MTAT.05.125 Introduction to Theoretical Computer Science

Autumn 2015

Vitaly Skachek
Estonian version by Reimo Palm
English version by Yauhen Yakimenka

Lecture 7. Regular expressions. Nonregular languages.

Regular expressions

We defined operations on languages: union $\cup$, concatenation $\circ$ and operation *. As with algebraic expressions, we can then build expressions of languages, e.g. $(L_1 \cup L_2) \circ L_1^*$.

For instance, if $L_1 = \{0\}$ and $L_2 = \{1\}$, we can write language $(L_1 \cup L_2) \circ L_1^*$ shortly just as $(0 \cup 1)0^*$. We will call this short-hand regular expressions.

Let Σ be some finite alphabet. Then formal definition of regular expression is as follows.

Definition. *R is a regular expression if R is either of the following:*

1. *a for some $a \in \Sigma$ (correspond to language $\{a\}$);*
2. *empty string ε (correspond to language $\{\varepsilon\}$);*
3. *empty language $\varnothing$ (correspond to language $\{\}$);*
4. *$(R_1 \cup R_2)$, where R_1 and R_2 are some regular expressions;*
5. *$(R_1 \circ R_2)$, where R_1 and R_2 are some regular expressions;*
6. *R_1^*, where R_1 is some regular expression.*

This is *inductive definition*, i.e. in clauses 1–3 we define the smallest possible regular expression and then in clauses 4–6 we define the rules to deduce all the other regular expressions.

We will also occasionally use R^+ as a shorthand for $R \circ R^*$, i.e. regular expression R repeated one or more times.

Each regular expression is another way to *describe* a language (and a language is a set of words). So for example the regular expression a denotes the set of words $\{a\}$ (i.e. one word a), the regular expression ε denotes the language $\{\varepsilon\}$ (one word ε) and the regular expression $\varnothing$ denotes the language $\varnothing$ (i.e. empty set of words, i.e. no words). If R is some regular expression we will sometimes denote the language it describes as $L(R)$.

Example 1. Let us fix alphabet $\Sigma = \{0, 1\}$. Consider the following regular expressions.

- The regular expression $(0 \cup 1)^*$ describes all binary words:

  $$L\big((0 \cup 1)^*\big) = (\{0\} \cup \{1\})^* = \{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}.$$

  The same language could be written as Σ^*.

- The regular expression $(01^*) \cup (10^*)$ describes all binary words whose first symbol is different from all the remaining symbols (if any). So if the first symbol is 0, all the remaining symbols are 1's and vice a versa.

- The regular expression $(01^+)^*$ describes a language where every 0 is followed by at least one 1.

- The regular expression $(0(0 \cup 1)^*0) \cup (1(0 \cup 1)^*1) \cup 0 \cup 1$ describes the language of words that start and end with the same letter.

- The regular expression $(0 \cup \varepsilon)(1 \cup \varepsilon)$ describes the language $\{\varepsilon, 0, 1, 01\}$.

Example 2. Let R be some arbitrary regular expression.

- $R \cup \varnothing = R$ because union of any language with empty language does not change the language.

- $R \circ \varnothing = \varnothing$ because concatenation of a language with the empty language is the empty language.

- $R \circ \varepsilon = R$ because concatenation with ε does not change a language.

- $\varnothing^* = \varepsilon$.

- $R^* = R^+ \cup \varepsilon$.

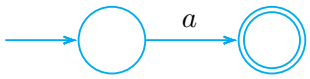
Regular expressions and regular languages

The names "regular expression" and "regular language" are similar. There is a reason for that – the set of all regular languages is exactly the set of languages described by all possible regular expressions. We will further prove it.

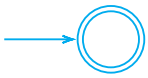
Lemma. *If a language is described by a regular expression, the it is regular.*

Proof. Let the language be described by some regular expression R (i.e. this language is $L(R)$). To prove that this language is regular, we will construct an NFA N that recognises this language.

