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

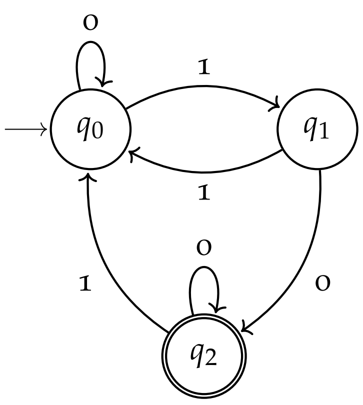
1 Warmup

Problem 1. Discussion. Look at the definitions of DFA and NFA.

1. What does "finite" refer to?
2. What does "automaton" refer to?
3. What does "deterministic" refer to?, i.e., in what sense are DFAs deterministic?
4. What does "non-deterministic" refer to? i.e., in what sense are NFAs non-deterministic?

2 DFAs

Problem 2. Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language recognised by this automaton?

Solution 2.

1. $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   $q_0 = q_0$
   $\delta : Q \times \Sigma \to Q$ is given by:

	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_2	q_0

   $F = \{q_2\}$
2. Strings of 1s and 0s which end in 0 and which contain an odd number of 1s

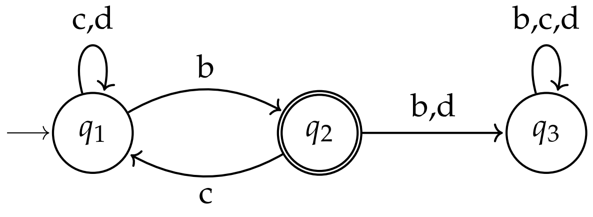
Problem 3. Draw the following automaton and give the language that it recognises:

- $Q = \{q_1, q_2, q_3\}$ 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

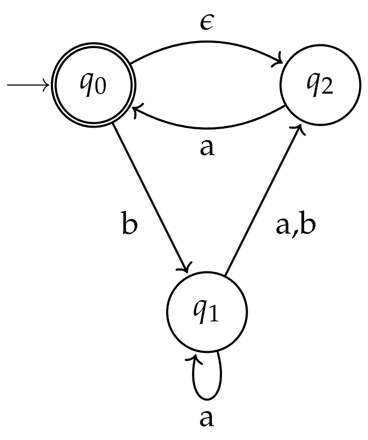
Solution 3.



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

3 NFAs

Problem 4. Consider the following NFA:



Does the NFA accept:

1. ϵ
2. a
3. baba
4. baa
5. aaab
6. aabb

Solution 4.

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. baa ACCEPTED (by path q_0, q_1, q_2, q_0)
5. aaab REJECTED (ended in the set of states $\{q_1\}$, which does not contain an accept state)
6. aabb REJECTED (ended in the set of states $\{q_2\}$, which does not contain an accept state)

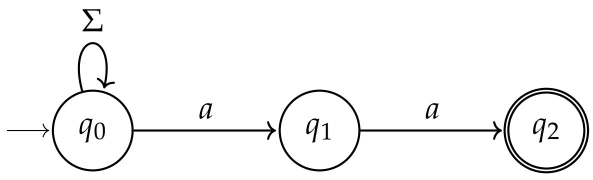
Problem 5. Draw an NFA for each of the following languages. The alphabet is $\Sigma = \{a, b, c\}$. You may use epsilon transitions.

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)

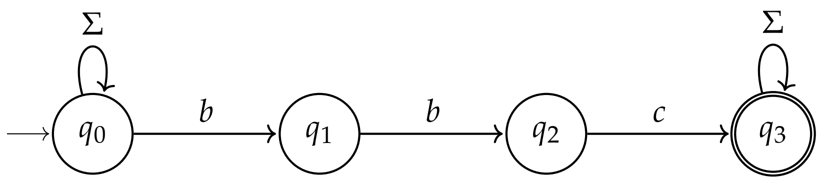
4. Strings that start with a and end in c (use only 3 states)
5. Strings of length 4 starting with b
6. Strings of length at most 4.

Solution 5.

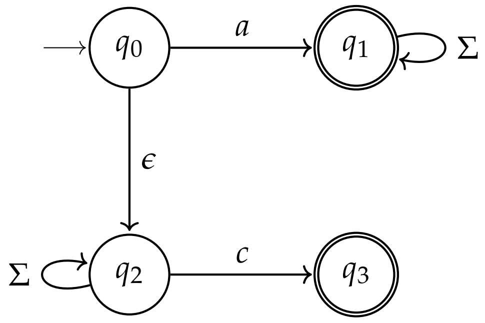
1. Strings that end with *aa* (use only 3 states)



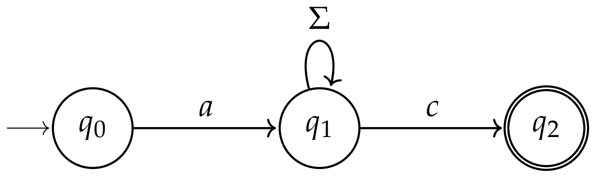
2. Strings that contain the substring ‘bbc’ (use only 4 states)



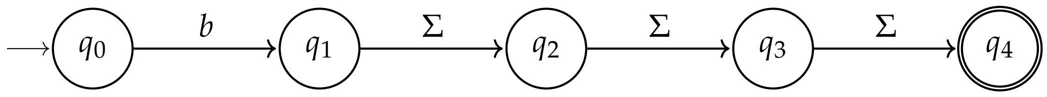
3. Strings that start with a or end in c (use only 4 states)



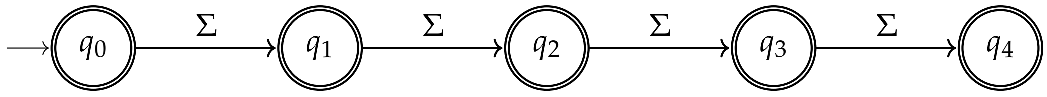
4. Strings that start with a and end in c (use only 3 states)



5. Strings of length 4 starting with b



6. Strings of length at most 4.

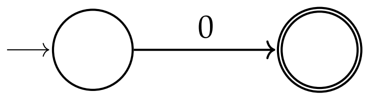


4 From RE to NFA

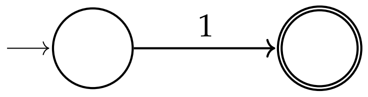
Problem 6. Use the construction demonstrated in lectures to construct an NFA for the following RE $0^\star110^\star$

Solution 6.

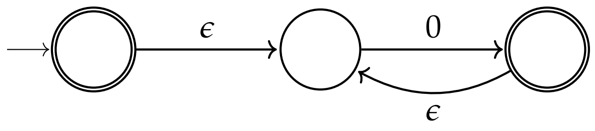
NFA for 0:



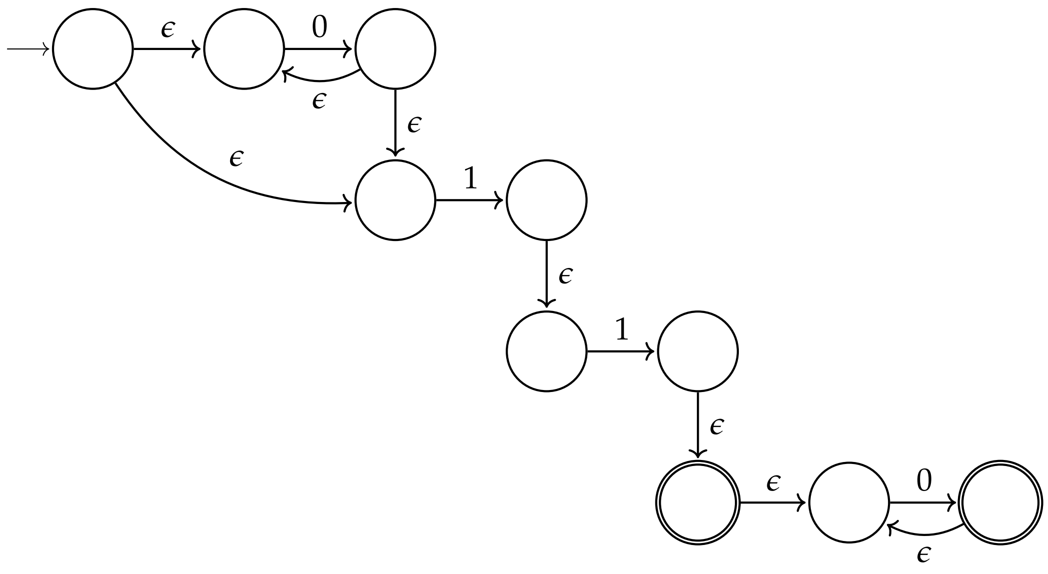
NFA for 1:



NFA for $0^\star$:



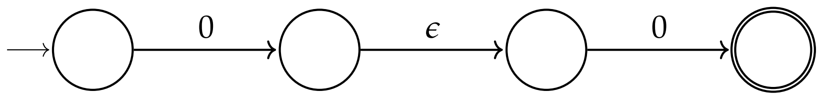
Concatenate the various machines to get the final answer:



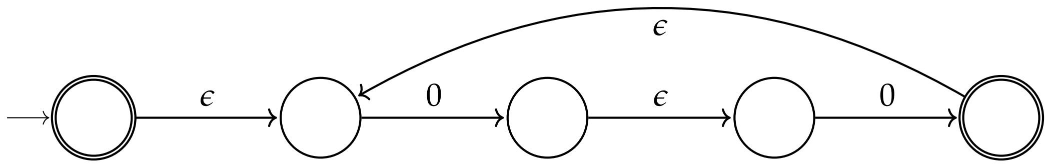
Problem 7. Use the construction demonstrated in lectures to construct an NFA for the following RE $((00)^\star(11) \mid 10)^\star$

Solution 7.

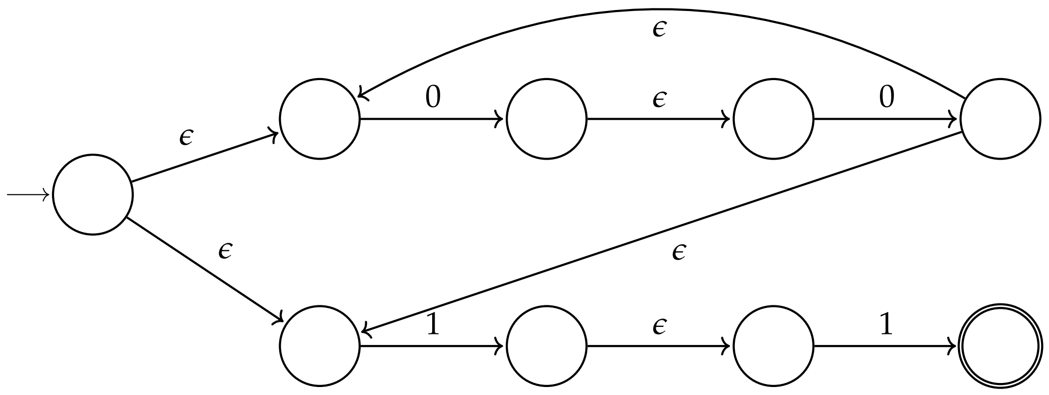
NFA for 00 (similarly for 11, 10):



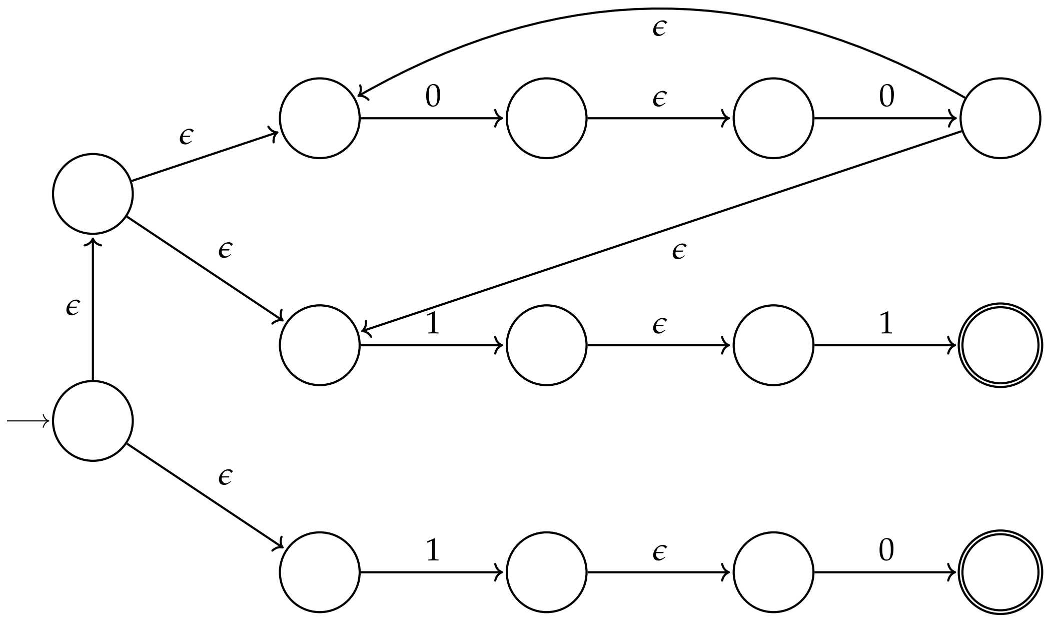
NFA for $(00)^\star$



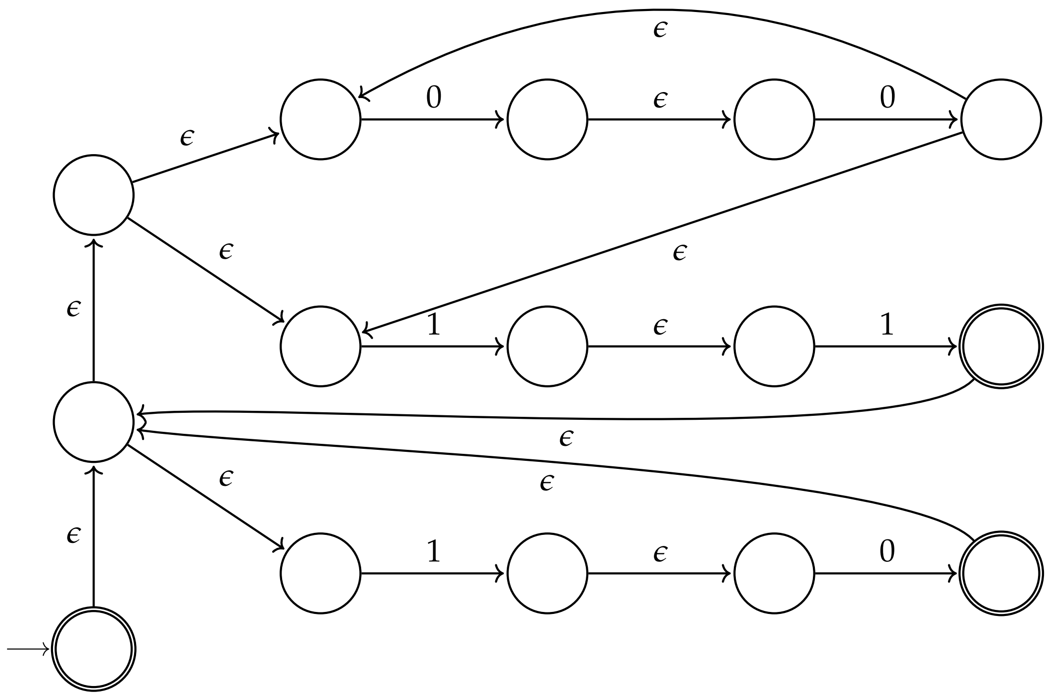
NFA for $(00)^\star(11)$



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



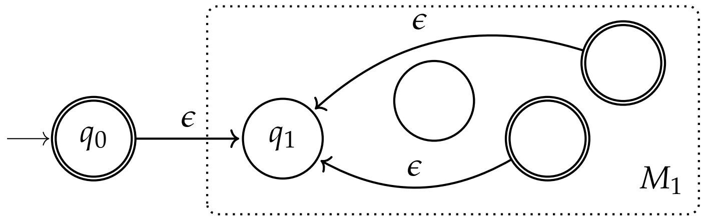
NFA for $((00)^\star(11) \mid 10)^\star$



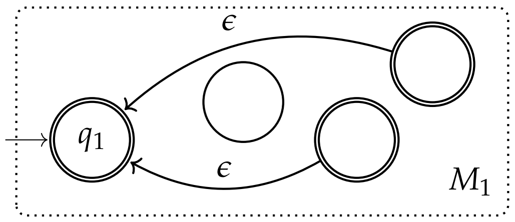
Solution 7.

1. Check if there are any accept states in the *connected component* of the graph containing the start state.
2. Build an NFA N recognising $L(N_1) \setminus L(N_2)$ and check if $L(N) = \emptyset$. Use that $L(N) = \emptyset$ iff $L(N_1) \subseteq L(N_2)$.

