2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
We want to define a program which recognises strings “start” and “stop”. We start by defining a program which recognises the letter “s”. This can be given by a system given by the following diagram, which will be our first automaton (automata will be introduced soon).
Example Recognising Strings, Step 1

This automaton has the following ingredients.

- States $q_0$, $q_1$.
- State $q_0$ is the starting state, indicated by the arrow into it coming from nowhere.
- State $q_1$, the state indicating that we have recognised letter “s”.
- A transition, which when recognising letter “s”, goes from state $q_0$ to $q_1$. 
Example Recognising Strings, Step 2

In order to recognise the letter “t”, we extend this automaton as follows:
Example Recognising Strings, Step 2

$q_0$ is the start state.
$q_1$ indicates we have read string “s”.
$q_2$ indicates we have read string “st”.

Diagram:

$q_0$\rightarrow s \rightarrow q_1 \rightarrow t \rightarrow q_2
For the 3rd letter, we have two choices: “a” as part of the word “start”, and “o” as part of the word “stop”.

![Diagram of a finite automaton](image-url)
Example Recognising Strings, Step 3

- $q_0$ is the start state.
- $q_1$ indicates we have read string “s”.
- $q_2$ indicates we have read string “st”.
- $q_3$ indicates we have read string “sta”.
- $q_6$ indicates we have read string “sto”.
Example Recognising Strings, Step 3

We can complete our automaton and obtain the following:

![Diagram of an automaton with states q0, q1, q2, q3, q4, q5, q6, q7 and transitions labeled with s, t, a, o, p, r, and t.](attachment:automaton_diagram.png)
Example Recognising Strings, Step 3

This diagram contains a new ingredient: States $q_5$ and $q_7$ are accepting states. If we have processed a word and reached such a state then the word is accepted as a string of the language of the automaton.
(In our example $L = \{\text{start, stop}\}$.)
Automata with Loops

In order to recognise infinite languages, we need automata with loops. The following automaton recognises $L = \ldots$
Automata with Loops

The following automaton recognises $L = ???$
Automata with Loops

The following automaton recognises $L = \cdots$
The language \{\text{start, stop}\} can be as well recognised by the following \textit{nondeterministic automaton}:
The automaton chooses in state $q_0$, non-deterministically, when in state $q_0$ and recognising a letter $s$, whether to go to $q_1$ or $q_6$.

The accepted language is the set of strings such that for each of them we obtain an accepting state for at least one non-deterministic choice.
If we try to accept the word “stop” by moving $q_0 \xrightarrow{s} q_1 \xrightarrow{t} q_2$ we get stuck at $q_2$.

That we fail to accept a word for one specific non-deterministic choice doesn’t imply that this word is not in the language.

A word is not accepted only if for all non-deterministic choices the corresponding run of the automaton doesn’t accept the string.
Why Nondeterministic Automata?

- When translating regular grammars into automata, we will obtain non-deterministic automata.
- We will show later that from a non-deterministic automaton we can obtain an equivalent deterministic automaton.
- Non-deterministic machine models play an important role in the theory of algorithms and complexity.
  - In some cases (as for automata), deterministic and non-deterministic are equivalent.
  - Sometimes they are not.
  - In other cases it is an open problem whether they are equivalent.
2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
**Definition NFA**

A **non-deterministic finite state automaton**, in short **NFA** $(Q, q_0, F, T, \delta)$ is given by

- A finite set $Q$ of **states**.
- A single **initial state** $q_0$.
- A set $F \subseteq Q$ of **accepting states**.
- A finite set of **terminal symbols** $T$.
- A function $\delta : Q \times T \rightarrow \mathcal{P}(Q)$.

Here $\mathcal{P}(Q)$ is the set of subsets of $Q$. $\delta(q, a)$ gives the set of possible next states that the automaton can be in after reading the terminal symbol $a \in T$ when in state $q$. 
Presentation of DFA by Picture

An NFA can be presented by a picture, like the following diagram:

- The arrow from nowhere into state $q_0$ denotes that $q_0$ is the initial state.
- Circles denote states.
- Arrows from a state $q$ to $q'$ labelled by $a$ mean that $q'$ is one element of $\delta(q, a)$.
- Double circle like for $q_5$ and $q_9$ denote the accepting states.
### Presentation of DFA by Table

<table>
<thead>
<tr>
<th>automaton</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, \ldots, q_n$</td>
</tr>
<tr>
<td>terminals</td>
<td>$a_0, \ldots, a_m$</td>
</tr>
<tr>
<td>start</td>
<td>$q_i$</td>
</tr>
<tr>
<td>final</td>
<td>$q_{i_0}, q_{i_1}, \ldots, q_{i_m}$</td>
</tr>
<tr>
<td>transitions</td>
<td>$\delta(q_j, a_i) = {q_{k_1}, \ldots, q_{k_l}}$</td>
</tr>
</tbody>
</table>

We omit $\delta(q, a)$, if $\delta(q, a) = \emptyset$. 
Example

The automaton given by the diagram

is represented by the table on the next slide.
### Example

<table>
<thead>
<tr>
<th>automaton</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7$</td>
</tr>
<tr>
<td>terminals</td>
<td>$a, o, p, r, s, t$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_5, q_9$</td>
</tr>
<tr>
<td>transitions</td>
<td></td>
</tr>
<tr>
<td>$\delta(q_0, s)$</td>
<td>${q_1, q_6}$,</td>
</tr>
<tr>
<td>$\delta(q_1, t)$</td>
<td>$q_2$,</td>
</tr>
<tr>
<td>$\delta(q_2, a)$</td>
<td>$q_3$,</td>
</tr>
<tr>
<td>$\delta(q_3, r)$</td>
<td>$q_4$,</td>
</tr>
<tr>
<td>$\delta(q_4, t)$</td>
<td>$q_5$,</td>
</tr>
<tr>
<td>$\delta(q_6, t)$</td>
<td>$q_7$,</td>
</tr>
<tr>
<td>$\delta(q_7, o)$</td>
<td>$q_8$,</td>
</tr>
<tr>
<td>$\delta(q_8, p)$</td>
<td>$q_9$.</td>
</tr>
</tbody>
</table>
The Extended Transition Function

We define an extended transition function, which determines the set of states, an NFA can reach from a state $q$, when reading a word $w \in T^*$:

**Definition**

Let $M = (Q, q_0, F, T, \delta)$ be an NFA. Then we define

$$\delta^* : Q \times T^* \to \mathcal{P}(Q)$$

by induction on the length of words $w \in T^*$:

- $\delta^*(q, \epsilon) := \{q\}$.
- If $w = aw'$ for $a \in T$, $w' \in T^*$, $\delta(q, a) = \{q_1, \ldots, q_m\}$ ($m = 0$ possible), then $\delta^*(q, w) = \bigcup_{i=1}^{n} \delta^*(q_i, w')$. 
Example

For the automaton

we give some definitions of $\delta^*$:
Example

- $\delta^*(q_0, \epsilon) = \{q_0\}$.
- $\delta^*(q_0, s) = \{q_1, q_6\}$.
- $\delta^*(q_0, st) = \{q_2, q_7\}$.
- $\delta^*(q_0, sta) = \{q_3\}$.
- $\delta^*(q_0, star) = \{q_4\}$.
- $\delta^*(q_0, start) = \{q_5\}$.
- $\delta^*(q_0, sto) = \{q_8\}$.
- $\delta^*(q_0, stop) = \{q_9\}$. 
The Language accepted by an NFA

Definition

Let $A = (Q, q_0, F, T, \delta)$ be an NFA. The **language accepted by** $A$ is defined as

$$L(A) := \{w \in T^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$
Operational Understanding of NFAs

An NFA $A = (Q, q_0, F, T, \delta)$ can be understood as well as a program. A run of an NFA on an input string $s$ is as follows:

- $q := q_0$
  - $p$ is a pointer pointing initially to the beginning of $s$
  - stopped := false.

- While $p$ doesn’t point to the end of $s$ and stopped = false do:
  - Let $a$ be the next symbol from the string.
  - If $\delta(q, a) = \emptyset$, stopped := true.
  - Otherwise choose non-deterministically an element $q' \in \delta(q, a)$, and set $q$ to $q'$.
    - Move the pointer to the next symbol of the input string.

- If stopped = true, the string is not accepted in this run.
- Otherwise the string is accepted if $q \in F$.

A string is accepted if there exist a sequence of non-deterministic choices, such that the string is accepted in the corresponding run.
Operational Understanding of NFAs

One can now see that a string $s$ is accepted by an automaton $A$ in the sense above iff $s \in L(A)$. 
2.3.3. Examples of Automata (13.3)

2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
Example 1: An Automaton accepting \{1, 2, 3\}
Example 1: An Automaton accepting \{1, 2, 3\}

Display style:

<table>
<thead>
<tr>
<th>automaton</th>
<th>(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>(q_0, q_1, q_2, q_3)</td>
</tr>
<tr>
<td>terminals</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>start</td>
<td>(q_0)</td>
</tr>
<tr>
<td>final</td>
<td>(q_1, q_2, q_3)</td>
</tr>
</tbody>
</table>
| transitions    | \(\delta(q_0, 1) = \{q_1\}\)  
                     \(\delta(q_0, 2) = \{q_2\}\)  
                     \(\delta(q_0, 3) = \{q_3\}\) |
Simplified Version
Simplified Version

Shorter pictorial presentation of the same automaton:

\[ q_0 \xrightarrow{1,2,3} q_1 \]
### Display style:

<table>
<thead>
<tr>
<th><strong>automaton</strong></th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>states</strong></td>
<td>$q_0, q_1$</td>
</tr>
<tr>
<td><strong>terminals</strong></td>
<td>$1, 2, 3$</td>
</tr>
<tr>
<td><strong>start</strong></td>
<td>$q_0$</td>
</tr>
<tr>
<td><strong>final</strong></td>
<td>$q_1$</td>
</tr>
<tr>
<td><strong>transitions</strong></td>
<td>$\delta(q_0, 0) = {q_1}$</td>
</tr>
<tr>
<td></td>
<td>$\delta(q_0, 1) = {q_1}$</td>
</tr>
<tr>
<td></td>
<td>$\delta(q_0, 2) = {q_1}$</td>
</tr>
</tbody>
</table>
Example 2: Automaton accepting ???

![Diagram of an automaton accepting 0, 1, ..., 9]
### Simplified Version

**Display style:**

<table>
<thead>
<tr>
<th>automaton</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0$</td>
</tr>
<tr>
<td>terminals</td>
<td>$0, \ldots, 9$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_0$</td>
</tr>
<tr>
<td>transitions</td>
<td>$\delta(q_0, a) = {q_0}$ ($a \in {0, 1, \ldots, 9}$)</td>
</tr>
</tbody>
</table>
Example 3: Automaton accepting $\ldots, 9$

$\begin{array}{c}
q_0 \\
1, 2, \ldots, 9 \\
q_1 \\
0, 1, \ldots, 9 \\
q_2
\end{array}$
Display Style

Display style:

<table>
<thead>
<tr>
<th>automaton</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, q_1, q_2$</td>
</tr>
<tr>
<td>terminals</td>
<td>$0, \ldots, 9$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_1, q_2$</td>
</tr>
</tbody>
</table>

transitions

$\delta(q_0, 0) = \{q_2\}$,
$\delta(q_0, a) = \{q_1\}$, ($a \in \{1, \ldots, 9\}$)
$\delta(q_1, a) = \{q_1\}$, ($a \in \{0, \ldots, 9\}$)
2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
Automaton with empty move transitions are like NFAs, but have the possibility to make transitions without consuming a letter. These transitions are labelled as $\lambda$.

\[ q_0 \xrightarrow{\lambda} q_1 \]

\[ a \xrightarrow{\lambda} q_0 \]

\[ q_1 \xrightarrow{\lambda} b \]
**Display style**

<table>
<thead>
<tr>
<th>automaton</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>states</td>
<td>$q_0, q_1$</td>
</tr>
<tr>
<td>terminals</td>
<td>$a, b$</td>
</tr>
<tr>
<td>start</td>
<td>$q_0$</td>
</tr>
<tr>
<td>final</td>
<td>$q_1$</td>
</tr>
</tbody>
</table>
| transitions | $\delta(q_0, a) = \{ q_0 \}$  
|             | $\delta(q_0, \lambda) = \{ q_1 \}$,  
|             | $\delta(q_1, b) = \{ q_1 \}$. |
Definition NFA with empty moves

A **non-deterministic finite state automaton with empty moves** \((Q, q_0, F, T, \delta)\) is given by

- A finite set \(Q\) of **states**.
- A single **initial state** \(q_0\).
- A set \(F \subseteq Q\) of **accepting states**.
- A finite set of **terminal symbols** \(T\).
- A function \(\delta: Q \times (T \cup \{\lambda\}) \rightarrow \mathcal{P}(Q)\).
We extend the function $\delta^*$ to NFA with empty moves as follows:

**Definition**

Let $M = (Q, q_0, F, T, \delta)$ be an NFA with empty moves. Then we define

$$\delta^*: Q \times (\{\lambda\} \cup T^*) \rightarrow \mathcal{P}(Q)$$

- We first define $\delta^*(q, \lambda)$ inductively:
  - $q \in \delta^*(q, \lambda)$.
  - If $q' \in \delta^*(q, \lambda)$ and $q'' \in \delta(q', \lambda)$, then $q'' \in \delta^*(q, \lambda)$.
- For $a \in T$ we define
  $$\delta^*(q, a) := \{q'' | \exists q' \in \delta^*(q, \lambda). q'' \in \delta(q, a)\}$$
2.3.4. Automata with Empty Move Transitions (13.4)

The Extended Transition Function

Definition (Cont)

Now we define \( \delta(q, w) \) by induction on the length of \( w \):

- \( \delta^*(q, \epsilon) := \delta^*(q, \lambda) \).
- \( \delta^*(q, a) \) is defined as before.
- If \( w = aw' \) for \( a \in T, w' \in T^+ \). Then

\[
\delta^*(q, aw) := \{ q'' | \exists q' \in \delta^*(q, a). q'' \in \delta^*(q', w) \}
\]
Informal Understanding

So \( q \in \delta(q, a_1 \cdots a_n) \) if we can reach from \( q \) using finitely many \( \lambda \)-transitions, then one \( a_1 \)-transition, then again finitely many \( \lambda \)-transitions, an \( a_2 \) transition, \ldots, finitely many \( \lambda \)-transitions, and an \( a_n \) transition:

\[
q_0 \xrightarrow{\lambda} q_1 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} q_1 \xrightarrow{a_1} q_1 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} q_2 \xrightarrow{a_2} q_2 \xrightarrow{\lambda} \cdots \xrightarrow{\lambda} q' \in F
\]
Language Accepted

**Definition**

Let $M = (Q, q_0, F, T, \delta)$ be an NFA with empty moves. Then $L(A) := \{w \in T^* \mid \exists q \in \delta^*(q_0, w).\delta^*(q, \lambda) \cap F \neq \emptyset\}$. 
NFA with Empty Moves are Equivalent to NFAs

We show that for every NFA $M$ with empty moves we can find an NFA $M'$ without any moves s.t. $L(M') = L(M)$.

This is done as follows:

Let $M = (Q, q_0, F, T, \delta)$.

$M'$ is obtained from $M$ by

- replacing $\delta$ by $\delta^*$, i.e. replacing $\delta(q, a)$ by $\delta^*(q, a)$.
- replacing $F$ by $F' := \{q \in Q. \delta^*(q, \lambda) \cap F \neq \emptyset\}$.

So the transitions are obtained by allowing first finitely many empty moves and then one proper transition.

At the end we might need to make finitely many empty moves before reaching the accepting state, therefore the set of accepting states is the set of states from which we can reach an accepting state of $M$ using empty moves.
Correctness of the Translation

- One can now easily see that $\delta_{M'}^*(q, w) = \delta_M^*(q, w)$.
- Now it follows

$$L(M') = \{ w \in T^* | \delta_{M'}^*(q, w) \cap F' \neq \emptyset \}$$
$$= \{ w \in T^* | \exists q' \in \delta_{M'}^*(q, w).\delta_M^*(q', \lambda) \cap F \neq \emptyset \}$$
$$= \{ w \in T^* | \exists q' \in \delta_M^*(q, w).\delta_M^*(q', \lambda) \cap F \neq \emptyset \}$$
$$= L(M)$$
Example

Consider the NFA with empty moves from above:

\[ q_0 \xrightarrow{\lambda} q_1 \]

\[ a \]
Example

The transformed automaton is as follows:

\[ q_0 \xrightarrow{a} q_0 \quad q_0 \xrightarrow{b} q_1 \quad q_1 \xrightarrow{b} q_1 \]
Theorem

For any NFA $M$ with empty moves there exist an NFA without empty moves s.t. $L(M') = L(M)$. 
2.3.1. String Recognition (13.1)
2.3.2. Nondeterministic Finite State Automata (13.2)
2.3.3. Examples of Automata (13.3)
2.3.4. Automata with Empty Move Transitions (13.4)
2.3.5. Deterministic Finite State Automata (13.6)
2.3.6. Regular Grammars and NFAs (13.5)
2.3.7. Translating NFAs into Regular Expressions (13.10)
2.3.8. Equivalence Theorem
2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)
2.3.10. Closure Properties and Decidability of Regular Languages
Definition DFA

Let $M = (Q, q_0, F, T, \delta)$ be an NFA. $M$ is a **deterministic finite state automaton**, in short **DFA**, if for all $q \in Q$, $a \in T$ $\delta(q, a)$ has at most one element.

So deterministic finite state automata are those automata corresponding to real programs: we have never to make a choice.
Notation

1. If $M = (Q, q_0, F, T, \delta)$ is an NFA, we write
   - $q \xrightarrow{a} q'$ for $q' \in \delta(q, a)$.
   - $q \xrightarrow{w} q'$ for $q' \in \delta^*(q, w)$.

2. If $M = (Q, q_0, F, T, \delta)$ is a DFA, we often write $\delta(q, a) = q'$ if $q \xrightarrow{a} q'$, and $\delta^*(q, w) = q'$ if $q \xrightarrow{w} q'$.

   So we consider $\delta, \delta^*$ as partial functions $Q \times A \to Q$.

Note that with this notation we have for DFAs:

$\delta^*(q, \epsilon) = q$, $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$.
Theorem

Let $M$ be an NFA. Then there exists a DFA $M'$ s.t. $L(M) = L(M')$. 
Proof Idea

- We will define a new automaton $M' = (Q', q'_0, F', T, \delta')$.
- $Q'$ is the set of all subsets of $Q$, i.e. $P(Q)$.
  - Having reached state $\{q_1, \ldots, q_k\}$ means that $\{q_1, \ldots, q_k\}$ are the set of states we could have reached in $M$ by making different choices, but following the same word.
- $q'_0 := \{q_0\}$.
  - Initially the states we have reached are the elements of $\{q_0\}$.
- We set
  $$\{q_1, \ldots, q_k\} \xrightarrow{a} \{q'_1, \ldots, q'_l\}$$
  if $\{q'_1, \ldots, q'_l\}$ are the set of states we can reach from a state $q_i$ following an arrow labelled by $a$.
  - If we could have reached any of the states $\{q_1, \ldots, q_k\}$, then after reading $a$ in addition, we could have reached states $\{q'_1, \ldots, q'_l\}$. 
Proof Idea

- The accepting states are the set of states containing at least one accepting state.
  - If having read word $w$ we can reach the states $\{q_1, \ldots, q_k\}$, then the word $w$ can be accepted, if one of $q_1, \ldots, q_k$ is an accepting state.
Resulting DFA

automaton  $M'$

states  $\mathcal{P}(Q)$

terminals  $T$

start  $\{q_0\}$

final  $A \in \mathcal{P}(Q)$ s.t. $A \cap F \neq \emptyset$

transitions  $\delta' (\{q_1, \ldots, q_k\}, a) = \delta(q_1, a) \cup \delta(q_2, a) \cup \cdots \cup \delta(q_k, a)$
Simplification

- Usually only some states of $M'$ are reachable.
- We can omit all unreachable states and get an equivalent automaton.
- We can construct the reachable states of $M'$ by starting with $\{q_0\}$, and constructing from there systematically all transitions and the states reached.
- Furthermore, there will be a state $\emptyset$.
  - When we have reached that state we have consumed a word for which there is no complete run of $M$.
  - $\emptyset \not\in F'$, $\delta'(\emptyset, a) = \emptyset$.
  - So $\emptyset$ is a sink, a state from which we cannot escape, and which doesn’t accept anything.
  - If we omit $\emptyset$, we obtain a DFA with the same language.
Example 1

Consider the following NFA accepting \{start, stop\}:
We obtain up to renaming of states the DFA we defined originally:
Example 2

Consider the following NFA accepting $L = \ldots$:
Corresponding DFA
Formal Proof

Consider the DFA as given above.

We show for \( A := \{q_1, \ldots, q_k\} \subseteq Q \) that
\[
\delta'^*(A, w) = \delta^*(q_1, w) \cup \cdots \cup \delta^*(q_k, w)
\]
by induction on the length of \( w \):

We write \( \delta^*(\{q_1, \ldots, q_k\}, w) \) for \( \delta^*(q_1, w) \cup \cdots \cup \delta^*(q_k, w) \), and have to show for \( A \subseteq Q \) that
\[
\delta'^*(A, w) = \delta^*(A, w).
\]

Case \( w = \epsilon \): \( \delta'^*(A, \epsilon) = A = \delta^*(A, \epsilon) \).

Case \( w = aw' \):

\[
\begin{align*}
\delta'^*(A, w) &= \delta'^*(A, aw') \\
&= \delta'^*(\delta'(A, a), w') \\
&= \delta'^*(\delta(A, a), w') \\
\text{IH} &= \delta^*(\delta(A, a), w') \\
&= \delta^*(A, aw') = \delta^*(A, w)
\end{align*}
\]
Proof

We obtain now

\[
L(M') = \{ w \in T^* | \delta'((\{q_0\}, w)) \in F' \}
\]

\[
= \{ w \in T^* | \delta'((\{q_0\}, w)) \cap F \neq \emptyset \}
\]

\[
= \{ w \in T^* | \delta((q_0, w)) \cap F \neq \emptyset \}
\]

\[
= L(M)
\]
2.3.1. String Recognition (13.1)
2.3.2. Nondeterministic Finite State Automata (13.2)
2.3.3. Examples of Automata (13.3)
2.3.4. Automata with Empty Move Transitions (13.4)
2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)
2.3.7. Translating NFAs into Regular Expressions (13.10)
2.3.8. Equivalence Theorem
2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)
2.3.10. Closure Properties and Decidability of Regular Languages
We will show that regular expressions coincide with regular languages and with languages recognised by a DFA or NFA. Here we prove one part of this result:

**Theorem**

*For every right linear grammar $G$ there exists an NFA $M$ s.t.*

$$L(G) = L(M)$$
Proof Idea

- A derivation of a word in $G$ has the form

$$S = A_0 \longrightarrow a_1 A_1 \longrightarrow a_1 a_2 A_2 \longrightarrow \cdots \longrightarrow a_1 a_2 \cdots a_{n-1} A_{n-1} \longrightarrow a_1 a_2 \cdots a_{n-1} a_n$$

where we have productions

$$A_i \longrightarrow a_{i+1} A_{i+1} \quad A_n \longrightarrow a_n$$

or

$$S = A_0 \longrightarrow a_1 A_1 \longrightarrow a_1 a_2 A_2 \longrightarrow \cdots \longrightarrow a_1 a_2 \cdots a_{n-1} A_{n-1} \longrightarrow a_1 a_2 \cdots a_{n-1}$$

where we have productions

$$A_i \longrightarrow a_{i+1} A_{i+1} \quad A_n \longrightarrow \epsilon$$
Proof Idea

Define $M$ with states $N \cup \{q_F\}$ for a special new accepting state $q_F$ s.t. the derivation

$$S = A_0 \rightarrow a_1A_1 \rightarrow a_1a_2A_2 \rightarrow \cdots \rightarrow a_1a_2\cdots a_{n-1}A_{n-1} \rightarrow a_1a_2\cdots a_n$$

corresponds to a sequence of transitions

$$S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} q_F$$

and a derivation

$$S = A_0 \rightarrow a_1A_1 \rightarrow a_1a_2A_2 \rightarrow \cdots \rightarrow a_1a_2\cdots a_{n-1}A_{n-1} \rightarrow a_1a_2\cdots a_{n-1}$$

corresponds to a sequence of transitions

$$S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \in F$$
Proof Idea

So we have:

- If $A \rightarrow aA'$, then $A \xrightarrow{a} A'$.
- If $A \rightarrow a$ then $A \xrightarrow{a} q_F$.
- $q_F \in F$.
- If $A \rightarrow \epsilon$, then $A \in F$. 
We obtain from $G = (N, T, S, P)$ the following NFA:

- **automaton** $M$
- **states** $N \cup \{ q_F \}$
- **terminals** $T$
- **start** $S$
- **final** $A \in N$ s.t. $A \rightarrow \epsilon$.
  - $q_F$
- **transitions** $\delta(A, a) = \{ A' \mid A \rightarrow aA' \} \cup \{ q_F \mid A \rightarrow a \}$
Formal Proof

We show that \( L(M) = L(G) \):

- Assume \( w = a_1 \cdots a_n \in L(M) \).
  
  Then there exists a sequence of transitions in \( A \)
  
  \[
  S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} q_F
  \]
  
  or
  
  \[
  S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} A_n \in F
  \]
  
  But from this we obtain derivations
  
  \[
  S = A_0 \rightarrow a_1 A_1 \rightarrow a_1 a_2 A_2 \rightarrow \cdots \rightarrow a_1 a_2 \cdots a_{n-1} A_{n-1} \rightarrow a_1 a_2 \cdots a_{n-1}a_n = w
  \]
  
  or
  
  \[
  S = A_0 \rightarrow a_1 A_1 \rightarrow a_1 a_2 A_2 \rightarrow \cdots \rightarrow a_1 a_2 \cdots a_n A_n \rightarrow a_1 a_2 \cdots a_n = w
  \]
  
  So \( w \in L(G) \).
Assume $w = a_1 \cdots a_n \in L(G)$. A derivation will have the form

$$S = A_0 \rightarrow a_1 A_1 \rightarrow a_1 a_2 A_2 \rightarrow \cdots \rightarrow a_1 a_2 \cdots a_{n-1} A_{n-1} \rightarrow a_1 a_2 \cdots a_{n-1} a_n = w$$

or

$$S = A_0 \rightarrow a_1 A_1 \rightarrow a_1 a_2 A_2 \rightarrow \cdots \rightarrow a_1 a_2 \cdots a_n A_n \rightarrow a_1 a_2 \cdots a_n = w$$

Then there exists a sequence of transitions in $A$

$$S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_{n-1}} A_{n-1} \xrightarrow{a_n} q_F$$

or

$$S = A_0 \xrightarrow{a_1} A_1 \xrightarrow{a_2} A_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} A_n \in F$$

So $w \in L(M)$. 
Example

Consider the Grammar:

<table>
<thead>
<tr>
<th>grammar</th>
<th>$G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminals</td>
<td>$0, 1$</td>
</tr>
<tr>
<td>nonterminals</td>
<td>$S, T$</td>
</tr>
<tr>
<td>start symbol</td>
<td>$S$</td>
</tr>
</tbody>
</table>
| productions | $S \rightarrow 0, S \rightarrow 1T,$  
               $T \rightarrow 0T, T \rightarrow 1T,$  
               $T \rightarrow \epsilon, T \rightarrow 0, T \rightarrow 1$ |
Corresponding Automaton

(Note that it is nondeterministic).
Corresponding Automaton

With corresponding rules:

- $S \rightarrow 0$
- $S \rightarrow 1$
- $T \rightarrow 0$
- $T \rightarrow 1$
- $T \rightarrow 0T$
- $T \rightarrow 1T$
- $T \rightarrow \epsilon$

Accepting state because of $T \rightarrow \epsilon$
2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
Theorem

Let $M = (Q, q_0, F, T, \delta)$ be an NFA. Then there exist a regular expression $L'$ s.t. $L' = L(M)$.

Before proving it we give an example:
Example

Consider the following automaton for the language $L = \Sigma$:

We derive regular expressions and simplify them at each intermediate step in order to keep them simple. We write \{0, 1\} for \{0\}|\{1\} and similar notations.
From $M$ to $L_{q,q'}^\emptyset$

Original automaton:

We derive $L_{q,q'}^\emptyset$, which is the set of strings which allows us to get from $q$ to $q'$ with intermediate states in $\emptyset$, i.e. without any intermediate states: This language is

$\left\{ a_1 \right\} | \cdots | \left\{ a_n \right\},$ if $q \neq q'$ and we have transitions $q \xrightarrow{a_i} q'$,

$\left\{ a_1 \right\} | \cdots | \left\{ a_n \right\} | \{ \epsilon \},$ if $q = q'$ and we have transitions $q \xrightarrow{a_i} q'$. 
Calculation of $L_{q,q'}^\emptyset$

Original automaton:

```
  q0  0  q1
  1   0,1
```

- $L_{q_0,q_0}^\emptyset = \{1\} \cup \{\epsilon\} = \{1, \epsilon\}$
- $L_{q_0,q_1}^\emptyset = \{0\}$
- $L_{q_1,q_0}^\emptyset = \emptyset$
- $L_{q_1,q_1}^\emptyset = \{0\} \cup \{1\} \cup \{\epsilon\} = \{0, 1, \epsilon\}$
2.3.7. Translating NFAs into Regular Expressions (13.10)

From $M$ to $L_{q,q'}^\emptyset$

Original automaton:

States with $L_{q,q'}^\emptyset$:
We derive $L_{q,q'}^{q_0}$, which is the set of strings which allows us to get from $q$ to $q'$ with intermediate states in $\{q_0\}$. This language is

$$L_{q,q'}^{q_0} = L_{q,q'}^{\emptyset} \mid (L_{q,q'}^{\emptyset} (L_{q_0,q_0}^{\emptyset})^* L_{q_0,q'}^{\emptyset})$$
2.3.7. Translating NFAs into Regular Expressions (13.10)

Calculation of \( L_{q_0}^{q_0} \)

\[
L_{q_0}^{q_0} = L_{q_0, q_0}^{q_0} \mid (L_{q_0, q_0}^{q_0} (L_{q_0, q_0}^{q_0})^* L_{q_0, q_0}^{q_0})
\]

\[
L_{q_0, q_0}^{q_0} = \{ 1, \epsilon \} \mid (\{ 1, \epsilon \}.\{ 1, \epsilon \}^*.\{ 1, \epsilon \})
\]
\[
= \{ 1 \}^*
\]

\[
L_{q_0, q_1}^{q_0} = \{ 0 \} \mid (\{ 1, \epsilon \}.\{ 1, \epsilon \}^*.\{ 0 \})
\]
\[
= \{ 1 \}^*.\{ 0 \}
\]

\[
L_{q_1, q_0}^{q_0} = \emptyset \mid (\emptyset.\{ 1, \epsilon \}^*.\{ 0 \})
\]
\[
= \emptyset
\]

\[
L_{q_1, q_1}^{q_0} = \{ 0, 1, \epsilon \} \mid (\emptyset.\{ 1, \epsilon \}^*.\{ 0 \})
\]
\[
= \{ 0, 1, \epsilon \}
\]
From $L_{q,q'}^\emptyset$ to $L_{q,q'}^{q_0}$

States with $L_{q,q'}^\emptyset$:

States with $L_{q,q'}^{q_0}$:
From $L_{q, q'}^{q_0}$ to $L_{q, q'}^{q_0, q_1}$

We derive $L_{q, q'}^{q_0, q_1}$, which is the set of strings which allows us to get from $q$ to $q'$ with intermediate states in $\{q_0, q_1\}$. This language is

$$L_{q, q'}^{q_0, q_1} = L_{q, q'}^{q_0} \mid (L_{q, q_0}^{q_0} (L_{q_0, q_0}^{q_0})^* L_{q_0, q_1}^{q_0})$$
Calculation of $L_{q_0, q_1}^{q_0, q_1}$

$L_{q_0, q_1}^{q_0}$:

\[
L_{q_0, q_1}^{q_0} = \{1\}^* \cdot \{0\} \cdot \{0, 1, \epsilon\}^* \cdot \emptyset
\]

$L_{q_0, q_1}^{q_1}$:

\[
L_{q_0, q_1}^{q_1} = (\{1\}^* \cdot \{0\}) \mid (\{1\}^* \cdot \{0\} \cdot \{0, 1, \epsilon\}^* \{0, 1, \epsilon\})
\]

\[
= \{1\}^* \cdot \{0\} \cdot \{0, 1\}^*
\]

$L_{q_1, q_0}^{q_0}$:

\[
L_{q_1, q_0}^{q_0} = \emptyset \mid (\{0, 1, \epsilon\} \cdot \{0, 1, \epsilon\}^* \cdot \emptyset)
\]

\[
= \emptyset
\]

$L_{q_1, q_1}^{q_0}$:

\[
L_{q_1, q_1}^{q_0} = \{0, 1, \epsilon\} \mid \{0, 1, \epsilon\} \cdot \{0, 1, \epsilon\}^* \cdot \{0, 1, \epsilon\}
\]

\[
= \{0, 1\}^*
\]
From $L_{q,q'}^{q_0}$ to $L_{q,q'}^{q_0,q_1}$

States with $L_{q,q'}^{q_0}$:

$\{1\}^*.\{0\}$

States with $L_{q,q'}^{q_0,q_1}$, the complete language between those states:

$\{1\}^*.\{0\}.\{0,1\}^*$
The Language of $M$: $L(M)$

States with $L_{q,q'}^{q_0,q_1}$:

1. $L_{q,q'}^{q_0,q_1}$ is the set of strings which allows us to get from $q_0$ to $q_1$ using any intermediate states.
2. The language $L(M)$ is the set of strings which allow us to get from $q_0$ to any accepting state.
3. In the example there is only one accepting state ($q_1$), so the language accepted by $M$ is

$$L(M) = L_{q_0,q_1}^{q_0,q_1} = \{1\}^* \cdot \{0\} \cdot \{0,1\}^*$$
The Language of $M$: $L(M)$

States with $L_{q_0, q_1}^{q, q'}$:

Let $M'$ be as $M$, but with additional accepting state $q_0$, then we get

$$L(M') = L_{q_0, q_0}^{q_0, q_1} \mid L_{q_0, q_1}^{q_0, q_1} = \{1\}^* \mid (\{1\}^*.\{0\}.\{0, 1\}^*) = \{0, 1\}^*$$
Proof of the Theorem

We construct for states \( q, q' \in M \) a regular expression \( L_{q,q'} \) s.t.

\[
L_{q,q'} = \{ w \in T^* | q \xrightarrow{w} q' \}
\]

If \( F = \{ q_1, \ldots, q_k \} \) then we obtain

\[
L(M) = L_{q_0,q_1} \mid L_{q_0,q_2} \mid \cdots \mid L_{q_0,q_k}
\]

which is a regular expression.

(If \( F \) is empty, then \( L(M) = \emptyset \), which is a regular expression).
Proof

$L_{q,q'}$ is defined in stages by referring to $L^{q_1,\ldots,q_l}_{q,q'}$, where

$$L^{q_1,\ldots,q_l}_{q,q'} = \{a_1 \cdots a_k \in T^* \mid \exists p_i \in \{q_1, \ldots, q_l\}. \quad q \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{q_3} \cdots \xrightarrow{a_{k-1}} p_{k-1} \xrightarrow{a_k} q' \}$$

So $L^{q_1,\ldots,q_l}_{q,q'}$ is the set of words which allow us to get from $q$ to $q'$ by using as intermediate states $q_1, \ldots, q_l$ only.

We show that $L^{q_1,\ldots,q_{k}}_{q,q'}$ is a regular expression by induction on $k$.

Then $L_{q,q'} = L^Q_{q,q'}$ is a regular expression as well.
Proof

Base case $k = 0$: Let $a_1, \ldots, a_k$ be the $a_i$ s.t. $q \xrightarrow{a_i} q'$. Then

$$L_{q,q'}^\emptyset = \begin{cases} \{a_1\} \mid \cdots \mid \{a_k\} & \text{if } q \neq q' \\
\{a_1\} \mid \cdots \mid \{a_k\} \mid \{\epsilon\} & \text{if } q = q' \end{cases}$$

(in case of $k = 0$ we have $L_{q,q'}^\emptyset = \emptyset$ or $= \{\epsilon\}$ which are regular expressions.)
Proof

Induction Step: Assume we have shown that \( L_{p,p'}^{q_1,\ldots,q_{k-1}} \) are regular expressions for all \( p, p' \in Q \).
We show the same for \( L_{q,q'}^{q_1,\ldots,q_{k-1}} \).

A transition \( q \xrightarrow{w} q' \) which uses only intermediate states \( q_1, \ldots, q_k \) can have two forms:

- Either we don’t use \( q_k \) as an intermediate state. So we have only intermediate states \( q_1, \ldots, q_{k-1} \) and have \( w \in L_{q,q'}^{q_1,\ldots,q_{k-1}} \).
- Or we reach \( q_k \) as an intermediate state. We single out
  - the first part of the transition which doesn’t use state \( q_k \) until one reaches for the first time as an intermediate state \( q_k \) (note that \( q = q_k \) or \( q' = q_k \) is possible)
  - the second part where we several times go from \( q_k \) to \( q_k \) with intermediate states \( \neq q_k \),
  - and the last part where we get from \( q_k \) to \( q' \) without using \( q_k \).
So we have

\[ q \xrightarrow{v} q_k \xrightarrow{w_1} q_k \xrightarrow{w_2} q_k \xrightarrow{w_2} \cdots \xrightarrow{w_j} q_k \xrightarrow{v'} q' \]

where \( j = 0 \) is possible, all intermediate transitions avoid \( q_k \) and \( w = vw_1w_2\cdots w_kv' \).
Proof

\[ q \xrightarrow{v} q_k \xrightarrow{w_1} q_k \xrightarrow{w_2} q_k \xrightarrow{w_2} \cdots \xrightarrow{w_j} q_k \xrightarrow{v'} q' \]

In the second part we have

- \[ v \in L_{q_1, \ldots, q_{k-1}}^{q, q_k} \]
- \[ w_i \in L_{q_1, \ldots, q_{k-1}}^{q_k, q_k} \]
- \[ v' \in L_{q_1, \ldots, q_{k-1}}^{q_1, q_k} \]
- Therefore \[ w = vw_1 \cdots w_kv' \in L_{q_1, \ldots, q_{k-1}}^{q, q_k} \cdot (L_{q_1, \ldots, q_{k-1}}^{q_k, q_k})^* \cdot L_{q_k, q'}^{q_1, \ldots, q_{k-1}} \]

- Therefore

\[ L_{q, q'}^{q_1, \ldots, q_{k-1}} \subseteq L_{q, q'}^{q_1, \ldots, q_{k-1}} \cap (L_{q, q_k}^{q_1, \ldots, q_{k-1}} \cdot (L_{q_k, q_k}^{q_1, \ldots, q_{k-1}})^* \cdot L_{q_k, q'}^{q_1, \ldots, q_{k-1}}) \]

- One can see easily as well that for an element \( w \) in the right hand side we can derive that \( w \) is in the left hand side as well, i.e.

\[ L_{q, q'}^{q_1, \ldots, q_{k-1}} \supseteq L_{q, q'}^{q_1, \ldots, q_{k-1}} \cap (L_{q, q_k}^{q_1, \ldots, q_{k-1}} \cdot (L_{q_k, q_k}^{q_1, \ldots, q_{k-1}})^* \cdot L_{q_k, q'}^{q_1, \ldots, q_{k-1}}) \]
Proof

So

\[ L_{q_1, \ldots, q_k} = L_{q, q'} \mid (L_{q, q_k} \cdot (L_{q_k, q_k})^* \cdot L_{q_k, q'}) \]

and the right hand side is by using the IH a regular expression.
2.3.8. Equivalence Theorem

2.3.1. String Recognition (13.1)
2.3.2. Nondeterministic Finite State Automata (13.2)
2.3.3. Examples of Automata (13.3)
2.3.4. Automata with Empty Move Transitions (13.4)
2.3.5. Deterministic Finite State Automata (13.6)
2.3.6. Regular Grammars and NFAs (13.5)
2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)
2.3.10. Closure Properties and Decidability of Regular Languages
We are going to show that

- Regular Expressions,
- languages definable by regular grammars,
- languages definable by NFAs with empty moves,
- languages definable by NFAs,
- languages definable by DFAs

all define the same language.

We already have everything in order to show the above with regular grammars restricted to right-linear grammars.
Theorem

Let $L$ be a language over an alphabet $T$. The following is equivalent:

1. $L$ is a regular expression.
2. $L$ is definable by a right-linear grammar.
3. $L$ is definable by an NFA with empty moves.
4. $L$ is definable by an NFA.
5. $L$ is definable by a DFA.
Proof

- (1) → (2) was shown in 2.2.3.
  - (Regular grammars are closed under the operations for forming regular languages).
- (2) → (4) was shown in 2.3.6.
  - Right-linear grammars can be simulated by an NFA.
- (4) → (1) was shown in 2.3.7.
  - We can determine the language between states of an NFA as a regular expression.
- So (1), (2), (4) are equivalent.
Proof

- $(3) \rightarrow (4)$ was shown in 2.3.4.
  - We can omit the empty moves in NFAs with empty moves.
- $(4) \rightarrow (5)$ was shown in 2.3.5.
  - NFAs can be translated into DFAs using as states sets of states.
- $(5) \rightarrow (4) \rightarrow (3)$ are trivial.
  - DFAs are special cases of NFAs,
    NFAs are special cases of NFAs with empty moves.
- So $(3), (4), (5)$ are equivalent.
- So $(1), (2), (3), (4), (5)$ are equivalent.
It remains to show that left-linear and right-linear grammars are equivalent.

This is shown as follows:

- The languages derived by left-linear grammars are $L^R$ for languages derived by right-linear grammars.
- Regular Expressions are closed under the reverse operation, i.e. if $L$ is a regular expression, so is $L^R$.
- Therefore as well right-linear grammars are closed under the reverse operation, therefore left-linear and right-linear grammars are equivalent.
2.3.8. Equivalence Theorem

Right-Linear Languages are the Reverse of Left-Linear Ones

Lemma

1. Let $G$ be a left-linear grammar. Then there exist a right-linear grammar $G'$ over the same alphabet s.t. $L(G) = L(G')^R$.

2. Let $G$ be a right-linear grammar. Then there exist a left-linear grammar $G'$ over the same alphabet s.t. $L(G) = L(G')^R$. 
Proof

We prove only (1), (2) is analogously.
Let $G$ be a left-linear grammar with alphabet $T$, nonterminals $N$ and start symbol $S$.
Let $G'$ be identical to $G$ but with rules

$$A \rightarrow aB$$

$(A, B \in N, a \in T)$ replaced by

$$A \rightarrow Ba$$

$G'$ is right-linear. Further it follows immediately for any $w \in (N \cup T)^*$ that

$$S \Rightarrow_G w \text{ iff } S \Rightarrow_{G'} w^R$$
2.3.8. Equivalence Theorem

Proof

Therefore

\[ L(G') = \{ w \in T^* | S \Rightarrow_{G'} w \} \]
\[ = \{ w^R \in T^* | S \Rightarrow_{G} w \} \]
\[ = \{ w \in T^* | S \Rightarrow_{G} w \}^R \]
\[ = L(G)^R \]
Lemma

1. Regular expressions are closed under \(L \mapsto L^R\), i.e. if \(L\) is a regular expression, so is \(L^R\).
2. The same applies to languages definable by right-linear grammars.
Proof

(1) We show that if \( L \) is a regular expression, so is \( L^R \) by induction on the definition of regular expressions:

- For \( L = \emptyset, \{ \epsilon \}, \{ a \} \) we have that \( L^R = L \) is a regular expression.
- For \( L = L_1 \mid L_2 \) we have that \( L^R = L_1^R \mid L_2^R \) which is a regular expression by IH.
- For \( L = L_1.L_2 \) we have that \( L^R = L_2^R.L_1^R \) which is a regular expression by IH.
- For \( L = L_1^* \) we have that \( L^R = (L_1^R)^* \) which is a regular expression by IH.

(2) Follows since languages definable by right-linear grammars are exactly the languages definable by regular expressions.
2.3.8. Equivalence Theorem

Left-Linear and Right-Linear Grammars are Equivalent

Lemma

Let $L$ be a language over an alphabet $T$. The following are equivalent:

1. $L = L(G)$ for a left-linear grammar $G$.
2. $L = L(G)$ for a right-linear grammar $G$. 
Proof

- Assume $L = L(G)$ for a left-linear grammar $G$.
  - Then $L^R = L(G')$ for a right-linear grammar $G'$.
  - Right-linear grammars are closed under $L \mapsto L^R$.
  - Therefore there exists a right-linear grammar $G''$ s.t.
    \[ L(G'') = L(G')^R = (L^R)^R = L. \]

- Assume $L = L(G)$ for a right-linear grammar $G$.
  - There exists a right-linear grammar $G'$ s.t. $L(G') = L^R$.
  - There exists a left-linear grammar $G''$ s.t. $L(G'') = L(G')^R$.
  - Now $L(G'') = L(G')^R = (L(G)^R)^R = L(G) = L$. 
Let $L$ be a language over an alphabet $T$. The following are equivalent:

1. $L$ is a regular expression.
2. $L$ is a regular.
3. $L$ is definable by a right-linear grammar.
4. $L$ is definable by a left-linear grammar.
5. $L$ is definable by an NFA with empty moves.
6. $L$ is definable by an NFA.
7. $L$ is definable by a DFA.
Proof

By the above.
2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.1. String Recognition (13.1)

2.3.2. Nondeterministic Finite State Automata (13.2)

2.3.3. Examples of Automata (13.3)

2.3.4. Automata with Empty Move Transitions (13.4)

2.3.5. Deterministic Finite State Automata (13.6)

2.3.6. Regular Grammars and NFAs (13.5)

2.3.7. Translating NFAs into Regular Expressions (13.10)

2.3.8. Equivalence Theorem

2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
Motivation

- We want to show that there are languages which are context-free but not regular.
- In order to do this we prove the pumping lemma, which uses the fact that an NFA has only finitely many states. (We could use as well the fact that a regular grammar has only finitely many nonterminals).
Consider an NFA

![NFA Diagram]

This NFA has 5 states.
Any run of the NFA for a word of length \( \geq 5 \) uses at least 6 states.
Therefore it must visit one state at least twice.
So there must be a loop within the first 5 letters of such a word.
Using the Finiteness of an NFA

Here is the run for the word $z = \text{ababa}$ using colours blue, red and green.

- The **blue part** is the part before we reached a state visited twice, corresponding to the word $u = a$.
- The **red part** is the part from the state visited twice until we reach it again, corresponding to the word $v = \text{bab}$.
- The **green part** is the remaining part, corresponding to the word $w = a$.
- The loop must occur within the first 5 letters, so $|uv| \leq 5$. Because $v$ is along a loop, $|v| \geq 1$. 
If we repeat the loop several times, we obtain as well an accepting word.

- If we start with $u = a$, then repeat the loop following the word $v = bab$ $i$ times, then the follow the word $w = a$, we obtain an accepting run.
- It accepts the word $a(bab)^i a$.
- In general we get that the word $uv^i w$ is an element of the language as well.
Generalisation

Assume an NFA $M$ having $n$ states. Then for every word $x \in L(M)$ s.t. $|x| \geq n$ there exist words $u, v, w$ s.t.

$$x = uvw, \; |uv| \leq n, \; |v| \geq 1$$

and s.t.

$$uv^i w \in L(M) \text{ for all } i \in \mathbb{N}$$

This follows by the above considerations.

So we have proved the following theorem:
Pumping Lemma for Regular Languages

Let \( L \) be a regular language. Then there exist a fixed number \( n \) depending on \( L \) only s.t. we have the following:

- If \( x \in L \) is a word, \( |x| \geq n \), then there exist words \( u, v, w \) s.t.

\[
    x = uvw, \quad |uv| \leq n, \quad |v| \geq 1
\]

and s.t.

\[
    uv^i w \in L(M) \text{ for all } i \in \mathbb{N}
\]
Example 1

**Lemma**

The language \( L := \{ a^i b^i \mid i \geq 1 \} \) is context-free but not regular.
Proof (Example 1)

- We have already seen that $L$ is context-free.
- Assume $L$ is regular.
- Let $n$ be as in the pumping lemma.
- Consider $x := a^n b^n \in L$.
- $|x| \geq n$, so there exist $u, v, w$ s.t.
  
  $x = uvw$, $|uv| \leq n$, $|v| \geq 1$,
  
  and s.t.
  
  $uv^i w \in L$ for all $i \geq n$.
- Since $|uv| \leq n$, $u$ and $v$ are substrings of $a^n$.
- Therefore $uv^2 w = a^{n+l} b^n$ where $l = |v|$.
- But $a^{n+l} b^n \notin L$, a contradiction.
Example 2

Lemma

The language $L := \{xx^R \mid x \in \{a, b\}^*\}$ is context-free but not regular.
Proof (Example 2)

- We have already seen that $L$ is context-free.
- Assume $L$ is regular.
- Let $n$ be as in the pumping lemma.
- Consider $x := a^n b b a^n \in L$.
- $|x| \geq n$, so there exist $u, v, w$ s.t.
  $x = uvw$, $|uv| \leq n$, $|v| \geq 1$,
  and s.t.
  $uv^i w \in L$ for all $i \geq n$.
- Since $|uv| \leq n$, $u$ and $v$ are substrings of $a^n$.
- Therefore $uv^2 w = a^{n+l} b b a^n$ where $l = |v|$.
- But $a^{n+l} b b a^n \not\in L$, a contradiction.
2.3.1. String Recognition (13.1)
2.3.2. Nondeterministic Finite State Automata (13.2)
2.3.3. Examples of Automata (13.3)
2.3.4. Automata with Empty Move Transitions (13.4)
2.3.5. Deterministic Finite State Automata (13.6)
2.3.6. Regular Grammars and NFAs (13.5)
2.3.7. Translating NFAs into Regular Expressions (13.10)
2.3.8. Equivalence Theorem
2.3.9. The Pumping Lemma for Regular Languages (12.4, 12.5)

2.3.10. Closure Properties and Decidability of Regular Languages
Closure Properties

Lemma

Regular languages are closed under

1. complement,
2. intersection,
3. the operation $L \mapsto L^R$.

So if $L, L'$ are regular languages over alphabet $T$, so are

1. $L^c$ (the complement of $L$, i.e. $\{ t \in T^* \mid t \notin L \}$),
2. $L \cap L'$,
3. $L^R$ (i.e. $\{ w^R \mid w \in L \}$, where $w^R$ is the result of rewriting $w$).
Closure under $L \mapsto L^R$

- We will use that regular expressions, languages definable by DFAs and regular languages are equivalent.
- We have seen in 2.3.8. that regular expressions and therefore regular languages are closed under $L \mapsto L^R$. 
Closure under Intersection

- Assume $L$, $L'$ regular languages.
- There exist DFAs $M = (Q, q_0, F, T, \delta)$ and $M' = (Q', q'_0, F', T', \delta')$ s.t. $L = L(M)$, $L' = L(M')$.
- By a lemma defined below we can assume $M$ and $M'$ have total transitions, i.e.

\[
\forall q \in Q. \forall a \in T. \exists q' \in Q. \delta(q, a) = q',
\]

similarly for $M'$.
- Let $M'' := (Q \times Q', (q_0, q'_0), F \times F', T, \delta'')$ where $\delta''((q, q'), a) = (\delta(q, a), \delta(q', a))$. 
Closure under Intersection

- We have in $M''$ for $w \in T^*$
  
  $$(q, q') \xrightarrow{w} (p, p') \text{ iff } q \xrightarrow{w} p \land q' \xrightarrow{w} p'$$

- Therefore
  
  $$L(M'') = \{ w \in T^* | \exists q \in F \times F'(q_0, q'_0) \xrightarrow{w} q \}$$
  
  $$= \{ w \in T^* | \exists q \in F, q' \in F'.(q_0, q'_0) \xrightarrow{w} (q, q') \}$$
  
  $$= \{ w \in T^* | \exists q \in F, q' \in F'.q_0 \xrightarrow{w} q \land q'_0 \xrightarrow{w} q' \}$$
  
  $$= \{ w \in T^* | w \in L(M) \land w \in L(M') \}$$
  
  $$= L(M) \cap L(M') = L \cap L'$$

- Therefore $L \cap L'$ is regular.
Closure under Complement

- Assume $L$ a regular languages.
- There exist DFA $M = (Q, q_0, F, T, \delta)$ s.t. $L = L(M)$.
- As before we assume that $\delta$ is total.
- Let $M' := (Qq_0, Q \setminus F, T, \delta)$. 
Closure under Complement

\[ L(M') = \{ w \in T^* \mid \exists q \in Q \setminus F. q_0 \xrightarrow{w} q \} \]
\[ = \{ w \in T^* \mid \neg \exists q \in F. q_0 \xrightarrow{w} q \} \]
\[ = T^* \setminus L(M) = L^c \]

\[ \Rightarrow \text{So } L^c \text{ is regular.} \]
From DFA to Total DFA

Lemma

Let $M = (Q, q_0, F, T, \delta)$ be a DFA. Then we can define a DFA $M' = (Q', q'_0, F', T', \delta')$, s.t.

- $\forall q \in Q'. \forall a \in T. \delta(q, a)$ is defined.
- $L(M') = L(M)$. 
Proof

- Take $M$ and modify it as follows:
- Add one additional state $q'$ to $Q$.
- $F$ stays at it is (so $q' \notin F'$).
- If for some $q \in Q$, $a \in T \, \delta(q, a)$ is not defined, change this to $\delta(q, a) = q'$.
- Further add $\delta(q', a) = q'$.
- Let the new automaton be called $M'$:
- This automaton operates as $M$, except when $M$ fails, then $M'$ switches to state $q'$ and stays there forever, without accepting the word.
- So $L(M) = L(M')$. 
Decision Problems

Theorem

- We can decide for regular languages whether $L = \emptyset$.
- We can decide for regular languages $L$ and $L'$ whether $L \subseteq L'$.
- We can decide for regular languages $L$ and $L'$ whether $L = L'$.
Proof

▶ Check for emptiness: We can decide for a regular expression whether it is the empty language:

▶ $\emptyset$ is the empty language.
▶ $\{a\}$, $\{\epsilon\}$ are not empty.
▶ $L \mid L'$ is empty iff both $L$ and $L'$ are empty.
▶ $L.L'$ is empty if $L$ is empty or $L'$ is empty.
▶ $L^*$ is never empty.

▶ Since any other description of a regular language can be transformed into a regular expression, we can decide for regular languages whether they are empty.
2.3.10. Closure Properties/Decidability of Regular Languages

Proof

- Decision of inclusion:

\[ L \subseteq L' \iff L \cap (L')^c = \emptyset \]

The right hand side is decidable.

- Decision for equality:

\[ L = L' \iff L \subseteq L' \land L' \subseteq L \]

The right hand side is decidable.