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

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

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:

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

Solution.

(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:

![Transducer Diagram]

[hard] 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 $\{[x, y] \in (\{0, 1\} \times \{0, 1\})^* : y = x\}$:

![Transducer Diagram]

Let $A_1$ be the transducer obtained in (a). For every $k > 1$, we define $A_k = \text{Join}(A_{k-1}, A_1)$. 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

$$z = (x + y + r) \mod 2.$$  \hspace{1cm} (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):

<table>
<thead>
<tr>
<th>$r$</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>1</td>
</tr>
</tbody>
</table>

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 $A = (Q_A, \{0, 1\}, \delta_A, q_0, F_A)$, $B = (Q_B, \{0, 1\}, \delta_B, q_0, F_B)$, and let $D = (Q_D, \{0, 1\}, \delta_D, q_0, F_D)$ be the transducer for addition obtained in (c). We define $C$ as $C = (Q_C, \delta_C, q_0, F_C)$ where

- $Q_C = Q_A \times Q_B \times Q_D$,
- $q_0 = (q_0, q_0, q_0)$,
- $F_C = F_A \times F_B \times F_D$, and

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 $k = c_0 \cdot 2^0 + c_1 \cdot 2^1 + \ldots + c_\ell \cdot 2^\ell$. 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

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

Let $i, j \in \mathbb{N}_0$ be such that $i \neq j$. We claim that $L^{n_i} \neq L^{n_j}$. We have

$$
u_i \nu_i = \begin{bmatrix} 0^i 10^i \\ 0^i 2^i \end{bmatrix} \quad \text{and} \quad \nu_j \nu_j = \begin{bmatrix} 0^j 10^j \\ 0^j 2^j \end{bmatrix}.
$$

Therefore, $u_i \nu_i$ encodes $[2^i, 2^i]$, and $u_i \nu_j$ encodes $[2^j, 2^{i+j}]$. We observe that $u_i \nu_i$ belongs to the language since $2^2 = (2^i)^2$. However, $u_j \nu_j$ does not belong to the language since $2^{i+j} \neq 2^j = (2^j)^2$. $\square$
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 $\Sigma$ 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 $\langle L_1 \cup L_2 \rangle$.

**Solution.**

(a) 

\[
\text{Input: A fixed-length language } L \subseteq \Sigma^k \text{ described explicitly by a set of words.}
\]

\[
\text{Output: State } q \text{ of the master automaton over } \Sigma \text{ such that } L(q) = L.
\]

\[
\text{Algorithm:}
\]

\[
\text{add-lang}(L) :=
\]

\[
\begin{align*}
\text{if } L = \emptyset & \text{ then } \text{ return } q_\emptyset \\
\text{else if } L = \{\varepsilon\} & \text{ then } \text{ return } q_\varepsilon \\
\text{else } & \text{ for } a_i \in \Sigma \text{ do } \\
& \quad L^{a_i} \leftarrow \{u \mid a_i u \in L\} \\
& \quad s_i \leftarrow \text{add-lang}(L^{a_i}) \\
& \text{ return make}(s_1, s_2, \ldots, s_n)
\end{align*}
\]

(b) Executing $\text{add-lang}(L_1)$ yields the following computation tree:

\[
\text{add-lang(}\{bba, aba, bbb\})
\]

\[
\text{make(}\text{add-lang(}\{ba\}), \text{add-lang(}\{ba, bb\}))
\]

\[
\text{make(}\text{add-lang(}\emptyset\)), \text{add-lang(}\{a\})\)
\]

\[
\text{make(}\text{add-lang(}\varepsilon\)), \text{add-lang(}\emptyset\))
\]

\[
\text{make(}\text{add-lang(}\varepsilon\)), \text{add-lang(}\{a, b\})\)
\]

The table obtained after the execution is as follows:
Executing add-lang($L_2$) yields the following computation tree:

```
add-lang(\{aba, abb\})
```

```
make(add-lang(\{ba, bb\}), add-lang(\{}))
```

```
make(add-lang(\{}), add-lang(\{a, b\}))
```

```
make(add-lang(\{}), add-lang(\{\}))
```

The table obtained after the execution is as follows:

<table>
<thead>
<tr>
<th>Ident.</th>
<th>$a$-succ</th>
<th>$b$-succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>$q_0$</td>
</tr>
<tr>
<td>4</td>
<td>$q_e$</td>
<td>$q_e$</td>
</tr>
<tr>
<td>5</td>
<td>$q_0$</td>
<td>4</td>
</tr>
</tbody>
</table>

The resulting master automaton fragment is:
(c) Let us first adapt the algorithm for intersection to obtain an algorithm for union:

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

1. \text{union}(p, q) :
2. \text{if } G(p, q) \text{ is not empty then}
3. \quad \text{return } G(p, q)
4. \text{else if } p = q_0 \text{ and } q = q_0 \text{ then}
5. \quad \text{return } q_0
6. \text{else if } p = q_e \text{ or } q = q_e \text{ then}
7. \quad \text{return } q_e
8. \text{else}
9. \quad \text{for } a_i \in \Sigma \text{ do}
10. \quad \quad s_i \leftarrow \text{union}(p^{a_i}, q^{a_i})
11. \quad \quad G(p, q) \leftarrow \text{make}(s_1, s_2, ..., s_n)
12. \quad \text{return } G(p, q)

Executing \text{union}(6, 7) yields the following computation tree:
The table obtained after the execution is as follows:

<table>
<thead>
<tr>
<th>Ident.</th>
<th>(a)-succ</th>
<th>(b)-succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>(q_0)</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>(q_\varepsilon)</td>
<td>(q_\varepsilon)</td>
</tr>
</tbody>
</table>

The new fragment of the master automaton is:

[hard] Note that \texttt{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:
\[ \langle L_1 \rangle = L_1, \]
\[ \langle L_2 \rangle = L_2, \]
\[ \langle L_1 \cup L_2 \rangle = \{ba, bb\}. \]

Exercise 7.3.

We define the *language* of a Boolean formula \( \varphi \) over variables \( x_1, \ldots, x_n \) as:
\[ L(\varphi) = \{a_1a_2 \cdots a_n \in \{0, 1\}^n : \text{the assignment } x_1 \mapsto a_1, \ldots, x_n \mapsto a_n \text{ satisfies } \varphi\}. \]

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

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

**Solution.**

(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:

<table>
<thead>
<tr>
<th>Input: state ( q ) recognizing a language of length ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output:</strong> formula ( \varphi_q ) such that ( L(\varphi_q) = L(q) )</td>
</tr>
<tr>
<td>1 DFAtoFormula((q, n)):</td>
</tr>
<tr>
<td>2 \text{ if } G(q) \text{ is not empty then}</td>
</tr>
<tr>
<td>3 \quad \text{ return } G(q)</td>
</tr>
<tr>
<td>4 \text{ if } q = q_0 \text{ then}</td>
</tr>
<tr>
<td>5 \quad \text{ return false}</td>
</tr>
<tr>
<td>6 \text{ else if } q = q_\epsilon \text{ then}</td>
</tr>
<tr>
<td>7 \quad \text{ return true}</td>
</tr>
<tr>
<td>8 \text{ else}</td>
</tr>
<tr>
<td>9 \quad \varphi_0 \leftarrow \text{DFAtoFormula}(q^0, n - 1)</td>
</tr>
<tr>
<td>10 \quad \varphi_1 \leftarrow \text{DFAtoFormula}(q^1, n - 1)</td>
</tr>
<tr>
<td>11 \quad \varphi_q \leftarrow (\neg x_1 \land \varphi_0) \lor (x_1 \land \varphi_1)</td>
</tr>
<tr>
<td>12 \quad G(q) \leftarrow \varphi_q</td>
</tr>
<tr>
<td>13 \text{ return } G(q)</td>
</tr>
</tbody>
</table>

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
\[
\varphi \leftarrow \text{DFAtoFormula}(q^0, n - 1)
\]
\[
\varphi_q \leftarrow \varphi
\]
\[
G(q) \leftarrow \varphi_q
\]
return \( G(q) \)

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

---

**Input:** formula \( \varphi \) over variables \( x_1, \ldots, x_n \), total number of variables \( n \), \( k \) initially equal to 1

**Output:** state \( q \) such that \( L(\varphi) = L(q) \)

1. **FormulatoDFA**\( (\varphi, n, k) \):
2. if \( G(\varphi) \) is not empty then
   return \( G(\varphi) \)
3. if \( \varphi = \text{true} \) then
   return \( q_\epsilon \)
4. else if \( \varphi = \text{false} \) then
   return \( q_\emptyset \)
5. else
   \( r_0 \leftarrow \text{FormulatoDFA}(\varphi[x_k/\text{false}], n, k + 1) \)
6. \( r_1 \leftarrow \text{FormulatoDFA}(\varphi[x_k/\text{true}], n, k + 1) \)
7. \( G(\varphi) \leftarrow \text{make}(r_0, r_1) \)
8. return \( G(\varphi) \)