Problem 8. Recall the construction of an NFA M recognising $L(M_1)^*$ given an NFA M_1:



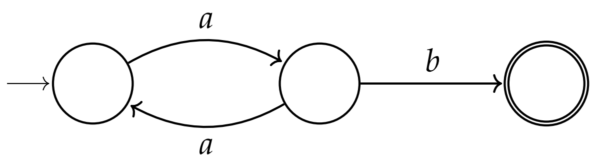
Let us now consider the following construction instead, where we do not add a new start state, and instead make the original start state accepting:



Show that the second construction does not work, i.e., find a finite automaton

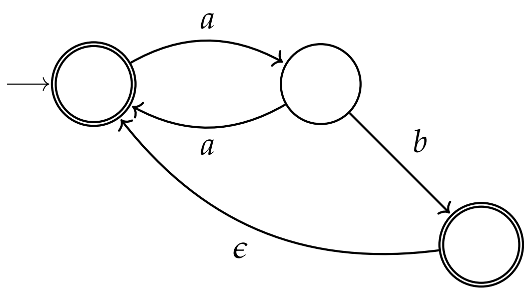
M_1 for which the automaton M built using the second construction method does *not* recognise $L(M_1)^*$

Solution 8. Let M_1 be:

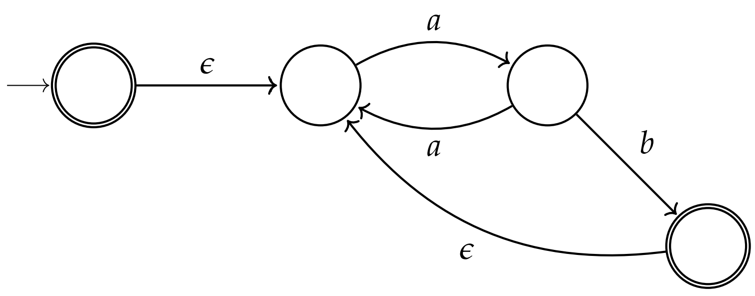


Then $L(M_1) = \{aa^{2k}b | k \geq 0\}$ (an odd number of a's followed by a b)

If we make an automata using the second approach, it will accept aa, but this is not in $L(M_1)^*$



The approach described in lectures would give us the following automata, which does work:

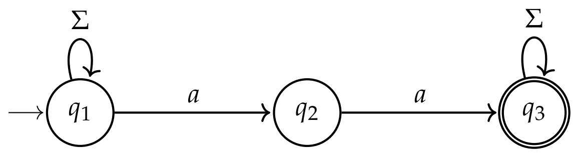


Problem 9. Fix alphabet $\Sigma = \{a, b\}$.

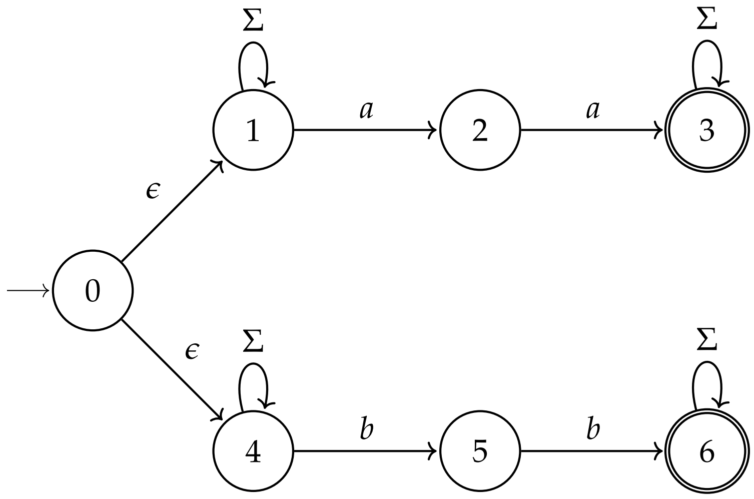
1. Devise an NFA which recognises the language of strings containing two consecutive a's
2. Apply the union construction to build an NFA which recognises the language of strings with two consecutive a's or two consecutive b's
3. Write a DFA for this language (in the next tutorial you will use a systematic procedure for converting NFAs into DFAs!)
4. Write an RE for this language (in the next tutorial you will use a systematic procedure for converting NFAs into REs!)

Solution 9.

1. Devise an NFA which accepts strings containing two consecutive a's



2. Apply the union construction to deduce an NFA which accepts all strings over $\{a, b\}$ with two consecutive a's or two consecutive b's



5 Closure properties

Problem 10. In this problem you will show that the regular languages are closed under union. That is, if L_1, L_2 are regular, then so is $L_1 \cup L_2$.

The idea is to simulate both automata at the same time using pairs of states. This is called a *product* construction.

Here is the 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 $L(M) = L(M_1) \cup L(M_2)$.

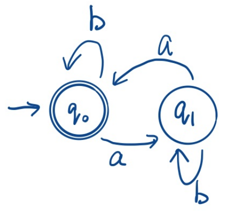
1. Argue why the construction is correct, i.e., why does M accept a string if and only if either M_1 accepts it or M_2 accepts it (or both).

2. 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.

Solution 10.

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.

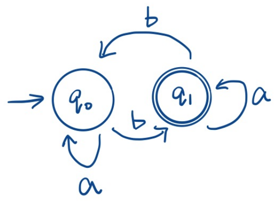
Draw a DFA for the set of all strings with an even number of a's



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

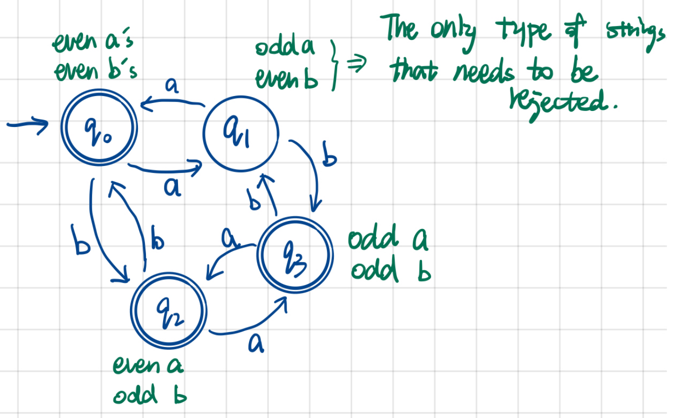
q_1 represents strings that contain odd number of a's and arbitrary number of b's.

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.



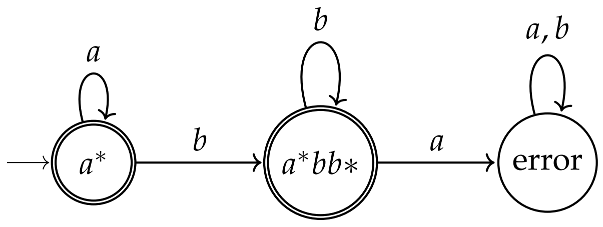
Problem 11.

1. The product technique for building a DFA recognising the union of the languages of two DFAs can be slightly modified to get a DFA recognising the intersection of the languages of two DFAs. How?
2. Use the product technique to build a DFA for this language over $\{a, b\}$:

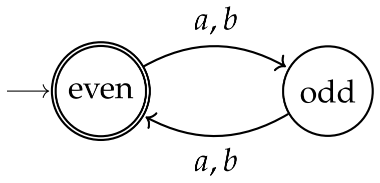
$$L = \{x \mid \text{there is no } b \text{ before an } a \text{ in } x \text{ and } |x| \text{ is even}\}$$

Solution 11.

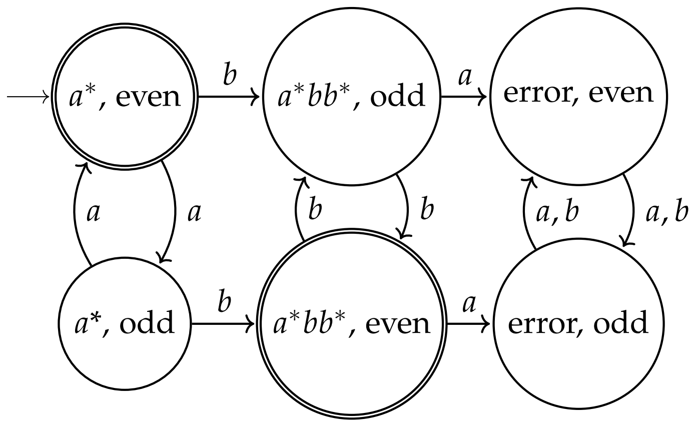
1. The only change is the set F of final states: these should be the pairs (q_1, q_2) such that $q_1 \in F_1$ *and* $q_2 \in F_2$.
2. $L = \{x \mid \text{there is no } b \text{ before an } a \text{ in } x\}$



$L = \{x \mid |x| \text{ is even}\}$



Cross product:



6 Algorithms

Problem 12. 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 12. 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!

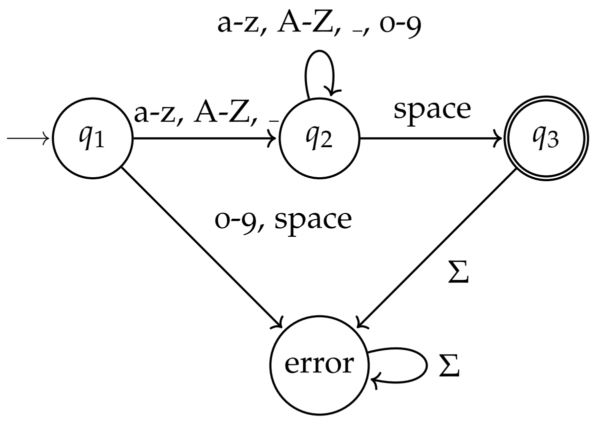
7 Additional problems

Problem 13.

1. Construct a DFA that recognises the language of valid identifiers. A valid identifier begins with a letter or with an underscore '_' and contains any combination of letters, underscores, or digits, followed by a blank space.
2. How would you change your automaton if the length of the identifier (excluding the blank) should not be longer than 6 symbols?

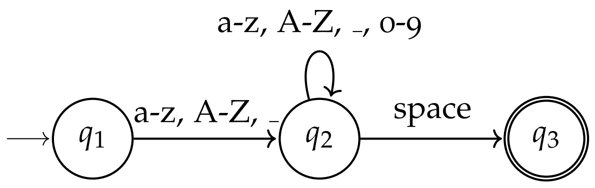
Solution 13.

1. If we don't need to count the length, it's fairly simple:



Often we're lazy, and omit to draw the 'error states', states from which no path leads to an accept state. If we do this we must write 'Error state not shown' on our diagram:

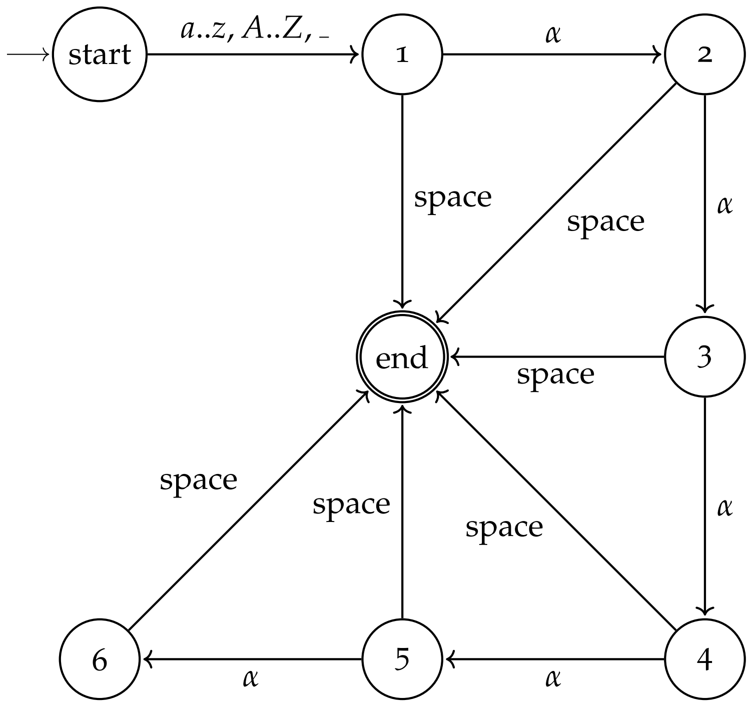
(Error state not shown)



