Exercise 5.1
Let $L_1 = \{baa, aaa, bab\}$ and $L_2 = \{baa, aab\}$.

(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 $\langle L_1 \cup L_2 \rangle$.

Exercise 5.2

(a) Give an recursive algorithm for the following operation:

**INPUT:** States $p$ and $q$ of the master automaton.

**OUTPUT:** State $r$ of the master automaton such that $L(r) = L(p) \cdot L(q)$.

Observe that the languages $L(p)$ and $L(q)$ can have different lengths. Try to reduce the problem for $p$, $q$ to the problem for $p^a$, $q$.

(b) Give an recursive algorithm for the following operation:

**INPUT:** A state $q$ of the master automaton.

**OUTPUT:** State $r$ of the master automaton such that $L(r) = L(q)^R$

where $R$ is the reverse operator.

(c) A *coding* over an alphabet $\Sigma$ is a function $h : \Sigma \mapsto \Sigma$. A coding $h$ can naturally be extended to a morphism over words, i.e. $h(\epsilon) = \epsilon$ and $h(w) = h(w_1)h(w_2)\cdots h(w_n)$ for every $w \in \Sigma^n$. Give an algorithm for the following operation:

**INPUT:** A state $q$ of the master automaton and a coding $h$.

**OUTPUT:** State $r$ of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

Can you make your algorithm more efficient when $h$ is a permutation?

Exercise 5.3
Let $k \in \mathbb{N}_{>0}$. Let $\text{flip} : \{0, 1\}^k \rightarrow \{0, 1\}^k$ be the function that inverts the bits of its input, e.g. $\text{flip}(010) = 101$.
Let $\text{val} : \{0, 1\}^k \rightarrow \mathbb{N}$ be such that $\text{val}(w)$ is the number represented by $w$ in the least significant bit first encoding.

(a) Describe the minimal transducer that accepts

$$L_k = \{[x, y] \in (\{0, 1\} \times \{0, 1\})^k \mid \text{val}(y) = \text{val}(\text{flip}(x)) + 1 \mod 2^k\}.$$ 

(b) Build the state $r$ of the master transducer for $L_3$, and the state $q$ of the master automaton for $\{010, 110\}$.

(c) Adapt the algorithm $\text{pre}$ seen in class to compute $\text{post}$ and compute using this algorithm $\text{post}(r, q)$. 
Solution 5.1

(a) 

**Input:** A fixed-length language $L \subseteq \Sigma^k$ described explicitly by a set of words.

**Output:** State $q$ of the master automaton over $\Sigma$ such that $L(q) = L$.

1. $\text{add-lang}(L)$:
2. if $L = \emptyset$ then
3. return $q_\emptyset$
4. else if $L = \{\varepsilon\}$ then
5. return $q_\varepsilon$
6. else
7. for $a_i \in \Sigma$ do
8. $L^{a_i} \leftarrow \{u \mid a_i u \in L\}$
9. $s_i \leftarrow \text{add-lang}(L^{a_i})$
10. return $\text{make}(s_1, s_2, \ldots, s_n)$

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

![Computation tree](image)

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>2</td>
<td>$q_\varepsilon$</td>
<td>$q_\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>$q_\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>$q_\varepsilon$</td>
<td>$q_\varepsilon$</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>$q_\emptyset$</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Calling $\text{add-lang}(L_2)$ adds the following rows to the table and returns 9:

<table>
<thead>
<tr>
<th>Ident.</th>
<th>$a$-succ</th>
<th>$b$-succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>$q_\emptyset$</td>
<td>$q_\varepsilon$</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>$q_\emptyset$</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>3</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. if \( G(p, q) \) is not empty then
   3. \hspace{1em} return \( G(p, q) \)
   4. else if \( p = q_{\emptyset} \) and \( q = q_{\emptyset} \) then
   5. \hspace{1em} return \( q_{\emptyset} \)
   6. else if \( p = q_{\varepsilon} \) or \( q = q_{\varepsilon} \) then
   7. \hspace{1em} return \( q_{\varepsilon} \)
   8. else
   9. \hspace{1em} for \( a_i \in \Sigma \) do
   10. \hspace{2em} \( s_i \leftarrow \text{union}(p^{a_i}, q^{a_i}) \)
   11. \hspace{1em} \( G(p, q) \leftarrow \text{make}(s_1, s_2, \ldots, s_n) \)
   12. return \( G(p, q) \)
Executing $\text{union}(6, 9)$ yields the following computation tree:

Calling $\text{union}(6, 9)$ adds the following row to the table and returns 10:

<table>
<thead>
<tr>
<th>Ident.</th>
<th>a-succ</th>
<th>b-succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

The new fragment of the master automaton is:

★ Note that $\text{union}$ 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:

\[
\langle L_1 \rangle = L_1,
\langle L_2 \rangle = L_2,
\langle L_1 \cup L_2 \rangle = \{aa, ab\}.
\]

**Solution 5.2**

(a) Let \( L \) and \( L' \) be fixed-length languages. The following holds:

\[
L \cdot L' = \begin{cases} 
\emptyset & \text{if } L = \emptyset, \\
L' & \text{if } L = \{\varepsilon\}, \\
\bigcup_{a \in \Sigma} \{a\} \cdot L \cdot L' & \text{otherwise.}
\end{cases}
\]

These identities give rise to the following algorithm:

| Input: States \( p \) and \( q \) of the master automaton. |
| Output: State \( r \) of the master automaton such that \( L(r) = L(p) \cdot L(q) \). |
| concat\( (p, q) \): |
| if \( G(p, q) \) is not empty then |
| return \( G(p, q) \) |
| else if \( p = q\emptyset \) then |
| return \( q\emptyset \) |
| else if \( p = q\varepsilon \) then |
| return \( q \) |
| else |
| for \( a_i \in \Sigma \) do |
| \( s_i \leftarrow \) concat\( (p^{a_i}, q) \) |
| \( G(p, q) \leftarrow \) make\( (s_1, s_2, \ldots, s_n) \) |
| return \( G(p, q) \) |

(b) Let \( L \) be a fixed-length language. The following holds:

\[
L^R = \begin{cases} 
\emptyset & \text{if } L = \emptyset, \\
\{\varepsilon\} & \text{if } L = \{\varepsilon\}, \\
\bigcup_{a \in \Sigma} (L^R)^a \cdot \{a\} & \text{otherwise.}
\end{cases}
\]

These identities give rise to the following algorithm:

★ Note that Lines 11 and 12 are introduced in order to represent the language \( \{a_i\} \) in Line 13 as a state \( \text{make}(s_1, s_2, \ldots, s_n) \) of the master automaton. This can be avoided by using the algorithm from Exercise 8.1, namely the state that represents \( \{a_i\} \) is \( \text{add-lang}(\{a_i\}) \). Thus, Lines 11-13 can be replaced just by \( r \leftarrow \text{concat}(\text{reverse}(q^{a_i}), \text{add-lang}(\{a_i\})) \)
Input: A state $q$ of the master automaton.
Output: State $r$ of the master automaton such that $L(r) = L(q)^R$.

```plaintext
reverse(q):
    if $G(q)$ is not empty then
        return $G(q)$
    else if $q = q_0$ then
        return $q_0$
    else if $q = q_\epsilon$ then
        return $q_\epsilon$
    else
        $p \leftarrow q_\emptyset$
        for $a_i \in \Sigma$ do
            $s_i \leftarrow q_\epsilon$
            $s_j \leftarrow q_0$ for every $i \neq j$
            $r \leftarrow \text{concat}(reverse(q^{a_i}), \text{make}(s_1, s_2, \ldots, s_n))$
        $p \leftarrow \text{union}(p, r)$
        $G(q) \leftarrow p$
    return $G(q)$
```

(c) Let $L$ be a fixed-length language and let $h$ be a coding. The following holds:

$$h(L) = \begin{cases} 
\emptyset & \text{if } L = \emptyset, \\
\{\epsilon\} & \text{if } L = \{\epsilon\}, \\
\bigcup_{a \in \Sigma} h(a) \cdot h(L^a) & \text{otherwise.}
\end{cases}$$

These identities give rise to the following algorithm:

Input: A state $q$ of the master automaton and a coding $h$.
Output: State $r$ of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

```plaintext
coding(q, h):
    if $G(q)$ is not empty then
        return $G(q)$
    else if $q = q_0$ then
        return $q_0$
    else if $q = q_\epsilon$ then
        return $q_\epsilon$
    else
        $p \leftarrow q_\emptyset$
        for $a \in \Sigma$ do
            $r \leftarrow \text{coding}(q^a, h)$
            $s_{h(a)} \leftarrow r$
            $s_b \leftarrow q_0$ for every $b \neq h(a)$
            $p \leftarrow \text{union}(p, \text{make}(s))$
        $G(q) \leftarrow p$
    return $G(q)$
```

The above algorithm makes use of `union` because the coding may be the same for distinct letters, i.e. $h(a) = h(b)$ for $a \neq b$ is possible. However, if the coding is a permutation, then this is not possible, and thus each letter maps to a unique residual. Therefore, the algorithm can be adapted as follows:
Input: A state $q$ of the master automaton and a coding $h$ which is a permutation.
Output: State $r$ of the master automaton such that $L(r) = \{h(w) : w \in L(q)\}$.

```
coding-permutation(q, h):
  1 if $G(q)$ is not empty then
  2   return $G(q)$
  3 else if $q = q_0$ then
  4     return $q_0$
  5 else if $q = q_\varepsilon$ then
  6     return $q_\varepsilon$
  7 else
  8     for $a \in \Sigma$ do
  9       $s_{h(a)} \leftarrow$ coding-permutation($q^a, h$)
 10     $G(q) \leftarrow$ make($s$)
 11     return $G(q)$
```

Solution 5.3

(a) Let $[x, y] \in L_k$. We may flip the bits of $x$ at the same time as adding 1. If $x_1 = 1$, then $\neg x_1 = 0$, and hence adding 1 to val(flip($x$)) results in $y_1 = 1$. Thus, for every $1 < i \leq k$, we have $y_i = \neg x_i$. If $x_1 = 0$, then $\neg x_1 = 1$. Adding 1 yields $y_1 = 0$ with a carry. This carry is propagated as long as $\neg x_i = 1$, and thus as long as $x_i = 0$. If some position $j$ with $x_j = 1$ is encountered, the carry is “consumed”, and we flip the remaining bits of $x$. These observations give rise to the following minimal transducer for $L_k$:

(b) The minimal transducer accepting $L_3$ is

State 4 of the following master automaton fragment accepts $\{010, 110\}$:
(c) We can establish the following identities similar to those obtained for pre:

\[
post_R(L) = \begin{cases} 
\emptyset & \text{if } R = \emptyset \text{ or } L = \emptyset, \\
\{\varepsilon\} & \text{if } R = \{[\varepsilon,\varepsilon]\} \text{ and } L = \{\varepsilon\}, \\
\bigcup_{a,b \in \Sigma} b \cdot post_{R[a,b]}(L^a) & \text{otherwise.}
\end{cases}
\]

To see that these identities hold, let \(b \in \Sigma\) and \(v \in \Sigma^k\) for some \(k \in \mathbb{N}\). We have,

\[
bv \in post_R(L) \iff \exists a \in \Sigma, u \in \Sigma^k \text{ s.t. } au \in L \text{ and } [au,bv] \in R \\
\iff \exists a \in \Sigma, u \in L^a \text{ s.t. } [au,bv] \in R \\
\iff \exists a \in \Sigma, u \in L^a \text{ s.t. } [u,v] \in R^{[a,b]} \\
\iff \exists a \in \Sigma \text{ s.t. } v \in Post_{R[a,b]}(L^a) \\
\iff v \in \bigcup_{a \in \Sigma} Post_{R[a,b]}(L^a) \\
\iff bv \in \bigcup_{a \in \Sigma} b \cdot Post_{R[a,b]}(L^a).
\]

We obtain the following algorithm:

Input: A state \(r\) of the master transducer and a state \(q\) of the master automaton.

Output: State \(p\) of the master automaton such that \(L(p) = Post_R(L)\) where \(R = L(r)\) and \(L = L(q)\).

1. \(post(r,q)\):
2. \quad if \(G(r,q)\) is not empty then
3. \quad \quad return \(G(r,q)\)
4. \quad else if \(r = r_\emptyset\) or \(q = q_\emptyset\) then
5. \quad \quad return \(q_\emptyset\)
6. \quad else if \(r = r_\varepsilon\) and \(q = q_\varepsilon\) then
7. \quad \quad return \(q_\varepsilon\)
8. \quad else
9. \quad \quad for \(b_i \in \Sigma\) do
10. \quad \quad \quad \(p \leftarrow q_\emptyset\)
11. \quad \quad \quad for \(a \in \Sigma\) do
12. \quad \quad \quad \quad \(p \leftarrow \text{union}(p, post(r[a,b], q^a))\)
13. \quad \quad \quad \(s_i \leftarrow p\)
14. \quad \quad \(G(q,r) \leftarrow \text{make}(s_1, s_2, \ldots, s_n)\)
15. \quad return \(G(q,r)\)
Note that the transducer for $L_3$ has some “strong” deterministic property. Indeed, for every state $r$ and $b \in \{0, 1\}$, if $r[a,b] \neq r_0$ then $r[\neg a,b] = r_0$. Hence, for a fixed $b \in \{0, 1\}$, at most one term of the form “$\text{post}(r[a,b], q^a)$” can differ from $q_0$ at line 12 of the algorithm. Thus, unions made by the algorithm on this transducer are trivial, and executing $\text{post}(6, 4)$ yields the following computation tree:

![Computation Tree](image-url)

Calling $\text{post}(6, 4)$ adds the following rows to the master automaton table and returns 8:

<table>
<thead>
<tr>
<th>Ident.</th>
<th>0-succ</th>
<th>1-succ</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$q_0$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>6</td>
<td>$q_0$</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>$q_0$</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

The resulting master automaton fragment: