After this tutorial you should be able to:

1. Remove ϵ-transitions from NFAs
2. Convert NFAs to DFAs
3. Convert DFAs to REs
4. Devise algorithms for solving certain decision problems about DFAs/NFAs

---

Problem 1. Write pseudo-code for EC, i.e., a procedure that takes an NFA N and a pair (q, r) of states as input, and returns whether or not there is a path from q to r labeled by zero or more ϵ-transitions.

Solution 1. One way to do this is to do a graph traversal starting in q, following only epsilon transitions, and seeing if r is reached. We will use something called a Depth First Search (DFS).

```
1  def EC(p,q):
2      visited = [0]*len(V) # initially no node is visited
3      DFS(p) # do a depth first search of the graph
4      return visited[q]
5
6  def DFS(p):
7      visited[p]=1
8      for v in edge[p]:
9          if not visited[v]:
10             DFS(v)
11
12 V = {0,1,2,3,4} # the nodes of the NFA
13 edge = { # the epsilon edges of the NFA
14     0: [],
15     1: [2],
16     2: [1,3],
17     3: [3]
18     }
19
20 print(EC(1,3))
```

Here is another approach. We give an iterative solution for computing the relation EC:

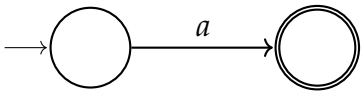
1. Let $E = \emptyset$.
2. For $q \in Q$ put $(q, q) \in \text{EC}$.
3. While EC $\neq E$ do:
   (a) let $E = \text{EC}$
   (b) For every pair $(q, q'') \in Q \times Q \setminus \text{EC}$, add (q, q'') to EC if there is some $q' \in Q$ such that $(q, q') \in \text{EC}$ and $q'' \in \delta(q', \epsilon)$.
4. Return EC.

The idea is that we iteratively add pairs of states to EC until no new pairs are added. This type of algorithm is called a *saturation* algorithm. The base case is in line 2, and the inductive case is in line 3b.

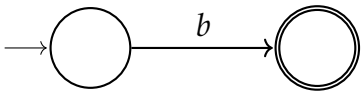
Problem 2.[Exam 2020] Using the methods from the course, convert the regular expression $a^* \mid b$ into an NFA, and then remove its epsilon transitions. Show all the steps.

Solution 2.

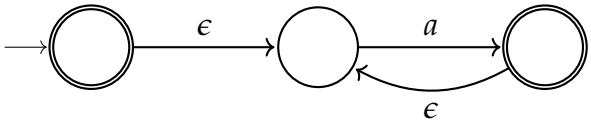
NFA for a:



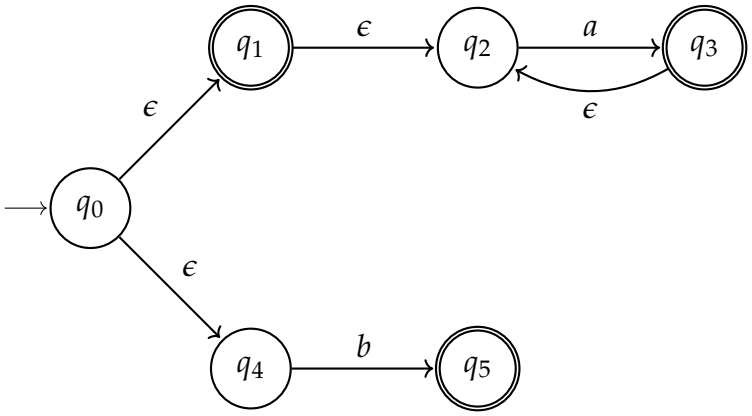
NFA for b:



NFA for a^*:



NFA for $a^* \mid b$ (states have been numbered):



To remove the ϵ-transitions, we compute EC. So it is easier to typeset, I write $\mathrm{EC}(q)$ for the set of states r such that $\mathrm{EC}(q, r)$:

- $\mathrm{EC}(q_0) = \{q_0, q_1, q_2, q_4\}$
- $\mathrm{EC}(q_1) = \{q_1, q_2\}$
- $\mathrm{EC}(q_2) = \{q_2\}$
- $\mathrm{EC}(q_3) = \{q_3, q_2\}$
- $\mathrm{EC}(q_4) = \{q_4\}$
- $\mathrm{EC}(q_5) = \{q_5\}$

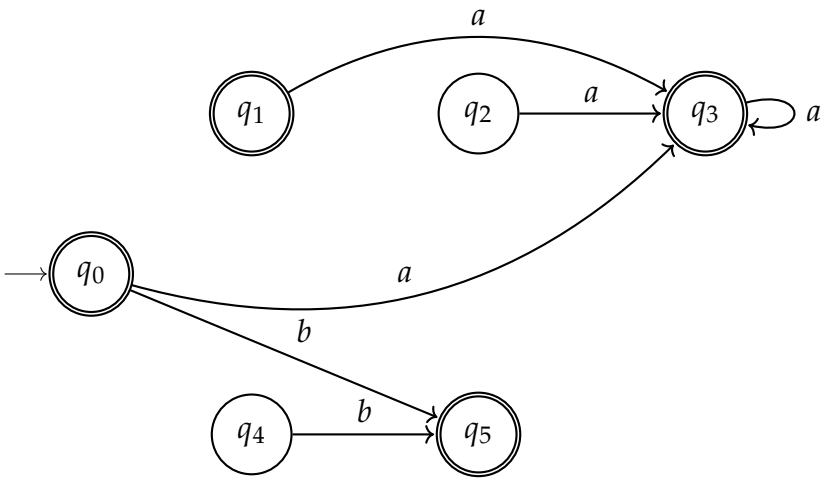
Since it is easier to visualise, let's write $p \rightsquigarrow q$ for $\mathrm{EC}(p, q)$.

Since q_1 is a final state, and $q_0 \rightsquigarrow q_1$, also q_0 is a final state of the NFA without epsilon transitions. Similarly, the following are alo final states of the NFA without epsilon transitions: q_1, q_3 and q_5.

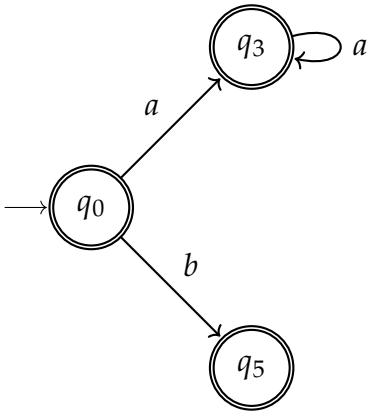
We then add new transitions to the NFA to "replace" the existing ϵ-transitions:

- $q_0 \rightsquigarrow q_2 \xrightarrow{a} q_3$, thus we add $q_0 \xrightarrow{a} q_3$.
- $q_0 \rightsquigarrow q_4 \xrightarrow{b} q_5$, thus we add $q_0 \xrightarrow{b} q_5$.
- $q_1 \rightsquigarrow q_2 \xrightarrow{a} q_3$, thus we add $q_1 \xrightarrow{b} q_3$.
- $q_3 \rightsquigarrow q_2 \xrightarrow{a} q_3$, thus we add $q_3 \xrightarrow{a} q_3$.

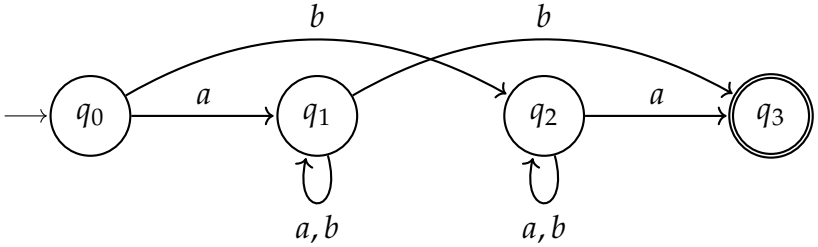
Thus, we get the following NFA:



After removing states that are unreachable from q_0, we get the following NFA:



Problem 3.[Exam 2022] Describe in words the language recognised by the following NFA (no additional justification is needed), and then give the DFA that results from applying the subset construction to the NFA:



You may draw or write the DFA, and no additional justification is needed. Note that you should only include the states in the DFA that are reachable from the initial state.

Solution 3.

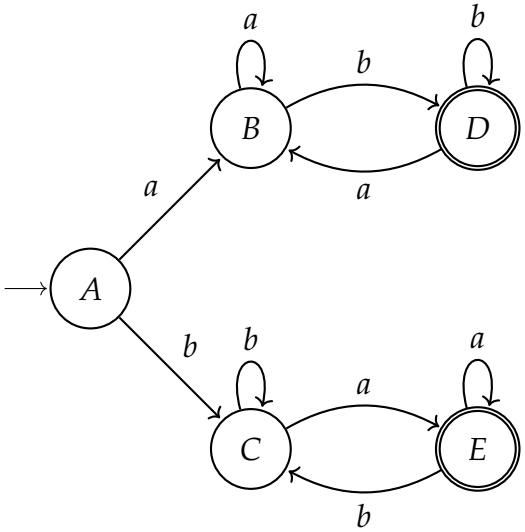
The provided NFA recognises strings of as and bs whose first and last symbols are different. To convert it into a DFA, we first generate the following table:

state	a	b
$\{q_0\}$	$\{q_1\}$	$\{q_2\}$
$\{q_1\}$	$\{q_1\}$	$\{q_1, q_3\}$
$\{q_2\}$	$\{q_2, q_3\}$	$\{q_2\}$
$\{q_1, q3\}$	$\{q_1\}$	$\{q_1, q_3\}$
$\{q_2, q_3\}$	$\{q_2, q_3\}$	$\{q_2\}$

We then relabel the states:

state	a	b
A	B	C
B	B	D
C	E	C
D	B	D
E	E	C

The start state is A, and the set of accept states is $\{D, E\}$.

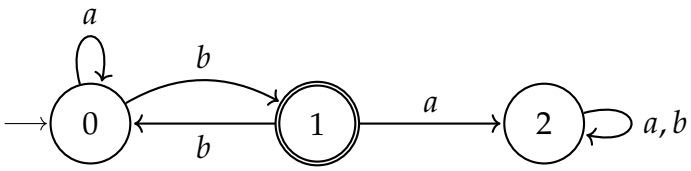


Problem 4.[Exam 2021] Consider the DFA M with states $Q = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $q_0 = 0$, $F = \{1\}$, and δ is as follows:

	a	b
0	0	1
1	2	0
2	2	2

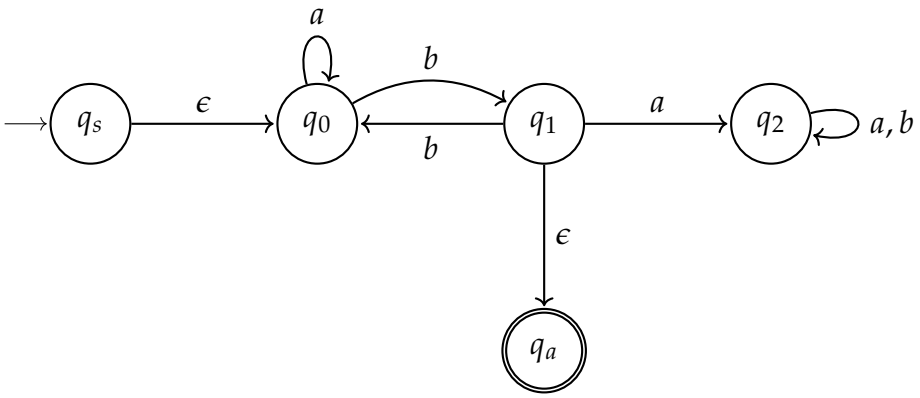
Give a short English description of $L(M)$ and a regular expression for $L(M)$. No additional justifications are needed. (In the exam the question did not specify how this should be done; in the tutorial, you should use the methods from lectures).

Solution 4.

