Automata and Formal Languages — Homework 4

Due 11.11.2016

Exercise 4.1
The *perfect shuffle* of two languages $L, L' \in \Sigma^*$ is defined as:

$$L \mathbin{\widetilde{\sqcup\!\sqcup}} L' = \{w \in \Sigma^* : \exists a_1, \ldots, a_n, b_1, \ldots, b_n \in \Sigma \text{ s.t. } \begin{array}{l} a_1 \cdots a_n \in L \text{ and} \\ b_1 \cdots b_n \in L' \text{ and} \\ w = a_1 b_1 \cdots a_n b_n \ \} \ . \end{array}$$

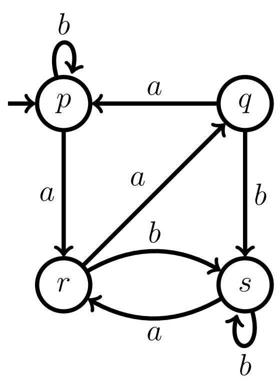
Give an algorithm that takes two DFAs A and B in input, and that returns a DFA accepting $L(A) \mathbin{\widetilde{\sqcup\!\sqcup}} L(B)$.

Exercise 4.2
Let Σ_1 and Σ_2 be alphabets. A *morphism* is a function $h : \Sigma_1^* \to \Sigma_2^*$ such that $h(\varepsilon) = \varepsilon$ and $h(uv) = h(u) \cdot h(v)$ for every $u, v \in \Sigma_1^*$. In particular, $h(a_1 a_2 \cdots a_n) = h(a_1) h(a_2) \cdots h(a_n)$ for every $a_1, a_2, \ldots, a_n \in \Sigma$. Hence, a morphism h is entirely determined by its image over letters.

1. Let A be an NFA over Σ_1. Give an NFA B that accepts $h(L(A)) = \{h(w) : w \in L(A)\}$.
2. Let A be an NFA over Σ_2. Give an NFA B that accepts $h^{-1}(L(A)) = \{w \in \Sigma_1^* : h(w) \in L(A)\}$.
3. Recall that $\{a^n b^n : n \in \mathbb{N}\}$ is not regular. Using this fact and the previous results, show that $L \subseteq \{a, b, c, d, e\}^*$ where
$$L = \{(ab)^m a^n e (cd)^m d^n : m, n \in \mathbb{N}\}$$
is also not regular.

Exercise 4.3
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.

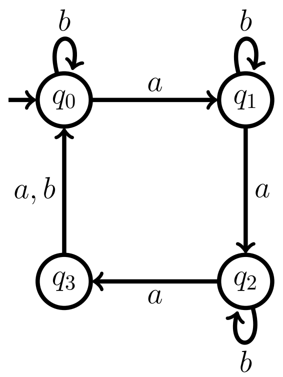
(c) Give an exponential time algorithm to decide whether a DFA is synchronizing. (Hint: use the powerset construction).

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

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

(f) Show that (d) implies that every synchronizing DFA with n states has a synchronizing word of length at most $(n^2 - 1)(n - 1)$. (Hint: you might need to reason in terms of the product construction.)

(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 4.1

Let $A = (Q, \Sigma, \delta, q_0, F)$ and $B = (Q', \Sigma, \delta', q'_0, F')$. Intuitively, we build a DFA C that alternates between reading a letter in A and reading a letter in B. To do so, we build two copies of the product of A and B. Reading a letter a in the first copy simulates reading a in A and then goes to the bottom copy, and vice versa. A word is accepted if it ends up in a state (p, q) of the top copy such that $p \in F$ and $q \in F'$.

Formally, $C = (Q'', \Sigma, \delta'', q''_0, F'')$ where

- $Q'' = Q \times Q' \times \{\top, \bot\}$,
- $\delta(p, a) = \begin{cases} (\delta(q, a), q', \bot) & \text{if } p = (q, q', r) \text{ and } r = \top, \\ (q, \delta'(q', a), \top) & \text{if } p = (q, q', r) \text{ and } r = \bot, \end{cases}$
- $F'' = \{(q, q', \top) : q \in F \text{ and } q' \in F'\}$.

As for most constructions seen in class, some states of C may be non reachable from the initial state. We give an algorithm that avoids this:

Input: DFAs $A = (Q, \Sigma, \delta, q_0, F)$ and $B = (Q', \Sigma, \delta', q'_0, F')$.
Output: A DFA $C = (Q'', \Sigma, \delta'', q''_0, F'')$ such that $L(C) = L(A) \,\widetilde{\sqcup\!\sqcup}\, L(B)$.

