Chapter 2

Finite Automata and Regular Languages

In this chapter, we introduce and analyze the class of languages that are known as regular languages. Informally, these languages can be “processed” by computers having a very small amount of memory.

2.1 An example: Controling a toll gate

Before we give a formal definition of a finite automaton, we consider an example in which such an automaton shows up in a natural way. We consider the problem of designing a “computer” that controls a toll gate.

When a car arrives at the toll gate, the gate is closed. The gate opens as soon as the driver has payed 25 cents. We assume that we have only three coin denominations: 5, 10, and 25 cents. We also assume that no excess change is returned.

After having arrived at the toll gate, the driver inserts a sequence of coins into the machine. At any moment, the machine has to decide whether or not to open the gate, i.e., whether or not the driver has paid 25 cents (or more). In order to decide this, the machine is in one of the following six states, at any moment during the process:

- The machine is in state $q_0$, if it has not collected any money yet.
- The machine is in state $q_1$, if it has collected exactly 5 cents.
- The machine is in state $q_2$, if it has collected exactly 10 cents.
• The machine is in state $q_3$, if it has collected exactly 15 cents.
• The machine is in state $q_4$, if it has collected exactly 20 cents.
• The machine is in state $q_5$, if it has collected 25 cents or more.

Initially (when a car arrives at the toll gate), the machine is in state $q_0$. Assume, for example, that the driver presents the sequence $(10,5,5,10)$ of coins.

• After receiving the first 10 cents coin, the machine switches from state $q_0$ to state $q_2$.
• After receiving the first 5 cents coin, the machine switches from state $q_2$ to state $q_3$.
• After receiving the second 5 cents coin, the machine switches from state $q_3$ to state $q_4$.
• After receiving the second 10 cents coin, the machine switches from state $q_4$ to state $q_5$. At this moment, the gate opens. (Remember that no change is given.)

The figure below represents the behavior of the machine for all possible sequences of coins. State $q_5$ is represented by two circles, because it is a special state: As soon as the machine reaches this state, the gate opens.

Observe that the machine (or computer) only has to remember which state it is in at any given time. Thus, it needs only a very small amount of memory: It has to be able to distinguish between any one of six possible cases and, therefore, it only needs a memory of $\lceil \log 6 \rceil = 3$ bits.
2.2 Deterministic finite automata

Let us look at another example. Consider the following state diagram:

We say that $q_1$ is the start state and $q_2$ is an accept state. Consider the input string 1101. This string is processed in the following way:

- Initially, the machine is in the start state $q_1$.
- After having read the first 1, the machine switches from state $q_1$ to state $q_2$.
- After having read the second 1, the machine switches from state $q_2$ to state $q_2$. (So actually, it does not switch.)
- After having read the first 0, the machine switches from state $q_2$ to state $q_3$.
- After having read the third 1, the machine switches from state $q_3$ to state $q_2$.

After the entire string 1101 has been processed, the machine is in state $q_2$, which is an accept state. We say that the string 1101 is accepted by the machine.

Consider now the input string 0101010. After having read this string from left to right (starting in the start state $q_1$), the machine is in state $q_3$. Since $q_3$ is not an accept state, we say that the machine rejects the string 0101010.

We hope you are able to see that this machine accepts every binary string that ends with a 1. In fact, the machine accepts more strings:

- Every binary string having the property that there are an even number of 0s following the rightmost 1, is accepted by this machine.
• Every other binary string is rejected by the machine. Observe that each such string is either empty, consists of 0s only, or has an odd number of 0s following the rightmost 1.

We now come to the formal definition of a finite automaton:

**Definition 2.2.1** A finite automaton is a 5-tuple \( M = (Q, \Sigma, \delta, q, F) \), where

1. \( Q \) is a finite set, whose elements are called states,
2. \( \Sigma \) is a finite set, called the alphabet; the elements of \( \Sigma \) are called symbols,
3. \( \delta : Q \times \Sigma \to Q \) is a function, called the transition function,
4. \( q \) is an element of \( Q \); it is called the start state,
5. \( F \) is a subset of \( Q \); the elements of \( F \) are called accept states.

You can think of the transition function \( \delta \) as being the “program” of the finite automaton \( M = (Q, \Sigma, \delta, q, F) \). This function tells us what \( M \) can do in “one step”:

• Let \( r \) be a state of \( Q \) and let \( a \) be a symbol of the alphabet \( \Sigma \). If the finite automaton \( M \) is in state \( r \) and reads the symbol \( a \), then it switches from state \( r \) to state \( \delta(r, a) \). (In fact, \( \delta(r, a) \) may be equal to \( r \).)

The “computer” that we designed in the toll gate example in Section 2.1 is a finite automaton. For this example, we have \( Q = \{q_0, q_1, q_2, q_3, q_4, q_5\} \), \( \Sigma = \{5, 10, 25\} \), the start state is \( q_0 \), \( F = \{q_5\} \), and \( \delta \) is given by the following table:

<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>10</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_1 )</td>
<td>( q_2 )</td>
<td>( q_5 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_2 )</td>
<td>( q_3 )</td>
<td>( q_5 )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_3 )</td>
<td>( q_4 )</td>
<td>( q_5 )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( q_4 )</td>
<td>( q_5 )</td>
<td>( q_5 )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( q_5 )</td>
<td>( q_5 )</td>
<td>( q_5 )</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>( q_5 )</td>
<td>( q_5 )</td>
<td>( q_5 )</td>
</tr>
</tbody>
</table>

The example given in the beginning of this section is also a finite automaton. For this example, we have \( Q = \{q_1, q_2, q_3\} \), \( \Sigma = \{0, 1\} \), the start state is \( q_1 \), \( F = \{q_2\} \), and \( \delta \) is given by the following table:
### 2.2. Deterministic finite automata

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>$q_1$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_2$</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

Let us denote this finite automaton by $M$. The language of $M$, denoted by $L(M)$, is the set of all binary strings that are accepted by $M$. As we have seen before, we have

$$L(M) = \{ w : w \text{ contains at least one 1 and ends with an even number of 0s} \}.$$  

We now give a formal definition of the language of a finite automaton:

**Definition 2.2.2** Let $M = (Q, \Sigma, \delta, q, F)$ be a finite automaton and let $w = w_1 w_2 \ldots w_n$ be a string over $\Sigma$. Define the sequence $r_0, r_1, \ldots, r_n$ of states, in the following way:

- $r_0 = q$;
- $r_{i+1} = \delta(r_i, w_{i+1})$, for $i = 0, 1, \ldots, n - 1$.

1. If $r_n \in F$, then we say that $M$ accepts $w$.
2. If $r_n \notin F$, then we say that $M$ rejects $w$.

In this definition, $w$ may be the *empty string*, which we denote by $\epsilon$, and whose length is zero; thus in the definition above, $n = 0$. In this case, the sequence $r_0, r_1, \ldots, r_n$ of states has length one; it consists of just the state $r_0 = q$. The empty string is accepted by $M$ if and only if the start state $q$ belongs to $F$.

**Definition 2.2.3** Let $M = (Q, \Sigma, \delta, q, F)$ be a finite automaton. The language $L(M)$ accepted by $M$ is defined to be the set of all strings that are accepted by $M$:

$$L(M) = \{ w : w \text{ is a string over } \Sigma \text{ and } M \text{ accepts } w \}.$$  

**Definition 2.2.4** A language $A$ is called *regular*, if there exists a finite automaton $M$ such that $A = L(M)$. 

We finish this section by presenting an equivalent way of defining the language accepted by a finite automaton. Let $M = (Q, \Sigma, \delta, q, F)$ be a finite automaton. The transition function $\delta : Q \times \Sigma \rightarrow Q$ tells us that, when $M$ is in state $r \in Q$ and reads symbol $a \in \Sigma$, it switches from state $r$ to state $\delta(r, a)$. Let $\Sigma^*$ denote the set of all strings over the alphabet $\Sigma$. ($\Sigma^*$ includes the empty string $\epsilon$.) We extend the function $\delta$ to a function $\overline{\delta} : Q \times \Sigma^* \rightarrow Q$, that is defined as follows. For any state $r \in Q$ and for any string $w$ over the alphabet $\Sigma$,

$$
\overline{\delta}(r, w) = \begin{cases} 
  r & \text{if } w = \epsilon, \\
  \delta(\overline{\delta}(r, v), a) & \text{if } w = va, \text{ where } v \text{ is a string and } a \in \Sigma.
\end{cases}
$$

What is the meaning of this function $\overline{\delta}$? Let $r$ be a state of $Q$ and let $w$ be a string over the alphabet $\Sigma$. Then

- $\overline{\delta}(r, w)$ is the state that $M$ reaches, when it starts in state $r$, reads the string $w$ from left to right, and uses $\delta$ to switch from state to state.

Using this notation, we have

$$L(M) = \{w : w \text{ is a string over } \Sigma \text{ and } \overline{\delta}(q, w) \in F\}.$$ 

### 2.2.1 A first example of a finite automaton

Let

$$A = \{w : w \text{ is a binary string containing an odd number of 1s}\}.$$ 

We claim that this language $A$ is regular. In order to prove this, we have to construct a finite automaton $M$ such that $A = L(M)$.

How to construct $M$? Here is a first idea: The finite automaton reads the input string $w$ from left to right and keeps track of the number of 1s it has seen. After having read the entire string $w$, it checks whether this number is odd (in which case $w$ is accepted) or even (in which case $w$ is rejected). Using this approach, the finite automaton needs a state for every integer $i \geq 0$, indicating that the number of 1s read so far is equal to $i$. Hence, to design a finite automaton that follows this approach, we need an infinite
2.2. Deterministic finite automata

number of states. But, the definition of finite automaton requires the number of states to be finite.

A better, and correct approach, is to keep track of whether the number of 1s read so far is even or odd. This leads to the following finite automaton:

- The set of states is $Q = \{q_e, q_o\}$. If the finite automaton is in state $q_e$, then it has read an even number of 1s; if it is in state $q_o$, then it has read an odd number of 1s.

- The alphabet is $\Sigma = \{0, 1\}$.

- The start state is $q_e$, because at the start, the number of 1s read by the automaton is equal to 0, and 0 is even.

- The set $F$ of accept states is $F = \{q_o\}$.

- The transition function $\delta$ is given by the following table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_e$</td>
<td>$q_e$</td>
<td>$q_o$</td>
</tr>
<tr>
<td>$q_o$</td>
<td>$q_o$</td>
<td>$q_e$</td>
</tr>
</tbody>
</table>

This finite automaton $M = (Q, \Sigma, \delta, q_e, F)$ can also be described by its state diagram, which is given in the figure below. The arrow that comes “out of the blue” and enters the state $q_e$, indicates that $q_e$ is the start state. The state depicted with double circles indicates the accept state.

We have constructed a finite automaton $M$ that accepts the language $A$. Therefore, $A$ is a regular language.
2.2.2 A second example of a finite automaton

Define the language $A$ as

$$A = \{ w : w \text{ is a binary string containing 101 as a substring} \}.$$ 

Again, we claim that $A$ is a regular language. In other words, we claim that there exists a finite automaton $M$ that accepts $A$, i.e., $A = L(M)$.

The finite automaton $M$ will do the following, when reading an input string from left to right:

- It skips over all 0s, and stays in the start state.
- At the first 1, it switches to the state “maybe the next two symbols are 01”.
  - If the next symbol is 1, then it stays in the state “maybe the next two symbols are 01”.
  - On the other hand, if the next symbol is 0, then it switches to the state “maybe the next symbol is 1”.
    * If the next symbol is indeed 1, then it switches to the accept state (but keeps on reading until the end of the string).
    * On the other hand, if the next symbol is 0, then it switches to the start state, and skips 0s until it reads 1 again.

By defining the following four states, this process will become clear:

- $q_1$: $M$ is in this state if the last symbol read was 1, but the substring 101 has not been read.
- $q_{10}$: $M$ is in this state if the last two symbols read were 10, but the substring 101 has not been read.
- $q_{101}$: $M$ is in this state if the substring 101 has been read in the input string.
- $q$: In all other cases, $M$ is in this state.

Here is the formal description of the finite automaton that accepts the language $A$:

- $Q = \{ q, q_1, q_{10}, q_{101} \}$. 
2.2. Deterministic finite automata

- $\Sigma = \{0, 1\}$,
- the start state is $q$,
- the set $F$ of accept states is equal to $F = \{q_{101}\}$, and
- the transition function $\delta$ is given by the following table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$q$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_{10}$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_{10}$</td>
<td>$q$</td>
<td>$q_{101}$</td>
</tr>
<tr>
<td>$q_{101}$</td>
<td>$q_{10}$</td>
<td>$q_{101}$</td>
</tr>
</tbody>
</table>

The figure below gives the state diagram of the finite automaton $M = (Q, \Sigma, \delta, q, F)$.

This finite automaton accepts the language $A$ consisting of all binary strings that contain the substring 101. As an exercise, how would you obtain a finite automaton that accepts the complement of $A$, i.e., the language consisting of all binary strings that do not contain the substring 101?

2.2.3 A third example of a finite automaton

The finite automata we have seen so far have exactly one accept state. In this section, we will see an example of a finite automaton having more accept states.
Let \( A \) be the language
\[
A = \{ w \in \{0, 1\}^* : \text{w has a 1 in the third position from the right} \},
\]
where \( \{0, 1\}^* \) is the set of all binary strings, including the empty string \( \epsilon \). We claim that \( A \) is a regular language. To prove this, we have to construct a finite automaton \( M \) such that \( A = L(M) \). At first sight, it seems difficult (or even impossible?) to construct such a finite automaton: How does the automaton “know” that it has reached the third symbol from the right? It is, however, possible to construct such an automaton. The main idea is to remember the last three symbols that have been read. Thus, the finite automaton has eight states \( q_{ijk} \), where \( i, j, \) and \( k \) range over the two elements of \( \{0, 1\} \). If the automaton is in state \( q_{ijk} \), then the following hold:

- If \( M \) has read at least three symbols, then the three most recently read symbols are \( ijk \).
- If \( M \) has read only two symbols, then these two symbols are \( jk \); moreover, \( i = 0 \).
- If \( M \) has read only one symbol, then this symbol is \( k \); moreover, \( i = j = 0 \).
- If \( M \) has not read any symbol, then \( i = j = k = 0 \).