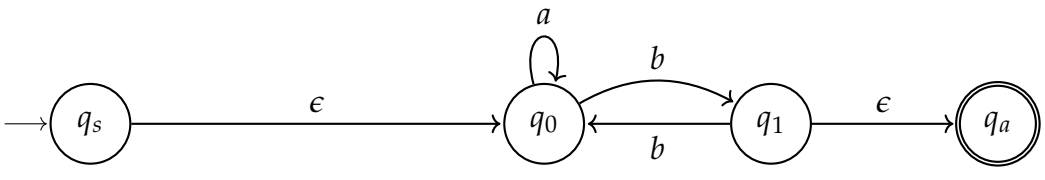


$L(M)$ is the set of strings formed by concatenating any number of a's and bb's, followed by a single b at the end.

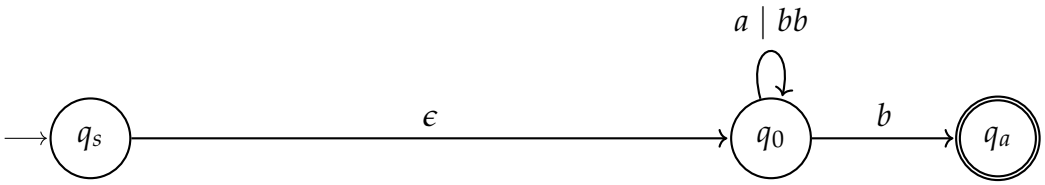
We now construct a regular expression for $L(M)$ using the method from the lectures.



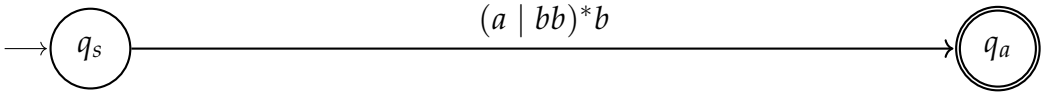
Eliminate q_2:



Eliminate q_1:



Eliminate q_0:



Thus, we obtain $(a \mid bb)^*b$ as a regular expression for $L(M)$.

Algorithms

Problem 5. Consider the decision problem which takes as input a regular expression R and string w, and outputs 1 if $w \in L(R)$, and 0 otherwise. This problem is called the *regular-expression membership problem*. In a previous tutorial we saw a recursive algorithm for solving this problem (called Match(R, w)). We gave the following algorithm in class:

1. Convert R into a NFA N such that $L(R) = L(N)$.
2. Check if $w \in L(N)$.

We say how to do step 1 in class. Write pseudocode for step 2.

Solution 5.

```
def membership(M,w):
    state = M.initial
    while not end_of_input(w):
        x = get_char(w)
        nextstate = set()
        for s in state:
            nextstate.union(M.δ(s,x))
        state = nextstate
    for s in state:
        if s in F:
            return "Accept"
    else:
        return "Reject"
```

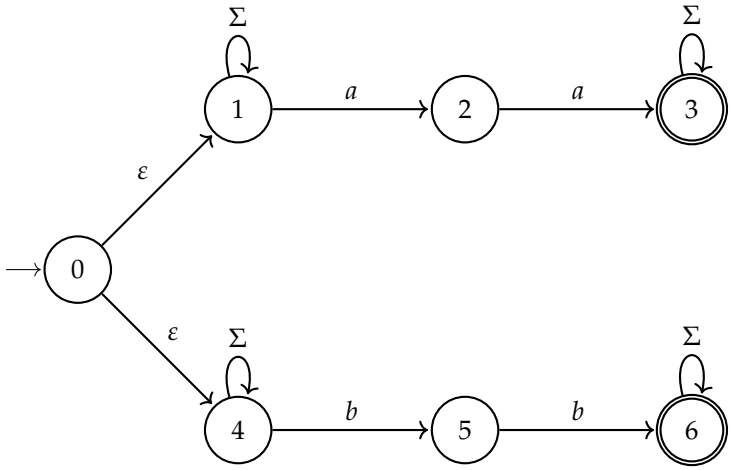
Problem 6. Consider the decision problem which takes as input regular expressions R_1, R_2, and outputs 1 if $L(R_1) = L(R_2)$, and 0 otherwise. This problem is called the *regular-expression equivalence problem*. Give an algorithm that solves this decision problem. (Hint: use automata).

Extra

Solution 6. Using the techniques from lectures:

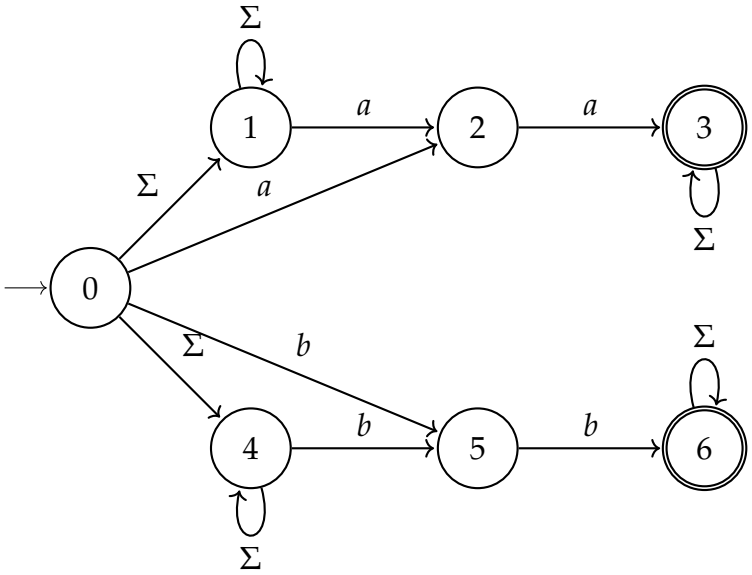
1. Convert R_i into a DFA M_i such that $L(R_i) = L(M_i)$.
2. Check if $L(M_1) = L(M_2)$ using the algorithm for the DFA equivalence problem.

