After this tutorial you should be able to:

1. Express in English the language of a DFA/NFA.
2. Design DFAs/NFAs for languages specified in English.
3. Convert RE to NFAs
4. Form a DFA for the union or intersection of two DFAs using the product construction.
5. Devise algorithms for solving certain decision problems about DFAs/NFAs

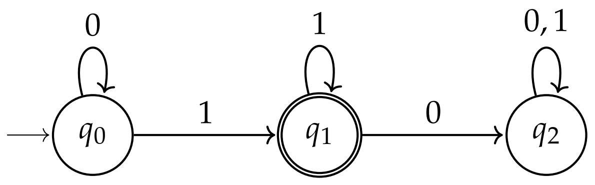
In your tutorial, I suggest you do one or two questions from each of the four sections: DFA, NFA, from RE to NFA, Algorithms. Use the rest of the problems and subproblems as practice, and to check you have mastered the material.

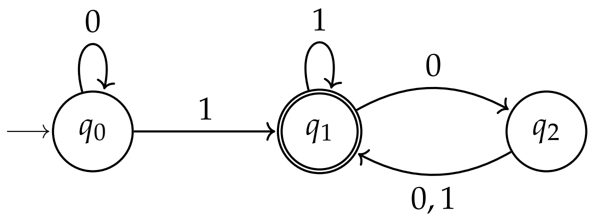
1 DFA

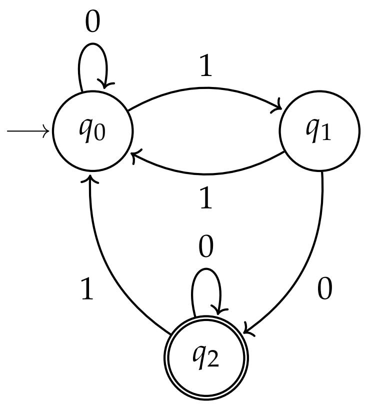
Problem 1. For each of the following DFAs:

(a) Give $(Q, \Sigma, \delta, q_0, F)$ for the automaton.

(b) Give a precise and concise English description of the language recognised by this automaton.







Solution 1.

1. (a) $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   q_0 is the initial state
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_2	q_2

   $F = \{q_1\}$

   (b) The set of binary strings in which there is at least one 1, and no 0 follows a 1.

2. (a) $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   q_0 is the initial state
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

   $F = \{q_1\}$

   (b) Strings of 1s and 0s which contain at least one 1 and which do not end with an odd number of 0s.

3. (a) $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   q_0 is the initial state
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_2	q_0

   $F = \{q_2\}$

   (b) Binary strings which end in 0 and which contain an odd number of 1s

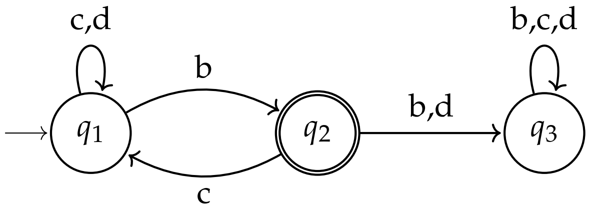
Problem 2. Draw the following automaton and give the language that it recognises:

- $Q = \{q_1, q_2, q_3\}$ is the set of states.
- $\Sigma = \{b, c, d\}$ is the alphabet,
- q_1 is the start state.
- $F = \{q_2\}$ is the set of accept states,

and the $\delta : Q \times \Sigma \to Q$ is given by the following table:

	b	c	d
q_1	q_2	q_1	q_1
q_2	q_3	q_1	q_3
q_3	q_3	q_3	q_3

Solution 2.



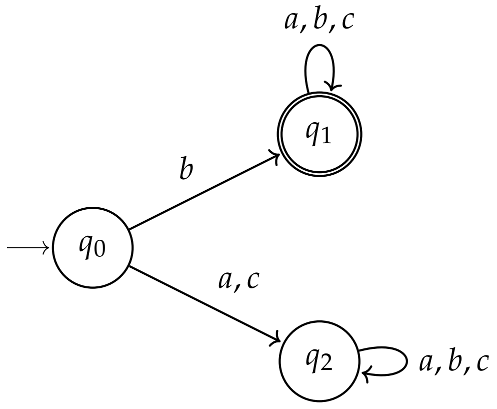
Accepts strings which end in b and in which b is never followed by b or d.

Problem 3. For each of the following languages, draw a DFAs which recognise them. The alphabet for all the languages is $\{a, b, c\}$.

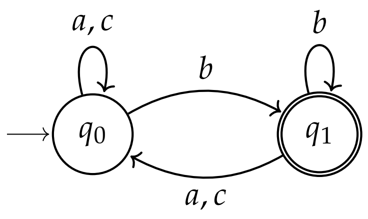
1. Strings that start with b
2. Strings that end with b
3. Strings that contain exactly one b
4. Strings that contain at least one b and are of length exactly 2

Solution 3.

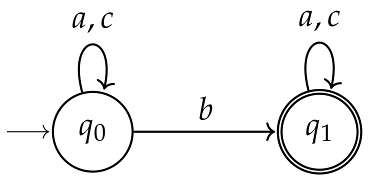
1. Start with b



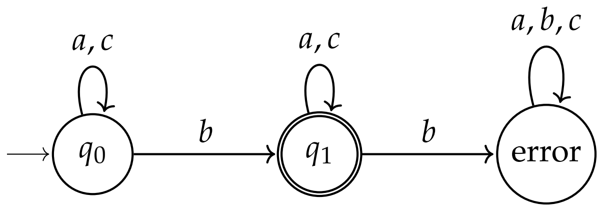
2. End with b



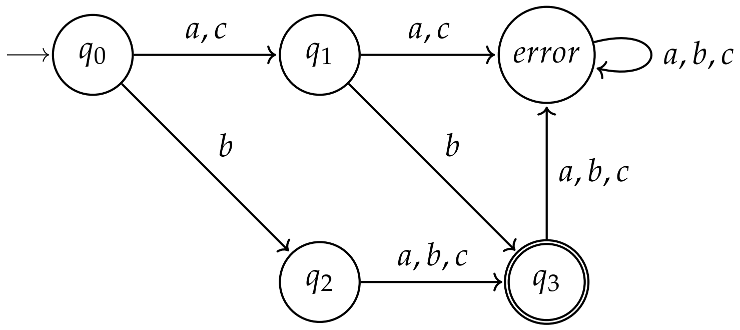
3. Contain exactly one b
   (Error states not shown)



   Or, with an error state:



4. Contain at least one b and are of length exactly 2

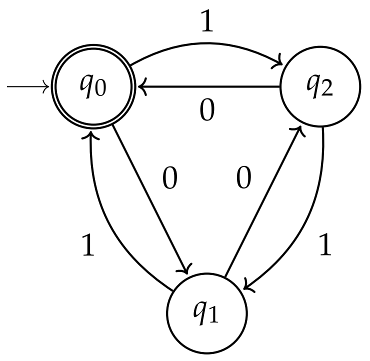


Problem 4.

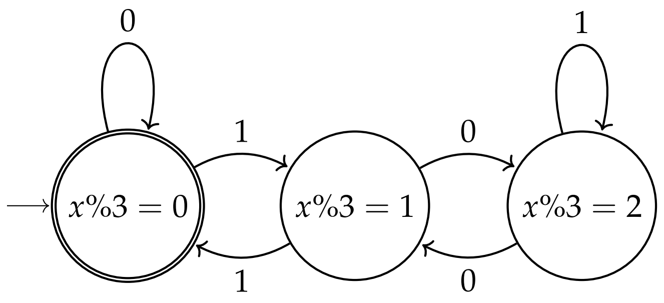
1. Construct a DFA which accepts strings of 0s and 1s, where the number of 1s modulo 3 equals the number of 0s modulo 3.
2. Construct a DFA which accepts binary numbers that are multiples of 3, scanning the most significant bit (i.e. leftmost) first. Hint: let each state 'remember' the remainder of the string read so far.
3. Using the previous automaton, construct an automaton which accepts binary numbers which are NOT multiples of 3.

Solution 4.

1. For the first language:

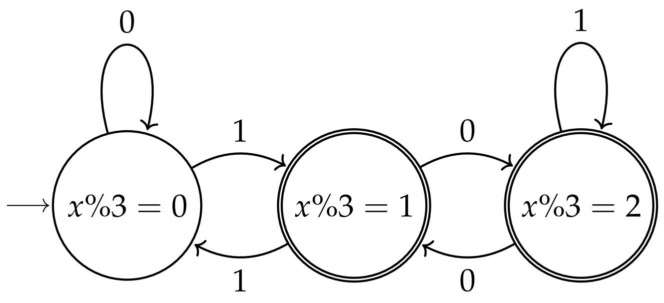


2. Each state will remember what the remainder is so far. We write $n\%3$ for the remainder of n divded by 3.



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 3. 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 3. 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.

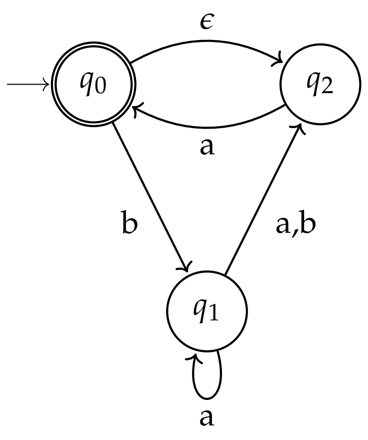
3. If we set $F' = Q \setminus F$ (i.e. invert the accept / non-accepting status of all the states), then the automaton will accept the complement of the original language.



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!)

2 NFA

Problem 5. Consider the following NFA:



Does the NFA accept:

1. ϵ
2. a
3. baba
4. aaab

Solution 5.

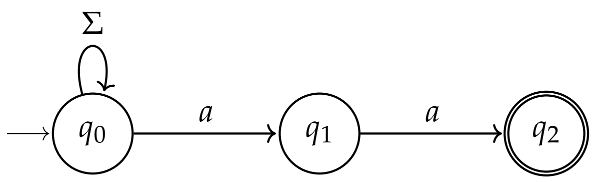
1. ϵ ACCEPTED (by path q_0)
2. a ACCEPTED (by path q_0, q_2, q_0)
3. baba ACCEPTED (by path q_0, q_1, q_1, q_2, q_0)
4. aaab REJECTED (ended in the set of states $\{q_1\}$, which does not contain an accept state)

Problem 6. Draw an NFA for each of the following languages. The alphabet is $\Sigma = \{a, b, c\}$. You may use epsilon transitions.

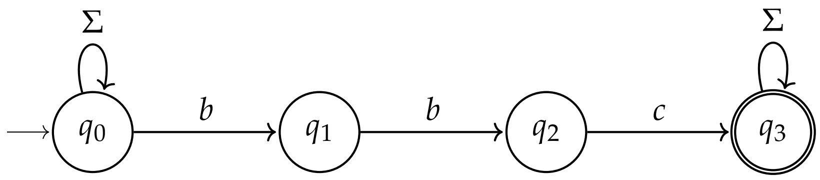
1. Strings that end with *aa* (use only 3 states)
2. Strings that contain the substring 'bbc' (use only 4 states)
3. Strings that start with a or end in c (use only 4 states)
4. Strings that start with a and end in c (use only 3 states)

Solution 6.

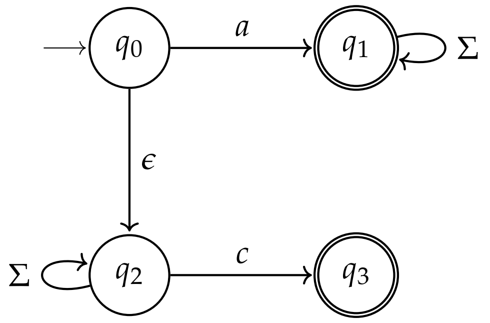
1. Strings that end with *aa* (use only 3 states)



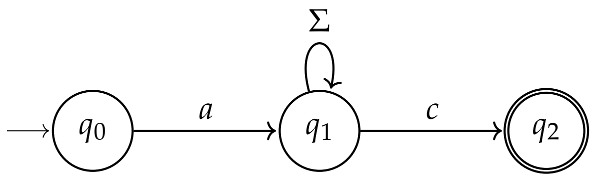
2. Strings that contain the substring 'bbc' (use only 4 states)



3. Strings that start with a or end in c (use only 4 states)



4. Strings that start with a and end in c (use only 3 states)



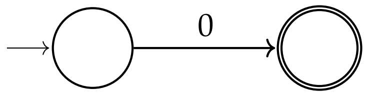
3 From RE to NFA

Problem 7. Use the construction demonstrated in lectures to construct an NFA for each of the following REs:

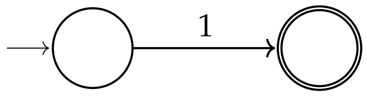
1. $0^\star 110^\star$
2. $(((00)^\star(11)) \mid 10)^\star$

Solution 7.

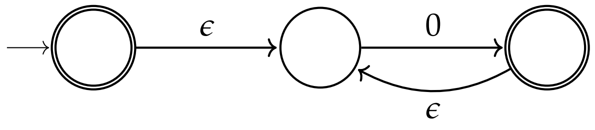
1. (a) NFA for 0:



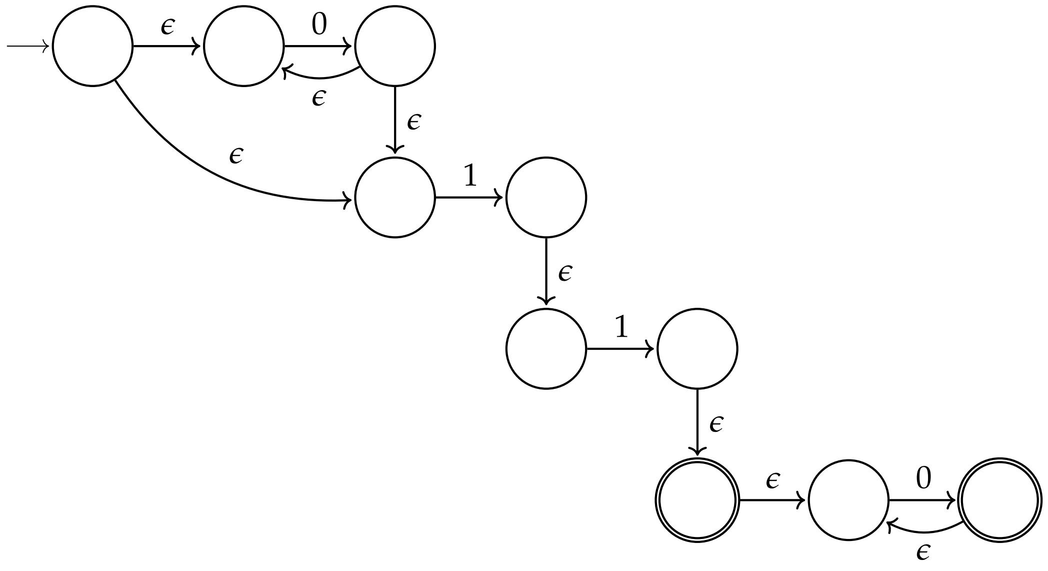
(b) NFA for 1:



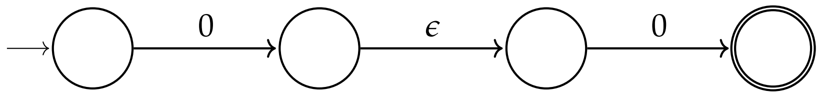
(c) NFA for $0^\star$:



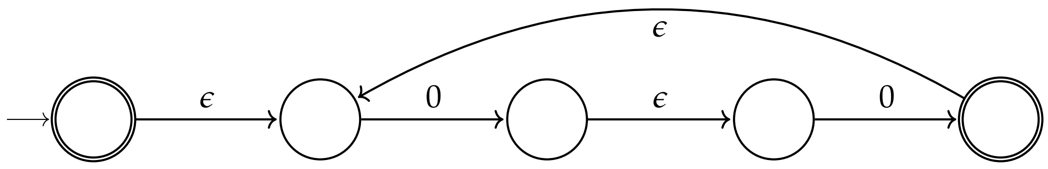
(d) Concatenate the various machines to get the final answer:



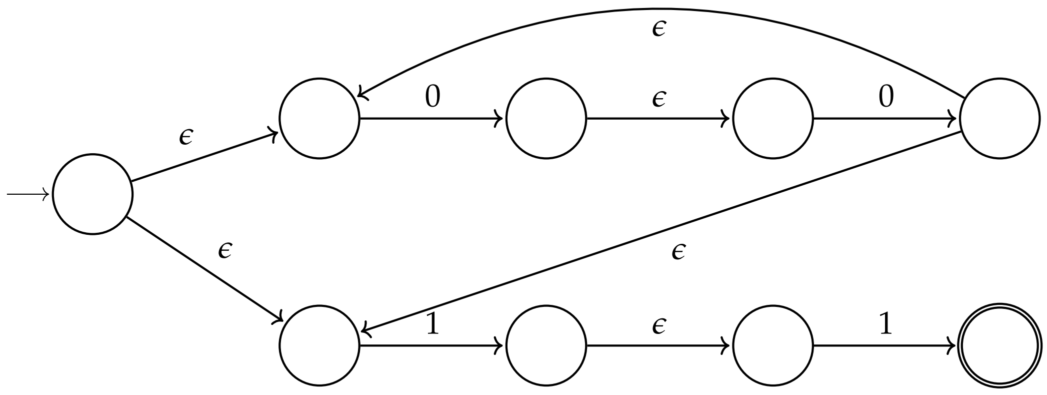
2. (a) NFA for 00 (similarly for 11, 10):



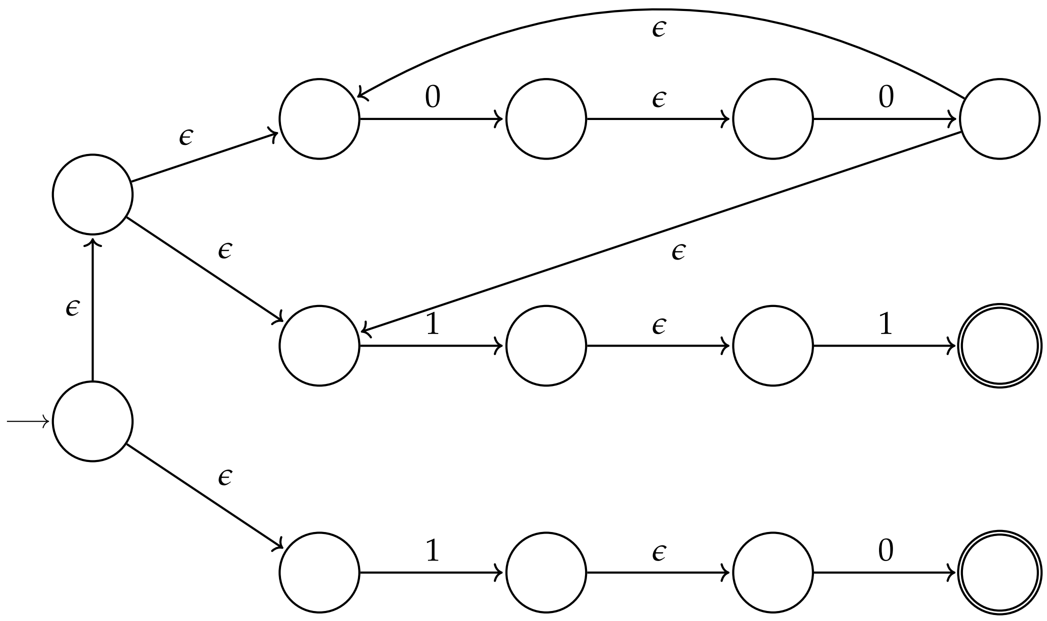
(b) NFA for $(00)^\star$



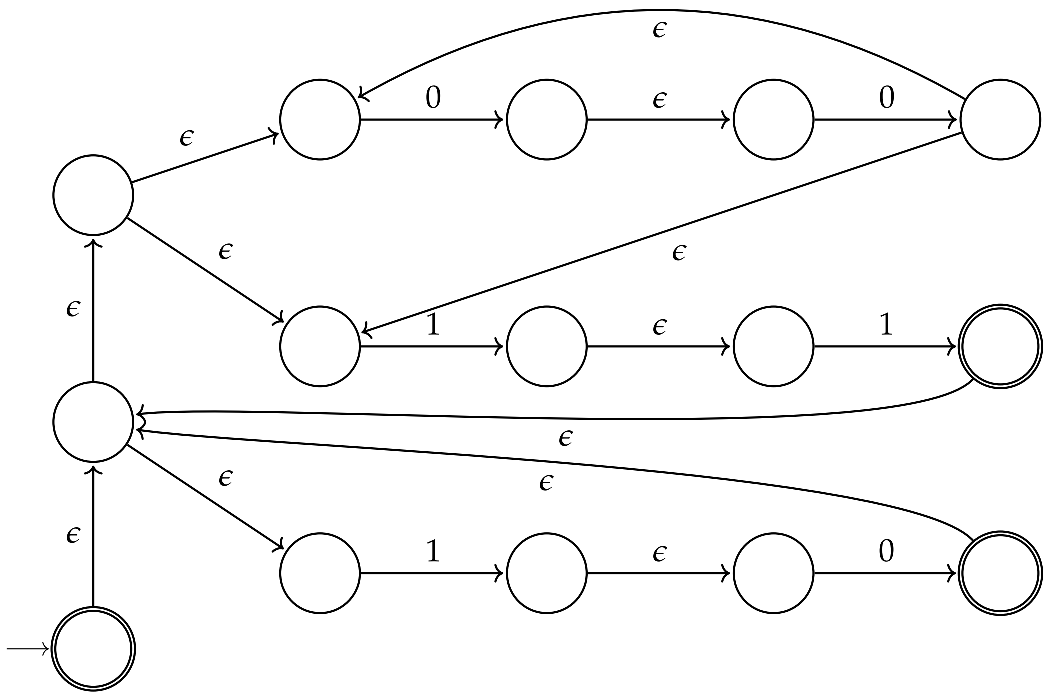
(c) NFA for $(00)^\star(11)$



(d) NFA for $(00)^{\star}(11) \mid 10$



(e) NFA for $(((00)^{\star}(11)) \mid 10)^{\star}$



4 Algorithms

Problem 8. In this problem you will show that the regular languages are closed under union. That is, if M_1, M_2 are DFAs, then so is $L(M_1) \cup L(M_2) = L(M)$ for some DFA M.

Hint. Simulate both automata at the same time using pairs of states.

1. Argue why the construction is correct.
2. Use the construction to draw a DFA for the language of strings over alphabet $\{a, b\}$ consisting of an even number of a's or an odd number of b's.

Show that the regular languages are closed under intersection. How would you change your union construction to do this?

Solution 8.

This is called a *product* construction. We are given DFA $M_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$ recognising L_i, $i = 1, 2$. We build a DFA $M = (Q, \Sigma, \delta, q_0, F)$ where

