Automata and Formal Languages — Homework 6

Due 27.11.2018

Exercise 6.1

(a) Let $n \in \mathbb{N}$ be such that $n \geq 2$. Show that $L_n = \{w \in \{a,b\}^* \mid |w| \equiv 0 \pmod{n}\}$ has exactly n residuals, without constructing any automaton for L_n.

(b) Consider the following "proof" showing that L_2 is non regular:

> Let $i, j \in \mathbb{N}$ be such that i is even and j is odd. By definition of L_2, we have $\varepsilon \in (L_2)^{a^i}$ and $\varepsilon \notin (L_2)^{a^j}$. Therefore, the a^i-residual and a^j-residual of L_2 are distinct. Since there are infinitely many even numbers i and odd numbers j, this implies that L_2 has infinitely many residuals, and hence that L_2 is not regular. □

Language L_2 is regular, so this "proof" must be incorrect. Explain what is wrong with the "proof".

Exercise 6.2

(a) Build B_p and C_p for the word pattern $p = abrababra$.

(b) How many transitions are taken when reading $t = abrar$ in B_p and C_p respectively?

(c) Let $n > 0$. Find a text $t \in \{a,b\}^*$ and a word pattern $p \in \{a,b\}^*$ such that testing whether p occurs in t takes n transitions in B_p and $2n - 1$ transitions in C_p.

Exercise 6.3

In order to make pattern-matching robust to typos we want to include also "similar" words in our results. For this we consider words with a small Levenshtein-distance (edit-distance) "similar".

We transform a word w to a new word w' using the following operations (with $a_i, b \in \Sigma$):

- *replace* (R): $a_1 \dots a_{i-1} a_i a_{i+1} \dots a_l \to a_1 \dots a_{i-1} b a_{i+1} \dots a_l$
- *delete* (D): $a_1 \dots a_{i-1} a_i a_{i+1} \dots a_l \to a_1 \dots a_{i-1} \varepsilon a_{i+1} \dots a_l$
- *insert* (I): $a_1 \dots a_{i-1} a_i a_{i+1} \dots a_l \to a_1 \dots a_{i-1} a_i b a_{i+1} \dots a_l$

The Levenshtein-distance (denoted $\Delta(w, w')$) of w and w' is the minimal number of operations (R,D,I) needed to transform w into w'. We denote with $\Delta_{L,i} = \{w \in \Sigma^* \mid \exists w' \in L.\, \Delta(w', w) \leq i\}$ the language of all words with edit-distance at most i to some word of L.

(a) Compute $\Delta(abcde, accd)$.

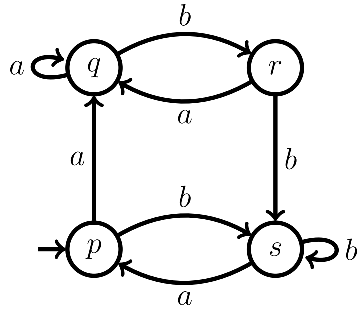
(b) Prove the following statement: If L is a regular language, then $\Delta_{L,n}$ is a regular language.

(c) Let p be the pattern *otto*. Construct an NFA locating the pattern or variations of it with edit-distance 1.

Exercise 6.4

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. A word $w \in \Sigma^*$ is a *synchronizing word* of A if reading w from any state of A leads to a common state, i.e. if there exists $q \in Q$ such that for every $p \in Q$, $p \xrightarrow{w} q$. A DFA is *synchronizing* if it has a synchronizing word.

(a) Show that the following DFA is synchronizing:



(b) Give a DFA that is not synchronizing.

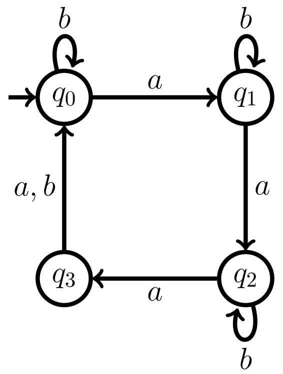
(c) Give an exponential time algorithm (reusing constructions from the lecture) to decide whether a DFA is synchronizing. [Hint:]

(d) Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. We say that A is (p, q)-*synchronizing* if there exist $w \in \Sigma^*$ and $r \in Q$ such that $p \xrightarrow{w} r$ and $q \xrightarrow{w} r$. Show that A is synchronizing if and only if it is (p, q)-synchronizing for every $p, q \in Q$.

