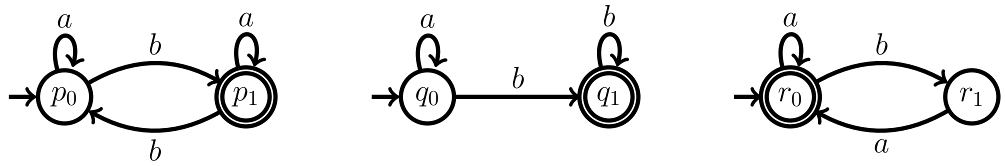
Technische Universität München **Winter term 2020/21**
I7
Prof. J. Esparza / M. Lazić / C. Weil-Kennedy

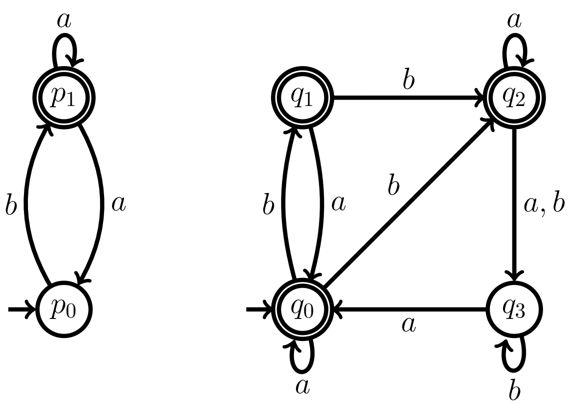
Automata and Formal Languages — Exercise Sheet 3

Exercise 3.1
Consider the following DFAs A, B and C:



Use pairings to decide *algorithmically* whether $L(A) \cap L(B) \subseteq L(C)$.

Exercise 3.2
Consider the following NFAs A and B:



(a) Use algorithm *UnivNFA* to determine whether $L(B) = \{a, b\}^*$.

(b) Use algorithm *InclNFA* to determine whether $L(A) \subseteq L(B)$.

Exercise 3.3

(a) We have seen that testing whether two NFAs accept the same language can be done by using algorithm *InclNFA* twice. Give an alternative algorithm, based on pairings, for testing equality.

(b) Give two NFAs A and B for which exploring only the minimal states of $[NFAtoDFA(A), NFAtoDFA(B)]$ is not sufficient to determine whether $L(A) = L(B)$.

(c) Show that the problem of determining whether an NFA and a DFA accept the same language is PSPACE-hard.

Exercise 3.4

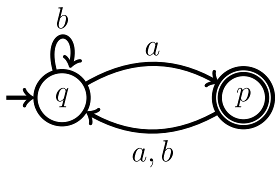
The *perfect shuffle* of two languages $L, L' \subseteq \Sigma^*$ is defined as:

$$L \mathbin{\widetilde{\sqcup\!\sqcup}} L' = \{w \in \Sigma^* : \exists a_1, \ldots, a_n, b_1, \ldots, b_n \in \Sigma \text{ s.t. } \begin{array}{l} a_1 \cdots a_n \in L \text{ and} \\ b_1 \cdots b_n \in L' \text{ and} \\ w = a_1 b_1 \cdots a_n b_n \text{ and } n \geq 0\} \ . \end{array}$$

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 3.5

Let $L \subseteq \Sigma^*$ be a language accepted by an NFA A. For every $u, v \in \Sigma^*$, we say that $u \preceq v$ if and only if u can be obtained by deleting zero, one or multiple letters of v. For example, $abc \preceq abca$, $abc \preceq acbac$, $abc \preceq abc$, $\varepsilon \preceq abc$ and $aab \not\preceq acbac$. Consider the following NFA A. Give an NFA-ε for each of the following languages and then generalize your approach to any NFA:

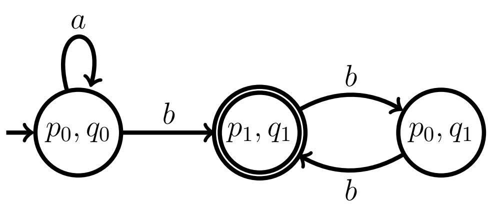


(a) $\downarrow L = \{w \in \Sigma^* \mid w \preceq w' \text{ for some } w' \in L\}$,