The start state is \( q_{000} \) and the set of accept states is \( \{q_{100}, q_{110}, q_{101}, q_{111}\} \). The transition function of \( M \) is given by the following state diagram.
2.3 Regular operations

In this section, we define three operations on languages. Later, we will answer the question whether the set of all regular languages is closed under these operations. Let $A$ and $B$ be two languages over the same alphabet.

1. The **union** of $A$ and $B$ is defined as
   \[ A \cup B = \{ w : w \in A \text{ or } w \in B \}. \]

2. The **concatenation** of $A$ and $B$ is defined as
   \[ AB = \{ ww' : w \in A \text{ and } w' \in B \}. \]
   In words, $AB$ is the set of all strings obtained by taking an arbitrary string $w$ in $A$ and an arbitrary string $w'$ in $B$, and gluing them together (such that $w$ is to the left of $w'$).

3. The **star** of $A$ is defined as
   \[ A^* = \{ u_1u_2\ldots u_k : k \geq 0 \text{ and } u_i \in A \text{ for all } i = 1, 2, \ldots, k \}. \]
   In words, $A^*$ is obtained by taking any finite number of strings in $A$, and gluing them together. Observe that $k = 0$ is allowed; this corresponds to the empty string $\epsilon$. Thus, $\epsilon \in A^*$.

To give an example, let $A = \{0, 01\}$ and $B = \{1, 10\}$. Then
   \[ A \cup B = \{0, 01, 1, 10\}, \]
   \[ AB = \{01, 010, 011, 0110\}, \]
and
   \[ A^* = \{\epsilon, 0, 01, 00, 001, 010, 0101, 000, 0001, 00101, \ldots\}. \]
As another example, if $\Sigma = \{0, 1\}$, then $\Sigma^*$ is the set of all binary strings (including the empty string). Observe that a string always has a finite length.

Before we proceed, we give an alternative (and equivalent) definition of the star of the language $A$: Define
   \[ A^0 = \{\epsilon\} \]
and, for $k \geq 1$,
\[ A^k = AA^{k-1}, \]
i.e., $A^k$ is the concatenation of the two languages $A$ and $A^{k-1}$. Then we have
\[ A^* = \bigcup_{k=0}^{\infty} A^k. \]

**Theorem 2.3.1** The set of regular languages is closed under the union operation, i.e., if $A$ and $B$ are regular languages over the same alphabet $\Sigma$, then $A \cup B$ is also a regular language.

**Proof.** Since $A$ and $B$ are regular languages, there are finite automata $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ that accept $A$ and $B$, respectively. In order to prove that $A \cup B$ is regular, we have to construct a finite automaton $M$ that accepts $A \cup B$. In other words, $M$ must have the property that for every string $w \in \Sigma^*$,
\[ M \text{ accepts } w \Leftrightarrow M_1 \text{ accepts } w \text{ or } M_2 \text{ accepts } w. \]

As a first idea, we may think that $M$ could do the following:

- Starting in the start state $q_1$ of $M_1$, $M$ “runs” $M_1$ on $w$.
- If, after having read $w$, $M_1$ is in a state of $F_1$, then $w \in A$, thus $w \in A \cup B$ and, therefore, $M$ accepts $w$.
- On the other hand, if, after having read $w$, $M_1$ is in a state that is not in $F_1$, then $w \notin A$ and $M$ “runs” $M_2$ on $w$, starting in the start state $q_2$ of $M_2$. If, after having read $w$, $M_2$ is in a state of $F_2$, then we know that $w \in B$, thus $w \in A \cup B$ and, therefore, $M$ accepts $w$. Otherwise, we know that $w \notin A \cup B$, and $M$ rejects $w$.

This idea does not work, because the finite automaton $M$ can read the input string $w$ only once. The correct approach is to run $M_1$ and $M_2$ simultaneously. We define the set $Q$ of states of $M$ to be the Cartesian product $Q_1 \times Q_2$. If $M$ is in state $(r_1, r_2)$, this means that

- if $M_1$ would have read the input string up to this point, then it would be in state $r_1$, and
• if $M_2$ would have read the input string up to this point, then it would be in state $r_2$.

This leads to the finite automaton $M = (Q, \Sigma, \delta, q, F)$, where

- $Q = Q_1 \times Q_2 = \{(r_1, r_2) : r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$. Observe that $|Q| = |Q_1| \times |Q_2|$, which is finite.
- $\Sigma$ is the alphabet of $A$ and $B$ (recall that we assume that $A$ and $B$ are languages over the same alphabet).
- The start state $q$ of $M$ is equal to $q = (q_1, q_2)$.
- The set $F$ of accept states of $M$ is given by
  
  $F = \{(r_1, r_2) : r_1 \in F_1 \text{ or } r_2 \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

- The transition function $\delta : Q \times \Sigma \to Q$ is given by
  
  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$,

  for all $r_1 \in Q_1$, $r_2 \in Q_2$, and $a \in \Sigma$.

To finish the proof, we have to show that this finite automaton $M$ indeed accepts the language $A \cup B$. Intuitively, this should be clear from the discussion above. The easiest way to give a formal proof is by using the extended transition functions $\overline{\delta_1}$ and $\overline{\delta_2}$. (The extended transition function has been defined after Definition 2.2.4.) Here we go: Recall that we have to prove that

$M$ accepts $w \iff M_1$ accepts $w$ or $M_2$ accepts $w$,

i.e.,

$M$ accepts $w \iff \overline{\delta_1}(q_1, w) \in F_1$ or $\overline{\delta_2}(q_2, w) \in F_2$.

In terms of the extended transition function $\overline{\delta}$ of the transition function $\delta$ of $M$, this becomes

$\overline{\delta}((q_1, q_2), w) \in F \iff \overline{\delta_1}(q_1, w) \in F_1$ or $\overline{\delta_2}(q_2, w) \in F_2$. \hfill (2.1)

By applying the definition of the extended transition function, as given after Definition 2.2.4, to $\delta$, it can be seen that

$\overline{\delta}((q_1, q_2), w) = (\overline{\delta_1}(q_1, w), \overline{\delta_2}(q_2, w))$. 
The latter equality implies that (2.1) is true and, therefore, $M$ indeed accepts the language $A \cup B$. □

What about the closure of the regular languages under the concatenation and star operations? It turns out that the regular languages are closed under these operations. But how do we prove this?

Let $A$ and $B$ be two regular languages, and let $M_1$ and $M_2$ be finite automata that accept $A$ and $B$, respectively. How do we construct a finite automaton $M$ that accepts the concatenation $AB$? Given an input string $u$, $M$ has to decide whether or not $u$ can be broken into two strings $w$ and $w'$ (i.e., write $u$ as $u = ww'$), such that $w \in A$ and $w' \in B$. In words, $M$ has to decide whether or not $u$ can be broken into two substrings, such that the first substring is accepted by $M_1$ and the second substring is accepted by $M_2$. The difficulty is caused by the fact that $M$ has to make this decision by scanning the string $u$ only once. If $u \in AB$, then $M$ has to decide, during this single scan, where to break $u$ into two substrings. Similarly, if $u \notin AB$, then $M$ has to decide, during this single scan, that $u$ cannot be broken into two substrings such that the first substring is in $A$ and the second substring is in $B$.

It seems to be even more difficult to prove that $A^*$ is a regular language, if $A$ itself is regular. In order to prove this, we need a finite automaton that, when given an arbitrary input string $u$, decides whether or not $u$ can be broken into substrings such that each substring is in $A$. The problem is that, if $u \in A^*$, the finite automaton has to determine into how many substrings, and where, the string $u$ has to be broken; it has to do this during one single scan of the string $u$.

As we mentioned already, if $A$ and $B$ are regular languages, then both $AB$ and $A^*$ are also regular. In order to prove these claims, we will introduce a more general type of finite automaton.

The finite automata that we have seen so far are deterministic. This means the following:

- If the finite automaton $M$ is in state $r$ and if it reads the symbol $a$, then $M$ switches from state $r$ to the uniquely defined state $\delta(r,a)$.

From now on, we will call such a finite automaton a deterministic finite automaton (DFA). In the next section, we will define the notion of a nondeterministic finite automaton (NFA). For such an automaton, there are zero or more possible states to switch to. At first sight, nondeterministic finite
automata seem to be more powerful than their deterministic counterparts. We will prove, however, that DFAs have the same power as NFAs. As we will see, using this fact, it will be easy to prove that the class of regular languages is closed under the concatenation and star operations.

2.4 Nondeterministic finite automata

We start by giving three examples of nondeterministic finite automata. These examples will show the difference between this type of automata and the deterministic versions that we have considered in the previous sections. After these examples, we will give a formal definition of a nondeterministic finite automaton.

2.4.1 A first example

Consider the following state diagram:

![State Diagram](image)

You will notice three differences with the finite automata that we have seen until now. First, if the automaton is in state $q_1$ and reads the symbol 1, then it has two options: Either it stays in state $q_1$, or it switches to state $q_2$. Second, if the automaton is in state $q_2$, then it can switch to state $q_3$ without reading a symbol; this is indicated by the edge having the empty string $\varepsilon$ as label. Third, if the automaton is in state $q_3$ and reads the symbol 0, then it cannot continue.

Let us see what this automaton can do when it gets the string 010110 as input. Initially, the automaton is in the start state $q_1$.

- Since the first symbol in the input string is 0, the automaton stays in state $q_1$ after having read this symbol.
- The second symbol is 1, and the automaton can either stay in state $q_1$ or switch to state $q_2$. 
– If the automaton stays in state \( q_1 \), then it is still in this state after having read the third symbol.
– If the automaton switches to state \( q_2 \), then it again has two options:
  * Either read the third symbol in the input string, which is 0, and switch to state \( q_3 \),
  * or switch to state \( q_3 \), without reading the third symbol.

If we continue in this way, then we see that, for the input string 010110, there are seven possible computations. All these computations are given in the figure below.

Consider the lowest path in the figure above:

- When reading the first symbol, the automaton stays in state \( q_1 \).
- When reading the second symbol, the automaton switches to state \( q_2 \).
- The automaton does not read the third symbol (equivalently, it “reads” the empty string \( \epsilon \)), and switches to state \( q_3 \). At this moment, the
2.4. Nondeterministic finite automata

automaton cannot continue: The third symbol is 0, but there is no edge leaving $q_3$ that is labeled 0, and there is no edge leaving $q_3$ that is labeled $\epsilon$. Therefore, the computation hangs at this point.

From the figure, you can see that, out of the seven possible computations, exactly two end in the accept state $q_4$ (after the entire input string 010110 has been read). We say that the automaton accepts the string 010110, because there is at least one computation that ends in the accept state.

Now consider the input string 010. In this case, there are three possible computations:

1. $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_1$
2. $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{0} q_3$
3. $q_1 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \xrightarrow{\epsilon} q_3 \rightarrow \text{hang}$

None of these computations ends in the accept state (after the entire input string 010 has been read). Therefore, we say that the automaton rejects the input string 010.

The state diagram given above is an example of a nondeterministic finite automaton (NFA). Informally, an NFA accepts a string, if there exists at least one path in the state diagram that (i) starts in the start state, (ii) does not hang before the entire string has been read, and (iii) ends in an accept state. A string for which (i), (ii), and (iii) does not hold is rejected by the NFA.

The NFA given above accepts all binary strings that contain 101 or 11 as a substring. All other binary strings are rejected.

2.4.2 A second example

Let $A$ be the language

$$A = \{w \in \{0, 1\}^* : w \text{ has a 1 in the third position from the right}\}.$$ 

The following state diagram defines an NFA that accepts all strings that are in $A$, and rejects all strings that are not in $A$. 

![State Diagram](attachment://state_diagram.png)
This NFA does the following. If it is in the start state $q_1$ and reads the symbol 1, then it either stays in state $q_1$ or it “guesses” that this symbol is the third symbol from the right in the input string. In the latter case, the NFA switches to state $q_2$, and then it “verifies” that there are indeed exactly two remaining symbols in the input string. If there are more than two remaining symbols, then the NFA hangs (in state $q_4$) after having read the next two symbols.

Observe how this guessing mechanism is used: The automaton can only read the input string once, from left to right. Hence, it does not know when it reaches the third symbol from the right. When the NFA reads a 1, it can guess that this is the third symbol from the right; after having made this guess, it verifies whether or not the guess was correct.

In Section 2.2.3, we have seen a DFA for the same language $A$. Observe that the NFA has a much simpler structure than the DFA.

### 2.4.3 A third example

Consider the following state diagram, which defines an NFA whose alphabet is \{0\}.

![State Diagram](image)

This NFA accepts the language

$$A = \{0^k : k \equiv 0 \text{ mod } 2 \text{ or } k \equiv 0 \text{ mod } 3\},$$

where $0^k$ is the string consisting of $k$ many 0s. (If $k = 0$, then $0^k = \epsilon$.) Observe that $A$ is the union of the two languages

$$A_1 = \{0^k : k \equiv 0 \text{ mod } 2\}$$
2.4. Nondeterministic finite automata

and

\[ A_2 = \{0^k : k \equiv 0 \mod 3\}. \]

The NFA basically consists of two DFAs: one of these accepts \(A_1\), whereas the other accepts \(A_2\). Given an input string \(w\), the NFA has to decide whether or not \(w \in A\), which is equivalent to deciding whether or not \(w \in A_1\) or \(w \in A_2\). The NFA makes this decision in the following way: At the start, it “guesses” whether (i) it is going to check whether or not \(w \in A_1\) (i.e., the length of \(w\) is even), or (ii) it is going to check whether or not \(w \in A_2\) (i.e., the length of \(w\) is a multiple of 3). After having made the guess, it verifies whether or not the guess was correct. If \(w \in A\), then there exists a way of making the correct guess and verifying that \(w\) is indeed an element of \(A\) (by ending in an accept state). If \(w \not\in A\), then no matter which guess is made, the NFA will never end in an accept state.