(e) Give a polynomial time algorithm to test whether a DFA is synchronizing. [Hint:]

(f) Show, from (d), that every synchronizing DFA with n states has a synchronizing word of length at most $(n^2 - 1)(n - 1)$. [Hint:]

(g) Show that the upper bound obtained in (f) is not tight by finding a synchronizing word of length $(4 - 1)^2$ for the following DFA:



Solution 6.1

(a) We claim that the residuals of L_n are

$$(L_n)^{a^0}, (L_n)^{a^1}, \ldots, (L_n)^{a^{n-1}}. \tag{1}$$

Let us first show that for every word w we have $(L_n)^w = (L_n)^{a^{|w| \bmod n}}$. Let $w \in \{a,b\}^*$. For every $u \in \{a,b\}^*$, we have

$$\begin{aligned} u \in (L_n)^w &\iff wu \in L_n \\ &\iff |wu| \equiv 0 \pmod{n} \\ &\iff |w| + |u| \equiv 0 \pmod{n} \\ &\iff (|w| \bmod n) + |u| \equiv 0 \pmod{n} \\ &\iff |a^{|w| \bmod n}| + |u| \equiv 0 \pmod{n} \\ &\iff |a^{|w| \bmod n} u| \equiv 0 \pmod{n} \\ &\iff a^{|w| \bmod n} u \in L_n \\ &\iff u \in (L_n)^{a^{|w| \bmod n}}. \end{aligned}$$

It remains to show that the residuals of (1) are distinct. Let $0 \leq i, j < n$ be such that $i \neq j$. We have $a^{n-i} \in (L_n)^{a^i}$, and $a^{n-i} \notin (L_n)^{a^j}$ since $|a^j a^{n-i}| \bmod n = j - i \neq 0$. Therefore, $(L_n)^{a^i} \neq (L_n)^{a^j}$. ☐

(b) The part of the "proof" showing that $(L_2)^{a^i} \neq (L_2)^{a^j}$, for every even i and odd j, is correct. However, this only shows that L_2 has at least two residuals. Indeed, even if there are infinitely many even and odd numbers, the following is not ruled out:

$$\begin{aligned} (L_2)^{a^0} &= (L_2)^{a^2} = (L_2)^{a^4} = \cdots, \\ (L_2)^{a^1} &= (L_2)^{a^3} = (L_2)^{a^5} = \cdots. \end{aligned}$$

In order to show that a language has infinitely many residuals, one must exhibit an infinite subset of residuals that are *pairwise* distinct.

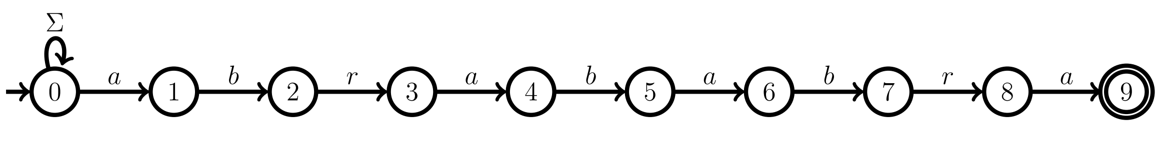
(c) We claim that the residuals $P^{a^1}, P^{a^2}, P^{a^4}, P^{a^8}, \ldots$ are pairwise distinct. Let $i, j \in \mathbb{N}$ be such that $i \neq j$. Let us show that $P^{a^{2^i}} \neq P^{a^{2^j}}$. We have $a^{2^i} \in P^{a^{2^i}}$ since $|a^{2^i} a^{2^i}| = 2^{i+1}$. Moreover, $a^{2^i} \notin P^{a^{2^j}}$ since $|a^{2^j} a^{2^i}| = 2^i + 2^j$ which is not a power of two since it lies in between two consecutive powers of two:

$$2^{\max(i,j)} < 2^i + 2^j < 2^{\max(i,j)} + 2^{\max(i,j)} = 2^{\max(i,j)+1}. \qquad \square$$

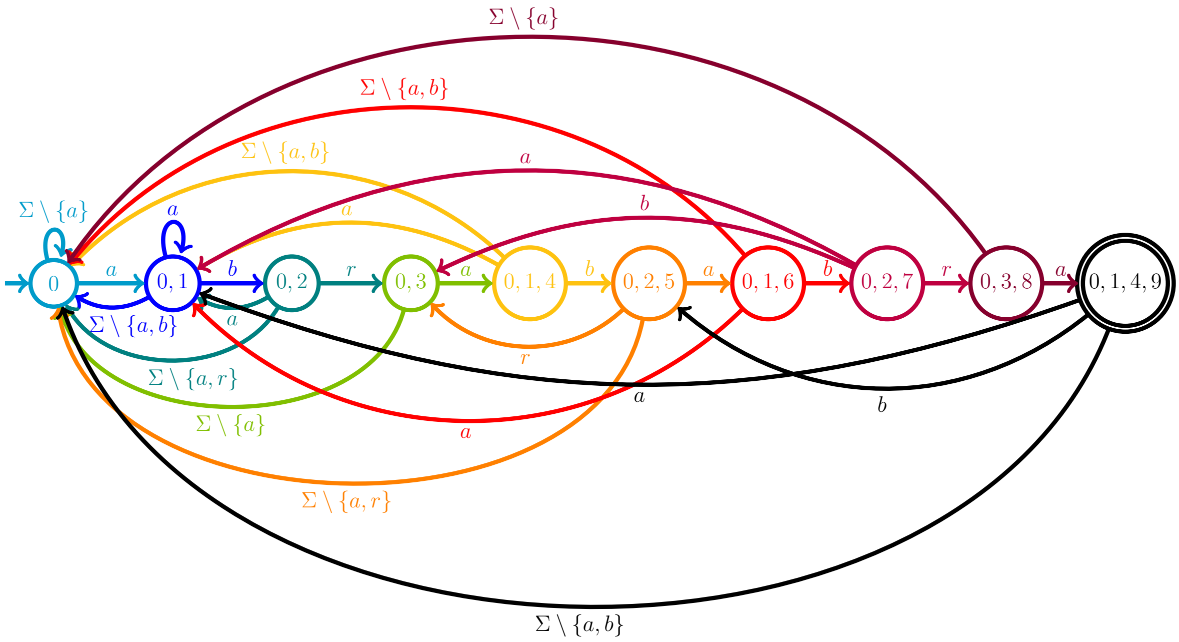
The language $P \cap \{a\}^*$ is also non regular since the above proof does not ever make use of letter b.

Solution 6.2

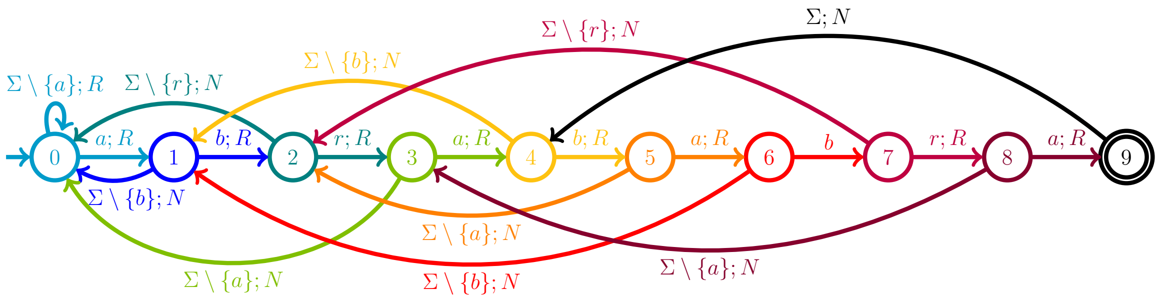
(a) A_p :



B_p :



C_p :

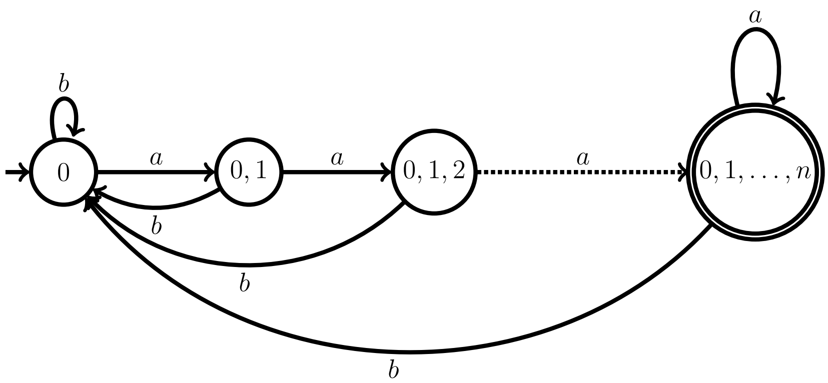


(b) Five transitions taken in B_p: $\{0\} \xrightarrow{a} \{0,1\} \xrightarrow{b} \{0,2\} \xrightarrow{r} \{0,3\} \xrightarrow{a} \{0,1,4\} \xrightarrow{r} \{0\}$.

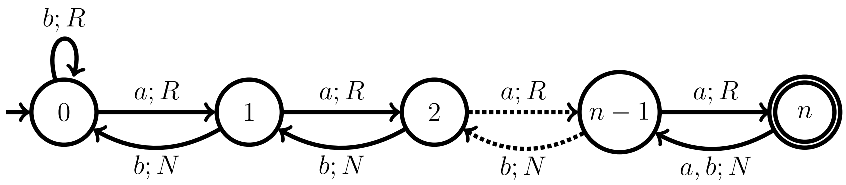
Seven transitions taken in C_p: $0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{r} 3 \xrightarrow{a} 4 \xrightarrow{r} 1 \xrightarrow{r} 0 \xrightarrow{r} 0$.

(c) $t = a^{n-1}b$ and $p = a^n$. The automata B_p and C_p are as follows:

B_p:



C_p:



The runs over t on B_p and C_p are respectively:

$$\{0\} \xrightarrow{a} \{0,1\} \xrightarrow{a} \{0,1,2\} \xrightarrow{a} \cdots \xrightarrow{a} \{0,1,\ldots,n-1\} \xrightarrow{b} \{0\},$$

and

$$0 \xrightarrow{a} 1 \xrightarrow{a} 2 \xrightarrow{a} \cdots \xrightarrow{a} (n-1) \xrightarrow{b} (n-2) \xrightarrow{b} (n-3) \xrightarrow{b} \cdots \xrightarrow{b} 0 \xrightarrow{b} 0.$$

Solution 6.3

1. $\Delta(abcde, accd) = 2$.
2. Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA für L. Wir erhalten einen NFA-ϵ N für $\Delta_{L,n}$, in dem wir n Fehlerebenen einführen. Der Automat darf nicht-deterministisch einen Fehler machen muss dann aber zu einer höheren Fehlerebene wechseln. Formal:

$$N = (Q \times [0,n], \Sigma, \delta', (q_0, 0), F \times [0,n])$$

mit

$$\begin{aligned}
\delta' &= \{((q,i), a, (p,i)) \mid q,p \in Q \wedge i \leq n \wedge a \in \Sigma \wedge \delta(q,a) = p\} && \text{kein Fehler}\\
&\cup \{((q,i), \varepsilon, (p,i+1)) \mid q,p \in Q \wedge i < n \wedge (\exists a \in \Sigma.\, \delta(q,a) = p)\} && \text{Delete}\\
&\cup \{((q,i), a, (q,i+1)) \mid q \in Q \wedge i < n \wedge a \in \Sigma\} && \text{Insert}\\
&\cup \{((q,i), b, (p,i+1)) \mid q,p \in Q \wedge i < n \wedge (\exists a \in \Sigma \setminus \{b\}.\, \delta(q,a) = p)\} && \text{Replace}
\end{aligned}$$

3. Anhand der bekannten Algorithmen berechnen wir $\delta'((q_0, 0), w) \cap F \times [0, n]$ und wählen ein (q, i) mit minimaler Fehlerebene i. Wir betrachten nun einen akzeptierenden Lauf für w der in (q, i) endet. Immer wenn eine "Fehlerkante" benutzt wird, ändern wir das Wort w zu w' ab, so dass der Lauf in Ebene 0 in den entsprechenden Zielzustand wechseln kann.
4. Wörter w mit $\Delta(b, w) = 1$: $\varepsilon, a, ab, ba, bb$.

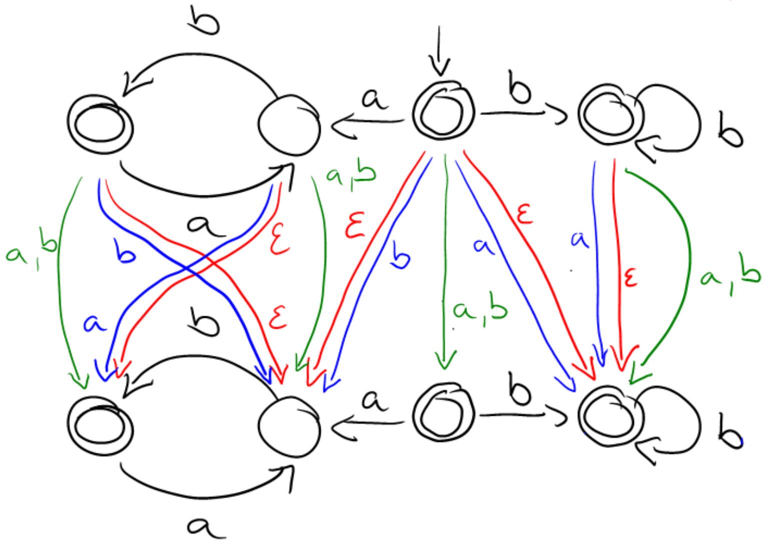


Figure 1: Konstruierter NFA-ϵ für L. Delete: rot, Insert: grün, Replace: blau.

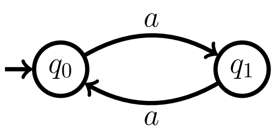
Solution 6.4

(a) abb is a synchronizing word:

$$p \xrightarrow{a} q \xrightarrow{b} r \xrightarrow{b} s,$$
$$q \xrightarrow{a} q \xrightarrow{b} r \xrightarrow{b} s,$$
$$r \xrightarrow{a} q \xrightarrow{b} r \xrightarrow{b} s,$$
$$s \xrightarrow{a} p \xrightarrow{b} s \xrightarrow{b} s.$$

★ As seen in class, aa and bb are also synchronizing words. In fact, one can prove that the set of synchronizing words of the automaton is: $(a+b)^*(aa+bb)(a+b)^*$.

(b) The following DFA is not synchronizing:



(c) Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and let $A_q = (Q, \Sigma, \delta, q, F)$ for every $q \in Q$. A word w is synchronizing for A if and only if reading w from each automaton A_q leads to the same state. Therefore, we may construct a DFA B that simulates all automata A_q simultaneously and tests whether a common state can be reached.

More formally, let $B = (\mathcal{P}(Q), \Sigma, \delta', \{Q\}, F')$ where

- $\delta'(P, a) = \{\delta(q, a) : q \in P\}$, and
- $F' = \{\{q\} : q \in Q\}$.

Automaton A is synchronizing if and only if $L(B) \neq \emptyset$. It is possible to compute B by adapting the algorithm *NFAtoDFA*(A) seen in class:

Input: DFAs $A = (Q, \Sigma, \delta, q_0, F)$.
Output: Is A synchronizing?

$Q' \leftarrow \emptyset$
$W \leftarrow \{Q\}$
while $W \neq \emptyset$ **do**
    pick P **from** W
    if $|P| = 1$ **then**
        return true
    else
        add P **to** Q'
    for $a \in \Sigma$ **do**
        $P' \leftarrow \{\delta(q, a) : q \in P\}$
        if $P' \notin Q'$ and $P' \notin W$ **then**
            add P' **to** W
return false

(d) $\Rightarrow$) Immediate.

$\Leftarrow$) Let $Q = \{q_0, q_1, \ldots, q_n\}$. Let us extend δ to words, i.e. $\delta(q_i, w) = r$ where $q_i \xrightarrow{w} r$. For every $i, j \in [n]$, let $w(i, j) \in \Sigma^*$ be such that $\delta(q_i, w(i, j)) = \delta(q_j, w(i, j))$. Let us define the following sequence of words:

$$\begin{aligned} u_1 &= w(q_0, q_1) \\ u_\ell &= w(\delta(q_\ell, u_1 u_2 \cdots u_{\ell-1}), \delta(q_{\ell-1}, u_1 u_2 \cdots u_{\ell-1})) && \text{for every } 2 \leq \ell \leq n. \end{aligned}$$

We claim that $u_1 u_2 \cdots u_n$ is a synchronizing word. To see that, let us prove by induction on ℓ that for every $i, j \in [\ell]$,

$$\delta(q_i, u_1 u_2 \cdots u_\ell) = \delta(q_j, u_1 u_2 \cdots u_\ell).$$

For $\ell = 1$, the claims holds by definition of u_1. Let $2 \leq \ell \leq n$. Assume the claim holds for $\ell - 1$. Let $i, j \in [\ell]$. If $i, j < \ell$, then

$$\begin{aligned} \delta(q_i, u_1 u_2 \cdots u_\ell) &= \delta(\delta(q_i, u_1 u_2 \cdots u_{\ell-1}), u_\ell) \\ &= \delta(\delta(q_j, u_1 u_2 \cdots u_{\ell-1}), u_\ell) && \text{(by induction hypothesis)} \\ &= \delta(q_j, u_1 u_2 \cdots u_\ell). \end{aligned}$$

If $i = \ell$ and $j < \ell$, then

$$\begin{aligned} \delta(q_\ell, u_1 u_2 \cdots u_\ell) &= \delta(\delta(q_i, u_1 u_2 \cdots u_{\ell-1}), u_\ell) \\ &= \delta(\delta(q_{i-1}, u_1 u_2 \cdots u_{\ell-1}), u_\ell) && \text{(by definition of } u_\ell) \\ &= \delta(\delta(q_j, u_1 u_2 \cdots u_{\ell-1}), u_\ell) && \text{(by induction hypothesis)} \\ &= \delta(q_j, u_1 u_2 \cdots u_\ell)\ . \end{aligned}$$

The case were $i < \ell$ and $i = \ell$ is symmetric, and the case where $i = j = \ell$ is trivial. □

(e) We use the approach used in (c), but instead of simulating all automata A_q at once, we simulate all pairs A_p and A_q. From (d), this is sufficient. The adapted algorithm is as follows:

```
   Input: DFAs A = (Q, Σ, δ, q_0, F).
   Output: A is synchronizing?
1  for p, q ∈ Q s.t. p ≠ q do
2      if ¬pair-synchronizable(p, q) then
3          return false
4  return true
5
6  pair-synchronizable(p, q) :
7      Q' ← ∅
8      W ← {{p, q}}
9      while W ≠ ∅ do
10         pick P from W
11         if |P| = 1 then
12             return true
13         else
14             add P to Q'
15         for a ∈ Σ do
16             P' ← {δ(q, a) : q ∈ P}
17             if P' ∉ Q' and P' ∉ W then
18                 add P' to W
19         return false
```

The `for` loop at line 1 is iterated at most $|Q|^2$ times. The `while` loop of `pair-synchronizable(`p, q`)` is iterated at most $|Q|^2$ times, the `for` loop at line 15 is taken iterated at most $|\Sigma|$ times, and line 16 requires time $O(|Q|)$. Hence, the total running time of the algorithm is in $O(|Q|^5 \cdot |\Sigma|)$.

★ In class, I mentioned that we should use the pairing $[A_p, A_q]$ to test whether A is (p, q)-synchronizing in polynomial time. This indeed works. However, using the approach of (c) as it is done above, i.e. starting from $\{p, q\}$ instead of $[p, q]$, also takes polynomial time. This works because A is deterministic and hence any reachable subset contains at most two states.

★ Our proof of (d) is constructive and yields an algorithm working in time $O(|Q|^4 + |Q|^3 \cdot |\Sigma|)$ to compute a sychronizing word of length $O(|Q|^3)$, if there exists one. See `synchronizing.py` for an implementation in Python. It is possible to do better. An algorithm presented in [1] computes a synchronizing word of length $O(|Q|^3)$, if there exists one, in time $O(|Q|^3 + |Q|^2 \cdot |\Sigma|)$.

(f) In the proof of (d), we built a synchronizing word $w = u_1 u_2 \cdots u_{|Q|-1}$ where each u_i is a (p, q)-synchronizing word for some $p, q \in Q$. We claim that if there exists a (p, q)-synchronizing word, then there exists one of length at most $|Q|^2 - 1$. This leads to the overall $(|Q| - 1)(|Q|^2 - 1)$ upper bound.

To see that the claim holds, assume for the sake of contradiction that every (p, q)-synchronizing word has length at least $|Q|^2$. Let w be such a minimal word. Let $r = \delta(p, w)$. We have

$$\begin{aligned} p &\xrightarrow{w} r, \\ q &\xrightarrow{w} r. \end{aligned}$$

This yields the following run in $[A, A]$:

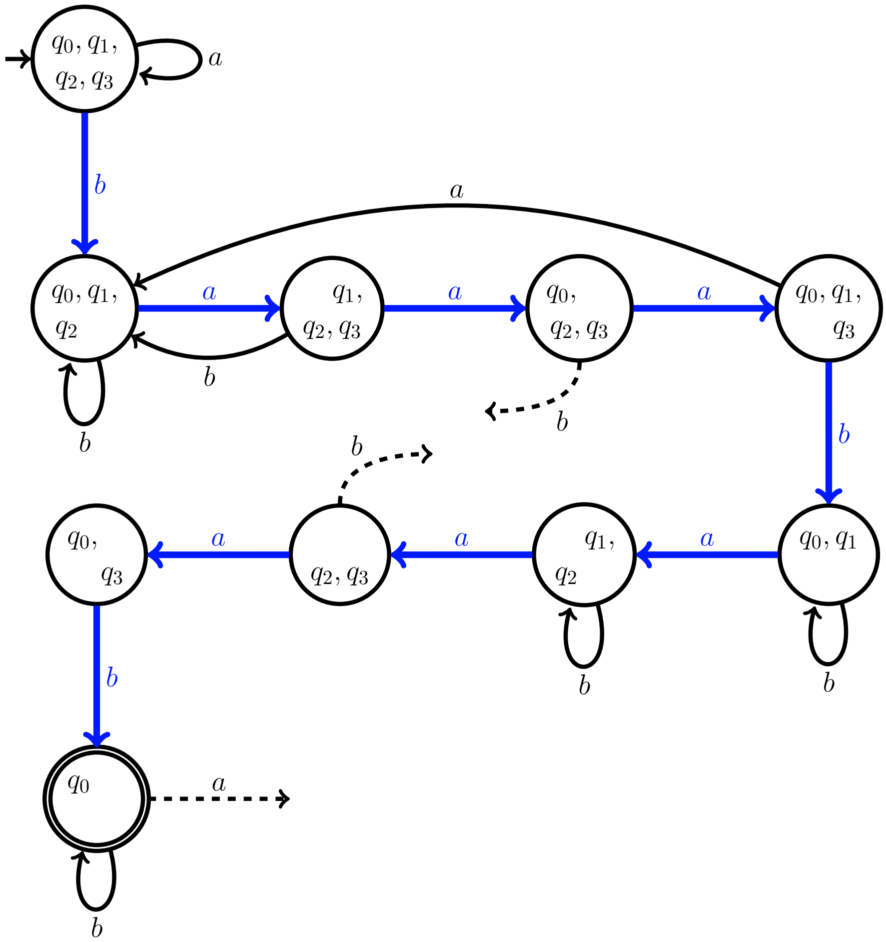
$$\begin{bmatrix} p \\ q \end{bmatrix} \xrightarrow{w} \begin{bmatrix} r \\ r \end{bmatrix}.$$

Since $|w(p, q)| \geq |Q|^2$, by the pigeonhole principle, there exist $s, t \in Q$, $x, z \in \Sigma^*$ and $y \in \Sigma^+$ such that $w = xyz$ and

$$\begin{bmatrix} p \\ q \end{bmatrix} \xrightarrow{x} \begin{bmatrix} s \\ t \end{bmatrix} \xrightarrow{y} \begin{bmatrix} s \\ t \end{bmatrix} \xrightarrow{z} \begin{bmatrix} r \\ r \end{bmatrix}.$$

Hence, xz is a smaller (p, q)-synchronizing word, which is a contradiction. □

(g) ba^3ba^3b is such a word. It can be obtained, e.g., from the algorithm designed in (c):



The *Černý conjecture* states that every synchronizing DFA has a synchronizing word of length at most $(|Q|-1)^2$. Since 1964, no one has been able to prove or disprove this conjecture. To this day, the best upper bound on the length of minimal synchronizing words is $((|Q|^3 - |Q|)/6) - 1$ (see [2]).

References

[1] David Eppstein. Reset sequences for monotonic automata. *SIAM Journal on Computing*, 19(3):500–510, 1990. Available online at `http://www.ics.uci.edu/~eppstein/pubs/Epp-SJC-90.pdf`.

[2] Jean-Éric Pin. On two combinatorial problems arising from automata theory. volume 17 of *Annals of Discrete Mathematics*, pages 535–548. North-Holland, 1983. Available online at `https://hal.archives-ouvertes.fr/hal-00143937/document`.