$Q'' \leftarrow \emptyset$
$\delta'' \leftarrow \emptyset$
$F'' \leftarrow \emptyset$
$W \leftarrow \{(q_0, q'_0, \top)\}$
while $W \neq \emptyset$ **do**
  pick $p = (q, q', r)$ **from** W
  add p **to** Q''
  if $q \in F$, $q' \in F'$ and $r = \top$ **then**
    add p **to** F''
  for $a \in \Sigma$ **do**
    if $r = \top$ **then**
      $p' \leftarrow (\delta(q, a), q', \bot)$
    else if $r = \bot$ **then**
      $p' \leftarrow (q, \delta(q', a), \top)$
    add (p, a, p') **to** δ''
    if $p' \notin Q''$ **then add** p' **to** W
return $(Q'', \Sigma, \delta'', (q_0, q'_0, \top), F'')$

Solution 4.2

1. Since h is determined by its image over letters, we simply replace each transition (p, a, q) of A by a sequence of transitions from p to q labeled by $h(a)$. Some ε-transitions may be introduced if $h(a) = \varepsilon$ for some letters a, but we can removed them as seen in class.

2. Let $A = (Q, \Sigma_2, \delta, Q_0, F)$. We keep the states of A unchanged, but we remove its transitions. For each $p, q \in Q$ and $a \in \Sigma_1$, we add a transition (p, a, q) to B for every q that can be reached from p by reading $h(a)$ in A. More formally, we let $B = (Q, \Sigma_1, \delta', Q_0, F)$ where
$$\delta'(p, a) = \{q \in Q : p \xrightarrow{h(a)}_A q\} .$$

3. Suppose L is regular. There exists an NFA A that accepts L. Let $h : \{a, b, c, d, e\} \to \{a, b\}$ be the morphism such that
$$\begin{aligned} h(a) &= a, \\ h(b) &= \varepsilon, \\ h(c) &= \varepsilon, \\ h(d) &= b, \\ h(e) &= \varepsilon. \end{aligned}$$

We have

$$
\begin{aligned}
h(L) &= \{(a\varepsilon)^m a^n \varepsilon(\varepsilon b)^m b^n : m, n \in \mathbb{N}\} \\
&= \{a^{m+n} b^{m+n} : m, n \in \mathbb{N}\} \\
&= \{a^n b^n : n \in \mathbb{N}\} \ .
\end{aligned}
$$

Therefore, by (1), there exists an NFA that accepts $\{a^n b^n : n \in \mathbb{N}\}$, which is a contradiction.

This contradiction can also be obtained from the following morphism:

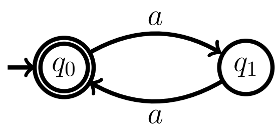
$$
\begin{aligned}
h(a) &= \varepsilon, \\
h(b) &= a, \\
h(c) &= b, \\
h(d) &= \varepsilon, \\
h(e) &= \varepsilon.
\end{aligned}
$$

Solution 4.3

(a) ba is a synchronizing word:

$$
\begin{aligned}
p &\xrightarrow{b} p \xrightarrow{a} r \ , \\
q &\xrightarrow{b} s \xrightarrow{a} r \ , \\
r &\xrightarrow{b} s \xrightarrow{a} r \ , \\
s &\xrightarrow{b} s \xrightarrow{a} r \ .
\end{aligned}
$$

(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 build a DFA B that simulates every automaton 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
- $F' = \{\{q\} : q \in Q\}$.

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: A is synchronizing?

```
if |Q| = 1 then return true
Q' ← ∅
W ← {Q}
while W ≠ ∅ do
    pick P from W
    add P to Q'
    for a ∈ Σ do
        P' ← {δ(q, a) : q ∈ P}
        if |P'| = 1 then
            return true
        else if P' ∉ Q' 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

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

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

(e) We use the approach used in (c), but instead of simulating every automaton 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, q0, 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         add P to Q′
12         for a ∈ Σ do
13             P′ ← {δ(q, a) : q ∈ P}
14             if |P′| = 1 then
15                 return true
16             else if P′ ∉ Q′ then
17                 add P′ to W
18         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 $|Q|^2$, and the `for` loop within it is iterated at most $|\Sigma|$ times. Hence, the total running time of the algorithm is in $O(|Q|^4 \cdot |\Sigma|)$.

★ Our proof of (d) is constructive and yields an algorithm working in time $O(|Q|^4 \cdot |\Sigma|)$ 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 existe one, in time $O(|Q|^3 + |Q|^2 \cdot |\Sigma|)$.

(f) We say that a word w is (p, q)-synchronizing if $\delta(p, w) = \delta(q, w)$. In the proof of (d), we have 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 the pair of A and itself:

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

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. □

★ As seen in class, it is possible to get a slightly better upper bound. If there exist $s, t \in Q$, $x, z \in \Sigma^*$ and $y \in \Sigma^+$ such that $w = xyz$ and

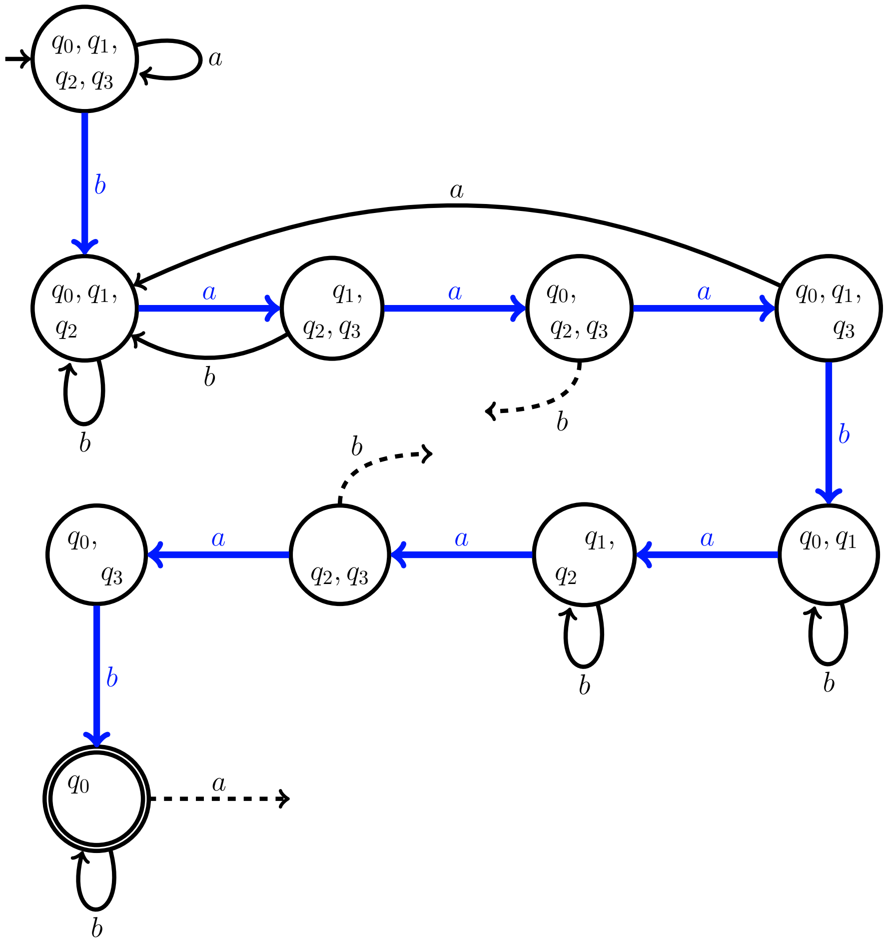
$$\begin{bmatrix} p \\ q \end{bmatrix} \xrightarrow{x} \begin{bmatrix} s \\ t \end{bmatrix} \xrightarrow{y} \begin{bmatrix} t \\ s \end{bmatrix} \xrightarrow{z} \begin{bmatrix} r \\ r \end{bmatrix} \ ,$$

then xz is a also a shorter (p,q)-synchronizing word. Moreover, if there exist $s \in Q$, $x \in \Sigma^*$ and $y \in \Sigma^+$ such that $w = xy$ and

$$\begin{bmatrix} p \\ q \end{bmatrix} \xrightarrow{x} \begin{bmatrix} s \\ s \end{bmatrix} \xrightarrow{z} \begin{bmatrix} r \\ r \end{bmatrix},$$

then x is a shorter (p,q)-synchronizing word. Thus, at most $\binom{n}{2}$ states of the form $[s\ t]$ appear along the path of a minimal (p,q)-synchronizing word, followed by a state of the form $[r\ r]$. Therefore, a minimal (p,q)-synchronizing word is of size at most $\binom{n}{2} = (n^2 - n)/2$. Overall, this yields a synchronizing word of length at most $(n-1)((n^2-n)/2) = n^3/2 - n^2 + n/2$.

(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`.