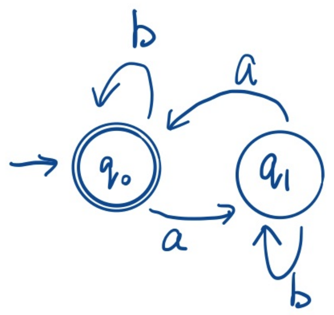
- $Q = Q_1 \times Q_2$, i.e., $\{(s_1, s_2) : s_1 \in Q_1, s_2 \in Q_2\}$
- $q_0 = (q_1, q_2)$
- $F = \{(s_1, s_2) : s_1 \in F_1 \text{ or } s_2 \in F_2\}$

- δ maps state (s_1, s_2) on input $a \in \Sigma$ to $(\delta_1(s_1, a), \delta_2(s_2, a))$.

Then $L(M) = L(M_1) \cup L(M_2)$.

Here is an informal argument (for a more formal argument, see Sipser Theorem 1.25). To see that M recognises $L(A_1) \cup L(A_2)$, we must reason that for every string w: M accepts w iff either M_1 accepts w or M_2 accepts w. Intuitively, this is because M simulates both M_1 and M_2. That is, a run of M is made up of a run from M_1 and a run from M_2. And the run of M is accepting iff at least one of the component runs is accepting.

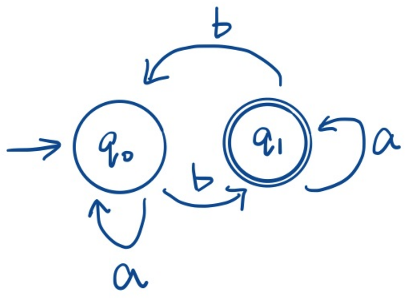
Draw a DFA for the set of all strings with an even number of a's



q_0 represents strings that contain even number of a's and arbitrary number of b's.

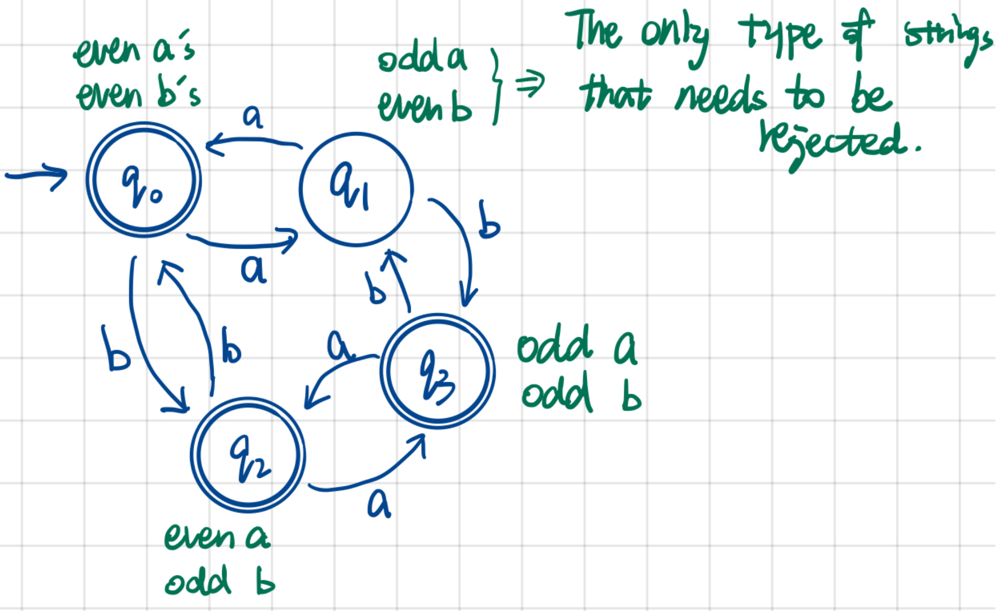
q_1 represents strings that contain odd number of a's and arbitrary number of b's.

Draw a DFA for the set of all strings with an odd number of b's



q_0 represents strings that contain even number of b's and arbitrary number of a's.

q_1 represents strings that contains odd number of b's and arbitrary number of a's.



For intersection, just change the set of final states: $F = \{(s_1, s_2) : s_1 \in F_1 \text{ and } s_2 \in F_2\}$.

Aside. Once we know regular languages are closed under union and complement, we can immediately conclude they are closed under intersection. Why?

Problem 9. The *non-emptiness problem for DFAs* is the following decision problem: given a DFA M, return "Yes" if $L(M) \neq \emptyset$ and "No" otherwise.

Devise an algorithm for solving this problem. What is the worst-case time complexity of your algorithm?

How would you adapt your algorithm to solve the non-emptiness problem for NFAs?

Solution 9. Idea: use a graph-traversal algorithm, starting at the initial state, to see if there is a final state that is reachable from the initial state. For instance, using an adjacency-list representation, depth-first search (DFS) runs in time $O(n + m)$ where n is the number of vertices and m the number of edges. The same approach works for NFAs!

Problem 10. The *equivalence problem for DFAs* is the following decision problem: given DFAs M_1, M_2, return "Yes" if $L(M_1) = L(M_2)$, and "No" otherwise.

Devise an algorithm for solving this problem. What is the worst-case time complexity of your algorithm? What about space complexity (i.e., amount of memory needed)?

Hint. Use the facts that the regular languages are closed under the Boolean operations (i.e., complement, union, intersection), and the fact that the emptiness problem for DFAs is decidable.

Solution 10. The idea behind the algorithm is to use the fact that two sets L_1 and L_2 are equal if and only if $L_1 \subseteq L_2$ and $L_2 \subseteq L_1$. But how does one test if one language is contained in another? Well, $L_1 \subseteq L_2$ if and only if $L_1 \cap (\Sigma^* \setminus L_2) = \emptyset$. Why?

So here is an algorithm for solving the equivalence problem for DFAs:

1. For $i = 1, 2$, build DFAs M'_i such that $L(M'_i) = \Sigma^* \setminus L(M_i)$.
2. Form DFAs J for $L(M_1) \cap L(M'_2)$, and K for $L(M_2) \cap L(M'_1)$.
3. Test if both $L(J) = \emptyset$ and $L(K) = \emptyset$. If so, return "Yes", else return "No".

What is the complexity? Step 1 can be done in linear time (just swap accepting and non-accepting states). Step 2 can be done using the product construction (see Problem 8 in this tutorial). Then J is a DFA whose size is the product of the sizes of M_1 and M'_2, i.e., polynomial in the size of the inputs. Step 3 is like Problem 9 in this tutorial — run the algorithm for the non-emptiness problem and flip the answer to solve the emptiness problem. This last step also takes polynomial (in fact linear) time. So the total time for the whole procedure is now polynomial in the sizes of the inputs M_1, M_2!

Problem 11. (Exam 2022) Fix $\Sigma = \{0, 1\}$. Consider the operation *delzero* that maps a string u to the string in which every 0 is deleted. So, e.g., $delzero(0) = \epsilon$, $delzero(01101) = 111$, $delzero(1) = 1$, and $delzero(\epsilon) = \epsilon$. For a language $L \subseteq \Sigma^*$, let $delzero(L) = \{delzero(u) : u \in L\}$. Explain why if L is regular, then $delzero(L)$ is regular.

Solution 11. Let $M = (Q, \Sigma, q_0, \delta, F)$ be a DFA for L. Build an NFA $M' = (Q, \Sigma, q_0, \delta', F)$ for $delzero(L)$ from M by replacing every transition on input letter 0, say $\delta(q, 0) = r$, by an epsilon transition $r \in \delta'(q, \epsilon)$.

Formally define δ' as follows. For every state q, define $\delta'(q, 1) = \{\delta(q, 1)\}$, $\delta'(q, \epsilon) = \{\delta(q, 0)\}$, and $\delta'(q, 0) = \emptyset$.

[Why is this correct? Every run $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \xrightarrow{w_3} \cdots \xrightarrow{w_n} q_n$ in M on a string w induces a run $q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} q_2 \xrightarrow{u_3} \cdots \xrightarrow{u_n} q_n$ on w in which $u_i = \epsilon$ if $w_i = 0$ and $u_i = w_i$ otherwise; thus $delzero(L) \subseteq L(M')$. For a similar reason, $L(M') \subseteq delzero(L)$.]

Problem 12. (Hard) A fundamental problem in string processing is to tell how close two strings are to each other. A simple measure of closeness is their Hamming distance, i.e., the number of position in which the strings differ, written

$H(u,v)$. For instance, $H(aaba, abbb) = 2$ since the strings disagree in positions 2 and 4 (starting the count at 1). Positions at which the the strings differ are called *errors*.

Let $\Sigma = \{a,b\}$. Show that if $A \subseteq \Sigma^*$ is a regular language, then the set of strings of Hamming distance at most 2 from some string in A, i.e., the set of strings $u \in \Sigma^*$ for which there is some $v \in A$ such that $len(u) = len(v)$ and $H(u,v) \leq 2$, is also regular.

Hint. Take a DFA M for A, say with state set Q, and build an NFA with state set $Q \times \{0,1,2\}$. The second component tells you how many errors you have seen so far. Use nondeterminism to guess the string $u \in A$ that the input string is similar to, as well as where the errors are.

How would you change the construction to handle Hamming distance 3 instead of 2?

How would you change the construction to handle that case that we don't require $len(u) = len(v)$? e.g., $H(abba, aba) = 2$.

p.s. The problem of telling how similar two strings are comes up in a variety of settings, including computational biology (how close are two DNA sequences), spelling correction (what are candidate correct spellings of a given misspelled word), machine translation, information extraction, and speech recognition. Often one has other distance measures, e.g., instead of just substituting characters, one might also allow insertions and deletions, and have different "costs" associated with each operation. The standard way of computing similarity between two given strings is by Dynamic Programming, an algorithmic technique you will delve into in `COMP3027:Algorithm Design`.

Solution 12. Say $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA recognising language A. Build the NFA $N = (Q \times \{0,1,2\}, \Sigma, \delta', (q_0, 0), F')$ where

- $F' = F \times \{0,1,2\}$,
- δ' is defined, for $i = 0,1$ by $\delta'((q,i),a) = \{(\delta(q,a),i), (\delta(q,b), i+1)\}$ and $\delta'((q,i),b) = \{(\delta(q,b),i), (\delta(q,a), i+1)\}$.

In other words, at each step N reads a symbol as input and makes a guess that the matching word in A has the same letter at that position (in which case it does not increase the number of errors), or has a different letter at that position (in which case it does increase the number of errors). If after reading the input word u, the NFA N has reached a state (q,i) then this means that there is a word v such that $H(u,v) = i$. If also $q \in F$ then $v \in A$ and the NFA has witnessed that there is a word of Hamming distance at most 2 from u. If there is no such word v then every run of the NFA on u will either 'crash' (i.e., the number of errors will exceed 2), or will end in a state of the form (q,i) with $q \notin F$, i.e., the guessed word v is not in A.