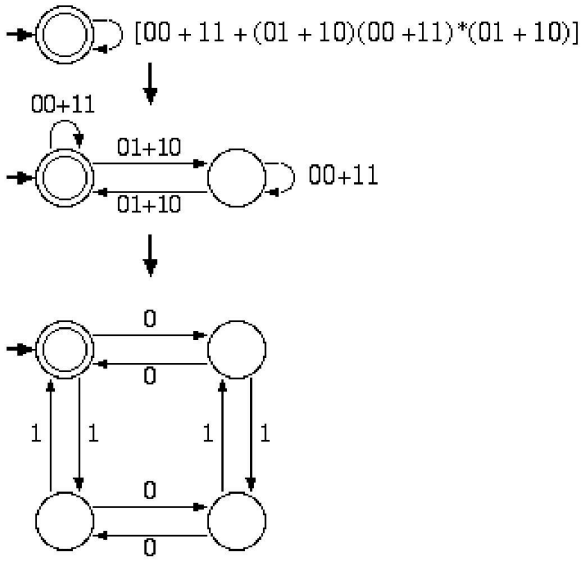
      Conversely, if L is generated by a DFA M with one final state, then L = Min(L)

(Min(L'))*, where L' is the language of the machine M' has the same states, transitions, and final state as M, and where we choose the final state of M to be the start state of M'. Since the Min of a language is always prefix free, L is of the form we claim.

10. Problems

a. Convert [00 + 11 + (01 + 10)(00 + 11)*(01 + 10)]* to a Finite Automaton.

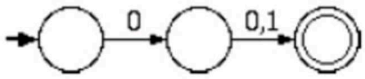
We just reverse the procedure for converting an NFA to a regular expression by ripping-in states.



(note: the rightmost state in the second diagram corresponds to the bottom right state in the third diagram.)

11. More Machines

Draw a finite state machine that accepts the complement of the language accepted by the non-deterministic machine:



answer: