The definition of an automaton requires the transition function to be a \textit{function}.

On the other hand, in an NFA the transition function should get mapped to a \textit{set} of values.

An NFA accepts a string if a sequence of possible states leads to a final state.

- Only if no such sequence exists will the NFA reject the input string.

E.g. the previous NFA accepts the string 010110 because it can be processed as follows:

\[(q_0, 010110) \rightarrow (q_0, 10110) \rightarrow (q_5, 0110)\]
\[(q_5, 110) \rightarrow (q_3, 10) \rightarrow (q_3, 0) \rightarrow (q_3, \epsilon)\]

On the other hand, we can end up in a rejecting state:

\[(q_0, 010110) \rightarrow (q_0, 10110) \rightarrow (q_5, 0110)\]
\[\rightarrow (q_5, 110) \rightarrow (q_3, 10) \rightarrow (q_3, 0) \rightarrow (q_3, \epsilon)\]
Definition of an NFA

- Let \( P(A) = \{ B \mid B \subseteq A \} \) denote the power set of the set \( A \) and for an alphabet \( \Sigma; \Sigma_e = \Sigma \cup \{ \varepsilon \} \)

- A nondeterministic finite automaton is a 5-tuple \( N = (Q, \Sigma, \delta, q_0, F) \)
  - \( Q \) is a finite set of states,
  - \( \Sigma \) is a finite alphabet,
  - \( \delta: Q \times \Sigma_e \rightarrow P(Q) \) is the (set-valued) transition function, that also allows \( \varepsilon \)-transitions
  - \( q_0 \in Q \) is the start state, and
  - \( F \subseteq Q \) is the set of (accepting) final states

The transition function of the previous automaton is

<table>
<thead>
<tr>
<th>( \varepsilon )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( {q_0} )</td>
<td>( {q_0, q_1} )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( {q_2} )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( \emptyset )</td>
<td>( {q_3} )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( {q_3} )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

- Now we can easily express the error state as an empty set of possible next states
An NFA \( N = (Q, \Sigma, \delta, q_0, F) \) accepts the string \( w \),

- If we can write it as \( w = y_1 y_2 ... y_m \in \Sigma^* \) and
  a sequence of states \( r_p, r_{p+1}, ..., r_m \) exists in \( Q \) s.t.
  - \( r_0 = q_0 \),
  - \( r_{i+1} \in \delta(r_i, y_{i+1}) \), \( i = 0, ..., m-1 \), and
  - \( r_m \in F \).

DFAs are a special case of NFAs \( \Rightarrow \)
all languages that can be recognized using the former can also be recognized using the latter
Also the other way around: DFAs and NFAs recognize the same set of languages

**Theorem 1.39** Let \( A = L(N) \) be the language recognized by some NFA \( N \). There exists a DFA \( M \) such that \( L(M) = A \)

**Proof.** Let \( N = (Q, \Sigma, \delta, q_0, F) \). We construct a DFA
\( M = (Q', \Sigma, \delta', q_0', F') \) that simulates the computation of \( N \) in parallel in all its possible states at all times. Let us first consider the easier situation where \( N \) has no \( \varepsilon \) arrows.

Every state of \( M \) is a set of states of \( N \)
\( Q' = P(Q) \)
\( q_0' = \{ q_0 \} \)
\( F' = \{ R \in Q' \mid R \text{ contains an accept state } r \in F \} \)
\( \delta'(R, a) = \bigcup_{r \in R} \delta(r, a) \)
Without ε arrows

After Minimization
Let us check that \( L(M) = L(N) \). The equivalence of the languages follows when we prove for all \( x \in \Sigma^* \) and \( r \in Q \) that

\[
(q_0, x) \xrightarrow{\text{\( N \)}} (r, \varepsilon) \iff (|q_0\rangle, x) \xrightarrow{\text{\( M \)}} (R, \varepsilon) \text{ and } r \in R,
\]

where the notation \((q_0, x) \xrightarrow{\text{\( N \)}} (r, \varepsilon)\) means that in automaton \( N \) we can process the string \( x \) starting from state \( q_0 \) so that we end up in state \( r \) and there are no more symbols to process (\( \varepsilon \)).

We prove it using induction over the length of the string \( x \):

1. **Basis**: \(|x| = 0\): \((q_0, \varepsilon) \xrightarrow{\text{\( N \)}} (r, \varepsilon) \iff r = q_0\).
   Similarly \((|q_0\rangle, \varepsilon) \xrightarrow{\text{\( M \)}} (R, \varepsilon) \iff R = \{q_0\}\).

2. **Induction hypothesis**: the claim holds when \(|x| \leq k\).

3. \(|x| = k + 1\): Then \( x = ya \) for some \( y \), \(|y| = k \), for which the claim holds by the induction hypothesis. Now,

\[
(q_0, x) = (q_0, ya) \xrightarrow{\text{\( N \)}} (r', \varepsilon) \iff \exists r' \in Q \text{ s.t. } (q_0, ya) \xrightarrow{\text{\( N \)}} (r', a) \text{ and } (r', a) \xrightarrow{\text{\( N \)}} (r, \varepsilon)
\]

\( \iff \exists r' \in Q \text{ s.t. } (q_0, y) \xrightarrow{\text{\( N \)}} (r', \varepsilon) \text{ and } (r', a) \xrightarrow{\text{\( N \)}} (r, \varepsilon) \text{ in one transition} \)

\( \iff \exists r' \in Q \text{ s.t. } ((q_0), y) \xrightarrow{\text{\( M \)}} (R', \varepsilon) \text{ and } r' \in R' \text{ and } r \in \delta(r', a) \)

By induction hypothesis we get

\( \iff ((q_0), y) \xrightarrow{\text{\( M \)}} (R', \varepsilon) \text{ and } \exists r' \in R' \text{ s.t. } r \in \delta(r', a) \text{ Rearranging yields} \)

By the definition of the transition function \( \delta' \)
Let us return a and name \( \delta'(R', a) \) and \( (q_0, y) \in M(R', a) \) and \( r \in \delta'(R', a) \)

Concluding

\( (q_0, x) = (q_0, y) \in M(R, a) \) and \( r \in R \)

Which completes the proof of the claim.

In order to take the \( \varepsilon \) arrows into account, we compute for each state \( R \subseteq Q \) of \( M \) the collection of states that can be reached from \( R \) by going only along \( \varepsilon \) arrows:

\[
E(R) = \{ q | q \text{ can be reached from } R \text{ by traveling along } 0 \text{ or more } \varepsilon \text{ arrows} \}
\]

It is enough to modify the transition function of \( M \) and start state to take the \( \varepsilon \) arrows into account:

\[
\delta'(R, a) = \bigcup_{r \in E} E(\delta(r, a))
\]

\[
q_0' = E(q_0)
\]
With ε arrows

After Minimization
Theorem 1.45 \ The class of regular languages is closed under the union operation.

Proof. Let the languages \( A_1 \) and \( A_2 \) be regular. Then, there exists (nondeterministic) finite automata \( N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \) and \( N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \), which recognize these two languages. Let us construct an automaton \( N = (Q, \Sigma, \delta, q_0, F) \) for recognizing the language \( A_1 \cup A_2 \).

- \( Q = \{ q_0 \} \cup Q_1 \cup Q_2 \)
- The start state of \( N \) is \( q_0 \),
- \( F = F_1 \cup F_2 \) and

\[
\delta(q,a) = \begin{cases} 
\delta_1(q,a), & q \in Q_1 \\
\delta_2(q,a), & q \in Q_2 \\
\{q_1,q_2\}, & q = q_0 \text{ and } a = \varepsilon \\
\emptyset, & q = q_0 \text{ and } a \neq \varepsilon 
\end{cases}
\]
Theorem 1.47  The class of regular languages is closed under the concatenation operation.

Proof. Let the languages $A_1$ and $A_2$ be regular. Then, there exists (nondeterministic) finite automata $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, which recognize these two languages. Let us construct an automaton $N = (Q, \Sigma, \delta, q_1, F)$ for recognizing $A_1 \cdot A_2$.

- $Q = Q_1 \cup Q_2$,
- The start state of $N$ is $q_1$,
- The final states of $N$ are those in $F_2$ and

$$
\delta(q,a) = \begin{cases} 
\delta_1(q,a), & q \in Q_i \text{ and } q \notin F_i \\
\delta_2(q,a), & q \in F_i \text{ and } a \neq \epsilon \\
\delta_1(q,a) \cup \{q_2\}, & q \in F_i \text{ and } a = \epsilon \\
\delta_2(q,a), & q \in Q_2 
\end{cases}
$$

\[\square\]
Theorem 1.49. The class of regular languages is closed under the star operation.

Proof. Let the language $A$ be regular. Then, there exists a (nondeterministic) finite automaton $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, which recognizes the language.

Let us construct an automaton $N = (Q, \Sigma, \delta, q_0, F)$ for recognizing $A^*$.

- $Q = \{ q_0 \} \cup Q_1$,
- The new start state of $N$ is $q_0$,
- $F = \{ q_0 \} \cup F_1$ and

\[
\delta(q, a) = \begin{cases} 
\delta_1(q, a), & q \in Q_1 \text{ and } a \in \Sigma \\
\delta_1(q, a), & q \in F_1 \text{ and } a \in \Sigma \\
\delta_1(q, a) \cup \{ q_1 \}, & q \in F_1 \text{ and } a = \varepsilon \\
\{ q_1 \}, & q = q_0 \text{ and } a = \varepsilon \\
Q, & q = q_0 \text{ and } a \neq \varepsilon
\end{cases}
\]

$\square$
1.3 Regular Expressions

- These have an important role in describing patterns in searching for strings in many applications (e.g. awk, grep, Perl, ...)

All regular expressions of alphabet $\Sigma$ are
1. $\emptyset$ and $\varepsilon$ are regular expressions,
2. $a$ is a regular expression of $\Sigma$ for all $a \in \Sigma$,
3. if $R_1$ and $R_2$ are regular expressions, then also
   - $(R_1 \cup R_2)$,
   - $(R_1 : R_2)$ and
   - $R_1^*$
   are regular expressions

Each regular expression $R$ of $\Sigma$ represents a language $L(R)$
1. $L(\emptyset) = \emptyset$,
2. $L(\varepsilon) = \{\varepsilon\}$,
3. $L(a) = \{a\} \forall a \in \Sigma$,
4. $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$,
5. $L((R_1 : R_2)) = L(R_1) \cdot L(R_2)$ and
6. $L(R_1^*) = (L(R_1))^*$

Proper closure: $R^*$ is a shorthand for $RR^*$ (Kleene plus)
Observe: $R^* \cup \varepsilon = R^*$
   - Let $R^k$ be shorthand for the concatenation of $k$ $R$'s with each other.
Examples

\[0^*10^* = \{ w \mid \text{w contains a single 1}\}\]
\[\Sigma^*001\Sigma^* = \{ w \mid \text{w contains the string 001 as a substring}\}\]
\[1^*(01^*1)^* = \{ w \mid \text{every 0 in w is followed by at least one 1}\}\]
\[(\Sigma^*\epsilon)^* = \{ w \mid \text{w is a string of even length}\}\]
\[01 \cup 10 = \{ 01, 10 \}\]
\[0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{ w \mid \text{w starts and ends with the same symbol}\}\]
\[(0 \cup \epsilon)^*1^* = 01^* \cup 1^*\]
\[(0 \cup \epsilon) (1 \cup \epsilon) = \{ \epsilon, 0, 1, 01 \}\]
\[1^*\epsilon = \emptyset\]
\[\emptyset^* = \{ \epsilon \}\]

For any regular expression \(R\)

- \(R \cup \emptyset = R\) and
- \(R\epsilon = R\)

However, it may hold that

- \(R \cup \epsilon \neq R\) and
- \(R\emptyset \neq R\)

For example, the unsigned real numbers that can be recognized using the previous automaton can be expressed with the regular expression

\[d^*(d^* \cup \epsilon)(E (\cup \cup \cup \epsilon) d^* \cup \epsilon),\]

where \(d = \{ 0 \cup \ldots \cup 9 \}\)
Theorem 1.54  A language is regular if and only if some regular expression describes it.

We state and prove both directions of this theorem separately.

Lemma 1.55  If a language is described by a regular expression, then it is regular.

Proof. Any regular expression can be converted into a finite automaton, which recognizes the same language as that described by the regular expression.

There are only six rules by which regular expressions can be composed. The following pictures illustrate the NFA for each of these cases.
\[
\begin{align*}
r &= \emptyset \\
r &= \varepsilon \\
r &= a \\
r &= s \cup t
\end{align*}
\]
Lemma 1.60 If a language is regular, then it is described by a regular expression.

Proof. By definition a regular language can be recognized with a (nondeterministic) finite automaton, which can be converted into a generalized nondeterministic finite automaton (GNFA). The GNFA finally yields a regular expression that is equivalent with the original automaton.

Let $\text{RE}_\Sigma$ denote the set of regular expressions over $\Sigma$

- In a GNFA the transition function $\delta$ is a finite mapping $\delta: Q \times \text{RE}_\Sigma \rightarrow P(Q)$
- $(q, w) > (q', w')$ if $q' \in \delta(q, r)$ for some $r \in \text{RE}_\Sigma$ s.t. $w = zw', z \in L(r)$
A GNFA $M$ can be reduced into a regular expression which describes the language recognized by $M$.

1. We compress $M$ into a GNFA with only 2 states (so that the language recognized remains equivalent).
   1. The accept states of $M$ are replaced by a single one ($\varepsilon$ arrows).
   2. We remove all other states $q$ except the start state and final state.
      Let $q_i$ and $q_j$ be the predecessor and successor of $q$ on some route passing through $q$.
      Now we can remove $q$ and rename the arrow between $q_i$ and $q_j$ with a new expression.
2. Eventually the GNFA contains at most two states. It is easy to convert the language recognized into a regular expression.