Problem 7. Here is an NFA over $\Sigma = \{a, b\}$. What language does it describe? Transform the NFA into an equivalent DFA, using the method shown in lectures.



Solution 7. The language is all strings with two consecutive a's or two consecutive b's.

Here is the transformation. First we remove the ϵ-transitions. To do this, we compute the $\text{EC}(q)$ of every state q: $\text{EC}(0) = \{0, 1, 4\}$, and $\text{EC}(q) = \{q\}$ for every other state q. So, after removing the ϵ-transitions we have the following NFA:



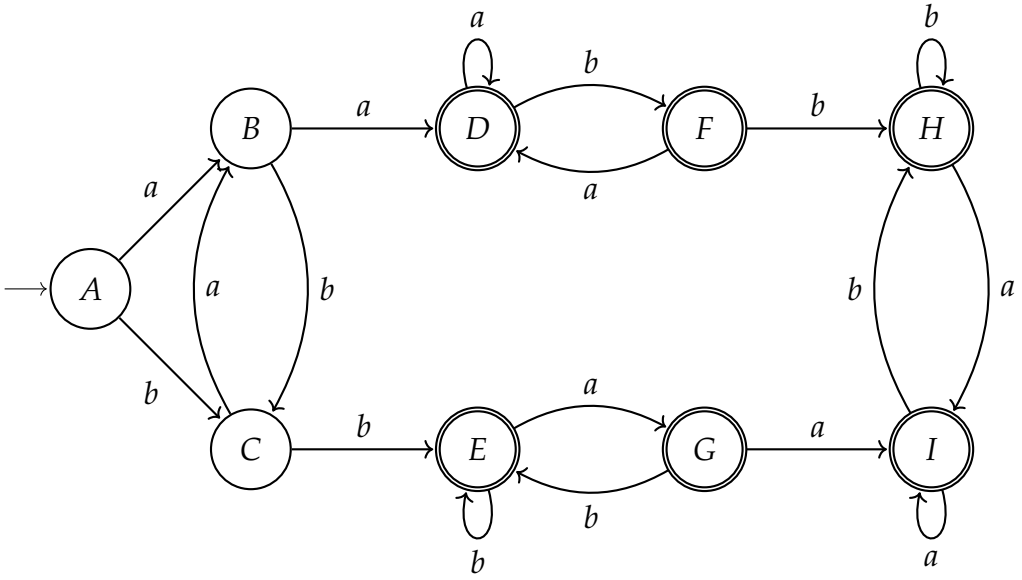
state	a	b
$\{0\}$	$\{1,2,4\}$	$\{1,4,5\}$
$\{1,2,4\}$	$\{1,2,3,4\}$	$\{1,4,5\}$
$\{1,4,5\}$	$\{1,2,4\}$	$\{1,4,5,6\}$
$\underline{\{1,2,3,4\}}$	$\{1,2,3,4\}$	$\{1,3,4,5\}$
$\underline{\{1,4,5,6\}}$	$\{1,2,4,6\}$	$\{1,4,5,6\}$
$\underline{\{1,3,4,5\}}$	$\{1,2,3,4\}$	$\{1,3,4,5,6\}$
$\underline{\{1,2,4,6\}}$	$\{1,2,3,4,6\}$	$\{1,4,5,6\}$
$\underline{\{1,3,4,5,6\}}$	$\{1,2,3,4,6\}$	$\{1,3,4,5,6\}$
$\underline{\{1,2,3,4,6\}}$	$\{1,2,3,4,6\}$	$\{1,3,4,5,6\}$

Relabel the states to make it easier to read:

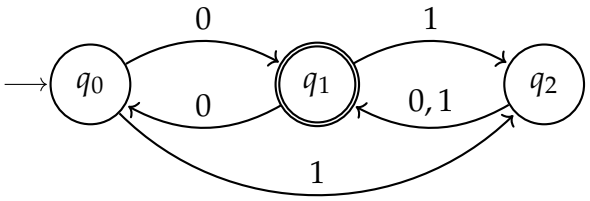
state	a	b
A	B	C
B	D	C
C	B	E
$\underline{D}$	D	F
$\underline{E}$	G	E
$\underline{F}$	D	H
$\underline{G}$	I	E
$\underline{H}$	I	H
$\underline{I}$	I	H

Start state is A

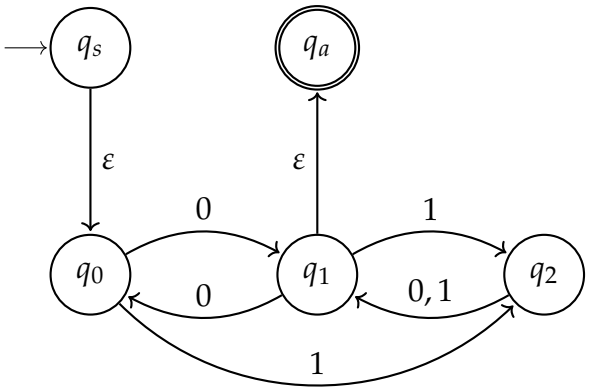
Set of accept states is $\{D, E, F, G, H, I\}$



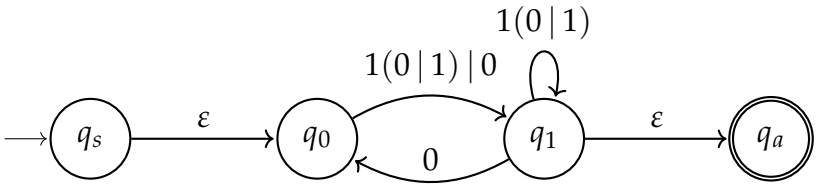
Problem 8. Convert the DFA to a regular expression.



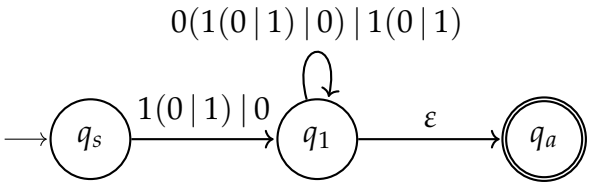
Solution 8.



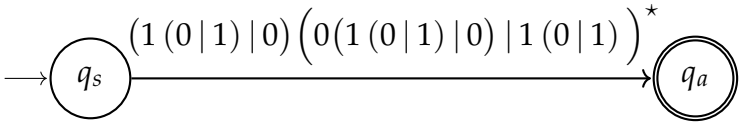
Eliminate q_2



Eliminate q_0



Eliminate q_0

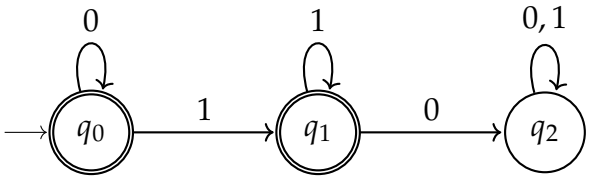


This is a very ugly regular expression. We should simplify it a bit.

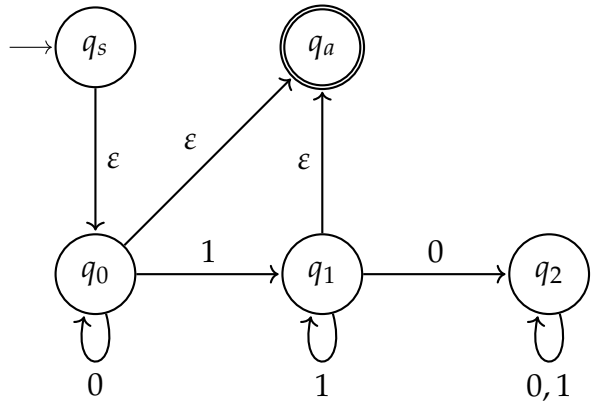
$$
\begin{aligned}
&\big(1\,(0\,|\,1)\,|\,0\big)\Big(0\big(1\,(0\,|\,1)\,|\,0\big)\,|\,1\,(0\,|\,1)\Big)^\star \\
&= (10\,|\,11\,|\,0)\,\Big(0\big(1\,(0\,|\,1)\,|\,0\big)\,|\,1\,(0\,|\,1)\Big)^\star \\
&= (10\,|\,11\,|\,0)\,\big(0\,(10\,|\,11\,|\,0)\,|\,1\,(0\,|\,1)\,\big)^\star \\
&= (10\,|\,11\,|\,0)\,\big(\,(010\,|\,011\,|\,00)\,|\,1\,(0\,|\,1)\,\big)^\star \\
&= (10\,|\,11\,|\,0)\,\big(\,(010\,|\,011\,|\,00)\,|\,(10\,|\,11)\,\big)^\star \\
&= (10\,|\,11\,|\,0)\,(010\,|\,011\,|\,00\,|\,10\,|\,11)^\star \\
&= (0\,|\,10\,|\,11)\,(00\,|\,010\,|\,011\,|\,10\,|\,11)^\star \\
&= (0\,|\,10\,|\,11)\,\big(00\,|(01\,|\,1)(0\,|\,1)\big)^\star
\end{aligned}
$$

When you have a fairly simple regular expression like this, it's a lot easier to see that it's correct.

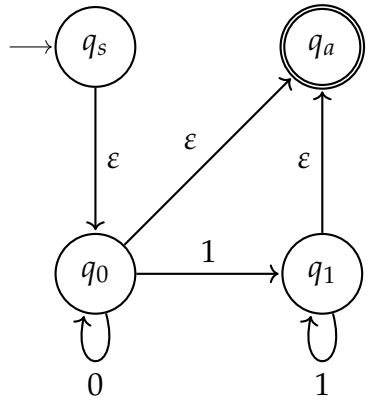
Problem 9. Convert the DFA to a regular expression.



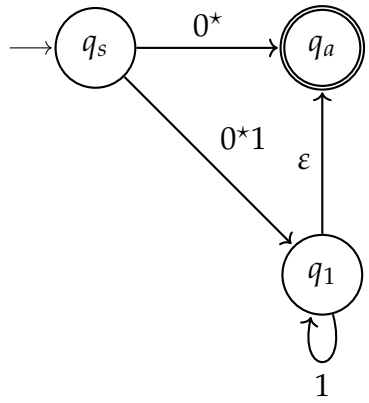
Solution 9.



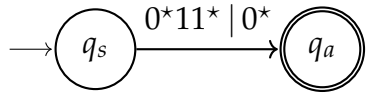
Eliminate q_2 (no outgoing transitions, very easy!)



Eliminate q_0



Eliminate q_1

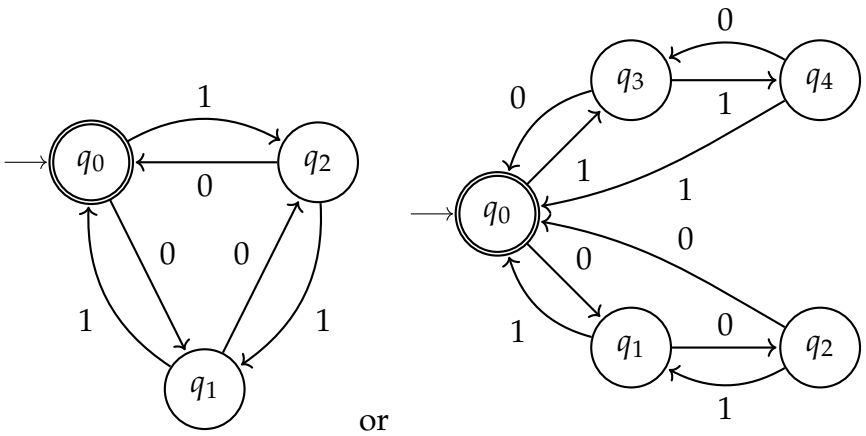


regular expression is $0^\star 1 1^\star \mid 0^\star$

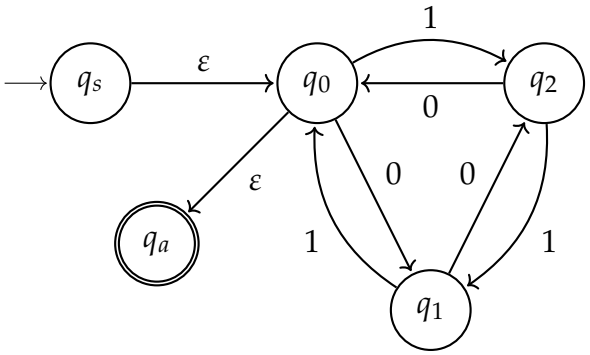
Problem 10. Construct a DFA which accepts strings of 0's and 1's, where the number of 1's modulo 3 equals the number of 0's modulo 3.

Give the corresponding regular expression

Solution 10. a)

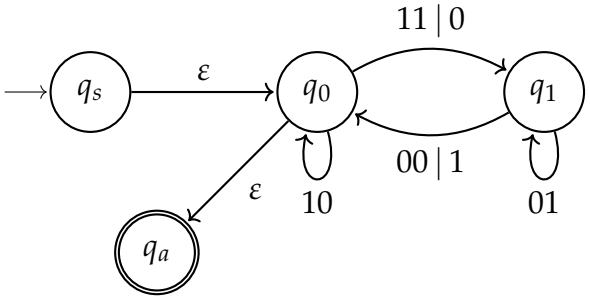


Using the first (smaller) automaton:



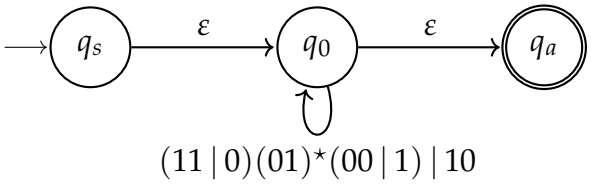
eliminate q_2:

$$
\begin{aligned}
(q_0, q_0) &: R_1 = 1, R_2 = \emptyset, R_3 = 0, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) &&= 10 \\
(q_0, q_1) &: R_1 = 1, R_2 = \emptyset, R_3 = 1, R_4 = 0, (R_1)(R_2)^\star(R_3) \,|\, (R_4) &&= 11 \,|\, 0 \\
(q_1, q_0) &: R_1 = 0, R_2 = \emptyset, R_3 = 0, R_4 = 1, (R_1)(R_2)^\star(R_3) \,|\, (R_4) &&= 00 \,|\, 1 \\
(q_1, q_1) &: R_1 = 0, R_2 = \emptyset, R_3 = 1, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) &&= 01
\end{aligned}
$$

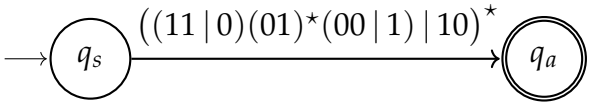


eliminate q_1:

$$
\begin{aligned}
(q_0, q_0) &: R_1 = 11 \,|\, 0, R_2 = 01, R_3 = 00 \,|\, 1, R_4 = 10 \\
&\quad (R_1)(R_2)^\star(R_3) \,|\, (R_4) = (11 \,|\, 0)(01)^\star(00 \,|\, 1) \,|\, 10
\end{aligned}
$$



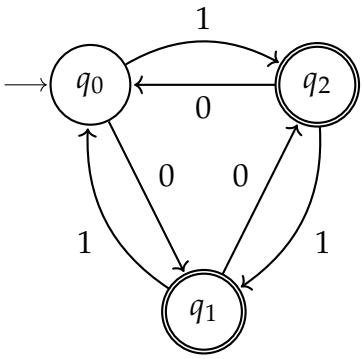
eliminate q_0

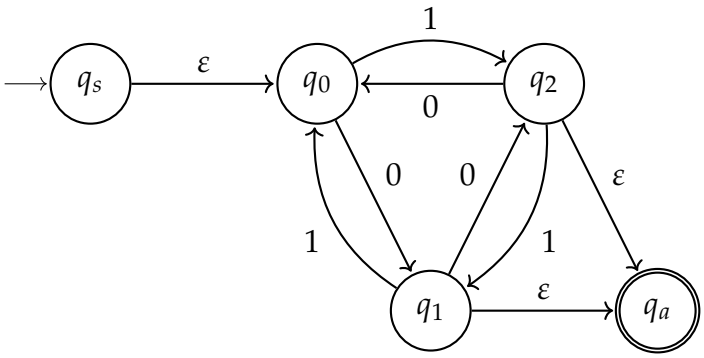


Problem 11. Construct a DFA which accepts strings of 0's and 1's, where the number of 1's modulo 3 does not equal the number of 0's modulo 3.

Give the corresponding regular expression.

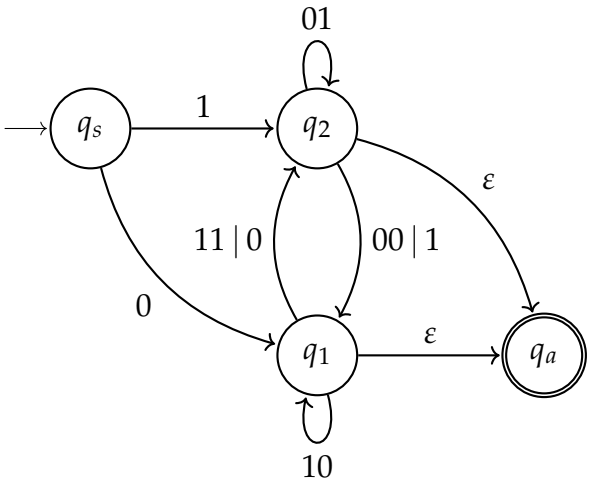
Solution 11.





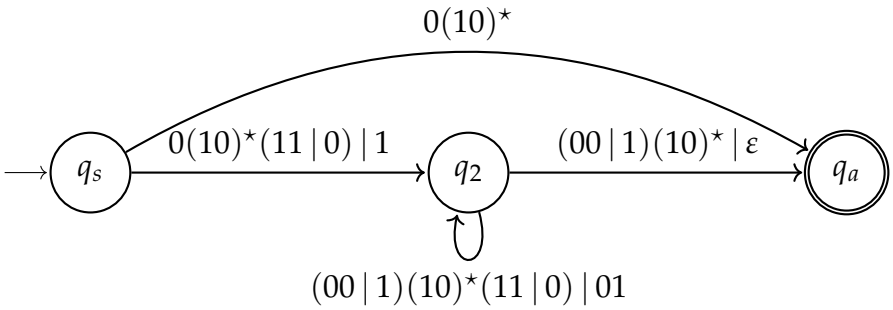
eliminate q_0:

$$
\begin{aligned}
(q_s, q_1) &: R_1 = \varepsilon, R_2 = \emptyset, R_3 = 0, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 0 \\
(q_s, q_2) &: R_1 = \varepsilon, R_2 = \emptyset, R_3 = 1, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 1 \\
(q_1, q_1) &: R_1 = 1, R_2 = \emptyset, R_3 = 0, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 10 \\
(q_1, q_2) &: R_1 = 1, R_2 = \emptyset, R_3 = 1, R_4 = 0, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 11 \,|\, 0 \\
(q_2, q_1) &: R_1 = 0, R_2 = \emptyset, R_3 = 0, R_4 = 1, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 00 \,|\, 1 \\
(q_2, q_2) &: R_1 = 0, R_2 = \emptyset, R_3 = 1, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 01
\end{aligned}
$$



eliminate q_1:

$$
\begin{aligned}
(q_s, q_a) &: R_1 = 0, R_2 = 10, R_3 = \varepsilon, R_4 = \emptyset, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 0(10)^\star \\
(q_s, q_2) &: R_1 = 0, R_2 = 10, R_3 = 11 \,|\, 0, R_4 = 1, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = 0(10)^\star(11 \,|\, 0) \,|\, 1 \\
(q_2, q_a) &: R_1 = 00 \,|\, 1, R_2 = 10, R_3 = \varepsilon, R_4 = \varepsilon, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = (00 \,|\, 1)(10)^\star \,|\, \varepsilon \\
(q_2, q_2) &: R_1 = 00 \,|\, 1, R_2 = 10, R_3 = 11 \,|\, 0, R_4 = 01, (R_1)(R_2)^\star(R_3) \,|\, (R_4) && = (00 \,|\, 1)(10)^\star(11 \,|\, 0) \,|\, 01
\end{aligned}
$$



Which gets us to

$$
\begin{aligned}
R_1 &= 0(10)^\star(11 \mid 0) \mid 1 \\
R_2 &= (00 \mid 1)(10)^\star(11 \mid 0) \mid 01 \\
R_3 &= (00 \mid 1)(10)^\star \mid \varepsilon \\
R_4 &= 0(10)^\star
\end{aligned}
$$

$$(R_1)(R_2)^\star(R_3) \mid (R_4) = \big(0\,(10)^\star\,(11 \mid 0) \mid 1\big)\big(\,(00 \mid 1)\,(10)^\star\,(11 \mid 0) \mid 01\big)^\star\big(\,(00 \mid 1)\,(10)^\star \mid \varepsilon\big) \mid 0\,(10)^\star$$

Yikes. That's a pretty ugly regular expression! It looks like it's probably correct. Lets see how sane it is.

Let n be the number of 0's - number of 1's

$R_1 = 0(10)^\star(11 \mid 0) \mid 1 \Rightarrow$	$n\%3 = 2$
$R_2 = (00 \mid 1)(10)^\star(11 \mid 0) \mid 01 \Rightarrow$	$n\%3 = 0$
$R_3 = (00 \mid 1)(10)^\star \mid \varepsilon \Rightarrow$	$n\%3 = 2$ or 0
$R_4 = 0(10)^\star \Rightarrow$	$n\%3 = 1$
$(R_1)(R_2)^\star(R_3) \Rightarrow$	$n\%3 = 2$ or 1
$(R_1)(R_2)^\star(R_3) \mid R_4 \Rightarrow$	$n\%3 = 2$ or 1

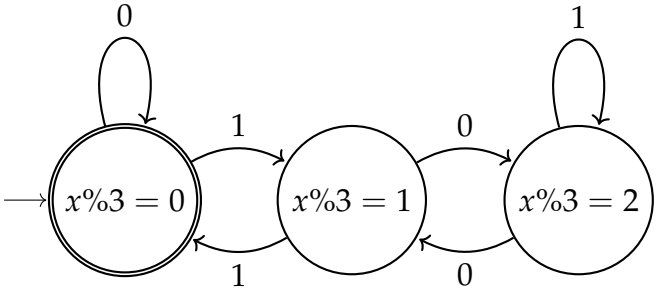
So, any string matching the regular expression does not have the same number of 0s and 1s mod 3. It's harder to confirm that this regular expression matches all such strings, but if there aren't any errors in the transformation from the NFA, then it will.

Problem 12.

1. Construct a DFA which accepts binary numbers that are multiples of 3, scanning the most significant bit (i.e. leftmost) first. Hint: consider the value of the remainder so far, as we scan across the number.
2. Construct an automaton which accepts binary numbers which are NOT multiples of 3.

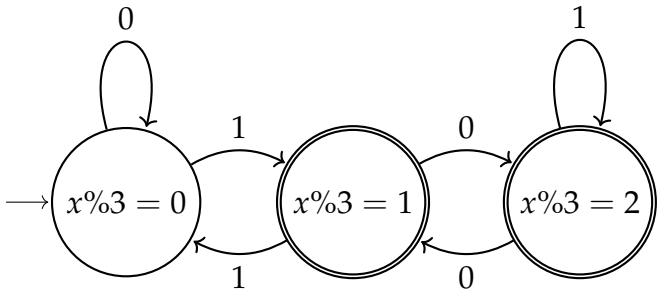
Solution 12. (For simplicity, we decided to treat empty binary strings as representing 0.)

1. Each state will remember what the remainder is so far.



To understand the transitions, use the following reasoning. Suppose the automaton has read a string u corresponding to number n and is in the state numbered $i = n\%3$. Then if the next symbol read is a 1, the string read so far is $u1$, which corresponds to the number $2n + 1$, which has remainder $2i + 1\%3$ when divided by three (this explains the transition on input 1 from state 0 to state 1, from state 1 to state 0, and from state 2 to state 2). Similarly, if the next symbol read is a 0, the string read so far is $u0$, which corresponds to the number $2n$, which has remainder $2n\%3$ when divided by three (this explains the transition on input 0 from state 0 to itself, from state 1 to state 2, and from state 2 to state 1).

2. If we set $F' = Q \setminus F$ (i.e. invert the accept / non-accepting status of all the states), then the automata will accept the complement of the original language.



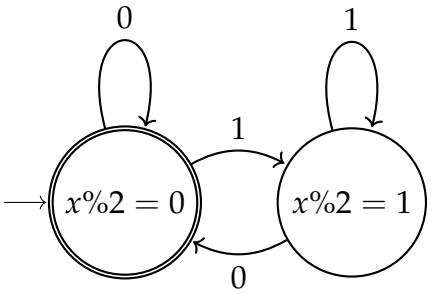
Inverting the accept states of any *DFA* M will always result in a machine which recognises the complement of $L(M)$. However, this approach does *not* work for all NFA (think about why!)

Problem 13.

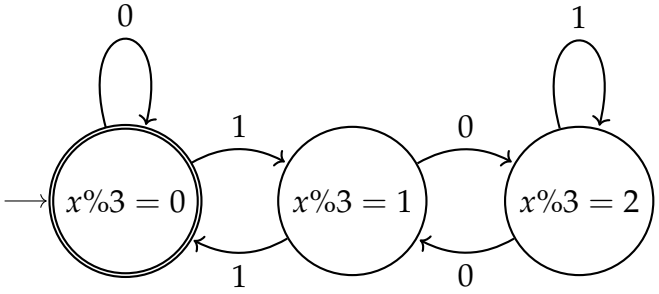
1. Construct a DFA which recognises binary numbers which are multiples of 2.
2. Construct a DFA which recognises binary numbers which are multiples of 3 (previous exercise)
3. Using these two DFA, construct an NFA which accepts binary numbers which are either not a multiple of 2, or not a multiple of 3, or both. (i.e. 2 is accepted, 3 is accepted, but 6 is not accepted)
4. Convert it to a DFA
5. Swap the accept and non-accept states. What is the language accepted by this new DFA?

Solution 13. (For simplicity, we decided to treat empty binary strings as representing 0.)

1. Construct a DFA which recognises binary numbers which are multiples of 2.

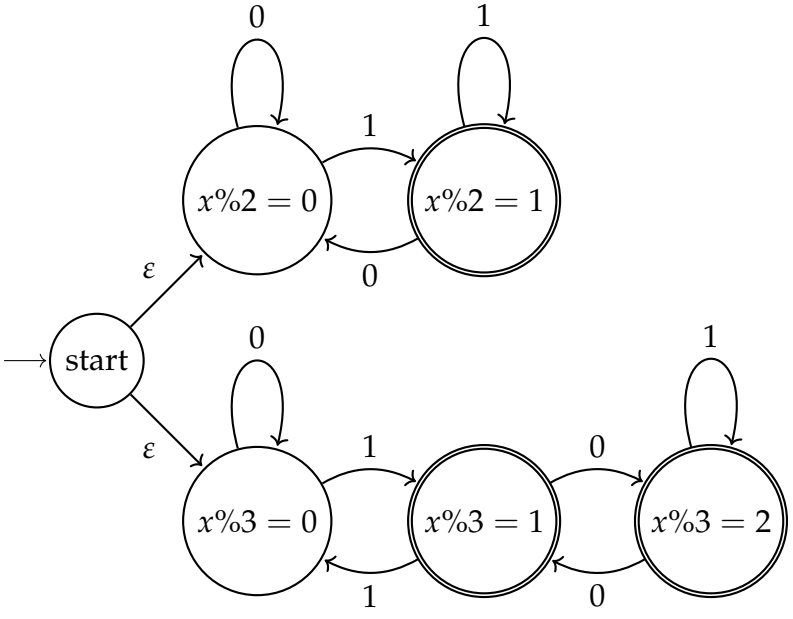


2. Construct a DFA which recognises binary numbers which are multiples of 3 (previous exercise)

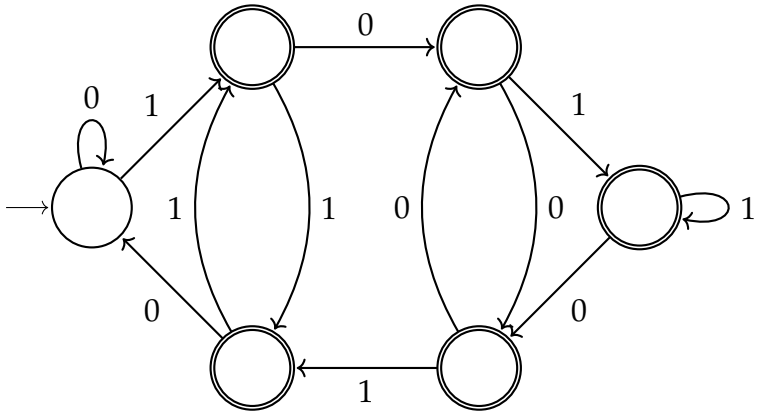


3. Using these two DFA, construct an NFA which accepts binary numbers which are either not a multiple of 2, or not a multiple of 3, or both. (i.e. 2 is accepted, 3 is accepted, but 6 is not accepted)

   We invert the accept/non accept states, then construct the union of the two DFA



4. Convert it to a DFA



5. Swap the accept and non-accept states. What is the language accepted by this new DFA?

   Binary numbers divisible by both 2 and 3 (i.e. binary numbers divisible by 6)

   We could have predicted this outcome using *DeMorgan's Law*: $(\neg A \vee \neg B) \equiv \neg(A \wedge B)$