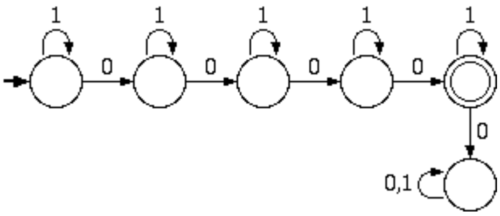
A R S D I G I T A   V N I V E R S I T Y

Month 8: Theory of Computation

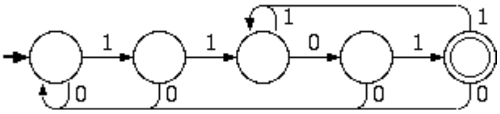
Problem Set 1 Solutions - Mike Allen and Dimitri Kountourogiannis

1. DFAs

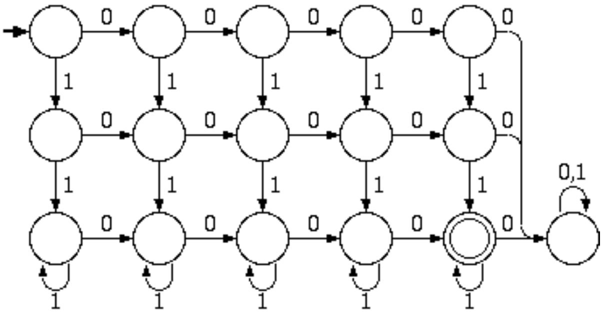
a. All strings that contain exactly 4 0s.



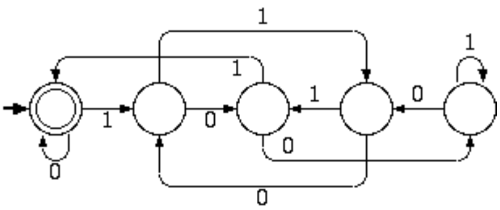
b. All strings ending in 1101.



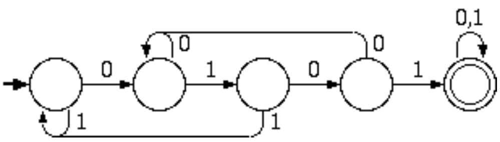
c. All strings containing exactly 4 0s and at least 2 1s.



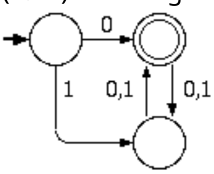
d. All strings whose binary interpretation is divisible by 5.

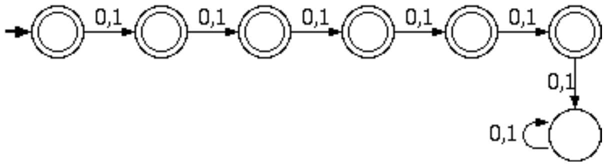


e. (1.4c) All strings that contain the substring 0101.

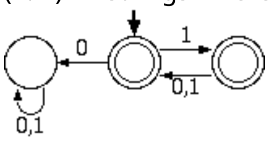


f. (1.4e) All strings that start with 0 and has odd length or start with 1 and has even length.



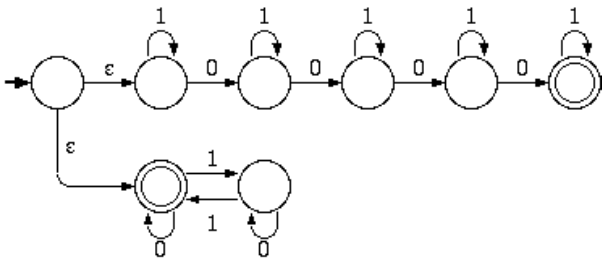


i. (1.4i) All strings where every odd position is a 1.

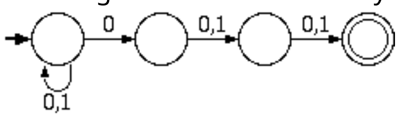


2. NFAs

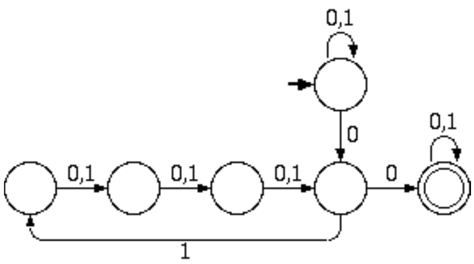
a. All strings containing exactly 4 0s or an even number of 1s. (8 states)