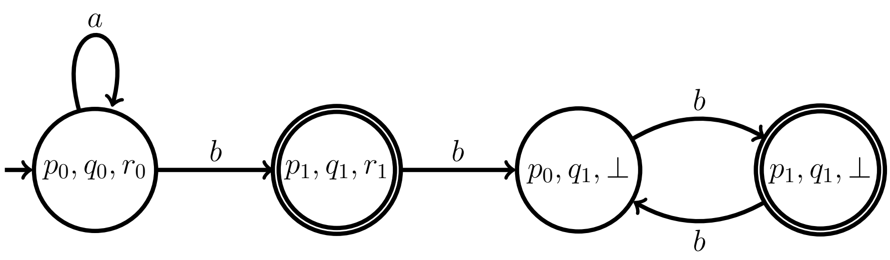
(b) $\uparrow L = \{w \in \Sigma^* \mid w' \preceq w \text{ for some } w' \in L\}$,

(c) $\sqrt{L} = \{w \in \Sigma^* \mid ww \in L\}$,

Solution 3.1
We first build the pairing accepting $L(A) \cap L(B)$. Note that it is not necessary to explore the implicit trap states of A and B as they cannot lead to final states in the pairing. We obtain:



Now, we build the pairing accepting $(L(A) \cap L(B)) \setminus L(C)$ from the above automaton and C. Note that we must explore the implicit trap state of C as it may be part of final states in the pairing. We obtain:



Since the above automaton contains final states, its language is non empty and hence $L(A) \cap L(B) \not\subseteq L(C)$. Note that we can reach this conclusion as soon as we construct state (p_1, q_1, r_1). For example, the word ab belongs to $L(a)$ and $L(b)$, but not to $L(c)$.

Solution 3.2

(a) The trace of the execution is as follows:

Iter.	$\mathcal{Q}$	$\mathcal{W}$
0	$\emptyset$	$\{\{q_0\}\}$
1	$\{\{q_0\}\}$	$\{\{q_1, q_2\}\}$
2	$\{\{q_0\}, \{q_1, q_2\}\}$	$\{\{q_2, q_3\}\}$
3	$\{\{q_0\}, \{q_1, q_2\}, \{q_2, q_3\}\}$	$\{q_3\}$

At the fourth iteration, the algorithm tests state $\{q_3\}$ which is minimal and non final, and hence it returns *false*. Therefore, $L(B) \neq \{a, b\}^*$.

(b) The trace of the algorithm is as follows:

Iter.	$\mathcal{Q}$	$\mathcal{W}$
0	$\emptyset$	$\{[p_0, \{q_0\}]\}$
1	$\{[p_0, \{q_0\}]\}$	$\{[p_1, \{q_1, q_2\}]\}$
2	$\{[p_0, \{q_0\}], [p_1, \{q_1, q_2\}]\}$	$\{[p_1, \{q_0, q_2, q_3\}]\}$
3	$\{[p_0, \{q_0\}], [p_1, \{q_1, q_2\}], [p_1, \{q_0, q_2, q_3\}]\}$	$\emptyset$

At the third iteration, $\mathcal{W}$ becomes empty and hence the algorithm returns *true*. Therefore $L(A) \subseteq L(B)$.

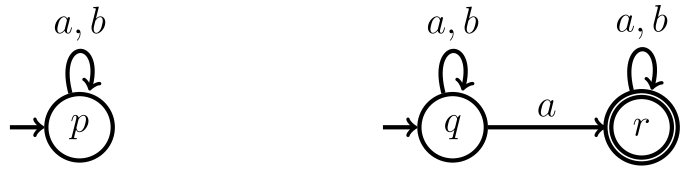
Input: NFAs $A = (Q, \Sigma, \delta, Q_0, F)$ and $A' = (Q', \Sigma, \delta', Q'_0, F')$.
Output: $L(A) = L(A')$?

$Q \leftarrow \emptyset$
$W \leftarrow \{[Q_0, Q'_0]\}$
while $W \neq \emptyset$ **do**
  pick $[P, P']$ **from** W
  if $(P \cap F = \emptyset) \neq (P' \cap F' = \emptyset)$ **then**
    return false
  for $a \in \Sigma$ **do**
    $q \leftarrow [\delta(P, a), \delta'(P', a)]$
    if $q \notin Q \wedge q \notin W$ **then**
      add q **to** W
return true

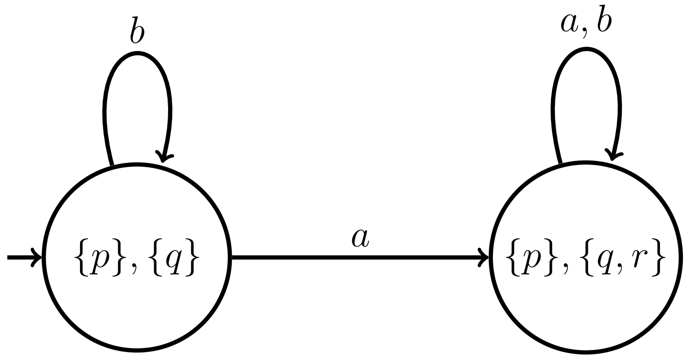
Solution 3.3

(a) We construct the pairing $[\mathit{NFAtoDFA}(A), \mathit{NFAtoDFA}(B)]$ on the fly. The algorithm returns *false* if it encounters a state $[P, P']$ such that only one of P and P' contains a final state. If no such state is encountered, the algorithm returns *true*.

(b) Let A and B be the following NFAs:



The pairing of A and B is as follows:



State $[\{p\}, \{q\}]$ does not allow us to conclude anything since both p and q are non final. However, state $[\{p\}, \{q, r\}]$, which is not minimal, allows us to conclude that $L(A) \neq L(B)$ since r is final.

(c) To show PSPACE-hardness, it suffices to give a reduction from NFA universality. Let A be an NFA. Let B the one state DFA that accepts Σ^*. The following holds:

$$L(A) = \Sigma^* \iff L(A) = L(B).$$

Therefore, $\langle A \rangle \mapsto \langle A, B \rangle$ is a reduction from NFA universality to NFA/DFA equality.

Solution 3.4

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

- $Q'' = Q \times Q' \times \{\top, \bot\}$,
- $q_0'' = (q_0, q_0', \top)$,
- $\delta(p, a) = \begin{cases} (\delta(q,a), q', \bot) & \text{if } p = (q, q', r) \text{ and } r = \top, \\ (q, \delta'(q', a), \top) & \text{if } p = (q, q', r) \text{ and } r = \bot, \end{cases}$
- $F'' = \{(q, q', \top) : q \in F \text{ and } q' \in F'\}$.

As for most constructions, 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 $L(C) = L(A) \mathbin{\widetilde{\sqcup\!\sqcup}} L(B)$.

$Q'' \leftarrow \emptyset$
$\delta'' \leftarrow \emptyset$
$F'' \leftarrow \emptyset$
$W \leftarrow \{(q_0, q_0', \top)\}$
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 3.5
Let $A = (Q, \Sigma, \delta, Q_0, F)$ be an NFA that accepts L.

(a) We add a ε-transition "parallel" to every transition of A. This simulates the deletion of letters from words of L. More formally, let $B = (Q, \Sigma, \delta', Q_0, F)$ be such that, for every $q \in Q$ and $a \in \Sigma \cup \{\varepsilon\}$,

$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{if } a \in \Sigma, \\ \{q \in Q : q \in \delta(q, b) \text{ for some } b \in \Sigma\} & \text{if } a = \varepsilon. \end{cases}$$

(b) For every state of Q, we add self-loops for each letter of Σ. This corresponds to the insertion of letters in words of L. More formally, let $B = (Q, \Sigma, \delta', Q_0, F)$ be such that $\delta'(q, a) = \delta(q, a) \cup \{q\}$ for every $q \in Q$ and $a \in \Sigma$.

(c) Intuitively, we construct an automaton B that guesses an intermediate state p and then reads w simultaneously from an initial state q_0 and from p. The automaton accepts if it simultaneously reaches p and and an accepting state q_F. More formally, let $B = (Q', \Sigma, \delta', Q_0', F')$ be such that

$$\begin{aligned} Q' &= Q \times Q \times Q, \\ Q_0' &= \{(p, q, p) : p \in Q, q \in Q_0\}, \\ F' &= \{(p, p, q) : p \in Q, q \in F\}, \end{aligned}$$

and, for every $p, q, r \in Q$ and $a \in \Sigma$,

$$\delta'((p, q, r), a) = \{(p, q', r') : q' \in \delta(q, a), r' \in \delta(r, a)\}.$$