2.4.4 Definition of nondeterministic finite automaton

The previous examples give you an idea what nondeterministic finite automata are and how they work. In this section, we give a formal definition of these automata.

For any alphabet \(\Sigma\), we define \(\Sigma_\epsilon\) to be the set

\[ \Sigma_\epsilon = \Sigma \cup \{\epsilon\}. \]

Recall the notion of a power set: For any set \(Q\), the power set of \(Q\), denoted by \(\mathcal{P}(Q)\), is the set of all subsets of \(Q\), i.e.,

\[ \mathcal{P}(Q) = \{R : R \subseteq Q\}. \]

Definition 2.4.1 A **nondeterministic finite automaton** (NFA) is a 5-tuple \(M = (Q, \Sigma, \delta, q, F)\), where

1. \(Q\) is a finite set, whose elements are called **states**,
2. \(\Sigma\) is a finite set, called the **alphabet**; the elements of \(\Sigma\) are called **symbols**,
3. \(\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)\) is a function, called the **transition function**,
4. \(q\) is an element of \(Q\); it is called the **start state**,
5. \(F\) is a subset of \(Q\); the elements of \(F\) are called **accept states**.
As for DFAs, the transition function \( \delta \) can be thought of as the “program” of the finite automaton \( M = (Q, \Sigma, \delta, q, F) \):

- Let \( r \in Q \), and let \( a \in \Sigma \). Then \( \delta(r, a) \) is a (possibly empty) subset of \( Q \). If the NFA \( M \) is in state \( r \), and if it reads \( a \) (where \( a \) may be the empty string \( \epsilon \)), then \( M \) can switch from state \( r \) to any state in \( \delta(r, a) \).
- If \( \delta(r, a) = \emptyset \), then \( M \) cannot continue and the computation hangs.

The example given in Section 2.4.1 is an NFA, where \( Q = \{q_1, q_2, q_3, q_4\} \), \( \Sigma = \{0, 1\} \), the start state is \( q_1 \), the set of accept states is \( F = \{q_4\} \), and the transition function \( \delta \) is given by the following table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>{( q_1 )}</td>
<td>{( q_1, q_2 )}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>{( q_3 )}</td>
<td>\emptyset</td>
<td>{( q_3 )}</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>\emptyset</td>
<td>\emptyset</td>
<td>{( q_4 )}</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>{( q_1 )}</td>
<td>{( q_4 )}</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

**Definition 2.4.2** Let \( M = (Q, \Sigma, \delta, q, F) \) be an NFA, and let \( w \in \Sigma^* \). We say that \( M \) accepts \( w \), if \( w \) can be written as \( w = y_1y_2\ldots y_m \), where \( y_i \in \Sigma \) for all \( i \) with \( 1 \leq i \leq m \), and there exists a sequence \( r_0, r_1, \ldots, r_m \) of states in \( Q \), such that

- \( r_0 = q \).
- \( r_{i+1} \in \delta(r_i, y_{i+1}) \), for \( i = 0, 1, \ldots, m - 1 \), and
- \( r_m \in F \).

Otherwise, we say that \( M \) rejects the string \( w \).

The NFA in the example in Section 2.4.1 accepts the string 01100. This can be seen by taking

- \( w = 01\epsilon100 = y_1y_2y_3y_4y_5y_6 \), and
- \( r_0 = q_1, r_1 = q_1, r_2 = q_2, r_3 = q_3, r_4 = q_4, r_5 = q_4, \) and \( r_6 = q_4 \).

**Definition 2.4.3** Let \( M = (Q, \Sigma, \delta, q, F) \) be an NFA. The *language* \( L(M) \) accepted by \( M \) is defined as

\[
L(M) = \{ w \in \Sigma^* : M \text{ accepts } w \}.
\]
2.5 Equivalence of DFAs and NFAs

You may have the impression that nondeterministic finite automata are more powerful than deterministic finite automata. In this section, we will show that this is not the case. That is, we will prove that a language can be accepted by a DFA if and only if it can be accepted by an NFA. In order to prove this, we will show how to convert an arbitrary NFA to a DFA that accepts the same language.

What about converting a DFA to an NFA? Well, there is (almost) nothing to do, because a DFA is also an NFA. This is not quite true, because

- the transition function of a DFA maps a state and a symbol to a state, whereas
- the transition function of an NFA maps a state and a symbol to a set of zero or more states.

The formal conversion of a DFA to an NFA is done as follows: Let \( M = (Q, \Sigma, \delta, q, F) \) be a DFA. Recall that \( \delta \) is a function \( \delta : Q \times \Sigma \rightarrow Q \). We define the function \( \delta' : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q) \) as follows. For any \( r \in Q \) and for any \( a \in \Sigma_\epsilon \),

\[
\delta'(r, a) = \begin{cases} 
\{ \delta(r, a) \} & \text{if } a \neq \epsilon, \\
\emptyset & \text{if } a = \epsilon.
\end{cases}
\]

Then \( N = (Q, \Sigma, \delta', q, F) \) is an NFA, whose behavior is exactly the same as that of the DFA \( M \); the easiest way to see this is by observing that the state diagrams of \( M \) and \( N \) are equal. Therefore, we have \( L(M) = L(N) \).

In the rest of this section, we will show how to convert an NFA to a DFA:

**Theorem 2.5.1** Let \( N = (Q, \Sigma, \delta, q, F) \) be a nondeterministic finite automaton. There exists a deterministic finite automaton \( M \), such that \( L(M) = L(N) \).

**Proof.** Recall that the NFA \( N \) can (in general) perform more than one computation on a given input string. The idea of the proof is to construct a DFA \( M \) that runs all these different computations simultaneously. (We have seen this idea already in the proof of Theorem 2.3.1.) To be more precise, the DFA \( M \) will have the following property:

- the state that \( M \) is in after having read an initial part of the input string corresponds exactly to the set of all states that \( N \) can reach after having read the same part of the input string.
We start by presenting the conversion for the case when $N$ does not contain $\varepsilon$-transitions. In other words, the state diagram of $N$ does not contain any edge that has $\varepsilon$ as a label. (Later, we will extend the conversion to the general case.) Let the DFA $M$ be defined as $M = (Q', \Sigma, \delta', q', F')$, where

- the set $Q'$ of states is equal to $Q' = \mathcal{P}(Q)$; observe that $|Q'| = 2^{|Q|}$,
- the start state $q'$ is equal to $q' = \{q\}$; so $M$ has the “same” start state as $N$,
- the set $F'$ of accept states is equal to the set of all elements $R$ of $Q'$ having the property that $R$ contains at least one accept state of $N$, i.e.,
  $$F' = \{R \in Q' : R \cap F \neq \emptyset\},$$
- the transition function $\delta' : Q' \times \Sigma \rightarrow Q'$ is defined as follows: For each $R \in Q'$ and for each $a \in \Sigma$,
  $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

Let us see what the transition function $\delta'$ of $M$ does. First observe that, since $N$ is an NFA, $\delta(r, a)$ is a subset of $Q$. This implies that $\delta'(R, a)$ is the union of subsets of $Q$ and, therefore, also a subset of $Q$. Hence, $\delta'(R, a)$ is an element of $Q'$.

The set $\delta(r, a)$ is equal to the set of all states of the NFA $N$ that can be reached from state $r$ by reading the symbol $a$. We take the union of these sets $\delta(r, a)$, where $r$ ranges over all elements of $R$, to obtain the new set $\delta'(R, a)$. This new set is the state that the DFA $M$ reaches from state $R$, by reading the symbol $a$.

In this way, we obtain the correspondence that was given in the beginning of this proof.

After this warming-up, we can consider the general case. In other words, from now on, we allow $\varepsilon$-transitions in the NFA $N$. The DFA $M$ is defined as above, except that the start state $q'$ and the transition function $\delta'$ have to be modified. Recall that a computation of the NFA $N$ consists of the following:

1. Start in the start state $q$ and make zero or more $\varepsilon$-transitions.
2. Read one “real” symbol of $\Sigma$ and move to a new state (or stay in the current state).
3. Make zero or more $\epsilon$-transitions.

4. Read one “real” symbol of $\Sigma$ and move to a new state (or stay in the current state).

5. Make zero or more $\epsilon$-transitions.


The DFA $M$ will simulate this computation in the following way:

- Simulate 1. in one single step. As we will see below, this simulation is implicitly encoded in the definition of the start state $q'$ of $M$.

- Simulate 2. and 3. in one single step.

- Simulate 4. and 5. in one single step.

- Etc.

Thus, in one step, the DFA $M$ simulates the reading of one “real” symbol of $\Sigma$, followed by making zero or more $\epsilon$-transitions.

To formalize this, we need the notion of $\epsilon$-closure. For any state $r$ of the NFA $N$, the $\epsilon$-closure of $r$, denoted by $C_\epsilon(r)$, is defined to be the set of all states of $N$ that can be reached from $r$, by making zero or more $\epsilon$-transitions. For any state $R$ of the DFA $M$ (hence, $R \subseteq Q$), we define

$$C_\epsilon(R) = \bigcup_{r \in R} C_\epsilon(r).$$

How do we define the start state $q'$ of the DFA $M$? Before the NFA $N$ reads its first “real” symbol of $\Sigma$, it makes zero or more $\epsilon$-transitions. In other words, at the moment when $N$ reads the first symbol of $\Sigma$, it can be in any state of $C_\epsilon(q)$. Therefore, we define $q'$ to be

$$q' = C_\epsilon(q) = C_\epsilon(\{q\}).$$

How do we define the transition function $\delta'$ of the DFA $M$? Assume that $M$ is in state $R$, and reads the symbol $a$. At this moment, the NFA $N$ would have been in any state $r$ of $R$. By reading the symbol $a$, $N$ can switch to any state in $\delta(r, a)$, and then make zero or more $\epsilon$-transitions. Hence, the
NFA can switch to any state in the set $C_\epsilon(\delta(r, a))$. Based on this, we define $\delta'(R, a)$ to be

$$\delta'(R, a) = \bigcup_{r \in R} C_\epsilon(\delta(r, a)).$$

To summarize, the NFA $N = (Q, \Sigma, \delta, q, F)$ is converted to the DFA $M = (Q', \Sigma, \delta', q', F')$, where

- $Q' = \mathcal{P}(Q)$,
- $q' = C_\epsilon(\{q\})$,
- $F' = \{R \in Q' : R \cap F \neq \emptyset\}$,
- $\delta' : Q' \times \Sigma \rightarrow Q'$ is defined as follows: For each $R \in Q'$ and for each $a \in \Sigma$,

$$\delta'(R, a) = \bigcup_{r \in R} C_\epsilon(\delta(r, a)).$$

The results proved until now can be summarized in the following theorem.

**Theorem 2.5.2** Let $A$ be a language. Then $A$ is regular if and only if there exists a nondeterministic finite automaton that accepts $A$.

### 2.5.1 An example

Consider the NFA $N = (Q, \Sigma, \delta, q, F)$, where $Q = \{1, 2, 3\}$, $\Sigma = \{a, b\}$, $q = 1$, $F = \{2\}$, and $\delta$ is given by the following table:

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$\epsilon$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${3}$</td>
<td>$\emptyset$</td>
<td>${2}$</td>
</tr>
<tr>
<td>2</td>
<td>${1}$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>${2}$</td>
<td>${2, 3}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

The state diagram of $N$ is as follows:
We will show how to convert this NFA $N$ to a DFA $M$ that accepts the same language. Following the proof of Theorem 2.5.1, the DFA $M$ is specified by $M = (Q', \Sigma, \delta', q', F')$, where each of the components is defined below.

- $Q' = \mathcal{P}(Q)$. Hence,
  \[ Q' = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\} \}. \]

- $q' = C_\epsilon(\{q\})$. Hence, the start state $q'$ of $M$ is the set of all states of $N$ that can be reached from $N$'s start state $q = 1$, by making zero or more $\epsilon$-transitions. We obtain
  \[ q' = C_\epsilon(\{q\}) = C_\epsilon(\{1\}) = \{1, 2\}. \]

- $F' = \{R \in Q' : R \cap F \neq \emptyset\}$. Hence, the accept states of $M$ are those states that contain the accept state 2 of $N$. We obtain
  \[ F' = \{\{2\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\} \}. \]

- $\delta' : Q' \times \Sigma \to Q'$ is defined as follows: For each $R \in Q'$ and for each $a \in \Sigma$, 
  \[ \delta'(R, a) = \bigcup_{r \in R} C_\epsilon(\delta(r, a)). \]
In this example $\delta'$ is given by

\[
\begin{align*}
\delta'(\emptyset, a) &= \emptyset & \delta'(\emptyset, b) &= \emptyset \\
\delta'(\{1\}, a) &= \{3\} & \delta'(\{1\}, b) &= \emptyset \\
\delta'(\{2\}, a) &= \{1, 2\} & \delta'(\{2\}, b) &= \emptyset \\
\delta'(\{3\}, a) &= \{2\} & \delta'(\{3\}, b) &= \{2, 3\} \\
\delta'(\{1, 2\}, a) &= \{1, 2, 3\} & \delta'(\{1, 2\}, b) &= \emptyset \\
\delta'(\{1, 3\}, a) &= \{2, 3\} & \delta'(\{1, 3\}, b) &= \{2, 3\} \\
\delta'(\{2, 3\}, a) &= \{1, 2\} & \delta'(\{2, 3\}, b) &= \{2, 3\} \\
\delta'(\{1, 2, 3\}, a) &= \{1, 2, 3\} & \delta'(\{1, 2, 3\}, b) &= \{2, 3\}
\end{align*}
\]

The state diagram of the DFA $M$ is as follows:

We make the following observations:
• The states \{1\} and \{1, 3\} do not have incoming edges. Therefore, these two states cannot be reached from the start state \{1, 2\}.

• The state \{3\} has only one incoming edge; it comes from the state \{1\}. Since \{1\} cannot be reached from the start state, \{3\} cannot be reached from the start state.

• The state \{2\} has only one incoming edge; it comes from the state \{3\}. Since \{3\} cannot be reached from the start state, \{2\} cannot be reached from the start state.

Hence, we can remove the four states \{1\}, \{2\}, \{3\}, and \{1, 3\}. The resulting DFA accepts the same language as the DFA above. This leads to the following state diagram, which depicts a DFA that accepts the same language as the NFA \(N\):
2.6 Closure under the regular operations

In Section 2.3, we have defined the regular operations union, concatenation, and star. We proved in Theorem 2.3.1 that the union of two regular languages is a regular language. We also explained why it is not clear that the concatenation of two regular languages is regular, and that the star of a regular language is regular. In this section, we will see that the concept of NFA, together with Theorem 2.5.2, can be used to give a simple proof of the fact that the regular languages are indeed closed under the regular operations. We start by giving an alternative proof of Theorem 2.3.1:

**Theorem 2.6.1** The set of regular languages is closed under the union operation, i.e., if $A_1$ and $A_2$ are regular languages over the same alphabet $\Sigma$, then $A_1 \cup A_2$ is also a regular language.

**Proof.** Since $A_1$ is regular, there is, by Theorem 2.5.2, an NFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, such that $A_1 = L(M_1)$. Similarly, there is an NFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, such that $A_2 = L(M_2)$. We may assume that $Q_1 \cap Q_2 = \emptyset$, because otherwise, we can give new “names” to the states of $Q_1$ and $Q_2$. From these two NFAs, we will construct an NFA $M = (Q, \Sigma, \delta, q_0, F)$, such that $L(M) = A_1 \cup A_2$. The construction is illustrated in Figure 2.1. The NFA $M$ is defined as follows:

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$, where $q_0$ is a new state.
2. $q_0$ is the start state of $M$.
3. $F = F_1 \cup F_2$.
4. $\delta : Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q)$ is defined as follows: For any $r \in Q$ and for any $a \in \Sigma_{\epsilon}$,

$$
\delta(r, a) = \begin{cases} 
\delta_1(r, a) & \text{if } r \in Q_1, \\
\delta_2(r, a) & \text{if } r \in Q_2, \\
\{q_1, q_2\} & \text{if } r = q_0 \text{ and } a = \epsilon, \\
\emptyset & \text{if } r = q_0 \text{ and } a \neq \epsilon.
\end{cases}
$$

\end{document}
Theorem 2.6.2 The set of regular languages is closed under the concatenation operation, i.e., if $A_1$ and $A_2$ are regular languages over the same alphabet $\Sigma$, then $A_1A_2$ is also a regular language.

Proof. Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA, such that $A_1 = L(M_1)$. Similarly, let $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ be an NFA, such that $A_2 = L(M_2)$. As in the proof of Theorem 2.6.1, we may assume that $Q_1 \cap Q_2 = \emptyset$. We will construct an NFA $M = (Q, \Sigma, \delta, q_0, F)$, such that $L(M) = A_1A_2$. The construction is illustrated in Figure 2.2. The NFA $M$ is defined as follows:

1. $Q = Q_1 \cup Q_2$.
2. $q_0 = q_1$.
3. $F = F_2$. 

Figure 2.1: The NFA $M$ accepts $L(M_1) \cup L(M_2)$. 
4. $\delta : Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q)$ is defined as follows: For any $r \in Q$ and for any $a \in \Sigma$, 

$$
\delta(r, a) = \begin{cases} 
\delta_1(r, a) & \text{if } r \in Q_1 \text{ and } r \notin F_1, \\
\delta_1(r, a) & \text{if } r \in F_1 \text{ and } a \neq \epsilon, \\
\delta_1(r, a) \cup \{q_2\} & \text{if } r \in F_1 \text{ and } a = \epsilon, \\
\delta_2(r, a) & \text{if } r \in Q_2.
\end{cases}
$$

Theorem 2.6.3 The set of regular languages is closed under the star operation, i.e., if $A$ is a regular language, then $A^*$ is also a regular language.

Proof. Let $\Sigma$ be the alphabet of $A$ and let $N = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA, such that $A = L(N)$. We will construct an NFA $M = (Q, \Sigma, \delta, q_0, F)$, such that $L(M) = A^*$. The construction is illustrated in Figure 2.3. The NFA $M$ is defined as follows:
1. \( Q = \{q_0\} \cup Q_1 \), where \( q_0 \) is a new state.

2. \( q_0 \) is the start state of \( M \).

3. \( F = \{q_0\} \cup F_1 \). (Since \( \epsilon \in A^* \), \( q_0 \) has to be an accept state.)

4. \( \delta : Q \times \Sigma_\epsilon \to P(Q) \) is defined as follows: For any \( r \in Q \) and for any \( a \in \Sigma_\epsilon \),

\[
\delta(r, a) = \begin{cases} 
\delta_1(r, a) & \text{if } r \in Q_1 \text{ and } r \notin F_1, \\
\delta_1(r, a) \cup \{q_1\} & \text{if } r \in F_1 \text{ and } a = \epsilon, \\
\{q_1\} & \text{if } r = q_0 \text{ and } a = \epsilon, \\
\emptyset & \text{if } r = q_0 \text{ and } a \neq \epsilon.
\end{cases}
\]

In the final theorem of this section, we mention (without proof) two more closure properties of the regular languages:

**Theorem 2.6.4** The set of regular languages is closed under the complement and intersection operations:

1. If \( A \) is a regular language over the alphabet \( \Sigma \), then the complement

\[
\overline{A} = \{w \in \Sigma^* : w \not\in A\}
\]

is also a regular language.
2. If $A_1$ and $A_2$ are regular languages over the same alphabet $\Sigma$, then the intersection

$$A_1 \cap A_2 = \{ w \in \Sigma^* : w \in A_1 \text{ and } w \in A_2 \}$$

is also a regular language.

### 2.7 Regular expressions

In this section, we present regular expressions, which are a means to describe languages. As we will see, the class of languages that can be described by regular expressions coincides with the class of regular languages.

Before formally defining the notion of a regular expression, we give some examples. Consider the expression

$$(0 \cup 1)01^*.$$ 

The language described by this expression is the set of all binary strings

1. that start with either 0 or 1 (this is indicated by $(0 \cup 1)$),
2. for which the second symbol is 0 (this is indicated by 0), and
3. that end with zero or more 1s (this is indicated by $1^*$).

That is, the language described by this expression is

$$\{00, 001, 0011, 00111, \ldots, 10, 101, 1011, 10111, \ldots\}.$$ 

Here are some more examples (in all cases, the alphabet is $\{0, 1\}$):

- The language $\{ w : w \text{ contains exactly two 0s} \}$ is described by the expression
  $$1^*01^*01^*.$$

- The language $\{ w : w \text{ contains at least two 0s} \}$ is described by the expression
  $$(0 \cup 1)^*0(0 \cup 1)^*0(0 \cup 1)^*.$$ 

- The language $\{ w : 1011 \text{ is a substring of } w \}$ is described by the expression
  $$(0 \cup 1)^*1011(0 \cup 1)^*.$$
2.7. Regular expressions

- The language \( \{ w : \text{the length of } w \text{ is even} \} \) is described by the expression
  \[ ((0 \cup 1)(0 \cup 1))^* . \]

- The language \( \{ w : \text{the length of } w \text{ is odd} \} \) is described by the expression
  \[ (0 \cup 1)((0 \cup 1)(0 \cup 1))^* . \]

- The language \( \{ 1011, 0 \} \) is described by the expression
  \[ 1011 \cup 0 . \]

- The language \( \{ w : \text{the first and last symbols of } w \text{ are equal} \} \) is described by the expression
  \[ 0((0 \cup 1)^*0 \cup 1(0 \cup 1)^*1 \cup 0 \cup 1. \]

After these examples, we give a formal (and inductive) definition of regular expressions:

**Definition 2.7.1** Let \( \Sigma \) be a non-empty alphabet.

1. \( \epsilon \) is a regular expression.
2. \( \emptyset \) is a regular expression.
3. For each \( a \in \Sigma \), \( a \) is a regular expression.
4. If \( R_1 \) and \( R_2 \) are regular expressions, then \( R_1 \cup R_2 \) is a regular expression.
5. If \( R_1 \) and \( R_2 \) are regular expressions, then \( R_1 R_2 \) is a regular expression.
6. If \( R \) is a regular expression, then \( R^* \) is a regular expression.

You can regard 1., 2., and 3. as being the “building blocks” of regular expressions. Items 4., 5., and 6. give rules that can be used to combine regular expressions into new (and “larger”) regular expressions. To give an example, we claim that

\[ (0 \cup 1)^*101(0 \cup 1)^* \]

is a regular expression (where the alphabet \( \Sigma \) is equal to \( \{ 0, 1 \} \)). In order to prove this, we have to show that this expression can be “built” using the “rules” given in Definition 2.7.1. Here we go:
• By 3., 0 is a regular expression.

• By 3., 1 is a regular expression.

• Since 0 and 1 are regular expressions, by 4., $0 \cup 1$ is a regular expression.

• Since $0 \cup 1$ is a regular expression, by 6., $(0 \cup 1)^*$ is a regular expression.

• Since 1 and 0 are regular expressions, by 5., 10 is a regular expression.

• Since 10 and 1 are regular expressions, by 5., 101 is a regular expression.

• Since $(0 \cup 1)^*$ and 101 are regular expressions, by 5., $(0 \cup 1)^*101$ is a regular expression.

• Since $(0 \cup 1)^*101$ and $(0 \cup 1)^*$ are regular expressions, by 5., $(0 \cup 1)^*101(0 \cup 1)^*$ is a regular expression.

Next we define the language that is described by a regular expression:

**Definition 2.7.2** Let Σ be a non-empty alphabet.

1. The regular expression $\epsilon$ describes the language \{\$\epsilon\$\}.

2. The regular expression $\emptyset$ describes the language $\emptyset$.

3. For each $a \in \Sigma$, the regular expression $a$ describes the language \{a\}.

4. Let $R_1$ and $R_2$ be regular expressions and let $L_1$ and $L_2$ be the languages described by them, respectively. The regular expression $R_1 \cup R_2$ describes the language $L_1 \cup L_2$.

5. Let $R_1$ and $R_2$ be regular expressions and let $L_1$ and $L_2$ be the languages described by them, respectively. The regular expression $R_1R_2$ describes the language $L_1L_2$.

6. Let $R$ be a regular expression and let $L$ be the language described by it. The regular expression $R^*$ describes the language $L^*$.

We consider some examples:

• The regular expression $(0 \cup \epsilon)(1 \cup \epsilon)$ describes the language \{01, 0, 1, $\epsilon$\}.
2.7. Regular expressions

- The regular expression \(0 \cup \epsilon\) describes the language \(\{0, \epsilon\}\), whereas the regular expression \(1^*\) describes the language \(\{\epsilon, 1, 11, 111, \ldots\}\). Therefore, the regular expression \((0 \cup \epsilon)1^*\) describes the language \(\{0, 01, 011, 0111, \ldots, \epsilon, 1, 11, 111, \ldots\}\).

Observe that this language is also described by the regular expression \(01^* \cup 1^*\).

- The regular expression \(1^*\emptyset\) describes the empty language, i.e., the language \(\emptyset\). (You should convince yourself that this is correct.)

- The regular expression \(\emptyset^*\) describes the language \(\{\epsilon\}\).

**Definition 2.7.3** Let \(R_1\) and \(R_2\) be regular expressions and let \(L_1\) and \(L_2\) be the languages described by them, respectively. If \(L_1 = L_2\) (i.e., \(R_1\) and \(R_2\) describe the same language), then we will write \(R_1 = R_2\).

Hence, even though \((0 \cup \epsilon)1^*\) and \(01^* \cup 1^*\) are different regular expressions, we write

\[(0 \cup \epsilon)1^* = 01^* \cup 1^*,\]

because they describe the same language.

In Section 2.8.2, we will show that every regular language can be described by a regular expression. The proof of this fact is purely algebraic and uses the following algebraic identities involving regular expressions.

**Theorem 2.7.4** Let \(R_1, R_2,\) and \(R_3\) be regular expressions. The following identities hold:

1. \(R_1\emptyset = \emptyset R_1 = \emptyset\).
2. \(R_1\epsilon = \epsilon R_1 = R_1\).
3. \(R_1 \cup \emptyset = \emptyset \cup R_1 = R_1\).
4. \(R_1 \cup R_1 = R_1\).
5. \(R_1 \cup R_2 = R_2 \cup R_1\).
6. \(R_1(R_2 \cup R_3) = R_1 R_2 \cup R_1 R_3\).
7. \((R_1 \cup R_2)R_3 = R_1R_3 \cup R_2R_3\).
8. \(R_1(R_2R_3) = (R_1R_2)R_3\).
9. \(\emptyset^* = \epsilon\).
10. \(\epsilon^* = \epsilon\).
11. \((\epsilon \cup R_1)^* = R_1^*\).
12. \((\epsilon \cup R_1)(\epsilon \cup R_1)^* = R_1^*\).
13. \(R_1^*(\epsilon \cup R_1) = (\epsilon \cup R_1)R_1^* = R_1^*\).
14. \(R_1^*R_2 \cup R_2 = R_1^*R_2\).
15. \(R_1(R_2R_1)^* = (R_1R_2)^*R_1\).
16. \((R_1 \cup R_2)^* = (R_1^*R_2)^*R_1^* = (R_2^*R_1)^*R_2^*\).

We will not present the (boring) proofs of these identities, but urge you to convince yourself informally that they make perfect sense. To give an example, we mentioned above that

\[(0 \cup \epsilon)^1 = 01^* \cup 1^*.\]

We can verify this identity in the following way:

\[
\begin{align*}
(0 \cup \epsilon)^1 & = 01^* \cup \epsilon 1^* \quad \text{(by identity 7)} \\
& = 01^* \cup 1^* \quad \text{(by identity 2)}
\end{align*}
\]

### 2.8 Equivalence of regular expressions and regular languages

In the beginning of Section 2.7, we mentioned the following result:

**Theorem 2.8.1** Let \(L\) be a language. Then \(L\) is regular if and only if there exists a regular expression that describes \(L\).

The proof of this theorem consists of two parts:
• In Section 2.8.1, we will prove that every regular expression describes a regular language.

• In Section 2.8.2, we will prove that every DFA $M$ can be converted to a regular expression that describes the language $L(M)$.

These two results will prove Theorem 2.8.1.

2.8.1 Every regular expression describes a regular language

Let $R$ be an arbitrary regular expression over the alphabet $\Sigma$. We will prove that the language described by $R$ is a regular language. The proof is by induction on the structure of $R$ (i.e., by induction on the way $R$ is “built” using the “rules” given in Definition 2.7.1).

The first base case: Assume that $R = \epsilon$. Then $R$ describes the language $\{\epsilon\}$. In order to prove that this language is regular, it suffices, by Theorem 2.5.2, to construct an NFA $M = (Q, \Sigma, \delta, q, F)$ that accepts this language. This NFA is obtained by defining $Q = \{q\}$, $q$ is the start state, $F = \{q\}$, and $\delta(q, a) = \emptyset$ for all $a \in \Sigma$. The figure below gives the state diagram of $M$:

![State Diagram](image1)

The second base case: Assume that $R = \emptyset$. Then $R$ describes the language $\emptyset$. In order to prove that this language is regular, it suffices, by Theorem 2.5.2, to construct an NFA $M = (Q, \Sigma, \delta, q, F)$ that accepts this language. This NFA is obtained by defining $Q = \{q\}$, $q$ is the start state, $F = \emptyset$, and $\delta(q, a) = \emptyset$ for all $a \in \Sigma$. The figure below gives the state diagram of $M$:

![State Diagram](image2)

The third base case: Let $a \in \Sigma$ and assume that $R = a$. Then $R$ describes the language $\{a\}$. In order to prove that this language is regular, it suffices, by Theorem 2.5.2, to construct an NFA $M = (Q, \Sigma, \delta, q_1, F)$ that accepts
this language. This NFA is obtained by defining $Q = \{q_1, q_2\}$, $q_1$ is the start state, $F = \{q_2\}$, and

$$\delta(q_1, a) = \{q_2\},$$
$$\delta(q_1, b) = \emptyset \text{ for all } b \in \Sigma \setminus \{a\},$$
$$\delta(q_2, b) = \emptyset \text{ for all } b \in \Sigma \epsilon.$$

The figure below gives the state diagram of $M$:

\[ \begin{array}{c}
q_1 \quad a \quad q_2
\end{array} \]

**The first case of the induction step:** Assume that $R = R_1 \cup R_2$, where $R_1$ and $R_2$ are regular expressions. Let $L_1$ and $L_2$ be the languages described by $R_1$ and $R_2$, respectively, and assume that $L_1$ and $L_2$ are regular. Then $R$ describes the language $L_1 \cup L_2$, which, by Theorem 2.6.1, is regular.

**The second case of the induction step:** Assume that $R = R_1R_2$, where $R_1$ and $R_2$ are regular expressions. Let $L_1$ and $L_2$ be the languages described by $R_1$ and $R_2$, respectively, and assume that $L_1$ and $L_2$ are regular. Then $R$ describes the language $L_1L_2$, which, by Theorem 2.6.2, is regular.

**The third case of the induction step:** Assume that $R = (R_1)^*$, where $R_1$ is a regular expression. Let $L_1$ be the language described by $R_1$ and assume that $L_1$ is regular. Then $R$ describes the language $(L_1)^*$, which, by Theorem 2.6.3, is regular.

This concludes the proof of the claim that every regular expression describes a regular language.

To give an example, consider the regular expression

$$(ab \cup a)^*,$$

where the alphabet is $\{a, b\}$. We will prove that this regular expression describes a regular language, by constructing an NFA that accepts the language described by this regular expression. Observe how the regular expression is “built”:

- Take the regular expressions $a$ and $b$, and combine them into the regular expression $ab$. 
2.8. Equivalence of regular expressions and regular languages 59

- Take the regular expressions $ab$ and $a$, and combine them into the regular expression $ab \cup a$.

- Take the regular expression $ab \cup a$, and transform it into the regular expression $(ab \cup a)^*$.

First, we construct an NFA $M_1$ that accepts the language described by the regular expression $a$:

Next, we construct an NFA $M_2$ that accepts the language described by the regular expression $b$:

Next, we apply the construction given in the proof of Theorem 2.6.2 to $M_1$ and $M_2$. This gives an NFA $M_3$ that accepts the language described by the regular expression $ab$:

Next, we apply the construction given in the proof of Theorem 2.6.1 to $M_3$ and $M_1$. This gives an NFA $M_4$ that accepts the language described by the regular expression $ab \cup a$:

Finally, we apply the construction given in the proof of Theorem 2.6.3 to $M_4$. This gives an NFA $M_5$ that accepts the language described by the regular expression $(ab \cup a)^*$:
2.8.2 Converting a DFA to a regular expression

In this section, we will prove that every DFA $M$ can be converted to a regular expression that describes the language $L(M)$. In order to prove this result, we need to solve recurrence relations involving languages.

**Solving recurrence relations**

Let $\Sigma$ be an alphabet, let $B$ and $C$ be “known” languages in $\Sigma^*$ such that $\epsilon \notin B$, and let $L$ be an “unknown” language such that

$$L = BL \cup C.$$ 

Can we “solve” this equation for $L$? That is, can we express $L$ in terms of $B$ and $C$?

Consider an arbitrary string $u$ in $L$. We are going to determine how $u$ looks like. Since $u \in L$ and $L = BL \cup C$, we know that $u$ is a string in $BL \cup C$. Hence, there are two possibilities for $u$.

1. $u$ is an element of $C$.

2. $u$ is an element of $BL$. In this case, there are strings $b \in B$ and $v \in L$ such that $u = bv$. Since $\epsilon \notin B$, we have $b \neq \epsilon$ and, therefore, $|v| < |u|$. (Recall that $|v|$ denotes the length, i.e., the number of symbols, of the string $v$.) Since $v$ is a string in $L$, which is equal to $BL \cup C$, $v$ is a string in $BL \cup C$. Hence, there are two possibilities for $v$. 
2.8. Equivalence of regular expressions and regular languages

(a) $v$ is an element of $C$. In this case,

$$u = bv,$$

where $b \in B$ and $v \in C$; thus, $u \in BC$.

(b) $v$ is an element of $BL$. In this case, there are strings $b' \in B$ and $w \in L$ such that $v = b'w$. Since $\epsilon \notin B$, we have $b' \neq \epsilon$ and, therefore, $|w| < |v|$. Since $w$ is a string in $L$, which is equal to $BL \cup C$, $w$ is a string in $BL \cup C$. Hence, there are two possibilities for $w$.

i. $w$ is an element of $C$. In this case,

$$u = bb'w,$$

where $b, b' \in B$ and $w \in C$; thus, $u \in BBC$.

ii. $w$ is an element of $BL$. In this case, there are strings $b'' \in B$ and $x \in L$ such that $w = b''x$. Since $\epsilon \notin B$, we have $b'' \neq \epsilon$ and, therefore, $|x| < |w|$. Since $x$ is a string in $L$, which is equal to $BL \cup C$, $x$ is a string in $BL \cup C$. Hence, there are two possibilities for $x$.

A. $x$ is an element of $C$. In this case,

$$u = bb'b''x,$$

where $b, b', b'' \in B$ and $x \in C$; thus, $u \in BBBC$.

B. $x$ is an element of $BL$. Etc., etc.

This process hopefully convinces you that any string $u$ in $L$ can be written as the concatenation of zero or more strings in $B$, followed by one string in $C$. In fact, $L$ consists of exactly those strings having this property:

Lemma 2.8.2 Let $\Sigma$ be an alphabet, and let $B$, $C$, and $L$ be languages in $\Sigma^*$ such that $\epsilon \notin B$ and

$$L = BL \cup C.$$

Then

$$L = B^*C.$$

Proof. First, we show that $B^*C \subseteq L$. Let $u$ be an arbitrary string in $B^*C$. Then $u$ is the concatenation of $k$ strings of $B$, for some $k \geq 0$, followed by one string of $C$. We proceed by induction on $k$.

The base case is when $k = 0$. In this case, $u$ is a string in $C$. Hence, $u$ is a string in $BL \cup C$. Since $BL \cup C = L$, it follows that $u$ is a string in $L$. 
Now let $k \geq 1$. Then we can write $u = vwc$, where $v$ is a string in $B$, $w$ is the concatenation of $k - 1$ strings of $B$, and $c$ is a string of $C$. Define $y = wc$. Observe that $y$ is the concatenation of $k - 1$ strings of $B$ followed by one string of $C$. Therefore, by induction, the string $y$ is an element of $L$. Hence, $u = vy$, where $v$ is a string in $B$ and $y$ is a string in $L$. This shows that $u$ is a string in $BL$. Hence, $u$ is a string in $BL \cup C$. Since $BL \cup C = L$, it follows that $u$ is a string in $L$. This completes the proof that $B^*C \subseteq L$.

It remains to show that $L \subseteq B^*C$. Let $u$ be an arbitrary string in $L$, and let $\ell$ be its length (i.e., $\ell$ is the number of symbols in $u$). We prove by induction on $\ell$ that $u$ is a string in $B^*C$.

The base case is when $\ell = 0$. Then $u = \epsilon$. Since $u \in L$ and $L = BL \cup C$, $u$ is a string in $BL \cup C$. Since $\epsilon \notin B$, $u$ cannot be a string in $BL$. Hence, $u$ must be a string in $C$. Since $C \subseteq B^*C$, it follows that $u$ is a string in $B^*C$.

Let $\ell \geq 1$. If $u$ is a string in $C$, then $u$ is a string in $B^*C$ and we are done. So assume that $u$ is not a string in $C$. Since $u \in L$ and $L = BL \cup C$, $u$ is a string in $BL$. Hence, there are strings $b \in B$ and $v \in L$ such that $u = bv$. Since $\epsilon \notin B$, the length of $b$ is at least one; hence, the length of $v$ is less than the length of $u$. By induction, $v$ is a string in $B^*C$. Hence, $u = bv$, where $b \in B$ and $v \in B^*C$. This shows that $u \in B(B^*C)$. Since $B(B^*C) \subseteq B^*C$, it follows that $u \in B^*C$.

The conversion

We will now use Lemma 2.8.2 to prove that every DFA can be converted to a regular expression.

Let $M = (Q, \Sigma, \delta, q, F)$ be an arbitrary deterministic finite automaton. We will show that there exists a regular expression that describes the language $L(M)$.

For each state $r \in Q$, we define

$$L_r = \{ w \in \Sigma^* : \text{the path in the state diagram of } M \text{ that starts in state } r \text{ and that corresponds to } w \text{ ends in a state of } F \}.$$ 

In words, $L_r$ is the language accepted by $M$, if $r$ were the start state.

We will show that each such language $L_r$ can be described by a regular expression. Since $L(M) = L_q$, this will prove that $L(M)$ can be described by a regular expression.
The basic idea is to set up equations for the languages \( L_r \), which we then solve using Lemma 2.8.2. We claim that

\[
L_r = \bigcup_{a \in \Sigma} a \cdot L_{\delta(r,a)} \quad \text{if } r \notin F.
\]  

(2.2)

Why is this true? Let \( w \) be a string in \( L_r \). Then the path \( P \) in the state diagram of \( M \) that starts in state \( r \) and that corresponds to \( w \) ends in a state of \( F \). Since \( r \notin F \), this path contains at least one edge. Let \( r' \) be the state that follows the first state (i.e., \( r \)) of \( P \). Then \( r' = \delta(r,b) \) for some symbol \( b \in \Sigma \). Hence, \( b \) is the first symbol of \( w \). Write \( w = bv \), where \( v \) is the remaining part of \( w \). Then the path \( P' = P \setminus \{r\} \) in the state diagram of \( M \) that starts in state \( r' \) and that corresponds to \( v \) ends in a state of \( F \). Therefore, \( v \in L_{r'} = L_{\delta(r,b)} \). Hence,

\[
L_r = \bigcup_{a \in \Sigma} a \cdot L_{\delta(r,a)} \subseteq \bigcup_{a \in \Sigma} a \cdot L_{\delta(r,a)}.
\]

Conversely, let \( w \) be a string in \( \bigcup_{a \in \Sigma} a \cdot L_{\delta(r,a)} \). Then there is a symbol \( b \in \Sigma \) and a string \( v \in L_{\delta(r,b)} \) such that \( w = bv \). Let \( P' \) be the path in the state diagram of \( M \) that starts in state \( \delta(r,b) \) and that corresponds to \( v \). Since \( v \in L_{\delta(r,b)} \), this path ends in a state of \( F \). Let \( P \) be the path in the state diagram of \( M \) that starts in \( r \), follows the edge to \( \delta(r,b) \), and then follows \( P' \). This path \( P \) corresponds to \( w \) and ends in a state of \( F \). Therefore, \( w \in L_r \). This proves the correctness of (2.2).

Similarly, we can prove that

\[
L_r = \epsilon \cup \left( \bigcup_{a \in \Sigma} a \cdot L_{\delta(r,a)} \right) \quad \text{if } r \in F.
\]  

(2.3)

So we now have a set of equations in the “unknowns” \( L_r \), for \( r \in Q \). The number of equations is equal to the size of \( Q \). In other words, the number of equations is equal to the number of unknowns. The regular expression for \( L(M) = L_q \) is obtained by solving these equations using Lemma 2.8.2.

Of course, we have to convince ourselves that these equations have a solution for any given DFA. Before we deal with this issue, we give an example.

An example

Consider the deterministic finite automaton \( M = (Q, \Sigma, \delta, q_0, F) \), where \( Q = \{q_0, q_1, q_2\} \), \( \Sigma = \{a, b\} \), \( q_0 \) is the start state, \( F = \{q_2\} \), and \( \delta \) is given in the
state diagram below. We show how to obtain the regular expression that describes the language accepted by $M$.

For this case, (2.2) and (2.3) give the following equations:

\[
\begin{align*}
L_{q_0} &= a \cdot L_{q_0} \cup b \cdot L_{q_2} \\
L_{q_1} &= a \cdot L_{q_0} \cup b \cdot L_{q_1} \\
L_{q_2} &= \epsilon \cup a \cdot L_{q_1} \cup b \cdot L_{q_0}
\end{align*}
\]

In the third equation, $L_{q_2}$ is expressed in terms of $L_{q_0}$ and $L_{q_1}$. Hence, if we substitute the third equation into the first one, and use Theorem 2.7.4, then we get

\[
L_{q_0} = a \cdot L_{q_0} \cup b \cdot (\epsilon \cup a \cdot L_{q_1} \cup b \cdot L_{q_0})
\]

\[
= (a \cup bb) \cdot L_{q_0} \cup ba \cdot L_{q_1} \cup b.
\]

We obtain the following set of equations.

\[
\begin{align*}
L_{q_0} &= (a \cup bb) \cdot L_{q_0} \cup ba \cdot L_{q_1} \cup b \\
L_{q_1} &= b \cdot L_{q_1} \cup a \cdot L_{q_0}
\end{align*}
\]

Let $L = L_{q_1}$, $B = b$, and $C = a \cdot L_{q_0}$. Then $\epsilon \notin B$ and the second equation reads $L = BL \cup C$. Hence, by Lemma 2.8.2,

\[
L_{q_1} = L = B^* C = b^* a \cdot L_{q_0}.
\]
2.8. Equivalence of regular expressions and regular languages

If we substitute $L_{q_1}$ into the first equation, then we get (again using Theorem 2.7.4)

$$L_{q_0} = (a \cup bb) \cdot L_{q_0} \cup ba \cdot b^* a \cdot L_{q_0} \cup b$$
$$= (a \cup bb \cup bab^* a) L_{q_0} \cup b.$$

Again applying Lemma 2.8.2, this time with $L = L_{q_0}$, $B = a \cup bb \cup bab^* a$ and $C = b$, gives

$$L_{q_0} = (a \cup bb \cup bab^* a)^* b.$$

Thus, the regular expression that describes the language accepted by $M$ is

$$(a \cup bb \cup bab^* a)^* b.$$

Completing the correctness of the conversion

It remains to prove that, for any DFA, the system of equations (2.2) and (2.3) can be solved. This will follow from the following (more general) lemma. (You should verify that the equations (2.2) and (2.3) are in the form as specified in this lemma.)

**Lemma 2.8.3** Let $n \geq 1$ be an integer and, for $1 \leq i \leq n$ and $1 \leq j \leq n$, let $B_{ij}$ and $C_i$ be regular expressions such that $\epsilon \notin B_{ij}$. Let $L_1, L_2, \ldots, L_n$ be languages that satisfy

$$L_i = \left( \bigcup_{j=1}^{n} B_{ij} L_j \right) \cup C_i \text{ for } 1 \leq i \leq n.$$

Then $L_1$ can be expressed as a regular expression only involving the regular expressions $B_{11}$ and $C_1$.

**Proof.** The proof is by induction on $n$. The base case is when $n = 1$. In this case, we have

$$L_1 = B_{11} L_1 \cup C_1.$$

Since $\epsilon \notin B_{11}$, it follows from Lemma 2.8.2 that $L_1 = B_{11}^* C_1$. This proves the base case.
Let \( n \geq 2 \) and assume the lemma is true for \( n - 1 \). We have

\[
L_n = \left( \bigcup_{j=1}^{n} B_{nj}L_j \right) \cup C_n \\
= B_{nn}L_n \cup \left( \bigcup_{j=1}^{n-1} B_{nj}L_j \right) \cup C_n.
\]

Since \( \epsilon \notin B_{nn} \), it follows from Lemma 2.8.2 that

\[
L_n = B_{nn}^* \left( \left( \bigcup_{j=1}^{n-1} B_{nj}L_j \right) \cup C_n \right) \\
= B_{nn}^* \left( \bigcup_{j=1}^{n-1} B_{nj}L_j \right) \cup B_{nn}^* C_n \\
= \left( \bigcup_{j=1}^{n-1} B_{nn}^* B_{nj}L_j \right) \cup B_{nn}^* C_n
\]

By substituting this equation for \( L_n \) into the equations for \( L_i \), \( 1 \leq i \leq n - 1 \), we obtain

\[
L_i = \left( \bigcup_{j=1}^{n} B_{ij}L_j \right) \cup C_i \\
= B_{in}L_n \cup \left( \bigcup_{j=1}^{n-1} B_{ij}L_j \right) \cup C_i \\
= \left( \bigcup_{j=1}^{n-1} (B_{in}B_{nn}^* B_{nj} \cup B_{ij})L_j \right) \cup B_{in}B_{nn}^* C_i \cup C_i.
\]

Thus, we have obtained \( n - 1 \) equations in \( L_1, L_2, \ldots, L_{n-1} \). Since \( \epsilon \notin B_{in}B_{nn}^* B_{nj} \cup B_{ij} \), it follows from the induction hypothesis that \( L_1 \) can be expressed as a regular expression only involving the regular expressions \( B_{ij} \) and \( C_i \). \[ \blacksquare \]
2.9  The pumping lemma and nonregular languages

In the previous sections, we have seen that the class of regular languages is closed under various operations, and that these languages can be described by (deterministic or nondeterministic) finite automata and regular expressions. These properties helped in developing techniques for showing that a language is regular. In this section, we will present a tool that can be used to prove that certain languages are not regular. Observe that for a regular language,

1. the amount of memory that is needed to determine whether or not a given string is in the language is finite and independent of the length of the string, and

2. if the language consists of an infinite number of strings, then this language should contain infinite subsets having a fairly repetitive structure.

Intuitively, languages that do not follow 1. or 2. should be nonregular. For example, consider the language

\[ \{0^n1^n : n \geq 0\} \]

This language should be nonregular, because it seems unlikely that a DFA can remember how many 0s it has seen when it has reached the border between the 0s and the 1s. Similarly the language

\[ \{0^n : n \text{ is a prime number}\} \]

should be nonregular, because the prime numbers do not seem to have any repetitive structure that can be used by a DFA. To be more rigorous about this, we will establish a property that all regular languages must possess. This property is called the pumping lemma. If a language does not have this property, then it must be nonregular.

The pumping lemma states that any sufficiently long string in a regular language can be pumped, i.e., there is a section in that string that can be repeated any number of times, so that the resulting strings are all in the language.
Theorem 2.9.1 (Pumping Lemma for Regular Languages) Let \( A \) be a regular language. Then there exists an integer \( p \geq 1 \), called the pumping length, such that the following holds: Every string \( s \) in \( A \), with \( |s| \geq p \), can be written as \( s = xyz \), such that

1. \( y \neq \epsilon \) (i.e., \( |y| \geq 1 \)),
2. \( |xy| \leq p \), and
3. for all \( i \geq 0 \), \( xy^i z \in A \).

In words, the pumping lemma states that by replacing the portion \( y \) in \( s \) by zero or more copies of it, the resulting string is still in the language \( A \).

Proof. Let \( \Sigma \) be the alphabet of \( A \). Since \( A \) is a regular language, there exists a DFA \( M = (Q, \Sigma, \delta, q_0, F) \) that accepts \( A \). We define \( p \) to be the number of states in \( Q \).

Let \( s = s_1s_2 \ldots s_n \) be an arbitrary string in \( A \) such that \( n \geq p \). Define \( r_1 = q, r_2 = \delta(r_1, s_1), r_3 = \delta(r_2, s_2), \ldots, r_{n+1} = \delta(r_n, s_n) \). Thus, when the DFA \( M \) reads the string \( s \) from left to right, it visits the states \( r_1, r_2, \ldots, r_{n+1} \). Since \( s \) is a string in \( A \), we know that \( r_{n+1} \) belongs to \( F \).

Consider the first \( p + 1 \) states \( r_1, r_2, \ldots, r_{p+1} \) in this sequence. Since the number of states of \( M \) is equal to \( p \), the pigeon hole principle implies that there must be a state that occurs twice in this sequence. That is, there are indices \( j \) and \( \ell \) such that \( 1 \leq j < \ell \leq p + 1 \) and \( r_j = r_{\ell} \).

We define \( x = s_1s_2 \ldots s_{j-1}, y = s_j \ldots s_{\ell-1}, \) and \( z = s_\ell \ldots s_n \). Since \( j < \ell \), we have \( y \neq \epsilon \), proving the first claim in the theorem. Since \( \ell \leq p + 1 \), we have \( |xy| = \ell - 1 \leq p \), proving the second claim in the theorem. To see that
the third claim also holds, recall that the string \( s = xyz \) is accepted by \( M \).
While reading \( x \), \( M \) moves from the start state \( q \) to state \( r_j \). While reading \( y \), it moves from state \( r_j \) to state \( r_\ell = r_j \), i.e., after having read \( y \), \( M \) is again in state \( r_j \). While reading \( z \), \( M \) moves from state \( r_j \) to the accept state \( r_{n+1} \).
Therefore, the substring \( y \) can be repeated any number \( i \geq 0 \) of times, and the corresponding string \( xy^iz \) will still be accepted by \( M \). It follows that \( xy^iz \in A \) for all \( i \geq 0 \).

\[ \begin{align*}
2.9.1 \quad \text{Applications of the pumping lemma} \\
\text{First example} \\
\text{Consider the language } A = \{0^n1^n : n \geq 0\}. \\
\text{We will prove by contradiction that } A \text{ is not a regular language.} \\
\text{Assume that } A \text{ is a regular language. Let } p \geq 1 \text{ be the pumping length,} \\
as given by the pumping lemma. Consider the string } s = 0^p1^p. \text{ It is clear} \\
\text{that } s \in A \text{ and } |s| = 2p \geq p. \text{ Hence, by the pumping lemma, } s \text{ can be} \\
\text{written as } s = xyz, \text{ where } y \neq \epsilon, |xy| \leq p, \text{ and } xy^iz \in A \text{ for all } i \geq 0. \\
\text{Observe that, since } |xy| \leq p, \text{ the string } y \text{ contains only 0s. Moreover,} \\
since y \neq \epsilon, y \text{ contains at least one 0. But now we are in trouble: None of} \\
\text{the strings } xy^0z = xz, xy^2z = xyyz, xy^3z = xyyyz, \ldots, \text{ is contained in } A. \\
\text{However, by the pumping lemma, all these strings must be in } A. \text{ Hence, we} \\
\text{have a contradiction and we conclude that } A \text{ is not a regular language.} \\
\text{Second example} \\
\text{Consider the language } A = \{w \in \{0,1\}^* : \text{ the number of 0s in } w \text{ equals the number of 1s in } w\}. \\
\text{Again, we prove by contradiction that } A \text{ is not a regular language.} \\
\text{Assume that } A \text{ is a regular language. Let } p \geq 1 \text{ be the pumping length,} \\
as given by the pumping lemma. Consider the string } s = 0^p1^p. \text{ Then } s \in A \\
\text{and } |s| = 2p \geq p. \text{ By the pumping lemma, } s \text{ can be written as } s = xyz, \\
\text{where } y \neq \epsilon, |xy| \leq p, \text{ and } xy^iz \in A \text{ for all } i \geq 0. \\
\text{Since } |xy| \leq p, \text{ the string } y \text{ contains only 0s. Since } y \neq \epsilon, y \text{ contains at} \\
\text{least one 0. Therefore, the string } xy^2z = xyyz \text{ contains more 0s than 1s,} \]
which implies that this string is not contained in \( A \). But, by the pumping lemma, this string is contained in \( A \). This is a contradiction and, therefore, \( A \) is not a regular language.

**Third example**

Consider the language

\[
A = \{ww : w \in \{0, 1\}^* \}.
\]

We prove by contradiction that \( A \) is not a regular language.

Assume that \( A \) is a regular language. Let \( p \geq 1 \) be the pumping length, as given by the pumping lemma. Consider the string \( s = 0^p10^p \). Then \( s \in A \) and \( |s| = 2p + 2 \geq p \). By the pumping lemma, \( s \) can be written as \( s = xyz \), where \( y \neq \epsilon \), \( |xy| \leq p \), and \( xy^iz \in A \) for all \( i \geq 0 \).

Since \( |xy| \leq p \), the string \( y \) contains only 0s. Since \( y \neq \epsilon \), \( y \) contains at least one 0. Therefore, the string \( xy^2z = xyyz \) is not contained in \( A \). But, by the pumping lemma, this string is contained in \( A \). This is a contradiction and, therefore, \( A \) is not a regular language.

You should convince yourself that by choosing \( s = 0^p \) (which is a string in \( A \) whose length is at least \( p \)), we do not obtain a contradiction. The reason is that the string \( y \) may have an even length. Thus, \( 0^p \) is the “wrong” string for showing that \( A \) is not regular. By choosing \( s = 0^p10^p \), we do obtain a contradiction; thus, this is the “correct” string for showing that \( A \) is not regular.

**Fourth example**

Consider the language

\[
A = \{0^m1^n : m > n \geq 0 \}.
\]

We prove by contradiction that \( A \) is not a regular language.

Assume that \( A \) is a regular language. Let \( p \geq 1 \) be the pumping length, as given by the pumping lemma. Consider the string \( s = 0^{p+1}1p \). Then \( s \in A \) and \( |s| = 2p + 1 \geq p \). By the pumping lemma, \( s \) can be written as \( s = xyz \), where \( y \neq \epsilon \), \( |xy| \leq p \), and \( xy^iz \in A \) for all \( i \geq 0 \).

Since \( |xy| \leq p \), the string \( y \) contains only 0s. Since \( y \neq \epsilon \), \( y \) contains at least one 0. Consider the string \( xy^0z = xz \). The number of 1s in this string
is equal to \( p \), whereas the number of 0s is at most equal to \( p \). Therefore, the string \( xy^0z \) is not contained in \( A \). But, by the pumping lemma, this string is contained in \( A \). This is a contradiction and, therefore, \( A \) is not a regular language.

**Fifth example**

Consider the language

\[
A = \{ 1^{n^2} : n \geq 0 \}.
\]

We prove by contradiction that \( A \) is not a regular language.

Assume that \( A \) is a regular language. Let \( p \geq 1 \) be the pumping length, as given by the pumping lemma. Consider the string \( s = 1^{p^2} \). Then \( s \in A \) and \( |s| = p^2 \geq p \). By the pumping lemma, \( s \) can be written as \( s = xyz \), where \( y \neq \epsilon \), \( |xy| \leq p \), and \( xy^i z \in A \) for all \( i \geq 0 \).

Observe that

\[
|s| = |xyz| = p^2
\]

and

\[
|x^{y^2}z| = |xyyz| = |xyz| + |y| = p^2 + |y|.
\]

Since \( |xy| \leq p \), we have \( |y| \leq p \). Since \( y \neq \epsilon \), we have \( |y| \geq 1 \). It follows that

\[
p^2 < |x^{y^2}z| \leq p^2 + p < (p + 1)^2.
\]

Hence, the length of the string \( x^{y^2}z \) is strictly between two consecutive squares. It follows that this length is not a square and, therefore, \( x^{y^2}z \) is not contained in \( A \). But, by the pumping lemma, this string is contained in \( A \). This is a contradiction and, therefore, \( A \) is not a regular language.

**Sixth example**

Consider the language

\[
A = \{ 1^n : n \text{ is a prime number} \}.
\]

We prove by contradiction that \( A \) is not a regular language.

Assume that \( A \) is a regular language. Let \( p \geq 1 \) be the pumping length, as given by the pumping lemma. Let \( n \geq p \) be a prime number, and consider the string \( s = 1^n \). Then \( s \in A \) and \( |s| = n \geq p \). By the pumping lemma, \( s \) can be written as \( s = xyz \), where \( y \neq \epsilon \), \( |xy| \leq p \), and \( xy^i z \in A \) for all \( i \geq 0 \).
Let $k$ be the integer such that $y = 1^k$. Since $y \neq \epsilon$, we have $k \geq 1$. For each $i \geq 0$, $n + (i - 1)k$ is a prime number, because $xy^iz = 1^{n+(i-1)k} \in A$. For $i = n + 1$, however, we have

$$n + (i - 1)k = n + nk = n(1 + k),$$

which is not a prime number, because $n \geq 2$ and $1 + k \geq 2$. This is a contradiction and, therefore, $A$ is not a regular language.

**Seventh example**

Consider the language

$$A = \{w \in \{0, 1\}^* : \text{the number of occurrences of 01 in } w \text{ is equal to the number of occurrences of 10 in } w \}.\$$

Since this language has the same flavor as the one in the second example, we may suspect that $A$ is not a regular language. This is, however, not true: As we will show, $A$ is a regular language.

The key property is the following one: Let $w$ be an arbitrary string in $\{0, 1\}^*$. Then

the absolute value of the number of occurrences of 01 in $w$ minus the number of occurrences of 10 in $w$ is at most one.

This property holds, because between any two consecutive occurrences of 01, there must be exactly one occurrence of 10. Similarly, between any two consecutive occurrences of 10, there must be exactly one occurrence of 01.

We will construct a DFA that accepts $A$. This DFA uses the following five states:

- $q$: start state; no symbol has been read.
- $q_{01}$: the last symbol read was 1; in the part of the string read so far, the number of occurrences of 01 is one more than the number of occurrences of 10.
- $q_{10}$: the last symbol read was 0; in the part of the string read so far, the number of occurrences of 10 is one more than the number of occurrences of 01.
The set of accept states is equal to \( \{ q, q^0_{\text{equal}}, q^1_{\text{equal}} \} \). The state diagram of the DFA is given below.

In fact, the key property mentioned above implies that the language \( A \) consists of the empty string \( \epsilon \) and all non-empty binary strings that start and end with the same symbol. As a result, \( A \) is the language described by the regular expression

\[
\epsilon \cup 0 \cup 1 \cup 0(0 \cup 1)^*0 \cup 1(0 \cup 1)^*1.
\]

This gives an alternative proof for the fact that \( A \) is a regular language.
2.10 Higman’s Theorem

Let $\Sigma$ be a finite alphabet. For any two strings $x$ and $y$ in $\Sigma^*$, we say that $x$ is a subsequence of $y$, if $x$ can be obtained by deleting zero or more symbols from $y$. For example, 10110 is a subsequence of 0010010101010001. For any language $L \subseteq \Sigma^*$, we define

$$\text{SUBSEQ}(L) := \{x : \text{there exists a } y \in L \text{ such that } x \text{ is a subsequence of } y\}.$$ 

That is, $\text{SUBSEQ}(L)$ is the language consisting of the subsequences of all strings in $L$. In 1952, Higman proved the following result:

**Theorem 2.10.1 (Higman)** For any finite alphabet $\Sigma$ and for any language $L \subseteq \Sigma^*$, the language $\text{SUBSEQ}(L)$ is regular.

2.10.1 Dickson’s Theorem

Our proof of Higman’s Theorem will use a theorem that was proved in 1913 by Dickson.

Recall that $\mathbb{N}$ denotes the set of positive integers. Let $n \in \mathbb{N}$. For any two points $p = (p_1, p_2, \ldots, p_n)$ and $q = (q_1, q_2, \ldots, q_n)$ in $\mathbb{N}^n$, we say that $p$ is dominated by $q$, if $p_i \leq q_i$ for all $i$ with $1 \leq i \leq n$.

**Theorem 2.10.2 (Dickson)** Let $S \subseteq \mathbb{N}^n$, and let $M$ be the set consisting of all elements of $S$ that are minimal in the relation “is dominated by”. Thus,

$$M = \{q \in S : \text{there is no } p \text{ in } S \setminus \{q\} \text{ such that } p \text{ is dominated by } q\}.$$ 

Then, the set $M$ is finite.

We will prove this theorem by induction on the dimension $n$. If $n = 1$, then either $M = \emptyset$ (if $S = \emptyset$) or $M$ consists of exactly one element (if $S \neq \emptyset$). Therefore, the theorem holds if $n = 1$. Let $n \geq 2$ and assume the theorem holds for all subsets of $\mathbb{N}^{n-1}$. Let $S$ be a subset of $\mathbb{N}^n$ and consider the set $M$ of minimal elements in $S$. If $S = \emptyset$, then $M = \emptyset$ and, thus, $M$ is finite. Assume that $S \neq \emptyset$. We fix an arbitrary element $q$ in $M$. If $p \in M \setminus \{q\}$, then $q$ is not dominated by $p$. Therefore, there exists an index $i$ such that $p_i \leq q_i - 1$. It follows that

$$M \setminus \{q\} \subseteq \bigcup_{i=1}^{n} (\mathbb{N}^{i-1} \times [1, q_i - 1] \times \mathbb{N}^{n-i}).$$
For all \(i\) and \(k\) with \(1 \leq i \leq n\) and \(1 \leq k \leq q_i - 1\), we define
\[
S_{ik} = \{ p \in S : p_i = k \}
\]
and
\[
M_{ik} = \{ p \in M : p_i = k \}.
\]
Then,
\[
M \setminus \{q\} = \bigcup_{i=1}^{n} \bigcup_{k=1}^{q_i-1} M_{ik}.
\] (2.4)

**Lemma 2.10.3** \(M_{ik}\) is a subset of the set of all elements of \(S_{ik}\) that are minimal in the relation “is dominated by”.

**Proof.** Let \(p\) be an element of \(M_{ik}\), and assume that \(p\) is not minimal in \(S_{ik}\). Then there is an element \(r\) in \(S_{ik}\), such that \(r \neq p\) and \(r\) is dominated by \(p\). Since \(p\) and \(r\) are both elements of \(S\), it follows that \(p \not\in M\). This is a contradiction.

Since the set \(S_{ik}\) is basically a subset of \(\mathbb{N}^{n-1}\), it follows from the induction hypothesis that \(S_{ik}\) contains finitely many minimal elements. This, combined with Lemma 2.10.3, implies that \(M_{ik}\) is a finite set. Thus, by (2.4), \(M \setminus \{q\}\) is the union of finitely many finite sets. Therefore, the set \(M\) is finite.

### 2.10.2 Proof of Higman’s Theorem

We give the proof of Theorem 2.10.1 for the case when \(\Sigma = \{0, 1\}\). If \(L = \emptyset\) or \(\text{SUBSEQ}(L) = \{0, 1\}^*\), then \(\text{SUBSEQ}(L)\) is obviously a regular language. Hence, we may assume that \(L\) is non-empty and \(\text{SUBSEQ}(L)\) is a proper subset of \(\{0, 1\}^*\).

We fix a string \(z\) of length at least two in the complement \(\overline{\text{SUBSEQ}(L)}\) of the language \(\text{SUBSEQ}(L)\). Observe that this is possible, because \(\overline{\text{SUBSEQ}(L)}\) is an infinite language. We insert 0s and 1s into \(z\), such that, in the resulting string \(z'\), 0s and 1s alternate. For example, if \(z = 00110101\), then \(z' = 010101010101\). Let \(n = |z'| - 1\), where \(|z'|\) denotes the length of \(z'\). Then, \(n \geq |z| - 1 \geq 1\).

A \((0,1)\)-*alternation* in a binary string \(x\) is any occurrence of 01 or 10 in \(x\). For example, the string 1101001 contains four \((0,1)\)-alternations. We define
\[
A = \{ x \in \{0, 1\}^* : \text{ \(x\) has at most \(n\) many \((0,1)\)-alternations} \}.
\]
Lemma 2.10.4 \( \text{SUBSEQ}(L) \subseteq A \). 

**Proof.** Let \( x \in \text{SUBSEQ}(L) \) and assume that \( x \not\in A \). Then, \( x \) has at least \( n + 1 = |z'| \) many \((0,1)\)-alternations and, therefore, \( z' \) is a subsequence of \( x \). In particular, \( z \) is a subsequence of \( x \). Since \( x \in \text{SUBSEQ}(L) \), it follows that \( z \in \text{SUBSEQ}(L) \), which is a contradiction. \( \blacksquare \)

Lemma 2.10.5 \( \overline{\text{SUBSEQ}(L)} = \left( A \cap \overline{\text{SUBSEQ}(L)} \right) \cup \overline{A} \).

**Proof.** Follows from Lemma 2.10.4. \( \blacksquare \)

Lemma 2.10.6 The language \( \overline{A} \) is regular.

**Proof.** The complement \( \overline{A} \) of \( A \) is the language consisting of all binary strings with at least \( n + 1 \) many \((0,1)\)-alternations. If, for example, \( n = 3 \), then \( \overline{A} \) is described by the regular expression 

\[
(00^*11^*00^*11^*0(0 \cup 1)^*) \cup (11^*00^*11^*00^*1(0 \cup 1)^*)
\]

This should convince you that the claim is true for any value of \( n \). \( \blacksquare \)

For any \( b \in \{0,1\} \) and for any \( k \geq 0 \), we define \( A_{bk} \) to be the language consisting of all binary strings that start with a \( b \) and have exactly \( k \) many \((0,1)\)-alternations. Then, we have 

\[
A = \{\epsilon\} \cup \left( \bigcup_{b=0}^{1} \bigcup_{k=0}^{n} A_{bk} \right).
\]

Thus, if we define 

\[
F_{bk} = A_{bk} \cap \overline{\text{SUBSEQ}(L)},
\]

and use the fact that \( \epsilon \in \text{SUBSEQ}(L) \) (which is true because \( L \neq \emptyset \)), then 

\[
A \cap \overline{\text{SUBSEQ}(L)} = \bigcup_{b=0}^{1} \bigcup_{k=0}^{n} F_{bk}.
\] (2.5)
For any \( b \in \{0, 1\} \) and for any \( k \geq 0 \), consider the relation “is a subsequence of” on the language \( F_{bk} \). We define \( M_{bk} \) to be the language consisting of all strings in \( F_{bk} \) that are minimal in this relation. Thus,

\[
M_{bk} = \{ x \in F_{bk} : \text{there is no } x' \in F_{bk} \setminus \{ x \} \text{ such that } x' \text{ is a subsequence of } x \}.
\]

It is clear that

\[
F_{bk} = \bigcup_{x \in M_{bk}} \{ y \in F_{bk} : x \text{ is a subsequence of } y \}.
\]

If \( x \in M_{bk}, y \in A_{bk}, \) and \( x \) is a subsequence of \( y \), then \( y \) must be in \( \text{SUBSEQ}(L) \) and, therefore, \( y \) must be in \( F_{bk} \). To prove this, assume that \( y \in \text{SUBSEQ}(L) \). Then, \( x \in \text{SUBSEQ}(L) \), contradicting the fact that \( x \in M_{bk} \subseteq F_{bk} \subseteq \text{SUBSEQ}(L) \). It follows that

\[
F_{bk} = \bigcup_{x \in M_{bk}} \{ y \in A_{bk} : x \text{ is a subsequence of } y \}. \tag{2.6}
\]

**Lemma 2.10.7** Let \( b \in \{0, 1\} \) and \( 0 \leq k \leq n \), and let \( x \) be an element of \( M_{bk} \). Then, the language

\[
\{ y \in A_{bk} : x \text{ is a subsequence of } y \}
\]

is regular.

**Proof.** We will prove the claim by means of an example. Assume that \( b = 1, k = 3, \) and \( x = 11110001000 \). Then, the language

\[
\{ y \in A_{bk} : x \text{ is a subsequence of } y \}
\]

is described by the regular expression

\[
1111*0000*11*0000*.
\]

This should convince you that the claim is true in general.

**Lemma 2.10.8** For each \( b \in \{0, 1\} \) and each \( 0 \leq k \leq n \), the set \( M_{bk} \) is finite.
Chapter 2. Finite Automata and Regular Languages

Proof. Again, we will prove the claim by means of an example. Assume that \( b = 1 \) and \( k = 3 \). Any string in \( F_{bk} \) can be written as \( 1^a 0^b 1^c 0^d \), for some integers \( a, b, c, d \geq 1 \). Consider the function \( \varphi : F_{bk} \to \mathbb{N}^4 \) that is defined by \( \varphi(1^a 0^b 1^c 0^d) = (a, b, c, d) \). Then, \( \varphi \) is an injective function, and the following is true, for any two strings \( x \) and \( x' \) in \( F_{bk} \):

\[
x \text{ is a subsequence of } x' \text{ if and only if } \varphi(x) \text{ is dominated by } \varphi(x').
\]

It follows that the elements of \( M_{bk} \) are in one-to-one correspondence with those elements of \( \varphi(F_{bk}) \) that are minimal in the relation “is dominated by”. The lemma thus follows from Dickson’s Theorem.

Now we can complete the proof of Higman’s Theorem:

- It follows from (2.6) and Lemmas 2.10.7 and 2.10.8, that \( F_{bk} \) is the union of finitely many regular languages. Therefore, by Theorem 2.3.1, \( F_{bk} \) is a regular language.

- It follows from (2.5) that \( A \cap \overline{\text{SUBSEQ}(L)} \) is the union of finitely many regular languages. Therefore, again by Theorem 2.3.1, \( A \cap \overline{\text{SUBSEQ}(L)} \) is a regular language.

- Since \( A \cap \overline{\text{SUBSEQ}(L)} \) is regular and, by Lemma 2.10.6, \( \overline{A} \) is regular, it follows from Lemma 2.10.5 that \( \overline{\text{SUBSEQ}(L)} \) is the union of two regular languages. Therefore, by Theorem 2.3.1, \( \overline{\text{SUBSEQ}(L)} \) is a regular language.

- Since \( \overline{\text{SUBSEQ}(L)} \) is regular, it follows from Theorem 2.6.4 that the language \( \text{SUBSEQ}(L) \) is regular as well.

Exercises

2.1 For each of the following languages, construct a DFA that accepts the language. In all cases, the alphabet is \( \{0, 1\} \).

1. \( \{w : \text{the length of } w \text{ is divisible by three}\} \)
2. \( \{w : 110 \text{ is not a substring of } w\} \)
3. \( \{w : w \text{ contains at least five 1s}\} \)
4. \( \{ w : w \text{ contains the substring 1011} \} \)
5. \( \{ w : w \text{ contains at least two 1s and at most two 0s} \} \)
6. \( \{ w : w \text{ contains an odd number of 1s or exactly two 0s} \} \)
7. \( \{ w : w \text{ begins with 1 and ends with 0} \} \)
8. \( \{ w : \text{ every odd position in } w \text{ is 1} \} \)
9. \( \{ w : w \text{ has length at least 3 and its third symbol is 0} \} \)
10. \( \{ \epsilon, 0 \} \)

2.2 For each of the following languages, construct an NFA, with the specified number of states, that accepts the language. In all cases, the alphabet is \( \{ 0, 1 \} \).

1. The language \( \{ w : w \text{ ends with 10} \} \) with three states.
2. The language \( \{ w : w \text{ contains the substring 1011} \} \) with five states.
3. The language \( \{ w : w \text{ contains an odd number of 1s or exactly two 0s} \} \) with six states.

2.3 For each of the following languages, construct an NFA that accepts the language. In all cases, the alphabet is \( \{ 0, 1 \} \).

1. \( \{ w : w \text{ contains the substring 11001} \} \)
2. \( \{ w : w \text{ has length at least 2 and does not end with 10} \} \)
3. \( \{ w : w \text{ begins with 1 or ends with 0} \} \)

2.4 Convert the following NFA to an equivalent DFA.

![Diagram](image-url)
2.5 Convert the following NFA to an equivalent DFA.

2.6 Convert the following NFA to an equivalent DFA.

2.7 In the proof of Theorem 2.6.3, we introduced a new start state $q_0$, which is also an accept state. Explain why the following is not a valid proof of Theorem 2.6.3:

Let $N = (Q_1, \Sigma, \delta_1, q_1, F_1)$ be an NFA, such that $A = L(N)$. Define the NFA $M = (Q_1, \Sigma, \delta, q_1, F)$, where

1. $F = \{q_1\} \cup F_1$.

2. $\delta : Q_1 \times \Sigma \rightarrow \mathcal{P}(Q_1)$ is defined as follows: For any $r \in Q_1$ and for any $a \in \Sigma$,

\[
\delta(r, a) = \begin{cases} 
\delta_1(r, a) & \text{if } r \in Q_1 \text{ and } a \notin F_1, \\
\delta_1(r, a) & \text{if } r \in F_1 \text{ and } a \neq \epsilon, \\
\delta_1(r, a) \cup \{q_1\} & \text{if } r \in F_1 \text{ and } a = \epsilon.
\end{cases}
\]
Then $L(M) = A^*$.

2.8 Prove Theorem 2.6.4.

2.9 Let $A$ be a language over the alphabet $\Sigma = \{0, 1\}$ and let $\overline{A}$ be the complement of $A$. Thus, $\overline{A}$ is the language consisting of all binary strings that are not in $A$.

Assume that $A$ is a regular language. Let $M = (Q, \Sigma, \delta, q, F)$ be a non-deterministic finite automaton (NFA) that accepts $A$.

Consider the NFA $N = (Q, \Sigma, \delta, q, \overline{F})$, where $\overline{F} = Q \setminus F$ is the complement of $F$. Thus, $N$ is obtained from $M$ by turning all accept states into nonaccept states, and turning all nonaccept states into accept states.

1. Is it true that the language accepted by $N$ is equal to $\overline{A}$?

2. Assume now that $M$ is a deterministic finite automaton (DFA) that accepts $A$. Define $N$ as above; thus, turn all accept states into nonaccept states, and turn all nonaccept states into accept states. Is it true that the language accepted by $N$ is equal to $\overline{A}$?

2.10 Recall the alternative definition for the star of a language $A$ that we gave just before Theorem 2.3.1.

In Theorems 2.3.1 and 2.6.2, we have shown that the class of regular languages is closed under the union and concatenation operations. Since $A^* = \bigcup_{k=0}^{\infty} A^k$, why doesn’t this imply that the class of regular languages is closed under the star operation?

2.11 Let $A$ and $B$ be two regular languages over the same alphabet $\Sigma$. Prove that the difference of $A$ and $B$, i.e., the language

$$A \setminus B = \{w : w \in A \text{ and } w \not\in B\}$$

is a regular language.

2.12 For each of the following regular expressions, give two strings that are members and two strings that are not members of the language described by the expression. The alphabet is $\Sigma = \{a, b\}$.

1. $a(ba)^*b$.

2. $(a \lor b)^*a(a \lor b)^*b(a \lor b)^*a(a \lor b)^*$. 


3. \((a \cup ba \cup bb)(a \cup b)^*\).

2.13 Give regular expressions describing the following languages. In all cases, the alphabet is \(\{0, 1\}\).

1. \(\{w : w \text{ contains at least three } 1\text{s}\}\).
2. \(\{w : w \text{ contains at least two } 1\text{s and at most one } 0\}\),
3. \(\{w : w \text{ contains an even number of } 0\text{s and exactly two } 1\text{s}\}\).
4. \(\{w : w \text{ contains exactly two } 0\text{s and at least two } 1\text{s}\}\).
5. \(\{w : w \text{ contains an even number of } 0\text{s and each } 0 \text{ is followed by at least one } 1\}\).
6. \(\{w : \text{ every odd position in } w \text{ is } 1\}\).

2.14 Convert each of the following regular expressions to an NFA.

1. \((0 \cup 1)^*000(0 \cup 1)^*
2. \(((10)^*(00)) \cup 10)^*
3. \((0 \cup 1)(11)^* \cup 0)^*

2.15 Convert the following DFA to a regular expression.

2.16 Convert the following DFA to a regular expression.
2.17 Convert the following DFA to a regular expression.

2.18 1. Let $A$ be a non-empty regular language. Prove that there exists an NFA that accepts $A$ and that has exactly one accept state.

2. For any string $w = w_1w_2\ldots w_n$, we denote by $w^R$ the string obtained by reading $w$ backwards, i.e., $w^R = w_nw_{n-1}\ldots w_1$. For any language $A$, we define $A^R$ to be the language obtained by reading all strings in $A$ backwards, i.e.,

$$A^R = \{w^R : w \in A\}.$$  

Let $A$ be a non-empty regular language. Prove that the language $A^R$ is also regular.

2.19 If $n \geq 1$ is an integer and $w = a_1a_2\ldots a_n$ is a string, then for any $i$ with $0 \leq i < n$, the string $a_1a_2\ldots a_i$ is called a proper prefix of $w$. (If $i = 0$, then $a_1a_2\ldots a_i = \epsilon$.)

For any language $L$, we define $\text{MIN}(L)$ to be the language

$$\text{MIN}(L) = \{w \in L : \text{ no proper prefix of } w \text{ belongs to } L\}.$$  

Prove the following claim: If $L$ is a regular language, then $\text{MIN}(L)$ is regular as well.
2.20 Use the pumping lemma to prove that the following languages are not regular.

1. \( \{a^nb^mc^{n+m} : n \geq 0, m \geq 0 \} \).
2. \( \{a^nb^nc^{2n} : n \geq 0 \} \).
3. \( \{a^nb^ma^n : n \geq 0, m \geq 0 \} \).
4. \( \{a^{2n} : n \geq 0 \} \). (Remark: \( a^{2n} \) is the string consisting of \( 2^n \) many \( a \)'s.)
5. \( \{a^nb^mc^k : n \geq 0, m \geq 0, k \geq 0, n^2 + m^2 = k^2 \} \).
6. \( \{uvu : u \in \{a, b\}^*, v \in \{a, b\}^* \} \).

2.21 Prove that the language

\[ \{a^nb^n : m \geq 0, n \geq 0, m \neq n \} \]

is not regular. (Using the pumping lemma for this one is a bit tricky. You can avoid using the pumping lemma by combining results about the closure under regular operations.)

2.22 1. Give an example of a regular language \( A \) and a non-regular language \( B \) for which \( A \subseteq B \).
2. Give an example of a non-regular language \( A \) and a regular language \( B \) for which \( A \subseteq B \).

2.23 Let \( A \) be a language consisting of finitely many strings.

1. Prove that \( A \) is a regular language.
2. Let \( n \) be the maximum length of any string in \( A \). Prove that every deterministic finite automaton (DFA) that accepts \( A \) has at least \( n + 1 \) states. (Hint: How is the pumping length chosen in the proof of the pumping lemma?)

2.24 Let \( L \) be a regular language, let \( M \) be a DFA whose language is equal to \( L \), and let \( p \) be the number of states of \( M \). Prove that \( L \neq \emptyset \) if and only if \( L \) contains a string of length less than \( p \).
2.25 Let $L$ be a regular language, let $M$ be a DFA whose language is equal to $L$, and let $p$ be the number of states of $M$. Prove that $L$ is an infinite language if and only if $L$ contains a string $w$ with $p \leq |w| \leq 2p - 1$.

2.26 Let $\Sigma$ be a non-empty alphabet, and let $L$ be a language over $\Sigma$, i.e., $L \subseteq \Sigma^*$. We define a binary relation $R_L$ on $\Sigma^* \times \Sigma^*$, in the following way: For any two strings $u$ and $u'$ in $\Sigma^*$,

$$uR_Lu' \text{ if and only if } (\forall v \in \Sigma^*: uv \in L \iff u'v \in L).$$

Prove that $R_L$ is an equivalence relation.

2.27 Let $\Sigma = \{0, 1\}$, let

$$L = \{w \in \Sigma^*: |w| \text{ is odd}\},$$

and consider the relation $R_L$ defined in Exercise 2.26.

1. Prove that for any two strings $u$ and $u'$ in $\Sigma^*$,

$$uR_Lu' \iff |u| - |u'| \text{ is even.}$$

2. Determine all equivalence classes of the relation $R_L$.

2.28 Let $\Sigma$ be a non-empty alphabet, and let $L$ be a language over $\Sigma$, i.e., $L \subseteq \Sigma^*$. Recall the equivalence relation $R_L$ that was defined in Exercise 2.26.

1. Assume that $L$ is a regular language, and let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that accepts $L$. Let $u$ and $u'$ be strings in $\Sigma^*$. Let $q$ be the state reached, when following the path in the state diagram of $M$, that starts in $q_0$ and that is obtained by reading the string $u$. Similarly, let $q'$ be the state reached, when following the path in the state diagram of $M$, that starts in $q_0$ and that is obtained by reading the string $u'$.

Prove the following: If $q = q'$, then $uR_Lu'$.

2. Prove the following claim: If $L$ is a regular language, then the equivalence relation $R_L$ has a finite number of equivalence classes.
2.29 Let $L$ be the language defined by

$$L = \{ uu^R : u \in \{0, 1\}^* \}.$$ 

In words, a string is in $L$ if and only if its length is even, and the second half is the reverse of the first half. Consider the equivalence relation $R_L$ that was defined in Exercise 2.26.

1. Let $m$ and $n$ be two distinct positive integers and consider the two strings $u = 0^m1$ and $u' = 0^n1$. Prove that $\neg (uR_Lu')$.

2. Prove that $L$ is not a regular language, without using the pumping lemma.

3. Use the pumping lemma to prove that $L$ is not a regular language.

2.30 In this exercise, we will show that the converse of the pumping lemma does, in general, not hold. Consider the language

$$A = \{ a^m b^n c^n : m \geq 1, n \geq 0 \} \cup \{ b^n c^k : n \geq 0, k \geq 0 \}.$$ 

1. Show that $A$ satisfies the conclusion of the pumping lemma for $p = 1$. Thus, show that every string $s$ in $A$ whose length is at least $p$ can be written as $s = xyz$, such that $y \neq \epsilon$, $|xy| \leq p$, and $xy^iz \in A$ for all $i \geq 0$.

2. Consider the equivalence relation $R_A$ that was defined in Exercise 2.26. Let $n$ and $n'$ be two distinct non-negative integers and consider the two strings $u = ab^n$ and $u' = ab^{n'}$. Prove that $\neg (uR_Au')$.

3. Prove that $A$ is not a regular language.