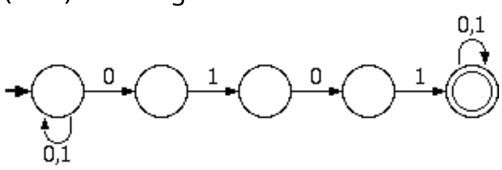
b. All strings such that the third symbol from the right end is a 0. (4 states)



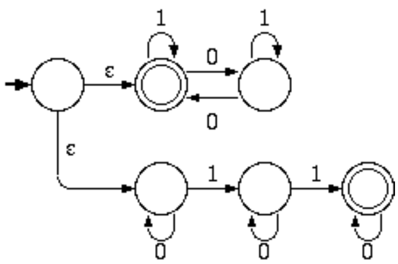
c. All strings such that some two zeros are separated by a string whose length is 4i for som
states)

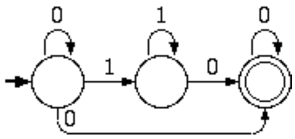


d. (1.5b) All strings that contain the substring 0101. (5 states)



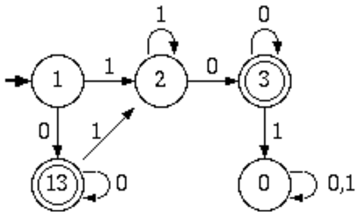
e. (1.5c) All strings that contains an even number of 0s or exactly two 1s. (6 states)



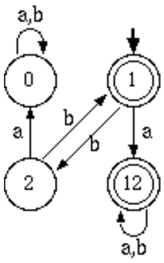


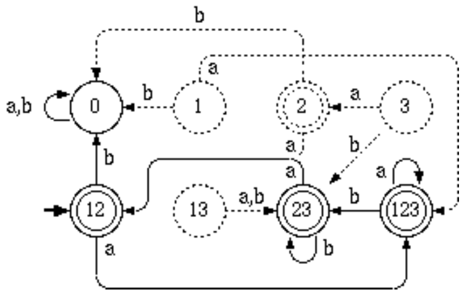
3. Converting NFAs to DFAs

   a. Convert the NFA in 2f into a DFA.



   b. 1.2a in the text
   c. 1.2b in the text





4. Discrete Math Review - Proofs

   L1: The set of strings where each string w has an equal number of zeros and ones; and any pr 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 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 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 r equal number of 0s and 1s. The prefix condition is slightly more difficult. We consid 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 prefix property.

   b. For those of you who are paying attention, this problem is extemely similar to the strean ghostbusters problem from algorithms. The proof is by induction on the length of string and w=uv do. Also, any prefix x of v cannot have more ones than zeros in it sir would be a prefix of w that had more ones than zeros. Therefore v must be in u and v are of length <= n, by the induction hypothesis they are in L2. Therefo 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 since w does. Also, no prefix x of u can have more ones than zeros, since then either have more ones than zeros which is impossible by hypothesis, or 0x wo same number of ones as zeros, which is also impossible by since j = n+1. Ther conclude that u is in L1, and since it is of length <=n it is in L2 by the induction

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

L1-L2 is the same as the intersection of L1 and the complement of L2. Since the set of re 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 ar 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 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 w in R where every proper prefix of w is in not in R. (Note that this is not simply the comp PREFIX).

We can construct a DFA to decide MIN(R) by taking the DFA for R and redirecting all outg 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 s infinite union is the set $\{0^i1^i \mid i>=0\}$ which we know is not regular. So the infinite union ca closed for regular languages.

e. What about infinite intersection?

We know that

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

Taking complements and applying DeMorgan's law gives us

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

Where we are using U to deonte union and ^ to denote intersection. Recall the complem regular language is regular, and hence the complement of a not-regular language is not we can conclude that the left hand side of the equation is not-regular, and each term in t and S_a denotes the set of states that can be reached from some state in S in one step wit simplify the notation, we are allowing multiple start states {(q, r, {r}) }. This could be also introducing a new state and introducing epsilon moves. The reason that this accepts Hal term r2 in (r1,r2,S) remembers the midpoint of a potential string in L, and S represents tl can reach from r2 in the number of steps it takes to get from the start q to r1. When r1=r contains all the paths that one can get to in twice as many steps as it took to get from q

An intuitive explanation The Half(L) problem is given a string w is there a string x of the as w such that wx is in the language L. This is hard to solve directly, so we break it into a subproblems of the following form: Fix a machine M that generates L and pick a state r i machine.

The problem Half(L,r)is then: Given a string w, is there a string x of the same length as w is in the language L *and* after reading in w, the machine M is in the state r.

We can reduce solving Half(L) to solving Half(L,r) for each state r in the machine M and o result. The reason this is good is that the problem Half(L,r) decomposes naturally into tw simple problems:

If we make the machine M' by making all accept states in M be reject states, and by maki accept state, does M' accept the string w?

and

If we make the machine M'' by making state r the start state, and changing all 0 transitio transitions and similarly all 1 transitions to 0,1 transitions, does the machine M'' accept t

What we have done in the second case is to ingnore what the value of any character in tl This is how to make a machine to accept all strings that have the same length as strings given machine. Putting all this together should result in a similar machine to what is give solution here, with possibly some missing extraneous states.

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 t contain a pair of 0s. So, the concatenation will generate all strings in which every occurre 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 tha 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 most one extra 1 or 0 in between. We need two terms, depending on whether the doubl double 0 comes first.

7. Converting Finite Automata to Regular Expressions

   a. (1.16a) `(a* + ba*b)*ba*`

terms that themselves are concatenations of arbitrary numbers of terms in r. This is the
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
(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 cre
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 counterex

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

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

Proof: We need the following lemma first: A prefix free regular language M can generate
machine with one final state. Suppose we have DFA representation of M that has multipl
Then all outgoing transitions from those final states must go to dead states since M is pr
when we mimize the DFA, all the dead states will become equivalent, and therefore all th
will become equivalent too.

We also need the following lemma: The Kleene star, M*, of prefix free regular language I
generated by a machine with one final state. From the previous lemma we know there is
generates M that has one final state. We can make M* by taking the minimal DFA that ac
removing the transitions from the final state and collapsing it together with the initial stá
keeping it a final state).

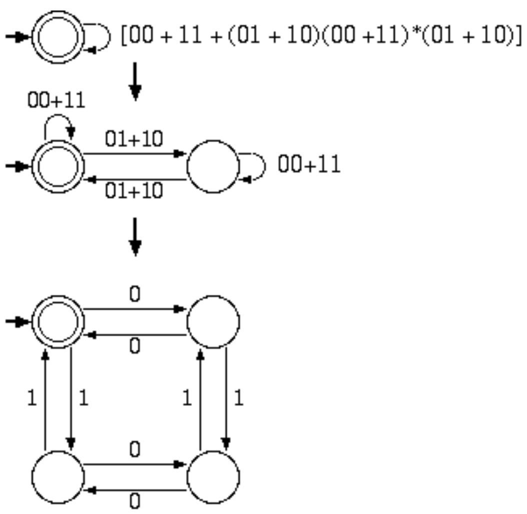
From these to lemmas it is clear that RS* can be generated by a machine with one final s
are prefix free, because we can just concatenate the machines for R and S*.

Conversely, if L is generated by a DFA M with one final state, then L = Min(L) (Min(L'))*, w
language of the machine M' has the same states, transitions, and final state as M, and w
choose the final state of M to be the start state of M'. Since the Min of a language is alwa
L is of the form we claim.

10. Optional Extra Problems

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

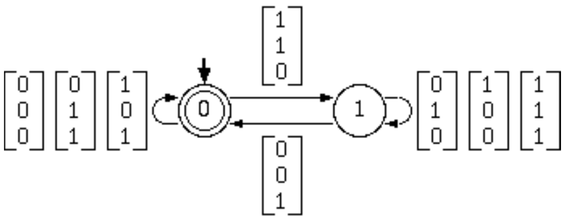
We just reverse the procedure for converting an NFA to a regular expression by ripping-i



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

b. (1.25) Let B = {w | the bottow row of w is the sum of the top two rows}. The reverse of B
decided by the NFA below, and since the set of regular languages is closed under revers
regular as well.

The NFA below determines if a string of columns composes a legal addition equation wh
two rows sum to the third. The two states correspond to whether the previous column le
carryout or not, and the legal transistions for each state correspond to columns which m
correctness of the equation. If an invalid column is added, no valid outgoing arrow is fou
computation dies (thus rejecting the input).



c. (1.41) Let D = {w | w contains an equal number of occurrences of 01 and 10}. This langu
decided by the DFA below, and so must be regular.

The DFA works because the number of 01 transistions must always we within one of the
transistions, so we need only remember which transistion came first (top path vs. botto
whether we have seen an even number or odd number of transistions (left state vs. righ