Automata and Formal Languages — Exercise Sheet 7

Exercise 7.1
Let $\text{val} : \{0,1\}^* \to \mathbb{N}$ be the function that associates to every word $w \in \{0,1\}^*$ the number $\text{val}(w)$ represented by w in the *least significant bit first* encoding.

(a) Give a transducer that doubles numbers, i.e. a transducer accepting

$$L_1 = \{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \text{val}(y) = 2 \cdot \text{val}(x)\}\,.$$

(b) Give an algorithm that takes $k \in \mathbb{N}$ as input, and that produces a transducer A_k accepting

$$L_k = \left\{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \text{val}(y) = 2^k \cdot \text{val}(x)\right\}.$$

Hint: use (a) and consider operations seen in class.

(c) Give a transducer for the addition of two numbers, i.e. a transducer accepting

$$\{[x,y,z] \in (\{0,1\} \times \{0,1\} \times \{0,1\})^* \mid \text{val}(z) = \text{val}(x) + \text{val}(y)\}\,.$$

(d) For every $k \in \mathbb{N}_{>0}$, let

$$X_k = \{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \text{val}(y) = k \cdot \text{val}(x)\}\,.$$

Sketch an algorithm that takes as input transducers A and B, accepting respectively X_a and X_b for some $a, b \in \mathbb{N}_{>0}$, and that produces a transducer C accepting X_{a+b}.

(e) Let $k \in \mathbb{N}_{>0}$. Using (b) and (d), how can you build a transducer accepting X_k?

(f) Show that the following language has infinitely many residuals, and hence that it is not regular:

$$\left\{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \text{val}(y) = \text{val}(x)^2\right\}.$$

Exercise 7.2
Let $L_1 = \{bba, aba, bbb\}$ and $L_2 = \{aba, abb\}$.

(a) Give an algorithm for the following operation:

INPUT:	A fixed-length language $L \subseteq \Sigma^k$ described explicitly as a set of words.
OUTPUT:	State q of the master automaton over Σ such that $L(q) = L$.

(b) Use the previous algorithm to build the states of the master automaton for L_1 and L_2.

(c) Compute the state of the master automaton representing $L_1 \cup L_2$.

(d) Identify the kernels $\langle L_1 \rangle$, $\langle L_2 \rangle$, and $\langle L_1 \cup L_2 \rangle$.

Exercise 7.3
We define the *language* of a Boolean formula φ over variables $x_1, \ldots, x_n$ as:

$$\mathcal{L}(\varphi) = \{a_1 a_2 \cdots a_n \in \{0,1\}^n : \text{ the assignment } x_1 \mapsto a_1, \ldots, x_n \mapsto a_n \text{ satisfies } \varphi\}.$$

(a) Give a polynomial-time algorithm that takes as input a DFA A recognizing a language of length n, and returns a Boolean formula φ such that $\mathcal{L}(\varphi) = \mathcal{L}(A)$.

(b) Give an exponential-time algorithm that takes a Boolean formula φ as input, and returns a DFA A recognizing $\mathcal{L}(\varphi)$.

Solution 7.1

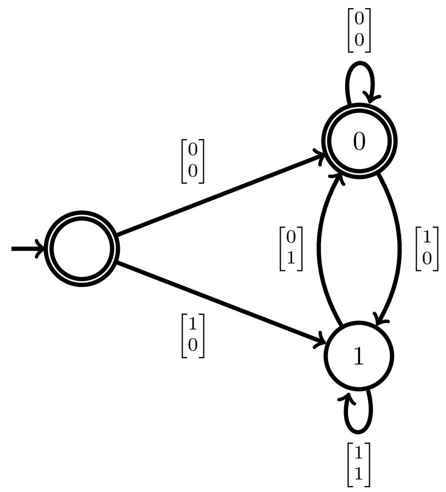
(a) Let $[x_1 x_2 \cdots x_n, y_1 y_2 \cdots y_n] \in (\{0,1\} \times \{0,1\})^n$ where $n \geq 2$. Multiplying a binary number by two shifts its bits and adds a zero. For example, the word

$$\begin{bmatrix}10110\\01011\end{bmatrix}$$

belongs to the language since it encodes $[13, 26]$. Thus, we have $\text{val}(y) = 2 \cdot \text{val}(x)$ if and only if $y_1 = 0$, $x_n = 0$, and $y_i = x_{i-1}$ for every $1 < i \leq n$. From this observation, we construct a transducer that

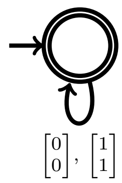
- tests whether the first bit of y is 0,
- tests whether y is consistent with x, by keeping the last bit of x in memory,
- accepts $[x, y]$ if the last bit of x is 0.

Note that words $[\varepsilon, \varepsilon]$ and $[0, 0]$ both encode the numerical values $[0, 0]$. Therefore, they should also be accepted since $2 \cdot 0 = 0$. We obtain the following transducer:



★ The initial state can be merged with state 0 as they have the same outgoing transitions.

(b) We construct A_0 as the following transducer accepting $\{[x, y] \in (\{0,1\} \times \{0,1\})^* : y = x\}$:



Let A_1 be the transducer obtained in (a). For every $k > 1$, we define $A_k = \mathit{Join}(A_{k-1}, A_1)$. A simple inductions show that $L(A_k) = L_k$ for every $k \in \mathbb{N}$.

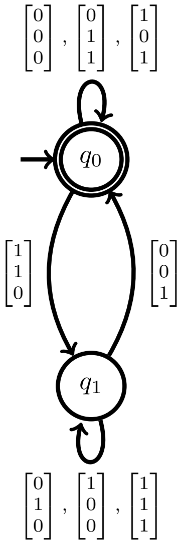
(c) We construct a transducer that computes the addition by keeping the current carry bit. Consider some tuple $[x, y, z] \in \{0,1\}^3$ and a carry bit r. Adding x, y and r leads to the bit

$$z = (x + y + r) \bmod 2. \tag{1}$$

Moreover, it yields a new carry bit r' such that $r' = 1$ if $x + y + r > 1$ and $r' = 0$ otherwise. The folllowing table identifies the new carry bit r' of the tuples that satisfy (1):

	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}$
$r = 0$	0	×	×	0	×	0	1	×
$r = 1$	×	0	1	×	1	×	×	1

We construct our transducer from the above table:



(d) We construct a transducer C that, intuitively, feeds its input to both A and B, and then feed the respective outputs of A and B to a transducer performing addition. More formally, let $A = (Q_A, \{0,1\}, \delta_A, q_{0A}, F_A)$, $B = (Q_B, \{0,1\}, \delta_B, q_{0B}, F_B)$, and let $D = (Q_D, \{0,1\}, \delta_D, q_{0D}, F_D)$ be the transducer for addition obtained in (c). We define C as $C = (Q_C, \{0,1\}, \delta_C, q_{0C}, F_C)$ where

- $Q_C = Q_A \times Q_B \times Q_D$,
- $q_{0C} = (q_{0A}, q_{0B}, q_{0D})$,
- $F_C = F_A \times F_B \times F_D$,

and

$$\delta_C((p,p',p''),[x,z]) = \{(q,q',q'') : \exists y,y' \in \{0,1\} \text{ s.t. } p \xrightarrow{[x,y]}_A q, p' \xrightarrow{[x,y']}_B q' \text{ and } p'' \xrightarrow{[y,y',z]}_D q''\}.$$

(e) Let $\ell = \lceil \log_2(k) \rceil$. There exist $c_0, c_1, \ldots, c_\ell \in \{0,1\}$ such that $k = c_0 \cdot 2^0 + c_1 \cdot 2^1 + \cdots + c_\ell \cdot 2^\ell$. Let $I = \{0 \leq i \leq \ell : c_i = 1\}$. Note that $k = \sum_{i \in I} 2^i$. Therefore, we may use transducer A_i from (b) for each $i \in I$, and combine these transducers using (d).

(f) For every $n \in \mathbb{N}_{>0}$, let

$$u_n = \begin{bmatrix} 0^n 1 \\ 0^n 0 \end{bmatrix} \text{ and } v_n = \begin{bmatrix} 0^{n-1} 0 \\ 0^{n-1} 1 \end{bmatrix}.$$

Let $i, j \in \mathbb{N}_{>0}$ be such that $i \neq j$. We claim that $L^{u_i} \neq L^{u_j}$. We have

$$u_i v_i = \begin{bmatrix} 0^i 1 0^i \\ 0^{2i} 1 \end{bmatrix} \text{ and } u_j v_i = \begin{bmatrix} 0^j 1 0^i \\ 0^{i+j} 1 \end{bmatrix}.$$

Therefore, $u_i v_i$ encodes $[2^i, 2^{2i}]$, and $u_i v_j$ encodes $[2^j, 2^{i+j}]$. We observe that $u_i v_i$ belongs to the language since $2^{2i} = (2^i)^2$. However, $u_j v_i$ does not belong to the language since $2^{i+j} \neq 2^{2j} = (2^j)^2$. $\square$

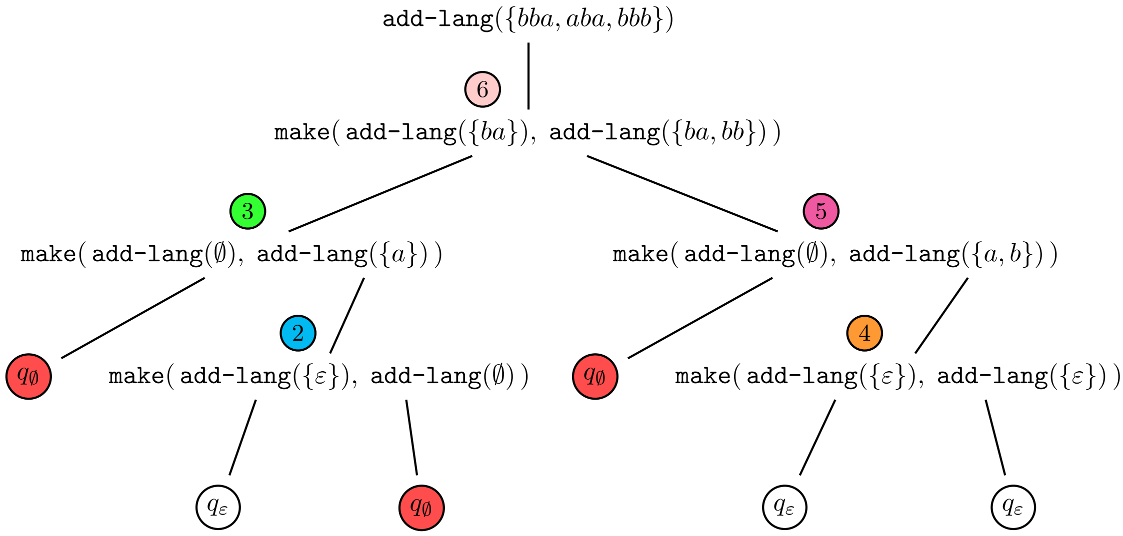
Solution 7.2

(a)

(b) Executing `add-lang`(L_1) yields the following computation tree:

Input: A fixed-length language $L \subseteq \Sigma^k$ described explicitely by a set of words.
Output: State q of the master automaton over Σ such that $L(q) = L$.

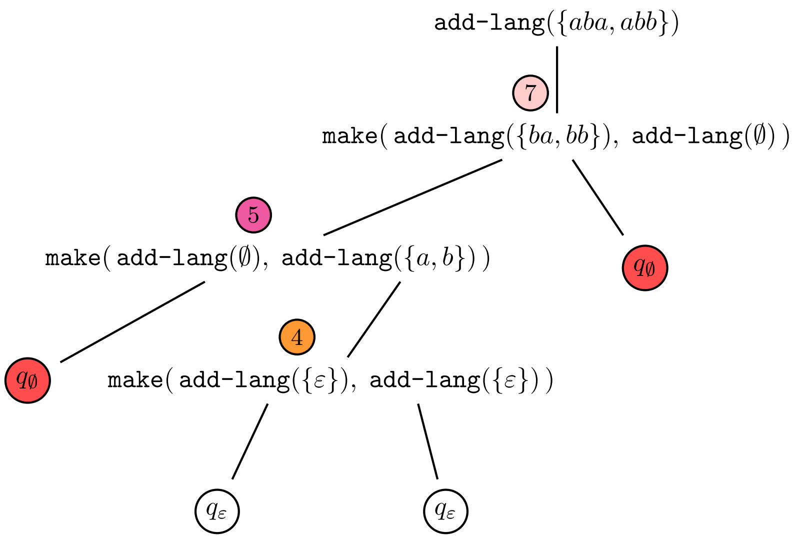
add-lang(L) :
    if $L = \emptyset$ **then**
        return $q_\emptyset$
    else if $L = \{\varepsilon\}$ **then**
        return q_ε
    else
        for $a_i \in \Sigma$ **do**
            $L^{a_i} \leftarrow \{u \mid a_i u \in L\}$
            $s_i \leftarrow$ add-lang(L^{a_i})
        return make($s_1, s_2, ..., s_n$)



The table obtained after the execution is as follows:

Ident.	a-succ	b-succ
2	q_ε	$q_\emptyset$
3	$q_\emptyset$	2
4	q_ε	q_ε
5	$q_\emptyset$	4
6	3	5

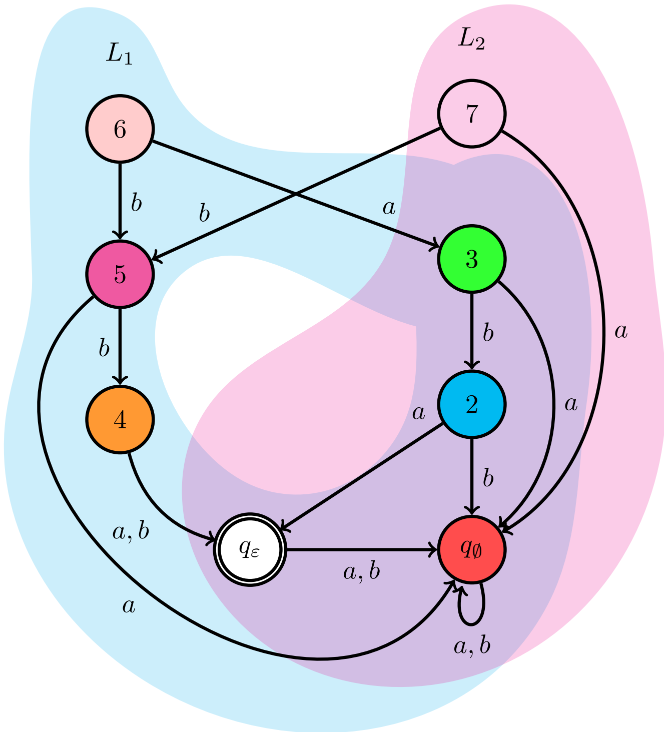
Executing add-lang(L_2) yields the following computation tree:



The table obtained after the execution is as follows:

Ident.	a-succ	b-succ
7	5	$q_\emptyset$
4	q_ε	q_ε
5	$q_\emptyset$	4

The resulting master automaton fragment is:

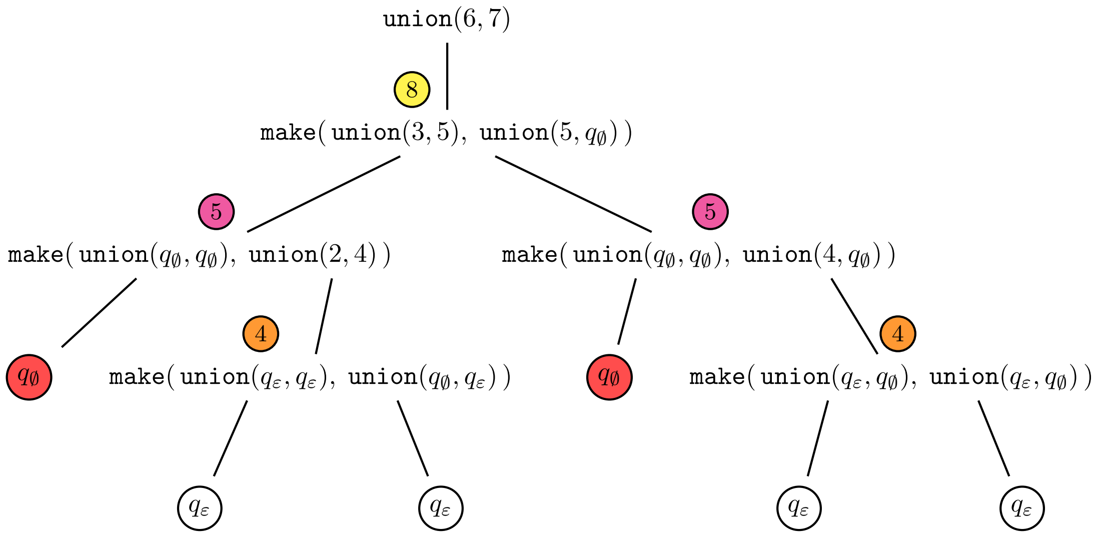


(c) Let us first adapt the algorithm for intersection to obtain an algorithm for union:

Input: States p and q of same length of the master automaton.
Output: State r of the master automaton such that $L(r) = L(p) \cup L(q)$.

`union`(p, q) :
  if $G(p, q)$ is not empty **then**
    return $G(p, q)$
  else if $p = q_\emptyset$ and $q = q_\emptyset$ **then**
    return $q_\emptyset$
  else if $p = q_\varepsilon$ or $q = q_\varepsilon$ **then**
    return q_ε
  else
    for $a_i \in \Sigma$ **do**
      $s_i \leftarrow$ `union`(p^{a_i}, q^{a_i})
    $G(p, q) \leftarrow$ `make`$(s_1, s_2, \ldots, s_n)$
    return $G(p, q)$

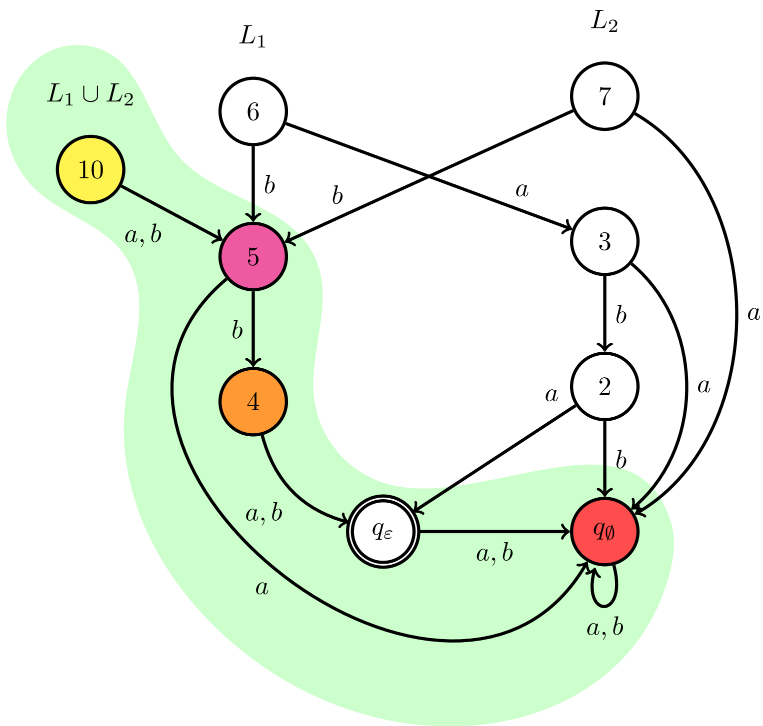
Executing `union`$(6,7)$ yields the following computation tree:



The table obtained after the execution is as follows:

Ident.	a-succ	b-succ
8	5	5
5	$q_\emptyset$	4
4	q_ε	q_ε

The new fragment of the master automaton is:



★ Note that `union` could be slightly improved by returning q whenever $p = q$, and by updating $G(q,p)$ at the same time as $G(p,q)$.

(d) The kernels are:

$$\langle L_1 \rangle = L_1,$$
$$\langle L_2 \rangle = L_2,$$
$$\langle L_1 \cup L_2 \rangle = \{ba, bb\}.$$

Solution 7.3

(a) The algorithm takes as input a state of the master automaton and the length of the language it recognizes, and recursively constructs a formula as follows:

Input: state q recognizing a language of length n
Output: formula φ_q such that $\mathcal{L}(\varphi_q) = \mathcal{L}(q)$

1 `DFAtoFormula`(q, n) *:*
2   **if** $G(q)$ is not empty **then**
3   **return** $G(q)$
4   **if** $q = q_\emptyset$ **then**
5   **return false**
6   **else if** $q = q_\epsilon$ **then**
7   **return true**
8   **else**
9         $\varphi_0 \leftarrow \mathit{DFAtoFormula}(q^0, n-1)$
10         $\varphi_1 \leftarrow \mathit{DFAtoFormula}(q^1, n-1)$
11         $\varphi_q \leftarrow (\neg x_1 \land \varphi_0) \lor (x_1 \land \varphi_1)$
12         $G(q) \leftarrow \varphi_q$
13   **return** $G(q)$

Observe that the parameter n is needed to identify the variable at line 11.

Our algorithm takes as input a table with the state identifiers and successors of all the descendants of q (i.e., the fragment of the master automaton starting at q). This is a polynomial time algorithm because we compute $\varphi_{q'}$ once for every descendant q' of q.

Note that this algorithm could be improved by adding an *else* that checks if $q^0 = q^1$ before the last else:

---

else if $q^0 = q^1$ **then**
    $\varphi \leftarrow \mathit{DFAtoFormula}(q^0, n-1)$
    $\varphi_q \leftarrow \varphi$
    $G(q) \leftarrow \varphi_q$
    return $G(q)$

(b) Given a formula φ over variables $x_1, \ldots, x_n$, we write $\varphi[x_i/\textbf{true}]$ and $\varphi[x_i/\textbf{false}]$ to denote the formulas obtained by replacing all occurrences of x_i in φ by **true** and **false**, respectively. We have that $\mathcal{L}(\varphi[x_1/\textbf{false}]) = \mathcal{L}(\varphi)^0$ and $\mathcal{L}(\varphi[x_1/\textbf{true}]) = \mathcal{L}(\varphi)^1$. This yields the following algorithm:

Input: formula φ over variables $x_1, \ldots, x_n$, total number of variables n, k initially equal to 1
Output: state q such that $L(\varphi) = L(q)$

`FormulatoDFA(`φ, n, k`)`*:*
    if $G(\varphi)$ is not empty **then**
        return $G(\varphi)$
    if $\varphi =$ **true then**
        return q_ε
    else if $\varphi =$ **false then**
        return $q_\emptyset$
    else
        $r_0 \leftarrow FormulatoDFA(\varphi[x_k/\textbf{false}], n, k+1)$
        $r_1 \leftarrow FormulatoDFA(\varphi[x_k/\textbf{true}], n, k+1)$
        $G(\varphi) \leftarrow make(r_0, r_1)$
        return $G(\varphi)$