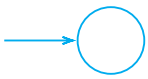
1. If $R = a$ then $L(R) = \{a\}$. Automaton N is



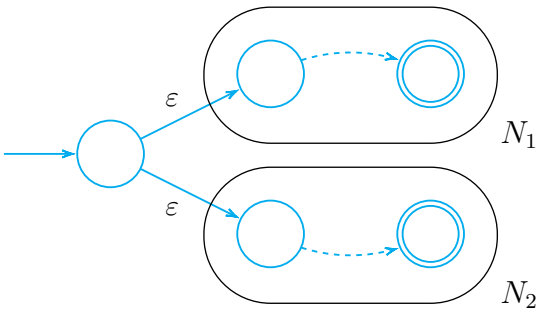
2. If $R = \varepsilon$ then $L(R) = \{\varepsilon\}$. Automaton N is



3. If $R = \varnothing$ then $L(R) = \varnothing$. Automaton N is



4. If $R = R_1 \cup R_2$ then let N_1 and N_2 be two NFAs that recognise languages $L(R_1)$ and $L(R_2)$, respectively. Automaton N is



5. If $R = R_1 \circ R_2$ then let N_1 and N_2 be two NFAs that recognise languages $L(R_1)$ and $L(R_2)$, respectively. Automaton N on is the same as in problem 2 of week 6 practise session.

6. If $R = R_1^*$ then let N_1 be the automaton recognising language $L(R_1)$. Automaton N is the same as in problem 3 of week 6 practise session.

This is recursive proof and it follows the recursive definition of regular expressions. We show in 1–3 how to build automaton for the elementary regular expressions and then in 4–6 show how to construct automaton for regular expression from operation on smaller regular expressions (and corresponding automata). □

Lemma. *If the language is regular, then there is a regular exception describing this language.*

The proof of this lemma is given in the appendix.

Nonregular languages

Consider the following language:

$$B = \{0^n 1^n \mid n \geq 0\}.$$

Is there a DFA that recognises B? No!

Intuition: an automaton should "remember" how many zeros it has seen. It needs an infinite number of states for doing so.

Another example

$$C = \{w \mid w \text{ has equal number of 0's and 1's}\}.$$

Pumping lemma

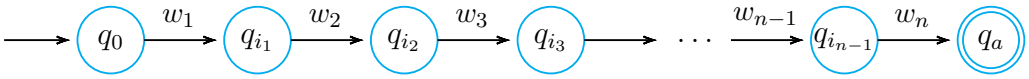
All regular languages have a certain property: each string of sufficiently large length contains a substring, which can be repeated any number of times, with the resulting strings remaining in the language.

Pumping lemma. *If L is a regular language then there exists a number p such that if $w \in L$, $|w| \geq p$, then w could be represented as $w = xyz$ and the following three conditions are satisfied:*

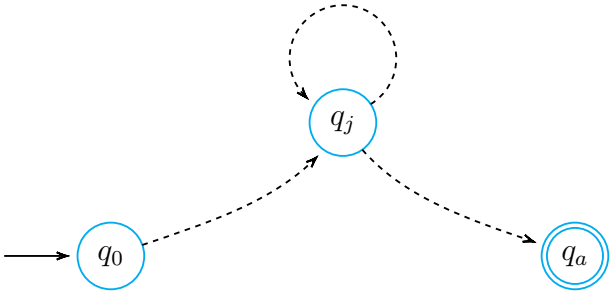
1. *for all $i \geq 0$ it holds that $xy^iz = x\underbrace{yy\cdots y}_{i \text{ times}}z \in L$;*
2. *$|y| > 0$;*
3. *$|xy| \leq p$.*

Here $|w|$ denotes the length of the string w. Either x or z (or both) can be ε, but not y.

Idea of the proof. Let p be the number of states of the automaton recognising L. Consider the run of the automaton on the input w of length $|w| = n$:



If $n \geq p$, there should be a state q_j which is repeated twice. We can repeat the substring between the appearances of q_j arbitrary number of times.



Proof. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA recognising L, and p be a number of states of M. Let $w = w_1 w_2 \dots w_n$ be a string in the language L with $n \geq p$. Let $q_0 = q_{i_0} \to q_{i_1} \to q_{i_2} \to \dots \to q_{i_{n-1}} \to q_{i_n} = q_a$ be a sequence of states that M enters when processing w: for each $j = 0, 1, \dots, n-1$ we have $\delta(q_{i_j}, w_j) = q_{i_{j+1}}$.

There are $n+1 \geq p+1$ states in this sequence. Since automaton has only p states, amongst the first $p+1$ states $q_0, q_{i_1}, \dots, q_{i_p}$ there exists a state q_j that appears at least twice:

$$q_0 = q_{i_0} \xrightarrow{w_1} \dots \xrightarrow{w_l} q_{i_l} = q_j \xrightarrow{w_{l+1}} \dots \xrightarrow{w_r} q_{i_r} = q_j \xrightarrow{w_{r+1}} \dots \xrightarrow{w_n} q_{i_n} = q_a.$$

Here q_{i_l} is the first appearance of q_j and q_{i_r} is the second appearance of it ($l < r \leq p$). Denote:

$$\begin{aligned} x &= w_1 w_2 \dots w_l, \\ y &= w_{l+1} w_{l+2} \dots w_r, \\ z &= w_{r+1} w_{r+2} \dots w_n. \end{aligned}$$

Substring x takes M from q_0 to q_j, y takes M from q_j to q_j (forms a loop) and z takes M from q_j to q_a. Therefore, M should accept $xy^i z$ for any $i \geq 0$. We proved the first condition of the lemma.

Since $l < r$ then $|y| = r - l > 0$. This proves the second condition of the lemma.

And finally to prove the third condition of the lemma, we note that $|xy| = r \leq p$. □

Example 3. Show that the language

$$B = \{0^n 1^n \mid n \geq 0\}$$

is not regular.

Assume, to the contrary, that B is regular. Let p be the length given by the pumping lemma.

Take $w = 0^p 1^p \in B$. Since $|w| \geq p$, we can write $w = xyz$, such that for all $i \geq 0$ it holds $xy^i z \in B$. From the condition 3 of the pumping lemma $|xy| \leq p$ and thus $|y| \leq |xy| \leq p$. So y is situated entirely in the first part of w and, hence, it contains only zeros. I.e. $xyyz$ has more zeros and therefore it does not belong to the language B. This contradicts the pumping lemma. Thus, the assumption that B is regular was wrong and B is not a regular language.

Practise session

1. What languages are described by the following regular expressions?

(a) $(0 \cup \varepsilon)^*(1 \cup \varepsilon)^*$

(b) $0\Sigma^*1$

(c) $0\varnothing 10^*$

(d) $0\varepsilon 10^*$

Solution.

(a) A string of zeros (can be empty) followed by a string of ones (can be empty).

(b) All strings that start with 0 and end with 1.

Note. We abused notation a bit: Σ usually denotes the alphabet[1] $\{0, 1\}$ but here by Σ we denote regular expression describing language equal to the alphabet, i.e. $\Sigma = 0 \cup 1$. Since this does not bring ambiguity, it's okay. However you should be careful with such mixtures.

[1] I.e. the set, not a regular expression

(c) Empty set: empty set concatenated with anything is empty set.

(d) Strings that start with 01.

2. True or false?

(a) $R \cup \varepsilon = R$

(b) $R \circ \varepsilon = R$

(c) $R \cup \varnothing = R$

(d) $R \circ \varnothing = R$

Solution.

(a) False. For example if $R = 0$ then $L(R \cup \varepsilon) = \{0, \varepsilon\}$, but $L(R) = \{0\}$.

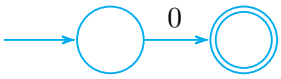
(b) True. Adding ε at the end does not change the strings.

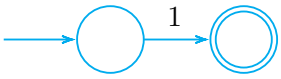
(c) True. Union of a language and the empty set does not change the language.

(d) False. For instance, if $R = 0$ then $L(R \circ \varnothing) = \varnothing$ but $L(R) = \{0\}$.