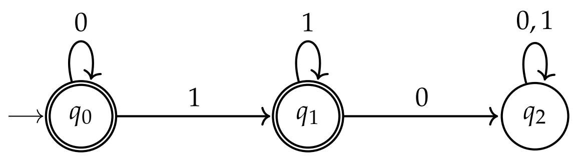
2. To be able to count the length of the identifier (up to to six characters) we will need to add more states, which can be used to count how many symbols we have read:

   Let $\Sigma = \{a..z, A..Z, 0..9, \_, space\}$ and $\alpha = \Sigma \setminus \{space\}$

   (Error state not shown)

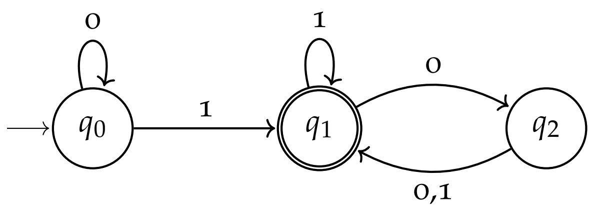


Problem 14. Consider the automaton drawn below:



1. Is the automaton deterministic or non-deterministic?
2. What is the language recognised by this automaton.

Problem 15. Consider the automaton drawn below:



1. Give $(Q, \Sigma, \delta, q_0, F)$ for this automaton.
2. What is language recognised by this automaton?

Solution 15.

1. $Q = \{q_0, q_1, q_2\}$
   $\Sigma = \{0, 1\}$
   $q_0 = q_0$
   $\delta : Q \times \Sigma \to Q$ is given by:

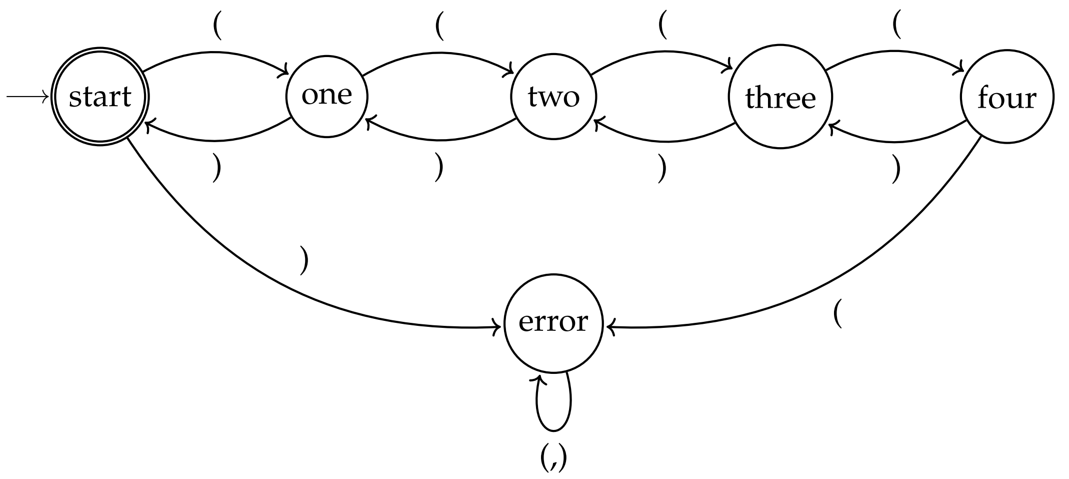
	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_1	q_1

   $F = \{q_1\}$

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

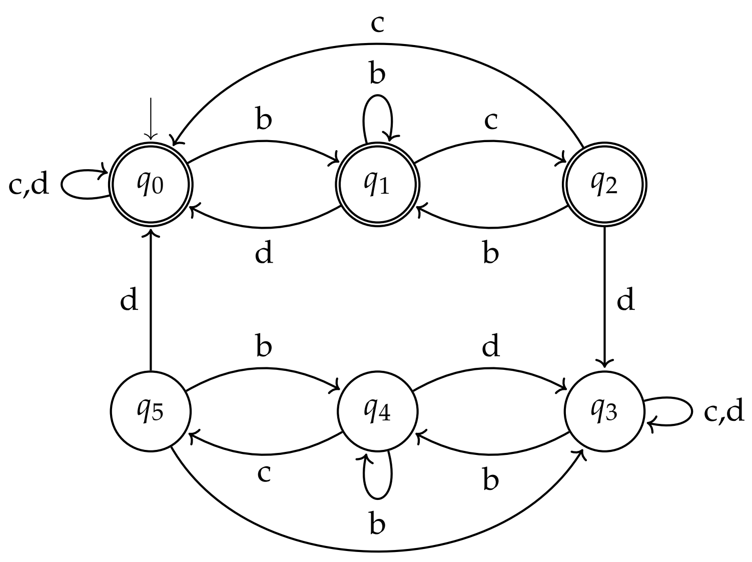
Problem 16. Construct a DFA which recgonises the set of strings that are valid bracket expressions with at most 4 levels of nesting. For example, it should accept (((()))) or ()(()()) but not (() or)(.

Solution 16.



Problem 17. Draw an automaton which recognises the language of strings over $\{b, c, d\}$ containing an even number of occurrences of the substring *bcd*. Recall that 0 is an even number.

Hint: think about what the automaton needs to remember as it moves through the input. Add a state for each of these conditions, then add appropriate transitions between them.



Solution 17.

In q_0, q_1, q_2 we've read an even number of occurrneces of *bcd*. In the other states we've seen an odd number. In q_1 or q_4 the last symbol read was a *b*. In q_2 or q_5 the last two symbols read were *bc*.

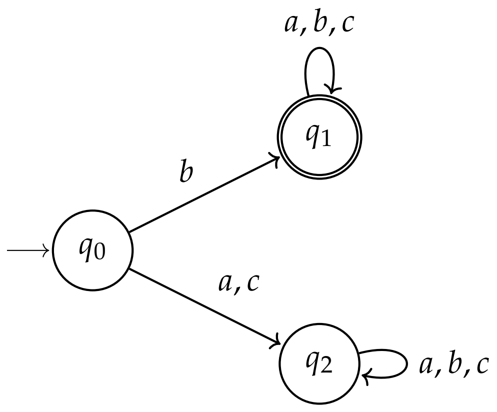
Problem 18. Prove that the following languages are regular, by drawing finite automata which recognise them. The alphabet for all the languages is $\{a, b, c\}$.

1. Start with *b*

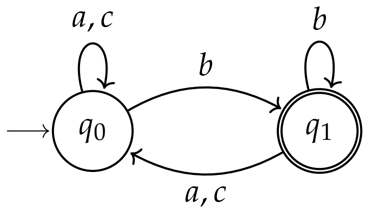
2. End with b
3. Contain exactly one b
4. Contain at least one b
5. Contain at least one b and are of length exactly 2
6. Are of length at most 2
7. Are of length at most n for some integer n

Solution 18.

1. Start with b

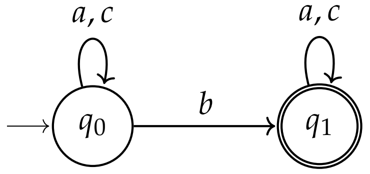


2. End with b

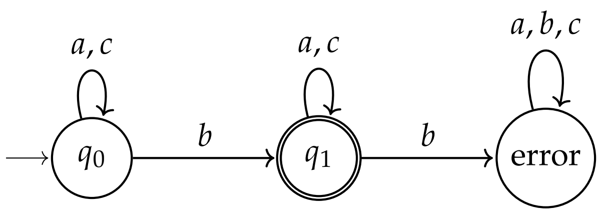


3. Contain exactly one b

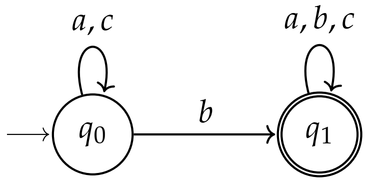
(Error states not shown)



Or, with an error state:



4. Contain at least one b



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

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "{a,c}"] \arrow[dr, "b"] & q_1 \arrow[r, "{a,c}"] \arrow[dr, "b"] & error \arrow[loop right, "{a,b,c}"] \\
 & & q_2 \arrow[r, "{a,b,c}"] & q_3 \arrow[u, "{a,b,c}"']
\end{tikzcd}$$

6. Are of length at most 2
   (error states not shown)

$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "{a,b,c}"] & q_1 \arrow[r, "{a,b,c}"] & q_2
\end{tikzcd}$$

7. Are of length at most n for some integer n
   As above, but with the set of $n \mid 2$ states $Q = \{q_0, ..., q_n, error\}$

Problem 19. Draw an automaton for comments in C or C++. These are strings which

1. begin with /*
2. end in */ (where the * cannot be the same as the first one)
3. have no other occurrences of */

Solution 19. (Error states not shown)

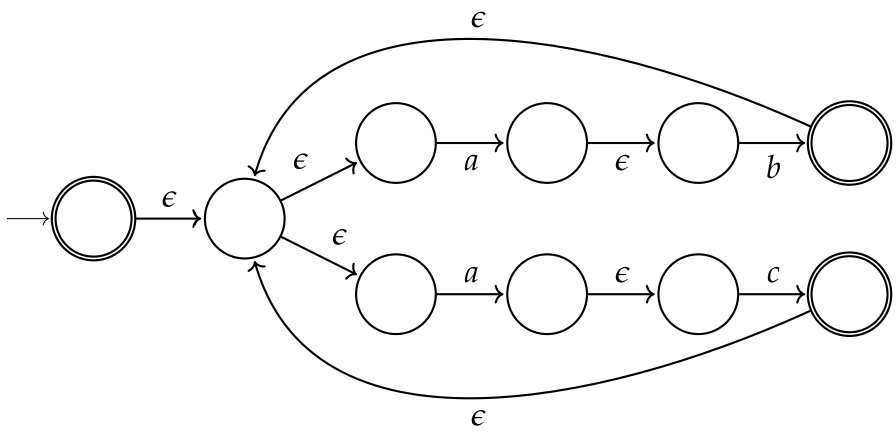
$$\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, "/"] & q_1 \arrow[r, "*"] & q_2 \arrow[loop above, "\Sigma \setminus \{*\}"] \arrow[r, bend left, "*"] & q_3 \arrow[loop above, "*"] \arrow[l, bend left, "\Sigma \setminus \{*, /\}"] \arrow[r, "/"] & q_4
\end{tikzcd}$$

Problem 20.

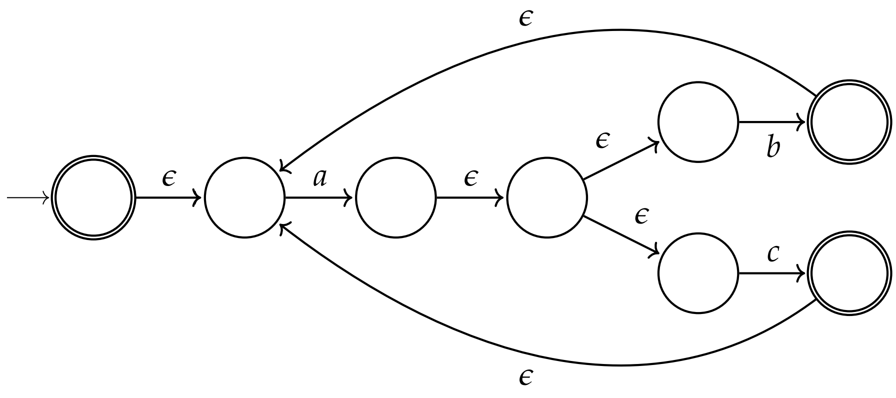
1. Give examples of strings described by the RE $(ab \mid ac)^{\star}$
2. Rewriting the RE in an equivalent form, construct two NFA (from the two regular expressions) which recognise the same language

Solution 20.

1. $\{\epsilon, ab, ac, abab, abac, acab, acac, ababab, ababac, \ldots\}$
2. $\big(a(b \mid c)\big)^\star$
   NFA for $(ab \mid ac)^\star$



NFA for $\big(a(b \mid c)\big)^\star$



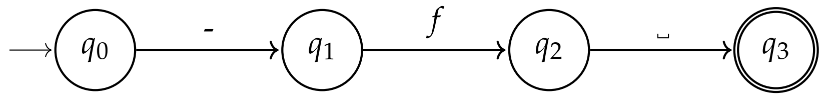
Problem 21.

1. Write a regular expression that matches strings composed of: "-F␣" or "-f␣", followed by filenames consisting only any number of "a" (possibly none) followed by the extension ".tmp"

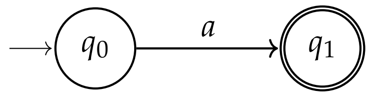
   Example: "-f aaaaa.tmp", "-f a.tmp", "-f .tmp" are all accepted, but "-fa.tmp", "-f b.temp", "-f aaa.pdf" are not.
2. Use the construction methods shown in lectures to convert the regular expression into an NFA.

Solution 21.

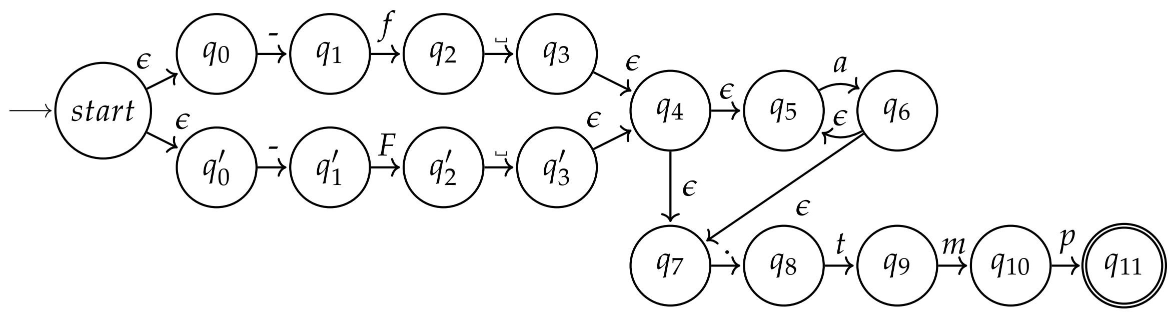
1. One such regular expression is $(((-F␣)|(-f␣))a^*(.tmp))$
2. (a) Here is an automaton recognising only the string "-f␣"



   (b) Here is an automaton recognising only the string "a"



   (c) Here is the NFA:



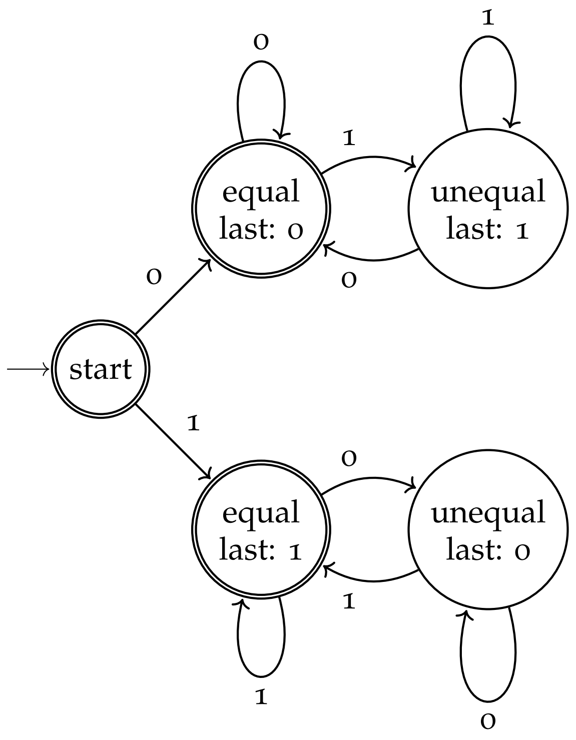
Problem 22. Try construct a DFA M such that

$L(M) = \{w|w$ contains an equal number of occurrences of the substrings 01 and 10$\}$

For example, $101 \in L(M)$, but $1010 \notin L(M)$. If you cannot do so, what is the difficulty?

Solution 22.

Observe that in any string of 1s and 0s, it's impossible for the number of occurrences of 01 and of 10 to differ by more than one at any point of scanning through the string. So, we just need our states to represent all the combinations of {equal count, unequal count} and {last seen 0, last seen 1, seen nothing}



Problem 23. Discussion: Do you think every context-free language is regular?

Problem 24. 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 24. 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 $F' = F \times \{0,1,2\}$ and 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.