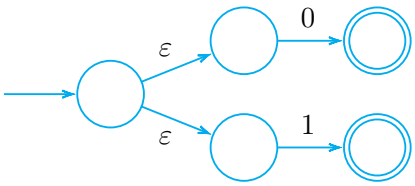
3. Find an NFA that recognises the language described by the regular expression $(0 \cup 1)^*010$.

Solution. We construct the automaton in the same manner as in the first lemma of this lecture.

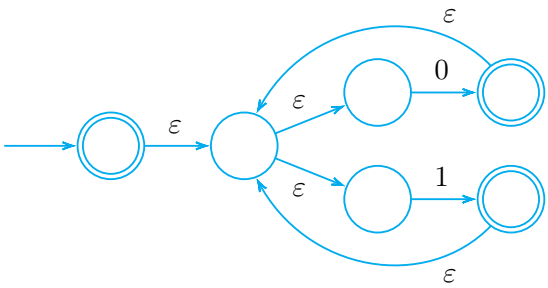
- Regular expression 0 corresponds to the automaton 

- Regular expression 1 corresponds to the automaton 

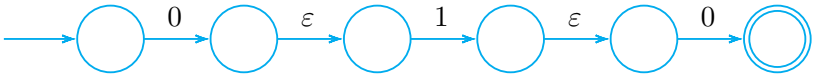
- Regular expression $0 \cup 1$ corresponds to the automaton



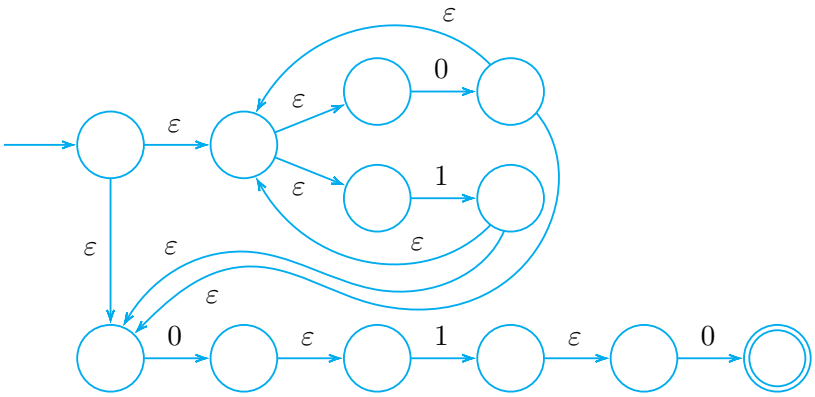
- Regular expression $(0 \cup 1)^*$ corresponds to the automaton



- Regular expression 010 corresponds to the automaton



- Regular expression $(0 \cup 1)^*010$ corresponds to the automaton



4. Show that the language

$$L = \{w \mid w \text{ has equal number of 0's and 1's}\}$$

is not regular.

Solution. Assume, to the contrary, that L is regular and let p be the "pumping length", given by the pumping lemma. Take $w = 0^p1^p \in L$. Clearly, $|w| \geq p$. Then we can apply pumping lemma and represent w as $w = xyz$ (with the three conditions holding).

From the condition 3, $|xy| \leq p$, therefore y should contain only zeros (because first p characters of w are zeros). Hence, $xyyz$ contains more zeros than xyz. And, since in xyz number of zeros and ones was the same, $xyyz$ contains more zeros that ones. But from the condition 1 of the lemma, $xyyz$ should also belong to the language L, i.e. should have equal number of zeros and ones. Contradiction!? Therefore the assumption that L is regular was wrong and L is nor regular.

5. Show that the language $L = \{ss \mid s \in \{0,1\}^*\}$ is not regular.

Solution. Assume, to the contrary, that L is regular and p is its "pumping length". Take $w = 0^p10^p1 \in L$. Obviously, $|w| \geq p$.

Then there should exist x, y, z, such that $w = xyz$ and conditions of the pumping lemma hold. Since $|xy| \leq p$, y consists of zeros only. Therefore $xyyz$ does not have the form ss as its first block of zeros is longer than the second block of zeros. So $xyyz \notin L$. Contradiction to the lemma's condition 1! That's why the assumption that L is regular was wrong.

Appendix (for self-reading)

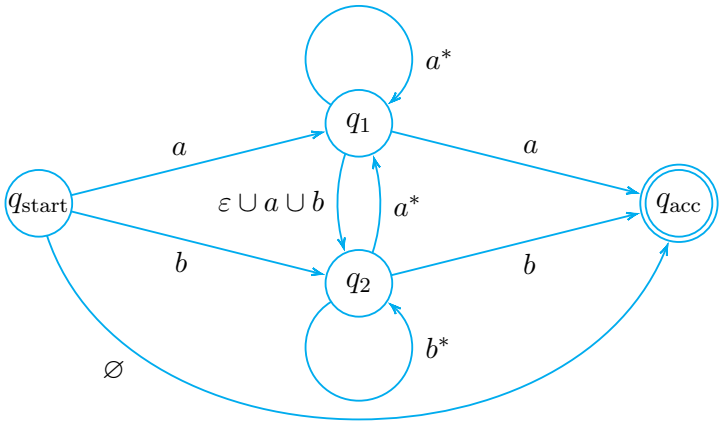
Lemma. *If the language is regular, then there is a regular exception describing this language.*

We will use a new type of automaton called *generalised* nondeterministic finite automaton (GNFA). It is an NFA where labels on the edges can have general regular expressions. GNFA reads blocks of symbols from the input, not just one symbol at a time. The GNFA moves from state to state if the input block represents a string, described by a regular expression on the corresponding edge.

GNFA has:

- Start state has no incoming edges and has outgoing edges to all other states.
- Accept state has no outgoing edges and has incoming edges from all states.
- Except for start and accept states, there are edges between each pair of states, including edges from each state to itself.

This is an example of GNFA:

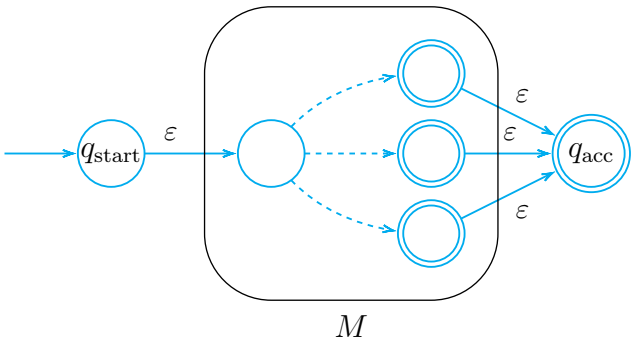


As a convention, we usually will not draw edges, which has $\varnothing$ on them.

Proof. If a language is regular, there exists DFA M that recognises the language.

1) First we convert DFA M into GNFA.

- We add a new start state and accept state.
- We connect the new start state with ε-arrow to the old start state.
- We connect the old accept states with ε-arrows to the new accept state.
- If there are parallel arrows (arrows with multiple labels) between two states, we replace them with a single arrow whose label is the union of the old labels.
- Add arrows labeled "$\varnothing$" if there is no arrow between two states.



The new automaton is GNFA that recognises the same language.

2) Further we want to convert obtained GNFA into regular expression.

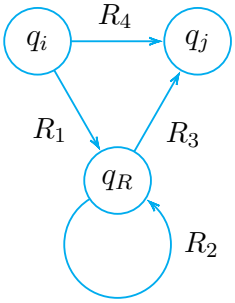
Algorithm (conversion of GNFA into regular expression):

1. If GNFA has only start and accept states then there is a single edge between them with a regular expression. Done.
2. If there is a state other than the start and accept states, we construct an equivalent GNFA without this particular state. This could be achieved by removing this state and updating all the labels on the remaining arrows.

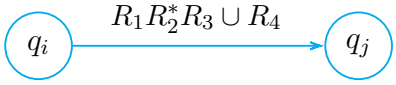
   More specifically, assume we want to remove state q_R. Let us see how the arrow from q_i to q_j should be updated[2]. Assume that before

[2]Note that the case $i = j$, i.e. $q_i = q_j$, should be also considered.
Also note that since arrows have orientation, we should also consider updating the arrow from q_j to q_i.

updating the arrows of interest have the following labels: $q_i \xrightarrow{R_1} q_{\mathrm{R}}$, $q_{\mathrm{R}} \xrightarrow{R_2} q_{\mathrm{R}}$, $q_{\mathrm{R}} \xrightarrow{R_3} q_j$, $q_i \xrightarrow{R_4} q_j$:



Then after updating the arrow from q_i to q_j should have the following label (it reflects the path via removed edge q_R):



3. Go to 1.

   End of algorithm.

Note that if either of R_1 and R_3 has label $\varnothing$, then resulting label will remain unchanged: $q_i \xrightarrow{R_4} q_j$. □

Definition. *GNFA is a 5-tuple* $(Q, \Sigma, \delta, q_{start}, q_{acc})$, *where*

- Q *is the set of states;*
- Σ *is the input alphabet;*
- $\delta\colon \big(Q \setminus \{q_{acc}\}\big) \times \big(Q \setminus \{q_{start}\}\big) \to \mathcal{R}$ *is a transition function, where* $\mathcal{R}$ *is a set of all regular expressions (so* $\delta(q_i, q_j)$ *is a label on the arrow from* q_i *to* q_j*);*
- $q_{start} \in Q$ *is a start state;*
- $q_{acc} \in Q$ *is an accept state.*

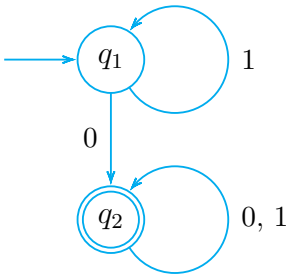
Definition. *Let* $M = (Q, \Sigma, \delta, q_{start}, q_{acc})$ *be a GNFA and* w *be a word over alphabet* Σ. *We say that automaton* M *accepts* w *if there are a split* $w = w_1 w_2 \dots w_k$, $w_i \in \Sigma^*$, *and sequence of states* $q_0, q_1, \dots, q_k$, *such that*

1. $q_0 = q_{start}$;
2. *for each* $i = 1, 2, \dots, k$ *we have* $w_i \in L(R_i)$, *where* $R_i = \delta(q_{i-1}, q_i)$;
3. $q_k = q_{acc}$.

Algorithm for conversion of GNFA into regular expression

1. Let k be the number of states in GNFA G.
2. If $k = 2$ then G consists of a start state, an accepting state and an arrow connecting them labelled with regular expression R. Return R.
3. If $k > 2$, choose any state $q_{\mathrm{R}} \in Q$, $q_{\mathrm{R}} \neq q_{\mathrm{start}}$, $q_{\mathrm{R}} \neq q_{\mathrm{acc}}$. Construct a new GNFA $G' = (Q', \Sigma, \delta', q_{\mathrm{start}}, q_{\mathrm{acc}})$, where
   (a) $Q' = Q \setminus \{q_{\mathrm{R}}\}$;
   (b) for each $q_i \in Q' \setminus \{q_{\mathrm{acc}}\}$, $q_j \in Q' \setminus \{q_{\mathrm{start}}\}$ let
   $$\delta'(q_i, q_j) = R_1 R_2^* R_3 \cup R_4,$$
   where $R_1 = \delta(q_i, q_{\mathrm{R}})$, $R_2 = \delta(q_{\mathrm{R}}, q_{\mathrm{R}})$, $R_3 = \delta(q_{\mathrm{R}}, q_j)$, $R_4 = \delta(q_i, q_j)$.
4. Assign $G := G'$ and go to 1.

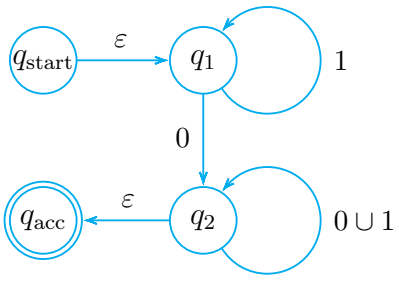
6. Given the following DFA:



what is the equivalent regular expression?

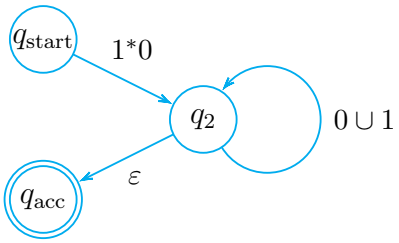
Solution.

1. Add new start and accept states:

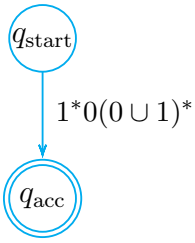


Note: again, we do not draw arrows with $\varnothing$ for the sake of clarity of the picture.

2. Remove q_1:

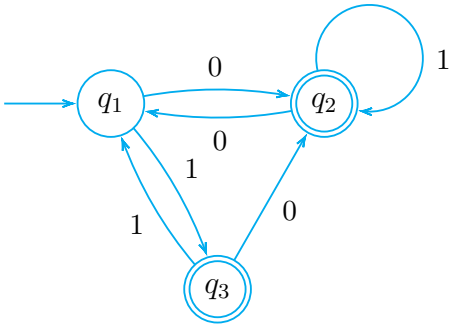


3. Remove q_2:



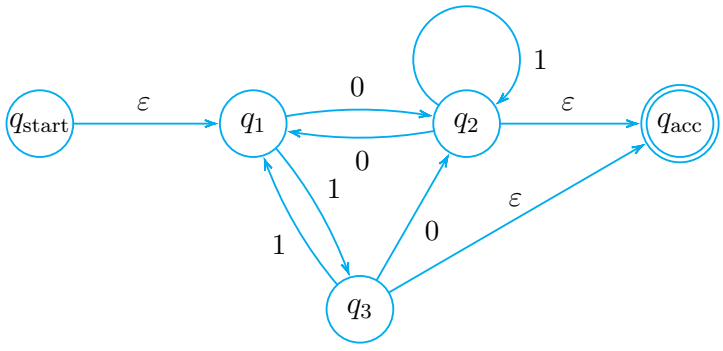
4. The resulting regular expression is $1^*0(0 \cup 1)^*$.

7. Given the following DFA, what is the equivalent regular expression?

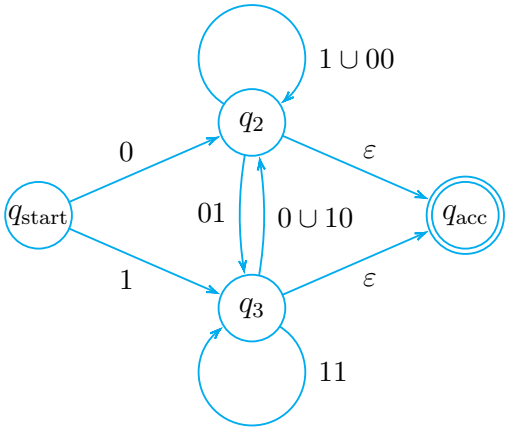


Solution.

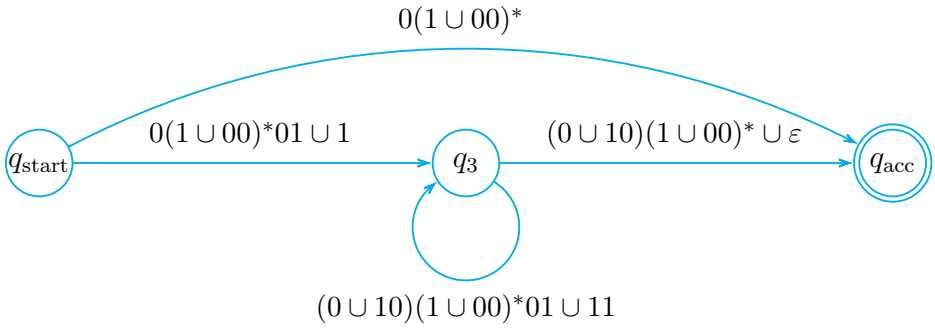
1. Add start and accept states:



2. Remove q_1:



3. Remove q_2:



4. Remove q_3: