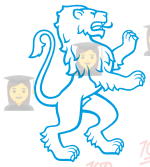


Kantonsschule Im Lee

Informatik

Endliche Automaten

Skript

Thomas Graf

Inhaltsverzeichnis

1 Alphabete, Wörter, Sprachen **2**
- 1.1 Alphabete, Wörter, Sprachen . . . 2
- 1.2 Lösungen der Aufgaben . . . 8
- 1.3 Kapiteltest . . . 11
- 1.4 Lösungen zum Kapiteltest . . . 12

2 Algorithmische Probleme **13**
- 2.1 Das Entscheidungsproblem . . . 14
- 2.2 Graphen . . . 15
  - 2.2.1 Adjazenzmatrix . . . 17
    - 2.2.1.1 Knotenüberdeckung (vertex cover) . . . 19
- 2.3 Lösungen der Aufgaben . . . 21

3 Endliche Automaten **22**
- 3.1 Darstellung endlicher Automaten . . . 22
  - 3.1.1 Darstellung durch gerichtete Graphen . . . 22
  - 3.1.2 Darstellung durch Programme . . . 26
  - 3.1.3 Formale Definition . . . 29
- 3.2 Beweise der Nichtexistenz . . . 33
- 3.3 Lösungen der Aufgaben . . . 38

4 Turingmaschinen **42**

Kapitel 1

Alphabete, Wörter, Sprachen

Wenn man sich mit der Funktionsweise von Rechnern (Computern) genauer beschäftigt, stellt man fest, dass Rechner im Grunde eine Transformation von Eingabedaten in Ausgabedaten realisieren. Sowohl die Eingabedaten als auch die Ausgabedaten lassen sich als Texte darstellen. Die Texte sind nichts anderes als Folgen von Symbolen aus einem bestimmten Alphabet. Programme können als Folge von Symbolen der Computertastatur dargestellt werden. In digitalen Rechnern sind alle Informationen als Folgen von Einsen und Nullen gespeichert. Damit realisiert der Rechner eine Transformation von Eingabetexten in Ausgabetexte.

In diesem Kapitel wollen wir den Formalismus für den Umgang mit Texten kennenlernen. Wir werden die fundamentalen Begriffe *Alphabet*, *Wort* und *Sprache* einführen. Diese werden uns später helfen, bekannte Probleme der Informatik wie beispielsweise das *Entscheidungsproblem* mathematisch sauber zu formulieren.

Dieses Kapitel folgt dem Abschnitt 2.2 aus dem Buch[1] sehr nahe. Es wurden einige Aufgaben, welche als Hilfestellung dienen, hinzugefügt und kleine Teile ausgelassen.

1.1 Alphabete, Wörter, Sprachen

Definition 1.1 (Alphabet)**:**
Eine **endliche nichtleere** Menge Σ heisst *Alphabet*. Die *Elemente eines Alphabets* werden *Buchstaben* (*Zeichen, Symbole*) genannt.

Wir werden später Alphabete verwenden, um eine schriftliche Darstellung einer Sprache zu erzeugen. Definition 1.1, entspricht unserer intuitiven Vorstellung eines Alphabets: Um Text darstellen zu können, muss ein Alphabet mindestens ein Symbol enthalten ($\rightarrow$ nichtleer). Damit man sich auf einen fixen Satz von Zeichen einigen kann, darf das Alphabet nicht unendlich gross sein ($\rightarrow$ endlich).

Wir listen nun einige der Alphabete auf, die in der Mathematik und Informatik häufig verwendet werden.

Beispiel 1.1: (a) $\Sigma_{\text{bool}} = \{0, 1\}$ ist das Boole'sche Alphabet, mit dem digitale Rechner arbeiten.
(b) $\Sigma_{\text{lat}} = \{a, b, c, \ldots, z, A, B, \ldots, Z\}$ ist das lateinische Alphabet.
(c) $\Sigma_{\text{Tastatur}} = \{a, b, c, \ldots, z, A, B, C, \ldots, Z, \sqcup, >, <, (,), \ldots, \#, ?, !\}$ ist das Alphabet aller

[1] J. Hromkovic: Theoretische Informatik. 5. Auflage, Springer Vieweg 2014., ISBN: 978-3-658-06432-7

Symbole, die mit der englischen Tastatur getippt werden können. Dabei ist ␣ das Symbol für das Leerzeichen / Leersymbol.

(d) $\Sigma_{\text{greek}} = \{\alpha, \beta, \gamma, \ldots, \omega, A, B, \Gamma, \ldots, \Omega\}$ ist das griechische Alphabet.

(e) $\Sigma_m = \{\ n \in \mathbb{N}\ ;\ n < m\ \}$ für jede fixe Wahl von $m \in \mathbb{N} \setminus \{0\}$, ist ein Alphabet für die m-adische Darstellung von Zahlen.

Wörter werden wir als endliche Folgen von Buchstaben ansehen.

Definition 1.2 (Wort)**:**
Sei Σ ein Alphabet. Ein *Wort* über Σ ist eine endliche (möglicherweise leere) Folge von Buchstaben aus Σ. Das *leere Wort* λ ist die leere Buchstabenfolge. Die *Länge* $|w|$ eines Wortes w ist die Länge des Wortes als Folge, das heisst die Anzahl der Vorkommen von Buchstaben in w.

Beispiel 1.2: (a) $w = 1, 0, 0, 1, 0$ ist ein Wort über dem Alphabet Σ_{bool} und $|w| = 5$, da w eine Folge von 5 Buchstaben ist.

(b) $u = M, ␣, j$ ist ein Wort über Σ_{Tastatur} und $|u| = 3$, da u eine Folge von 3 Buchstaben ist.

(c) Das leere Wort λ ist ein Wort über jedem Alphabet und es gilt $|\lambda| = 0$.

Aufgabe 1.1

Was ist Σ_{10}?

Definition 1.3 (Stern-Operator)**:**
Sei Σ ein Alphabet. Σ^* (gesprochen: *Sigma Stern*) ist **die Menge aller Wörter** über Σ, also die Menge aller endlichen Folgen von Symbolen aus Σ. Man nennt Σ^* auch den **Kleene'schen Stern**[a] von Σ. $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$ ist die Menge aller Wörter über Σ ohne das leere Wort.

[a] benannt nach dem US-Amerikanischen Mathematiker Stephen Cole Kleene

Wir werden im Folgenden Wörter ohne Kommmas schreiben. Anstelle von $1, 0, 0, 1, 0$ werden wir lediglich 10010 schreiben. Allgemein, werden wir anstelle von $x_1, x_2, \ldots, x_n$ einfach $x_1 x_2 \ldots x_n$ schreiben.

Bemerkung 1.1:
In der deutschen Sprache existieren die zwei Ausdrücke „Wörter" und „Worte". Dies sind keine Synonyme. *Wörter* bezeichnet den Plural von *Wort* („Im Duden stehen viele Wörter"). Der Ausdruck *Worte* bezieht sich auf Gedankenkonstrukte („Sie sprach weise Worte").

Wir können Wörter benutzen, um mathematische Objekte wie Zahlen, Formeln, Graphen und Computerprogramme darzustellen. Ein Wort $x = x_1 x_2 \ldots x_n \in (\Sigma_{\text{bool}})^*$ kann als binäre Darstellung der Zahl

$$\text{Nummer}(x) = \sum_{k=1}^{n} x_k \cdot 2^{n-k} \tag{1.1}$$

betrachtet werden.

Aufgabe 1.2

Sei $x = 1011$. Berechnen Sie Nummer(x).

Für eine Zahl $m \in \mathbb{N} \setminus \{0\}$ wird mit $\text{Bin}(m) \in (\Sigma_{\text{bool}})^*$ die kürzeste binäre Darstellung von m bezeichnet, also gilt $\text{Nummer}(\text{Bin}(m)) = m$. Man setzt $\text{Bin}(0) = 0$.

Aufgabe 1.3

Wie sehen die folgenden Mengen aus?

(a) $\{1\}^*$
(b) $(\Sigma_{\text{bool}})^*$

Aufgabe 1.4

In Definition 1.2, haben wir ein Wort als endliche Folge von Buchstaben definiert. Nun enthält zum Beispiel die Menge $\{1\}^*$ in Aufgabe 1.3 aber Wörter beliebiger Länge. Erklären Sie, warum dies kein Widerspruch ist.

Da Alphabete insbesondere auch Mengen sind, kann man auch das kartesische Produkt von Alphabeten bilden.

Beispiel 1.3:
Sei $\Sigma_1 = \{g, h\}$ und $\Sigma_2 = \{x, y\}$, dann ist das kartesische Produkt $\Sigma_1 \times \Sigma_2 = \{(g, x), (g, y), (h, x), (h, y)\}$.

Wir werden nun eine Operation einführen, welche uns erlaubt Wörter zu verketten. Diese Operation werden wir sehr häufig verwenden.

Definition 1.4 (Verkettung / Konkatenation)**:**
Die **Verkettung** (**Konkatenation**) für ein Alphabet Σ ist eine Abbildung $\text{Kon}: \Sigma^* \times \Sigma^* \to \Sigma^*$, sodass

$$\text{Kon}(x, y) = x \cdot y = xy$$

für alle $x, y \in \Sigma^*$.

Beispiel 1.4:
Sei $\Sigma = 8, 9, d, e$ und seien $x = e899dd$ und $y = de8$, dann ist $\text{Kon}(x, y) = x \cdot y = e899ddde8$.

Wir werden fast ausnahmslos xy schreiben und nur selten $x \cdot y$ oder $\text{Kon}(x, y)$.

Aufgabe 1.5

Erläutern Sie in Ihren eigenen Worten, was die Bedeutung des Ausdrucks $\Sigma^* \times \Sigma^* \to \Sigma^*$ in Definition 1.4 ist.

Die Verkettung Kon über Σ ist assoziativ über Σ^*, da offensichtlich

$$\text{Kon}(x, \text{Kon}(y, z)) = x \cdot (y \cdot z) = xyz = (x \cdot y) \cdot z = \text{Kon}(\text{Kon}(x, y), z)$$

gilt, für alle $x, y, z \in \Sigma^*$.

Aufgabe 1.6

Bei der Multiplikation in den reellen Zahlen ist die Zahl $1 \in \mathbb{R}$ das neutrale Element, da $1 \cdot x = x \cdot 1 = x$ für alle $x \in \mathbb{R}$ gilt. Welches ist das neutrale Element der Addition in den reellen Zahlen?

Für jedes $x \in \Sigma^*$ gilt

$$x \cdot \lambda = \lambda \cdot x = x.$$

Damit ist λ das neutrale Element der Verkettung über Σ^*.

Aufgabe 1.7

Sei $x = a01b$ und $y = c21$. Bestimmen Sie $|x|$, $|y|$ und $|xy|$. Seien allgemein $x, y \in \Sigma^*$ zwei Wörter für ein Alphabet Σ. Finden Sie einen Ausdruck für $|xy|$.

Definition 1.5 (Umkehrung / Reversal)**:**
Sei n eine natürliche Zahl. Für ein Wort $x = x_1 x_2 \dots x_n$, mit $x_i \in \Sigma$ für $i \in \{1, 2, \dots, n\}$ bezeichnet $x^R = x_n x_{n-1} \dots x_1$ die **Umkehrung** (**Reversal**) von x.

Beispiel 1.5:
Es sei $w := abcde$. Dann gilt $w^R = edcba$.

Aufgabe 1.8

Sei Σ ein Alphabet und $u, v \in \Sigma^*$ zwei Wörter. Beweisen oder widerlegen Sie die Aussage:

$$(uv)^R = v^R u^R$$

Definition 1.6 (Iteration eines Wortes)**:**
Sei Σ ein Alphabet. Für alle $x \in \Sigma^*$ und alle $i \in \mathbb{N}$ definieren wir die i-te **Iteration** x^i von x als

$$x^0 = \lambda, \quad x^1 = x, \quad x^i = xx^{i-1}.$$

Beispiel 1.6:
Sei $\Sigma = \{a, b, c\}$. Wir können nun schreiben:

$$aa = a^2$$
$$ababab = (ab)^4$$
$$cbbbbbab = cb^5ab = c(bb)^2bab$$

Mit der folgenden Definition wollen wir den Begriff *Teilwort*, für den wir eine gute Intuition haben, formalisieren. Ein Teilwort eines Wortes x ist ein zusammenhängender Teil von x.

Definition 1.7 (Teilwörter)**:**
Seien $u, w \in \Sigma^*$ für ein Alphabet Σ.

- v heisst **Teilwort** von $w \Leftrightarrow$ es existieren $x, y \in \Sigma^*$, sodass $w = xvy$
- v heisst **Präfix** von $w \Leftrightarrow$ es existiert $x \in \Sigma^*$, sodass $w = vx$
- v heisst **Suffix** von $w \Leftrightarrow$ es existiert $x \in \Sigma^*$, sodass $w = xv$
- $v \neq \lambda$ heisst **echtes** Teilwort (Präfix, Suffix) von $w \Leftrightarrow v \neq w$ und v ein Teilwort (Präfix, Suffix) von w ist

Aufgabe 1.9

Es seien $\Sigma := \{a, b, c\}$ und $w := abc$ ein Wort über Σ. Bestimmen Sie alle Teilwörter von w. Welche dieser Teilwörter sind auch Präfixe von w?

Beispiel 1.7:
Sei $\Sigma = \{a, b, c\}$. Das Wort abc ist ein echtes Teilwort, echtes Präfix und ein echtes Suffix von $(abc)^3$. Das leere Wort λ ist Teilwort von jedem Wort. Jedes Wort ist Teilwort von sich selbst (aber kein echtes Teilwort).

Aufgabe (Challenge) 1.10

Es sei x ein Wort der Länge $n \in \mathbb{N}$, welches aus lauter verschiedenen Buchstaben besteht (also aus n verschiedenen Buchstaben). Wie viele verschiedene Teilwörter hat x?

Wir wollen an dieser Stelle noch eine weitere nützliche Schreibweise definieren.

Sei $x \in \Sigma^*$ und $a \in \Sigma$, dann ist $|x|_a$ definiert als die Anzahl der Vorkommen von a in x.

Beispiel 1.8:
Sei $v = babaac$, dann ist $|v|_a = 3$, $|v|_b = 2$ und $|v|_c = 1$. Offensichtlich gilt für alle $x \in \Sigma^*$

$$|x| = \sum_{a \in \Sigma} |x|_a \,.$$

Nun kommen wir zur Definition einer Sprache. Dies wird für uns einer der wichtigsten Begriffe sein.

Definition 1.8 (Sprache)**:**
Eine **Sprache** L über einem Alphabet Σ ist eine Teilmenge von Σ^*. Das Komplement L^C der Sprache L bezüglich Σ ist die Sprache $\Sigma^* \setminus L$.

- $L_\emptyset = \emptyset$ ist die leere Sprache.
- $L_\lambda = \{\lambda\}$ ist die einelementige Sprache, die nur aus dem leeren Wort besteht.

Sind L_1 und L_2 Sprachen über Σ, so bezeichnet

$$L_1 \cdot L_2 = L_1 L_2 = \{\ vw\ ;\ v \in L_1 \text{ und } w \in L_2\ \}$$

die **Konkatenation** von L_1 und L_2.

Ist L eine Sprache über Σ, so definieren wir die Iterationen

$$L^0 = L_\lambda, L^{i+1} = L^i \cdot L \text{ für alle } i \in \mathbb{N},$$
$$L^* = \bigcup_{i \in \mathbb{N}} L^i \text{ und } L^+ = \bigcup_{i \in \mathbb{N}} L^i \cdot L.$$

Beispiel 1.9:
Die folgenden Mengen sind Beispiele von Sprachen über $\Sigma = \{a, b\}$.

- $L_1 = \emptyset$
- $L_2 = \{\lambda\}$
- $L_3 = \Sigma^* = \{\lambda, a, b, aa, \ldots\}$
- $L_4 = \Sigma^+ = \{a, b, aa, \ldots\}$
- $L_5 = \Sigma$
- $L_6 = \{\ a^p\ ;\ p \text{ ist eine Primzahl}\ \}$
- $L_5 = \{a\}^* = \{\lambda, a, aa, aaa, aaaa, \ldots\}$
- $\Sigma^2 = \{aa, ab, ba, bb\}$

Man beachte, dass $\Sigma^i = \{\ x \in \Sigma^*\ ;\ |x| = i\ \}$ für eine natürliche Zahl i, und dass $L_\emptyset \cdot L = \emptyset$, $L_\lambda \cdot L = L$.

Beispiel 1.10:

- Die Menge aller grammatikalisch korrekten englischen Texte ist eine Sprache über Σ_{Tastatur}.
- Die Menge aller syntaktisch korrekten Programme in `C++` ist auch eine Sprache über Σ_{Tastatur}.

Aufgabe 1.11

Sei $L_1 = \{\lambda, a, b\}$ und sei $L_2 = \{a^4, a^2b\}$. Welche Wörter Liegen in der Sprache $L_3 = L_1L_2$?

Aufgabe (Challenge) 1.12

Sei $k \in \mathbb{N}$. Geben Sie ein Alphabet Σ und zwei Sprachen L_1 und L_2 über Σ an, sodass

$$|L_1| = k \quad \text{und} \quad |L_1L_2| = k + 1.$$

1.2 Lösungen der Aufgaben

✔ Lösungsvorschlag zu Aufgabe 1.1 ✔

$$\Sigma_{10} = \{0, 1, 2, \ldots, 9\}$$

ist das Alphabet für die Darstellung von Zahlen im Dezimalsystem.

✔ Lösungsvorschlag zu Aufgabe 1.2 ✔

$$\text{Nummer}(x) = \text{Nummer}(1011) = \sum_{k=1}^{4} x_k \cdot 2^{4-k} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 2 + 1 = 11$$

✔ Lösungsvorschlag zu Aufgabe 1.3 ✔

$$\begin{aligned} \{1\}^* &= \{\lambda, 1, 11, 111, 1111, 11111, \ldots\} = \\ &= \{\lambda\} \cup \{\ x_1 x_2 \ldots x_i\ ;\ i \in \mathbb{N},\ x_j = 1 \text{ für } j = 1, 2, \ldots, i\ \} \end{aligned}$$

$$\begin{aligned} (\Sigma_{\text{bool}})^* = \{0,1\}^* &= \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 100, 011, \ldots\} = \\ &= \{\lambda\} \cup \{\ x_1 x_2 \ldots x_i\ ;\ i \in \mathbb{N},\ x_j \in \Sigma_{\text{bool}} \text{ für } j = 1, 2, \ldots, i\ \} \end{aligned}$$

✔ Lösungsvorschlag zu Aufgabe 1.4 ✔

Anhand dieser Frage lässt sich der Unterschied zwischen den Begriffen *unbeschränkt* und *unendlich* schön verdeutlichen. Betrachten wir dazu exemplarisch nochmals die Menge $\{1\}^*$ aus Aufgabe 1.3:

$$\begin{aligned} \{1\}^* &= \{\lambda, 1, 11, 111, 1111, 11111, \ldots\} = \\ &= \{\lambda\} \cup \{\ x_1 x_2 \ldots x_i\ ;\ i \in \mathbb{N},\ x_j = 1 \text{ für } j = 1, 2, \ldots, i\ \} \end{aligned}$$

- Für jede natürliche Zahl $n \in \mathbb{N}$, existiert in $\{1\}^*$ ein Wort w mit $|w| \geq n$, da für jedes $n \in \mathbb{N}$ (insbesondere) auch das Wort $w := 1^n$ in der Menge $\{1\}^*$ enthalten ist und $|w| \geq n$. Damit gibt es **keine obere Schranke** für die Länge der Wörter in $\{1\}^*$.
- Jedes Wort $w \in \{1\}^*$ hat die Form $w = 1^n$ für eine natürliche Zahl n. Damit hat jedes Wort in der Menge $\{1\}^*$ eine endliche Länge.

Die Länge der Wörter in $\{1\}^*$ können also nicht nach oben beschränkt werden, trotzdem hat jedes Wort in $\{1\}^*$ eine endliche Länge und ist somit tatsächlich ein Wort im Sinne von Definition 1.2.

✔ Lösungsvorschlag zu Aufgabe 1.5 ✔

Wir haben zwei Wörter x und y über dem Alphabet Σ. Weil Σ^* die Menge aller Wörter über Σ ist, gilt offensichtlich $x, y \in \Sigma^*$. Aus den zwei Wörtern x und y wird nun ein neues Wort xy gebildet. Man beachte, dass die Reihenfolge $xy \neq yx$ (im Allgemeinen) eine Rolle spielt. Die Verkettung nimmt also ein geordnetes Paar $(x, y) \in (\Sigma^* \times \Sigma^*)$ (dies ist gerade die Menge aller geordneten Paare von Wörtern über Σ) und bildet dieses auf ein neues Wort $xy \in \Sigma^*$ ab.

✔ Lösungsvorschlag zu Aufgabe 1.6 ✔

Die Zahl $0 \in \mathbb{R}$, da $0 + x = x + 0 = x$ für alle $x \in \mathbb{R}$.

✔ Lösungsvorschlag zu Aufgabe 1.7 ✔

Offensichtlich sind $|x| = |a01b| = 4$ und $|y| = |c21| = 3$ und somit $|xy| = |a01bc21| = 7$.

Seien $x, y \in \Sigma^*$ zwei Wörter über einem Alphabet Σ. Dann gilt

$$|xy| = |x| + |y| .$$

✔ Lösungsvorschlag zu Aufgabe 1.8 ✔

Da u und v Wörter sind, haben sie endliche Längen. Wir definieren n und m als $n := |u|$ und $m := |v|$. Dann hat u die Form $u = u_1u_2 \dots u_n$ und $v = v_1v_2 \dots v_m$, wobei $u_1, u_2, \dots, u_n, v_1, v_2, \dots, v_m \in \Sigma$. Damit ist

$$uv = u_1u_2 \dots u_nv_1v_2 \dots v_m$$

und somit

$$(uv)^R = v_m \dots v_2v_1u_n \dots u_2u_1 = v^Ru^R,$$

da $v^R = v_m \dots v_2v_1$ und $u^R = u_n \dots u_2u_1$ gilt.

✔ Lösungsvorschlag zu Aufgabe 1.9 ✔

Die Menge der Teilwörter von w ist

$$\{\lambda, a, b, c, ab, bc, abc\} ,$$

wobei nur $\{\lambda, a, ab, abc\}$ auch Präfixe von w sind.

✔ Lösungsvorschlag zu Challenge 1.10 ✔

Wir dürfen annehmen, dass x die Form $x = a_1a_2 \dots a_n$ hat. Wir werden die Anzahl der Teilwörter von x der Länge $i = 1, i = 2$ bis zur Länge $i = n$ zählen.

$\underline{i = 1}$:

$$\begin{aligned}
1: &\quad \underline{a_1}a_2a_3a_4 \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
2: &\quad a_1\underline{a_2}a_3a_4 \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
3: &\quad a_1a_2\underline{a_3}a_4 \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
&\quad \dots \\
(n-3): &\quad a_1a_2a_3a_4 \dots \underline{a_{n-3}}a_{n-2}a_{n-1}a_n \\
(n-2): &\quad a_1a_2a_3a_4 \dots a_{n-3}\underline{a_{n-2}}a_{n-1}a_n \\
(n-1): &\quad a_1a_2a_3a_4 \dots a_{n-3}a_{n-2}\underline{a_{n-1}}a_n \\
n: &\quad a_1a_2a_3a_4 \dots a_{n-3}a_{n-2}a_{n-1}\underline{a_n}
\end{aligned}$$

Offensichtlich gibt es n viele Teilwörter der Länge 1, nämlich genau die Teilwörter $a_1, a_2, \dots, a_n$. Wir haben jeweils die Anfangsposition des Teilwortes in x unterstrichen.

$\underline{i = 2}$:

$$\begin{aligned}
1: &\quad \underline{a_1a_2}a_3a_4 \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
2: &\quad a_1\underline{a_2a_3}a_4 \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
3: &\quad a_1a_2\underline{a_3a_4} \dots a_{n-3}a_{n-2}a_{n-1}a_n \\
&\quad \dots \\
(n-3): &\quad a_1a_2a_3a_4 \dots \underline{a_{n-3}a_{n-2}}a_{n-1}a_n \\
(n-2): &\quad a_1a_2a_3a_4 \dots a_{n-3}\underline{a_{n-2}a_{n-1}}a_n \\
(n-1): &\quad a_1a_2a_3a_4 \dots a_{n-3}a_{n-2}\underline{a_{n-1}a_n}
\end{aligned}$$

Es gibt $n-1$ viele Teilwörter der Länge 2. Man beachte, dass sich die Anfangsposition der Teilwörter in x um eine Position nach links verschoben hat, da das letzte Teilwort (ganz rechts) der Länge 2 beim zweitletzten Symbol in x beginnen muss.

$\underline{i = 3}$:

$$\begin{aligned} 1: &\quad \underline{a_1 a_2} a_3 a_4 \dots a_{n-3} a_{n-2} a_{n-1} a_n \\ 2: &\quad a_1 \underline{a_2 a_3} a_4 \dots a_{n-3} a_{n-2} a_{n-1} a_n \\ &\quad \dots \\ (n-3): &\quad a_1 a_2 a_3 a_4 \dots \underline{a_{n-3} a_{n-2}} a_{n-1} a_n \\ (n-2): &\quad a_1 a_2 a_3 a_4 \dots a_{n-3} \underline{a_{n-2} a_{n-1}} a_n \end{aligned}$$

Es gibt $n-2$ viele Teilwörter der Länge 3.

Nun nicht mehr schwierig zu sehen, dass es für ein Teilwort der Länge $i \in \{1, 2, \dots, n\}$ genau $n-i+1$ mögliche Anfangspositionen in x gibt, da die Positionen $n-i+2, 3, \dots, n$ nicht möglich sind („zu wenig Platz“). Wir summieren jetzt die Anzahl aller unterschiedlichen (nicht leeren) Teilwörter von x der Längen $i = 1, 2, \dots, n$:

$$\sum_{i=1}^{n}(n-i+1) = n(n+1) - \sum_{i=1}^{n} i = n(n+1) - \frac{n(n+1)}{2} = \frac{n(n+1)}{2}$$

Hinzu kommt noch das leere Wort (welches Teilwort jedes Wortes ist) und als Resultat finden wir, dass es

$$\frac{n(n+1)}{2} + 1$$

viele verschiedene Teilwörter von x gibt.

✔ Lösungsvorschlag zu Aufgabe 1.11 ✔

$$L_3 = \left\{a^4, a^2b, a^5, a^3b, ba^4, ba^2b\right\}$$

Bitte beachten Sie unbedingt, dass zwar $aa^4 = a^5$ aber $ba^2b \neq a^2b^2$, da $ba^2b = baab \neq a^2b^2 = aabb$.

✔ Lösungsvorschlag zu Challenge 1.12 ✔

Wir wählen $\Sigma = \{0\}$ und

$$L_1 = \left\{0^1, 0^2, \dots, 0^k\right\} = \left\{\ 0^i\ ;\ 1 \leq i \leq k\ \right\}$$

und

$$L_2 = \{\lambda, 0\}\,.$$

Dann gilt

$$\begin{aligned} L_1 \cdot L_2 &= L_1 \cdot \{\lambda\} \cup L_1 \cdot \{0\} \\ &= L_1 \cup \left\{\ 0^i 0\ ;\ 1 \leq i \leq k\ \right\} \\ &= L_1 \cup \left\{\ 0^i\ ;\ 2 \leq i \leq k+1\ \right\} \\ &= \left\{\ 0^i\ ;\ 1 \leq i \leq k+1\ \right\}, \end{aligned}$$

und somit $|L_1 L_2| = k+1$.

Alternativ könnten Sie L_2 auch definieren als $L_2 = \{a, aa\}$ (ohne L_1 zu verändern).

1.3 Kapiteltest

Aufgabe 1.13

Markieren Sie die zutreffende(n) Antwort(en).

(a) Es sei Σ ein Alphabet.
- ☐ Σ^* ist eine unendliche Menge.
- ☐ Σ^* ist eine endliche Menge.
- ☐ Σ^* ist keine Menge.

(b) Es seien Σ_1, Σ_2 Alphabete. Seien L_1 eine endliche Sprache über Σ_1 und L_2 eine endliche Sprache über Σ_2. Dann gilt
- ☐ $|L_1L_2| \leq |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| = |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| > |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| \neq |L_1| \cdot |L_2|$.

(c)
- ☐ λ ist Element jedes Alphabets
- ☐ λ ist ein Wort über jedem Alphabet
- ☐ $|\lambda| = 0$
- ☐ λ ist Suffix jedes Wortes über jedem Alphabet

Aufgabe 1.14

Gegeben sei eine natürliche Zahl n. Bestimmen Sie die Anzahl der Wörter über Σ_{bool} der Länge n, welche

(a) das Teilwort 01 nicht enthalten.
(b) weder das Teilwort 01 noch das Teilwort 00 enthalten.
(c) das Teilwort 00 nicht enthalten (schwierig!).

Aufgabe 1.15

(a) Es sei $L := \{a, ab, ba\}$. Bestimmen Sie die Sprache L^2. Zur Erinnerung: $L^2 := L \cdot L$ (Konkatenation einer Sprache mit sich selbst).

(b) Existiert eine nichtleere endliche Sprache $L \neq \{\lambda\}$ über Σ_{bool}, welche die Gleichung

$$L = L^2$$

erfüllt? Begründen Sie Ihre Antwort.

Aufgabe 1.16

Beweisen oder widerlegen Sie folgende Gleichung:

$$\left(\{0,1\}^2\right)^* = \left(\{0,1\}^*\right)^2.$$

1.4 Lösungen zum Kapiteltest

✔ Lösungsvorschlag zu Aufgabe 1.13 ✔

(a) Es sei Σ ein Alphabet.
- ☑ Σ^* ist eine unendliche Menge.
- ☐ Σ^* ist eine endliche Menge.
- ☐ Σ^* ist keine Menge.

(b) Es seien Σ_1, Σ_2 Alphabete. Seien L_1 eine endliche Sprache über Σ_1 und L_2 eine endliche Sprache über Σ_2. Dann gilt
- ☑ $|L_1L_2| \leq |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| = |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| > |L_1| \cdot |L_2|$.
- ☐ $|L_1L_2| \neq |L_1| \cdot |L_2|$.

(c)
- ☐ λ ist Element jedes Alphabets
- ☑ λ ist ein Wort über jedem Alphabet
- ☑ $|\lambda| = 0$
- ☑ λ ist Suffix jedes Wortes über jedem Alphabet

✔ Lösungsvorschlag zu Aufgabe 1.14 ✔

(a) Enthält ein binäres Wort w der Länge $n \in \mathbb{N}$ nicht 01 als Teilwort, dann hat w die Form $w = 1^k0^m$ für zwei natürliche Zahlen k, m, welche die Gleichung $k + m = n$ erfüllen. Somit ist ein solches Wort durch die Wahl von k eindeutig bestimmt. Für k gelten die Ungleichungen $0 \leq k \leq n$. Somit gibt es genau $n+1$ mögliche Wahlmöglichkeiten für k und damit also genau $n + 1$ solche (gesuchten) Wörter der Länge n.

(b) Wir haben in Teil (a) gesehen, dass jedes Wort mit Länge, welches 01 nicht als Teilwort enthält, die Form 1^k0^m haben muss mit $k + m = n$. Alle Wörter für $n \leq 1$ erfüllen diese Bedingung. Es gibt ein Wort der Länge 0 und zwei Wörter der Länge 1, welche diese Bedingung erfüllen. Sei nun $n \geq 2$. Die einzigen solchen Wörter, welche nicht 00 als Teilwort enthalten, sind $1^{n-1}0$ und 1^n. Also, gibt es für $n \geq 2$ genau zwei solche Wörter.

✔ Lösungsvorschlag zu Aufgabe 1.15 ✔

(a) Es sei $L := \{a, ab, ba\}$. Dann ist L^2 gegeben durch

$$L^2 = \left\{a^2, a^2b, aba, (ab)^2, ab^2a, ba^2, ba^2b, (ba)^2\right\}.$$

(b) Nein, eine solche Sprache existiert nicht. Angenommen es gäbe eine solche Sprache L. Da L nichtleer ist, existiert in L ein Wort $w \in L$ maximaler Länge, also $|w| = \max\{\, |u| \;;\; u \in L \,\}$ und es gilt $|w| \geq 1$. Da $L^2 = L$, gilt $w^2 \in L$. Doch dann gilt $|w^2| = 2\,|w| > |w|$ und somit ist w nicht das längste Wort in L. Dies ist ein Widerspruch.

✔ Lösungsvorschlag zu Aufgabe 1.16 ✔

Die beiden Mengen sind verschieden. Offensichtlich gilt $0 \in (\{0,1\}^*)^2$ aber $0 \notin \left(\{0,1\}^2\right)^*$.

Kapitel 2

Algorithmische Probleme

Der Titel dieses Kapitels stellt uns vor ein kleines Dilemma. Wir möchten in diesem Kapitel bereits über *algorithmische Probleme* sprechen, bevor wir den Begriff *Algorithmus* in einem späteren Kapitel formal durch das Modell der Turingmaschine definieren werden. Aus diesem Grund werden wir anstelle von *Algorithmus* den Begriff *Programm* verwenden. Dabei setzen wir voraus, dass die Leserin / der Leser eine gewisse intuitive Vorstellung von Programmen hat (die Programmiersprache spielt dabei keine Rolle).

Wenn wir Programme als Algorithmen anschauen, werden wir jedoch zusätzlich fordern, dass solch ein Programm für jede zulässige Eingabe seine Arbeit in endlicher Zeit beendet (also nicht unendlich lange läuft) und eine Ausgabe liefert. Insbesondere ist es einem Algorithmus nicht gestattet in einer Endlosschleife zu laufen. Wenn ein Programm nach endlicher Zeit seine Arbeit beendet, sagt man auch, dass das Programm *hält*.

Unter diesen Bedingungen realisiert ein Programm (Algorithmus) A typischerweise eine Abbildung

$$A : \Sigma_1^* \to \Sigma_2^*$$

für Alphabete Σ_1 und Σ_2. Dies sagt aus, dass

- sowohl die Eingaben als auch die Ausgaben für das Programm (Algorithmus) als Wörter kodiert sind und
- A für jede Eingabe eine eindeutige Ausgabe bestimmt.

Für jeden Algorithmus A und jede Eingabe x bezeichnen wir mit $A(x)$ die Ausgabe des Algorithmus A für die Eingabe x (das Resultat, welches der Algorithmus für die Eingabe x berechnet).

Definition 2.1 (Äquivalenz von Algorithmen)**:**
Man sagt, dass zwei Algorithmen (Programme) A und B **äquivalent** sind, falls beide über dem gleichen Eingabealphabet Σ arbeiten und $A(x) = B(x)$ für alle $x \in \Sigma^*$ gilt.

Zwei Algorithmen sind also genau dann äquivalent, wenn sie das gleiche Eingabealphabet haben und auf allen möglichen Eingaben die gleiche Ausgabe liefern. Dabei spielt es keine Rolle, wie die jeweiligen Algorithmen ihre Ausgaben berechnen.

Wir werden im Folgenden einige sehr bekannte und wichtige algorithmische Probleme kennenlernen.

2.1 Das Entscheidungsproblem

Das *Entscheidungsproblem* hat eine besonders simple Formulierung und wird typischerweise verwendet, um die Theorie der Berechenbarkeit zu entwickeln.

Definition 2.2 (Entscheidungsproblem)**:**
Das **Entscheidungsproblem** (Σ, L) für ein gegebenes Alphabet Σ und eine gegebene Sprache $L \subseteq \Sigma^*$ ist, für jedes $x \in \Sigma^*$ zu entscheiden, ob

$$x \in L \quad \text{oder} \quad x \notin L.$$

Ein Algorithmus A **löst** das Entscheidungsproblem (Σ, L), falls für **alle** $x \in \Sigma^*$ gilt:

$$A(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{falls } x \notin L. \end{cases}$$

Das Entscheidungsproblem ist also ein geordnetes Paar (Σ, L) eines Alphabets Σ und einer Sprache L (über Σ).

Man sagt auch, dass A die Sprache L **erkennt**. Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt, dann nennt man die Sprache L **rekursiv**.

Wir verwenden eine Sprache $L \subseteq \Sigma^*$ häufig, um gewisse Eigenschaften von Wörtern zu erzwingen. Die Wörter in der Sprache L haben diese geforderte Eigenschaft und alle Wörter im mengentheoretischen Komplement $L^{\complement} = \Sigma^* \setminus L$ haben diese Eigenschaft nicht. In dieser Betrachtungsweise kann man ein Entscheidungsproblem auch wie folgt darstellen:

- **Eingabe**: $x \in \Sigma^*$.
- **Ausgabe**: $A(x) \in \Sigma_{\text{bool}} = \{0, 1\}$, wobei gilt

$$A(x) = \begin{cases} 1, & \text{falls } x \in L \quad \text{(Ja, } x \text{ hat die Eigenschaft)}, \\ 0, & \text{falls } x \notin L \quad \text{(Nein, } x \text{ hat die Eigenschaft nicht)}. \end{cases}$$

Wir werden nun einige Beispiele von Sprachen geben, welche gewisse (interessante) Eigenschaften fordern.

Beispiel 2.1:
Sei $\Sigma_1 = \{0, 1\} = \Sigma_{\text{bool}}$. Wir definieren $L_1 = \{\, x \in \Sigma_1^* \,;\, x = 1v, \text{ für ein } v \in \{0, 1\}^* \,\}$. In der Sprache L_1 liegen alle binären Wörter (alle endlichen Folgen von Nullen und Einsen), die mit einer 1 beginnen (die 1 als Präfix haben). Seien beispielsweise $x_1 = 110011$, $x_2 = 1$ und $x_3 = 011110$. Dann gilt $x_1, x_2 \in L_1$ aber $x_3 \notin L_1$. Das zugehörige Entscheidungsproblem ist das geordnete Paar (Σ_1, L_1).

Beispiel 2.2:
Sei $\Sigma_2 = \{\#, \alpha, \beta\}$. Wir definieren $L_2 = \{\, x \in \Sigma_2^* \,;\, x = (\alpha\#)^n, \text{ für ein } n \in \mathbb{N} \,\}$. In der Sprache L_2 liegen also alle Wörter der Form $(\alpha\#)^n$ für eine natürliche Zahl n. Das zugehörige Entscheidungsproblem lautet (Σ_2, L_2). Das kürzeste Wort in L_2 ist λ, das zweitkürzeste ist $\alpha\#$ und das drittkürzeste ist $\alpha\#\alpha\# = (\alpha\#)^2$.

Beispiel 2.3:
Eines der bekanntesten Entscheidungsprobleme ist der *Primzahltest*. Für eine natürliche Zahl n möchte man entscheiden, ob n eine Primzahl ist. Der Primzahltest entspricht dem Entscheidungsproblem

$$(\Sigma_{\text{bool}}, \{\ x \in (\Sigma_{\text{bool}})^*\ ;\ \text{Nummer}(x) \text{ ist eine Primzahl}\ \})\,.$$

Häufig wird dieses wichtige Entscheidungsproblem wie folgt dargestellt:

- **Eingabe**: $x \in (\Sigma_{\text{bool}})^*$.
- **Ausgabe**:
  - Ja, falls Nummer(x) eine Primzahl ist.
  - Nein, sonst.

Beispiel 2.4:
Die *GNU Compiler Collection*[a] (GCC) umfasst (unter anderem) ein `C++` Compiler. Ein `C++` Compiler ist ein spezielles Computerprogramm, welches als Input ein in der Programmiersprache `C++` geschriebenes Programm erhält, und dieses Programm in ein, für den Computer „verständliches“ Programm, umwandelt.

Eine Teilaufgabe des Compilers ist zu prüfen, ob ein gegebenes `C++` Programm syntaktisch korrekt ist. Ein Programm ist syntaktisch korrekt, wenn es keine zwingenden Regeln seiner entsprechenden Programmiersprache verletzt. Ein typisches Beispiel eines syntaktischen Fehlers ist das Öffnen einer linken Klammer „(“, die aber nirgends im Programm durch eine rechte Klammer „)“ geschlossen wird.

Achtung: Syntaktische Korrektheit eines Programms P garantiert lediglich, dass P die zwingenden Regeln seiner Programmiersprache nicht verletzt. Es wird nicht garantiert, dass P ein sinnvolles Programm ist.

Wir definieren $L_{\texttt{C++}}$ als die Sprache aller syntaktisch korrekter `C++` Programme:

$$L_{\texttt{C++}} = \{\ x \in (\Sigma_{\text{Tastatur}})^*\ ;\ x \text{ ist ein syntaktisch korrektes Programm in } \texttt{C++}\ \}$$

Das entsprechende Entscheidungsproblem lautet:

- **Eingabe**: $x \in (\Sigma_{\text{Tastatur}})^*$.
- **Ausgabe**:
  - Ja, falls $x \in L_{\texttt{C++}}$.
  - Nein, sonst.

Man beachte, dass jedes Computerprogramm als ein Wort über dem Alphabet der Computertastatur aufgefasst werden kann — schliesslich ist jedes Programm nichts weiter als eine endliche Folge von Symbolen der Tastatur.

[a] https://de.wikipedia.org/wiki/GNU_Compiler_Collection

2.2 Graphen

Einige sehr interessante Probleme und Konzepte lassen sich besonders gut durch *Graphen* modellieren. Um ein Gefühl für Graphen zu erhalten, betrachten wir Zugverbindungen zwischen den drei Städten Zürich (Z), Effretikon (E) und Winterthur (W). Jede der drei Städte ist jeweils mit den anderen beiden verbunden. Diese Situation lässt sich mit einem Graphen wie in Abbildung Abbil-

dung 2.1 darstellen.

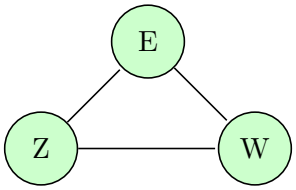


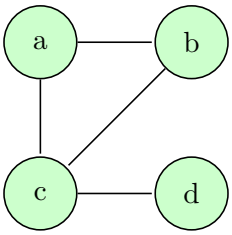
Abbildung 2.1: Zugverbindungen zwischen Zürich (Z), Effretikon (E) und Winterthur (W).

Die grünen Kreise (die Städte) bezeichnet man als *Knoten*, die schwarzen Verbindungslinien zwischen den Knoten nennt man *Kanten*. **Einen Graphen, in dem jeder Knoten mit jedem anderen Knoten verbunden ist, bezeichnet man als *vollständigen Graphen*** (der Graph in Abbildung Abbildung 2.1 ist ein vollständiger Graph).

Definition 2.3 (Graph)**:**
Ein Graph G ist ein geordnetes Paar (V, E), wobei V eine (endliche) nichtleere Menge von Knoten (englisch: *vertices*) ist. Die Menge E ist eine Teilmenge der zweielementigen Teilmengen von V, also $E \subseteq \{\ \{x, y\}\ ;\ x, y \in V, x \neq y\ \}$. Die Elemente der Menge E bezeichnet man als Kanten (englisch: *edges*). Falls die Knotenmenge V eine endliche Menge ist, nennt man den Graphen $G = (V, E)$ einen endlichen Graphen.

Eine Kante in einem Graphen verbindet zwei verschiedene Knoten. Die Kantenmenge E eines Graphen ist deshalb irgendeine Teilmenge (Auswahl) aller möglichen zweielementigen Teilmengen $\{x, y\}$ der Knotenmenge V. Dabei muss aber $x \neq y$ gelten, da wir die Verbindung eines Knotens zu sich selbst nicht als Kante anschauen möchten.

Für $V = \{a, b, c, d\}$ und $E = \{\{a, b\}, \{a, c\}, \{b, c\}, \{c, d\}\}$ würde der zugehörige Graph $G = (V, E)$ beispielsweise so aussehen:



Falls wir in unserem Beispiel mit den Zugverbindungen auch die Dauer der Verbindungen darstellen, dann könnten wir das wie in Abbildung 2.2 tun. Man sagt dann, dass die Kanten *gewichtet* sind. In diesem Beispiel sind die Gewichte die Fahrzeiten. Ein Graph mit gewichteten Kanten bezeichnet man als *gewichteten Graphen*. Graphen, ohne gewichtete Kanten nennt man *ungewichtete Graphen*.

Stellen wir uns nun vor, dass die Verbindung von Zürich nach Effretikon (und somit auch von Zürich nach Winterthur) aktuell unterbrochen ist, aber weiterhin Züge von Effretikon und Winterthur nach Zürich fahren können. Des Weiteren, haben wir bemerkt, dass die Verbindung von Effretikon nach Winterthur nur 9 Minuten dauert, die Verbindung von Winterthur nach Effretikon (wegen Stopp in Kemptthal) jedoch 11 Minuten. Diese Situation lässt sich durch *gerichtete* Kanten wie in Abbildung 2.3 veranschaulichen.

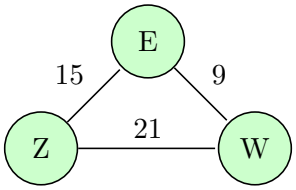


Abbildung 2.2: Gewichtete (ungerichtete) Zugverbindungen zwischen Zürich (Z), Effretikon (E) und Winterthur (W).

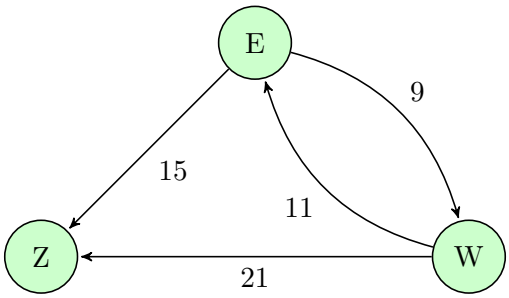
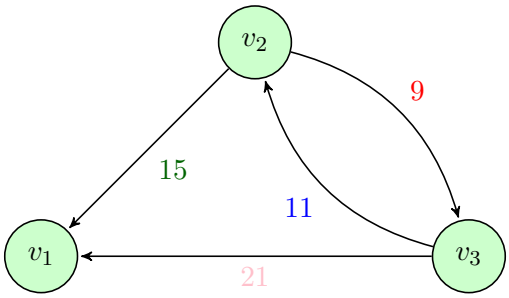


Abbildung 2.3: Gerichtete und gewichtete Zugverbindungen zwischen Zürich (Z), Effretikon (E) und Winterthur (W).

Natürlich ist es auch erlaubt Graphen zu konstruieren, die zwar gerichtet, aber nicht gewichtet sind. Um zu verdeutlichen, dass bei einer Gerichteten Kante die Reihenfolge (die Richtung) eine Rolle spielt, schreiben wir für gerichteten Kanten nicht eine zweielementige Menge $\{\text{Effretikon}, \text{Zürich}\}$, sondern ein geordnetes Paar $(\text{Effretikon}, \text{Zürich})$.

2.2.1 Adjazenzmatrix

Jeder endliche Graph $G = (V, E)$, egal ob gerichtet / ungerichtet oder gewichtet / ungewichtet, lässt sich einfach durch die ihm zugehörige *Adjazenzmatrix* A^G repräsentieren. Wir wollen für einen gegebenen Graphen G die Adjazenzmatrix A^G finden. Bevor wir das Vorgehen für einen beliebigen endlichen Graphen beschreiben werden, möchten wir die Adjazenzmatrix für den Graphen in Abbildung 2.3 finden. Wir führen diesen Graphen hier nochmals auf, allerdings haben wir zu didaktischen Zwecken die Kantengewichte des Graphen gefärbt und die Knoten (Städte) neu beschriftet (numeriert). Dazu haben wir die Bezeichnungen v_1 = Zürich, v_2 = Effretikon und v_3 = Winterthur gewählt.



Die zu diesem Graphen zugehörige Adjazenzmatrix A_G ist:

$$A^G = \begin{pmatrix} 0 & 0 & 0 \\ 15 & 0 & 9 \\ 21 & 11 & 0 \end{pmatrix} \tag{2.1}$$

Die Matrix A_G ist wie folgt zu verstehen:

Falls im Graphen eine Kantenverbindung vom Knoten v_i ($i \in \{1, 2, 3\}$) zum Knoten v_j ($j \in \{1, 2, 3\}$) besteht, dann setzt man den Matrix Eintrag $A^G_{i,j}$ (Eintrag in der i-ten Zeile und j-ten Spalte) auf das der Kante entsprechende Kantengewicht. Zum Beispiel besteht im Graphen die Kantenverbindung von Effretikon (v_2) nach Winterthur (v_3), mit einem Kantengewicht von 9. Damit wird der Eintrag $A^G_{2,3}$ der Adjazenzmatrix in der zweiten Zeile und der dritten Spalte auf den Wert 9 gesetzt. Besteht keine Verbindung von Knoten v_i nach v_j, so wird $A^G_{i,j} = 0$ gesetzt.

Bei ungewichteten Graphen ist der Matrix Eintrag $A^G_{i,j} = 1$, falls eine Kante vom Knoten v_i zum Knoten v_j besteht und $A^G_{i,j} = 0$, falls keine Verbindung besteht.

Es spielt keine Rolle, welche konkrete Nummerierung der Knoten im Graphen gewählt wird. Die Adjazenzmatrix wird immer genau eine vollständige Beschreibung des ihr entsprechenden Graphen liefern.

Vorgehen 2.1:
Aufstellen der Adjazenzmatrix bei gegebenem Graphen
Gegeben sei ein Graph $G = (V, E)$ als Bild oder als Beschreibung durch Mengen. Die zu G gehörige Adjazenzmatrix A^G wird wie folgt aufgestellt:

1. Sei $n := |V|$ die Anzahl der Knoten in G. Dann hat A^G eine Anzahl von n Zeilen und n Spalten (und ist somit eine quadratische Matrix).
2. Wähle eine beliebige Nummerierung der n Knoten von G und beschrifte den i-ten Knoten mit v_i für $i \in \{1, 2, \ldots, n\}$.
   Die genaue Form von A^G hängt von der konkreten Wahl der Nummerierung ab, die von A^G festgehaltene Information über G jedoch nicht.
3. Falls in G **keine** Kantenverbindung von v_i, $i \in \{1, 2, \ldots, n\}$, nach v_j, $j \in \{1, 2, \ldots, n\}$, existiert, dann setze $A^G_{i,j} = 0$. Falls eine solche Kantenverbindung existiert, dann setze
   - bei ungewichteten Graphen: $A^G_{i,j} = 1$,
   - bei gewichteten Graphen: $A^G_{i,j} = w_{i,j}$, wobei $w_{i,j}$ das Gewicht der Kante von v_i nach v_j ist.

Wir haben gesehen, wie ein gegebener Graph mit nummerierten Knoten eindeutig durch eine Adjazenzmatrix dargestellt werden kann. Umgekehrt, lässt sich aus einer gegebener Adjazenzmatrix ein eindeutiger Graph gewinnen. Damit ist eine Adjazenzmatrix eine alternative Repräsentation eines Graphen. Adjazenzmatrizen ungewichteter lassen sich einfach durch ein Wort über dem Alphabet $\{0, 1, \#\}$ darstellen (für gewichtete Graphen ist die Darstellung etwas weniger offensichtlich). Beispielsweise ist die Kodierung der Matrix

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

gegeben durch das Wort

0001#1010#1001#1100

Aufgabe 2.1

Überlegen Sie sich eine mögliche Kodierung der Adjazenzmatrix

$$\begin{pmatrix} 0 & 7 & 3 & 5 \\ 8 & 0 & 1 & 4 \\ 7 & 6 & 0 & 2 \\ 4 & 9 & 2 & 0 \end{pmatrix}$$

über dem Alphabet $\{0, 1, \#\}$

Aufgabe 2.2

Welche Einträge haben bei jeder Adjazenzmatrix immer den Wert 0?

2.2.1.1 Knotenüberdeckung (vertex cover)

Nun haben wir genügend Wissen über Graphen um ein sehr interessantes Entscheidungsproblem in Zusammenhang mit Graphen elegant beschreiben zu können. In diesem Unterunterabschnitt werden wir uns auf ungerichtete Graphen beschränken.

Definition 2.4 (Knotenüberdeckung (vertex cover))**:**
Man sagt, dass eine Kante $\{u, v\}$ ***inzident*** zu genau ihren Endpunkten u und v ist.

Eine ***Knotenüberdeckung*** (englisch: *vertex cover*) eines Graphen $G = (V, E)$ ist jede Knotenmenge $U \subseteq V$, sodass jede Kante aus E zu mindestens einem Knoten aus U inzident ist. Eine Teilmenge U der Knotenmenge V eines Graphen $G = (V, E)$ ist also eine Knotenüberdeckung, falls jede Kante $e \in E$ mindestens einen ihrer beiden Endpunkte in U hat.

Betrachten wir zum Beispiel den Graphen in Abbildung 2.4.

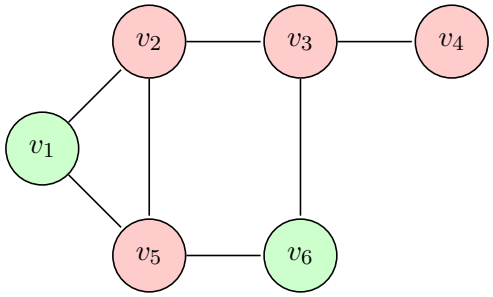


Abbildung 2.4: Graph mit 6 Knoten. Die Menge $\{v_2, v_3, v_4, v_5\}$ (rot markierte Knoten) ist eine Knotenüberdeckung des Graphen.

Die Menge $\{v_2, v_3, v_4, v_5\}$ der 4 rot markierten Knoten ist eine Knotenüberdeckung des Graphen, da jede der 7 Kanten des Graphen mindestens einer ihrer Endpunkte in dieser Menge hat.

Jedoch ist die Menge $\{v_2, v_3, v_4, v_5\}$ keine **minimale** Knotenüberdeckung, denn die Knotenmenge $\{v_2, v_3, v_5\}$ (siehe Abbildung 2.5) enthält einen Knoten weniger und ist ebenfalls eine Knotenüberdeckung.

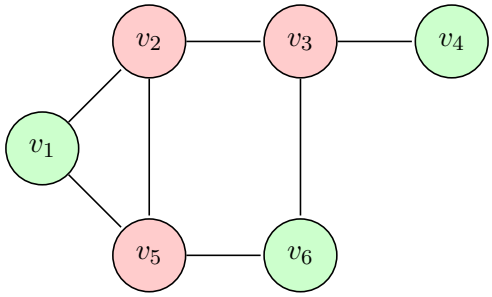


Abbildung 2.5: Die Menge $\{v_2, v_3, v_5\}$ (rot markierte Knoten) ist eine **minimale** Knotenüberdeckung des Graphen.

Definition 2.5 (Entscheidungsproblem der Knotenüberdeckung)**:**
Das **Entscheidungsproblem der Knotenüberdeckung**, besteht darin für einen gegebenen Graphen G und eine feste natürliche Zahl k zu entscheiden, ob G ein ungerichteter Graph ist, der eine Knotenüberdeckung mit maximal k vielen Knoten besitzt. Man möchte also für einen Graphen G und eine natürliche Zahl k entscheiden, ob das geordnete Paar (G, k) in der Menge

$$\text{VC} = \{(G,k)\ ;\ G \text{ ist ein ungerichteter Graph mit einer Knotenüberdeckung (vertex cover) der Mächtigkeit höchstens } k\}$$

liegt oder nicht.
Noch etwas genauer ausgedrückt:

$$\text{VC} = \{w \in \{0,1,\#\}^*\ ;\ w \text{ ist die Kodierung eines ungerichteten Graph mit einer Knotenüberdeckung (vertex cover) der Mächtigkeit höchstens } k\}$$

Wie man überprüfen kann, lässt der Graph in Abbildung 2.5 zum Beispiel keine Knotenüberdeckung der Mächtigkeit höchstens 2 zu.

2.3 Lösungen der Aufgaben

✔ Lösungsvorschlag zu Aufgabe 2.1 ✔

Jedes Kantengewicht w kodieren wir durch $\text{Bin}(w)$, die einzelnen Kodierungen der Gewichte werden durch # abgetrennt. Das Ende einer Zeile wird durch ## markiert. Damit erhalten wir folgende Kodierung der gegebenen Adjazenzmatrix:

0#111#11#101##1000#0#1#100##111#110#0#10##100#1001#10#0

✔ Lösungsvorschlag zu Aufgabe 2.2 ✔

Die sogenannten *Diagonalemente*. Dies sind die Einträge, bei denen die Zeilennummer mit der Spaltennummer übereinstimmt, also alle Einträge der Form $A^G_{i,i}$. Angenommen, ein Diagonalelement wäre nicht 0. Dann würde im zur Adjazenzmatrix gehörenden Graphen eine Verbindung der Form $\{v_i, v_i\}$ beziehungsweise (v_i, v_i) bestehen. Dies darf aber gemäss der Definition der Kantenmenge nicht sein, da sie Kantenverbindungen von einem Knoten zu sich selbst ausschliesst.

Kapitel 3

Endliche Automaten

Endliche Automaten sind das einfachste Berechnungsmodell, das man in der Informatik betrachtet. In einer ersten Phase möchten wir eine gute intuitive Vorstellung der Arbeitsweise von endlichen Automaten gewinnen. Dazu werden wir in Unterabschnitt 3.1.1 endliche Automaten mit (leicht modifizierten) gerichteten Graphen identifizieren. In Unterabschnitt 3.1.2 werden wir sehen, dass endliche Automaten äquivalent zu einer Darstellung durch spezielle Programme sind. Diese speziellen Programme lösen gewisses Entscheidungsproblem, ohne bei ihrer Arbeit Variablen zu benutzen.

Mithilfe dieser Intuition wird uns die formale Definition endlicher Automaten in Unterabschnitt 3.1.3 elegant und natürlich erscheinen.

Die Betrachtung endlicher Automaten ist hervorragend dazu geeignet, zentrale Begriffe der Informatik wie *Konfiguration*, *Berechnungsschritt*, *Simulation* und *Berechnung* schonend einzuführen.

3.1 Darstellung endlicher Automaten

Wir werden drei verschiedene Möglichkeiten zur Darstellung endlicher Automaten kennenlernen:

1. Darstellung durch gerichtete Graphen (Graphendarstellung)
2. Darstellung durch Programme (Programmdarstellung)
3. Formale Definition (durch Mengen und Funktionen)

3.1.1 Darstellung durch gerichtete Graphen

Die Darstellung endlicher Automaten durch (etwas modifizierte) gerichtete Graphen ist besonders prägnant und anschaulich. Wir werden deshalb endliche Automaten mit gerichteten Graphen identifizieren (Graphendarstellung). Wir werden im Folgenden nicht mehr zwischen einem endlichen Automaten A und seiner Darstellung $G(A)$ als gerichteter Graph unterscheiden.

Wenn man ein Berechnungsmodell definieren möchte, muss man folgende vier Fragen beantworten können:

1. Welcher Speicher steht zur Verfügung und wie wird dieser verwendet?
2. Wie wird die Eingabe (Input) eingegeben?
3. Wie wird die Ausgabe (Output) ausgegeben?
4. Welche elementaren Rechenoperationen kann das Berechnungsmodell durchführen?

Bei endlichen Automaten hat man keinen Speicher zur Verfügung ausser dem Speicher, in dem der (modifizierte) gerichtete Graph gespeichert ist und einem *Zeiger*, der auf den aktuell betrachteten

Knoten des Graphen zeigt. Die einzige wechselnde (nicht statische) Information ist der Name des Knoten, auf den der Zeiger aktuell zeigt.

Um konkret über endliche Automaten sprechen zu können, betrachten wir den einfachen endlichen Automaten A in Abbildung 3.1. Wir werden den Aufbau und die Funktionsweise endlicher Automaten anhand dieses konkreten endlichen Automaten A erklären.

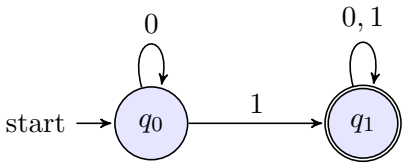


Abbildung 3.1: Darstellung von A als gerichteter Graph.

- A hat zwei **Zustände** q_0 und q_1. Die **Zustandsmenge** Q von A ist somit $Q = \{q_0, q_1\}$. Allgemein, entspricht die Zustandsmenge Q der Knotenmenge V des Graphen.
- Der Zeiger zeigt zu Beginn auf einen Zustand und ist mit dem Label „start“ besonders markiert. Der Zustand, auf den der Zeiger zu Beginn zeigt, wird **Anfangszustand** genannt. Manchmal schreibt man anstelle des Labels „start“ auch „λ“, weil sich der Automat nach dem Lesen des leeren Wortes λ im Anfangszustand befindet. Man darf auf das Label des Zeigers auch verzichten. Der Anfangszustand von A ist q_0.
- Wenn der Zeiger des endlichen Automaten auf den Zustand q zeigt, dann sagt man, dass sich der Automat im Zustand q befindet.
- Jedem endlichen Automaten ist ein **Eingabealphabet** Σ zugeordnet. Das Eingabealphabet von A ist $\Sigma = \{0, 1\}$. Alle zulässigen Eingaben w für eines endlichen Automaten müssen Wörter über seinem Eingabealphabet sein. Der endliche Automat erhält ein **Eingabewort** w über Σ und liest dieses Buchstabe um Buchstabe von links nach rechts. Wenn der endliche Automat das gesamte Wort w gelesen hat, ist seine Arbeit auf w beendet.
- Falls sich A zum Beispiel im Zustand q_0 befindet und das Symbol 1 liest, dann folgt A der Kante des Graphen, welche mit 1 beschriftet ist. Dadurch gelangt A in den Zustand q_1. Mit dem Lesen von 0 im Zustand q_0 bleibt A im Zustand q_0.
  Allgemein gilt: Falls sich ein endlicher Automat im Zustand q befindet und ein Symbol α von der Eingabe liest, dann folgt der Automat der Kante des Graphen, welche mit α beschriftet ist. Die Destination der Kante wird sein neuer Zustand sein. Mental kann man sich vorstellen, dass der Zeiger auf diesen neuen Zustand zeigt.
- Falls sich ein endlicher Automat **nach dem vollständigen Lesen** eines Eingabewortes w in einem doppelt umkreisten Zustand befindet, dann akzeptiert der Automat die Eingabe w, ansonsten akzeptiert der Automat die Eingabe w nicht (verwirft die Eingabe). A hat nur einen **akzeptierend Zustand**, nämlich q_1. Der Zustand q_0 ist **nicht akzeptierend**.

Um den Umgang mit endlichen Automaten zu üben, betrachten wir die Arbeit von A auf dem konkreten Eingabewort $w = 00101$. A befindet sich zu Beginn in seinem Anfangszustand q_0. Nun beginnt er damit das Eingabewort von links nach recht zu lesen (Buchstabe um Buchstabe). Halten Sie sich im Folgenden Abbildung 3.1 vor Augen.

1. Der erste Buchstabe (der Buchstabe ganz links) des Eingabeworts ist 0. Wir müssen schauen, wohin die gerichtete Kante, ausgehend vom aktuellen Zustand q_0, beim Lesen von 0 führt. Wie wir sehen, führt uns die Kante, die mit einer 0 markiert ist, zurück in den Zustand q_0. Wir bleiben also im Zustand q_0.
2. Danach liest A nochmals den Buchstaben 0. Genau wie im ersten Schritt, werden wir wieder der Kanten mit der 0, die vom Zustand q_0 ausgeht, folgen. Damit befindet sich A nach dem Lesen des Präfixes **00** von $w = \mathbf{00}101$ noch immer im Zustand q_0. Es bleibt noch das Suffix

101 von w zu lesen.
3. Nun liest A das Symbol 1 und wir müssen von unserem aktuellen Zustand q_0 aus, der mit 1 markierten Kante folgen. Dadurch gelangt A in den Zustand q_1.
4. A liest im Zustand q_1 das Symbol 0. Die mit 0 markierte Kante, die von q_1 ausgeht, führt zurück nach q_1.
5. A liest im Zustand q_1 das Symbol 1. Die mit 1 markierte Kante, der von q_1 ausgeht, führt ebenfalls zurück nach q_1.
6. A hat das Eingabewort w nun vollständig gelesen und beendet seine Arbeit im akzeptierenden Zustand q_1. Damit akzeptiert A das Eingabewort $w = 00101$.

A akzeptiert genau die Eingabewörter aus $\{0,1\}^*$, die mindestens eine 1 enthalten. Die Menge aller Wörter, welche von A akzeptiert werden ist somit

$$\{\ w \in \{0,1\}^*\ ;\ w = x1y \text{ mit } x,y \in \{0,1\}^*\ \} \tag{3.1}$$

Dies kann man wie folgt einsehen:

A hat nur den einen akzeptierenden Zustand q_1. Beginnend im Anfangszustand q_0, wird der A solange im Zustand q_0 bleiben (solange Nullen lesen), bis er das erste Mal eine 1 liest. Durch das Lesen dieser 1 gelangt A in den Zustand q_1. Beide (alle) von q_1 ausgehenden Kanten führen direkt zurück zu q_1. Deshalb wird der A nach dem erstmaligen Lesen einer 1, solange im akzeptierenden Zustand q_1 verbleiben, bis er die gesamte Eingabe gelesen hat und somit seine Arbeit in dem akzeptierenden Zustand beendet. A akzeptiert somit alle binären Eingabewörter, die mindestens eine 1 enthalten[1], was genau der Menge in Ausdruck Gleichung (3.1) entspricht.

Man bemerke, dass die Menge

$$L := \{\ w \in \{0,1\}^*\ ;\ w = x1y \text{ mit } x,y \in \{0,1\}^*\ \} \subseteq \{0,1\}^* \tag{3.2}$$

eine Teilmenge von $\{0,1\}^*$ ist. Somit ist L eine Sprache über dem Alphabet $\{0,1\}$. Die Menge L ist die Sprache aller Wörter, die von dem endlichen Automaten A akzeptiert werden.

Beispiel 3.1:
Sei L_{abb} die Sprache aller Wörter über $\Sigma = \{a,b,c\}$, welche mit a beginnen und mit bb enden, genauer:

$$L_{abb} := \{\ w \in \{a,b,c\}^*\ ;\ w = axbb,\ \text{für ein } x \in \{a,b,c\}^*\ \}\,. \tag{3.3}$$

Entwerfen Sie einen endlichen Automaten (in grafischer Form), welcher die Sprache L akzeptiert.

Lösung:

[1] Die Funktionsweise dieses endlichen Automaten könnte in Computer-Hardware zum Beispiel durch einen sogenannten *Latch* (*to latch* heisst auf Deutsch „einrasten“, „einklinken“) umgesetzt werden.

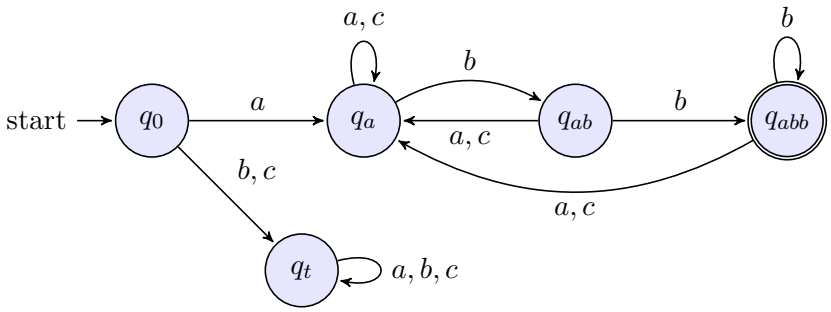


Abbildung 3.2: Endlicher Automat A für die Sprache L_{abb} in Ausdruck Gleichung (3.3).

Erklärung:

Sei $w \in \{a, b, c\}^*$ ein Eingabewort für den endlichen Automaten A in Abbildung 3.2. A beginnt seine Arbeit im Zustand q_0 und liest den ersten Buchstaben von w.

- Falls w nicht mit a beginnt (sondern mit b oder c), geht A beim Lesen des ersten Buchstabens von w in den nicht akzeptierend *Abfallzustand* (trash) q_t über. Dort wird der Automat bleiben, egal welche weiteren Buchstaben von w er liest. A wird für w im nicht akzeptierenden Zustand q_t seine Arbeit beenden ($\rightarrow w$ wird verworfen).
- Falls w mit a beginnt, wechselt A in den Zustand q_a.
- Falls A im Zustand q_a die Buchstaben a oder c liest, dann bleibt A im Zustand q_a.
- Falls A im Zustand q_a den Buchstaben b liest, geht er in den Zustand q_{ab} über.
- Die Wörter, welche den endlichen Automaten A in den Zustand q_{ab} führen, beginnen mit a, enden mit b, haben aber an ihrer zweitletzten Stelle nicht das Symbol b (siehe dazu Aufgabe 3.3).
- Falls A in q_{ab} die Buchstaben a oder c liest, dann kann das bislang gelesene Präfix von w nicht mit b enden und A geht zurück zu q_a.
- Falls A in q_{ab} nochmals ein b liest, endet das bislang gelesene Präfix von w mit bb und A wechselt in den akzeptierenden Zustand q_{abb}.
- A bleibt beim weiteren Lesen von b in q_{abb} (das gelesene Präfix von w endet dann immer noch mit bb), ansonsten muss der Automat zurück in den Zustand q_a.

Bemerkung 3.1:
Beachten Sie, dass ein endlicher Automat ein Eingabewort w nicht sofort akzeptiert, nur weil er während seiner Arbeit auf w akzeptierenden Zustände (einmal oder mehrfach) durchläuft. Die Entscheidung „*akzeptieren* oder *verwerfen*“ wird erst mit dem Lesen des letzten Buchstabens von w getroffen. Befindet sich der Automat nach dem Lesen des letzten Buchstabens von w in einem verwerfenden Zustand, dann verwirft der Automat w. Befindet er sich dann in einem akzeptieren Zustand, dann akzeptiert der Automat w.

Bei der Arbeit auf dem Wort $w = abbc$ zum Beispiel, durchläuft der endliche Automat in Abbildung 3.2 zwar den akzeptierenden Zustand q_{abb}, beendet seine Arbeit aber im nicht akzeptierenden Zustand q_a. Dies ist auch richtig, denn $w = abbc$ liegt nicht in der Sprache L_{abb}.

Bei der Graphendarstellung, müssen von jedem Knoten genauso viele Kanten (Pfeile) ausgehen, wie das Eingabealphabet des Automaten Symbole hat. Wenn der endliche Automat das Eingabealphabet Σ hat, dann muss der Ausgangsgrad (Anzahl der gerichteten Kanten, welche von dem Knoten

ausgehen) jedes Knoten genau $|\Sigma|$ sein. Wäre der Ausgangsgrad eines Knoten kleiner, dann wäre die Arbeit des endlichen Automaten auf mindestens einem Symbol seines Eingabealphabets nicht definiert, was man natürlich nicht möchte.

Aufgabe 3.1

Entwerfen Sie einen endlichen Automaten für die Sprache

$$L_1 := \{\ w \in \{0,1\}^*\ ;\ w = x0100y \text{ mit } x, y \in \{0,1\}^*\ \}\,. \tag{3.4}$$

Aufgabe 3.2

Beschreiben Sie die Sprache L_2, der von dem endlichen Automaten in Abbildung 3.3 akzeptierten Wörter.

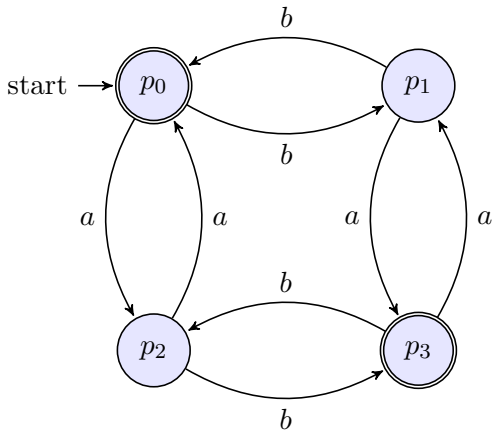


Abbildung 3.3: Endlicher Automat für die Sprache L_2.

Aufgabe 3.3

Betrachten Sie den endlichen Automaten in Abbildung 3.2. Beschreiben Sie mathematisch präzise die Teilmenge M_{ab} aller Wörter über $\{a, b, c\}$, welche in den Zustand q_{ab} führen.

3.1.2 Darstellung durch Programme

In diesem Unterabschnitt werden wir sehen, dass die Graphendarstellung von endlichen Automaten äquivalent zur Darstellung durch gewisse spezielle Programme (Programmdarstellung) ist.

Die Programme sind speziell in dem Sinne, dass sie lediglich Speicher zur Abspeicherung des Programms und zum Speichern eines „Zeigers“, der auf die aktuell auszuführende Zeile des Programms zeigt. Dies bedeutet aber, dass das Programm keinen Speicher für Variablen zur Verfügung hat. Der Inhalt des Zeigers, also die Nummer der aktuellen Programmzeile, ist die einzige wechselnde Information.

Um die Äquivalenz einzusehen, werden wir eine Methode demonstrieren, welche jede Graphendarstellung in ein äquivalentes Programm übersetzt (und umgekehrt). Bevor wir die Methode allgemein beschreiben, möchten wir das Vorgehen anhand eines Beispiels entwickeln. Betrachten Sie dazu den endlichen Automaten in Abbildung 3.4.

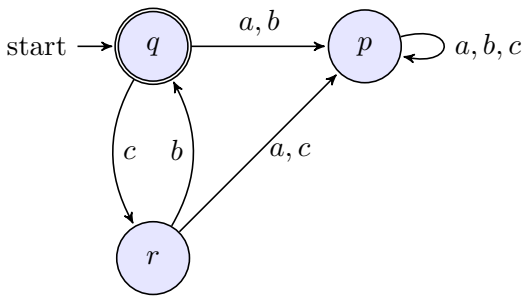


Abbildung 3.4: Endlicher Automat mit der Zustandsmenge $\{q, p, r\}$.

In einem ersten Schritt möchten wir die Zustände q, p, r durch die Nummern $0, 1, 2$ ersetzen. Die Vergabe der Nummern kann beliebig gewählt werden[2], aber dem Anfangszustand muss die Nummer 0 zugewiesen werden. Wir wählen für p die Nummer 1 und für r die Nummer 2. Damit erhalten wir den „nummerierten Graphen“ in Abbildung 3.5.

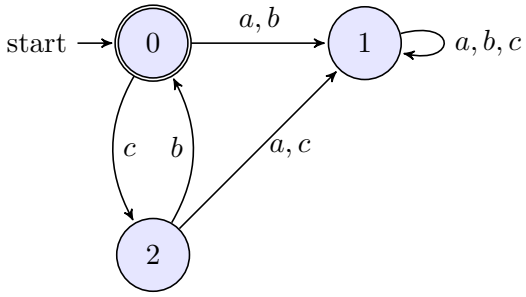


Abbildung 3.5: Nummerierte Version des endlichen Automaten in Abbildung 3.4 mit Zustandsmenge $\{0, 1, 2\}$.

Wir möchten nun ein Programm entwerfen, welches den endlichen Automaten in Abbildung 3.5 beschreibt, jedoch ohne Variablen auskommt. Das Programm soll genau so viele Zeilen haben, wie der endliche Automat Zustände (Knoten) hat. In diesem Fall wird das Programm entsprechend genau drei Zeilen haben. Die Nummerierung der Zustände wird dabei genau der Nummerierung der Programmzeilen entsprechen. Deshalb ist es wichtig, dass der Anfangszustand die Nummer 0 erhält, da das Programm von oben nach unten abgearbeitet wird und somit bei Zeile 0 seine Arbeit beginnt. Das Programm erhält das Eingabewort des endlichen Automaten als Input.

Falls das erste Symbol des Input a oder b ist, dann soll das Programm zu Zeile 1 springen. Falls das erste Symbol ein c ist, soll das Programm zu Zeile 2 springen. Nach dem Lesen eines Symbols des Eingabebandes wird dieses automatisch gelöscht und das nächste wird gelesen. Analog ist das Vorgehen für die anderen beiden Zeilen 1 und 2 des Programms. Insgesamt wird die Programmdarstellung des endlichen Automaten in Abbildung 3.5 durch folgendes Programm beschrieben:

```
if input = a goto 1, if input = b goto 1, if input = c goto 2;
if input = a goto 1, if input = b goto 1, if input = c goto 1;
if input = a goto 1, if input = b goto 0, if input = c goto 1;
```

Falls der endliche Automat in Abbildung 3.5 im Zustand 0 das Symbol a oder b liest, dann wechselt er in den Zustand 1. Falls er das Symbol c liest, dann wechselt er in den Zustand 2. Genau so geht auch unser obiges Programm vor:

[2]Die zugehörigen Programme werden die gleichen Entscheidungen treffen und bis auf einfache Vertauschungen von Nummern identisch sein. Siehe dazu auch Aufgabe 3.4.

Falls (`if`) das Programm in Zeile 0 das Symbol a vom Input erhält (liest), springt es mit dem `goto` Befehl zur Zeile 1. Dasselbe gilt, wenn in Zeile 0 der Input b gelesen wird. Falls in Zeile 0 der Input c gelesen wird, springt das Programm zur Zeile 2. Das Vorgehen für die Programmzeilen 1 und 2 (beziehungsweise die Zustände 1 und 2) ist völlig analog.

Aufgabe 3.4

Wie müsste obiges Programm angepasst werden, falls wir für die Zustände in Abbildung 3.4 die Nummerierung $q \leftrightarrow 0$, $r \leftrightarrow 1$ und $p \leftrightarrow 2$ gewählt hätten?

Nun sollte deutlicher geworden sein, wie auch ein beliebiger endlicher Automat in ein solch spezielles Programm ohne Variablen transformiert werden kann. Das zu einem endlichen Automaten A (beziehungsweise seiner Graphendarstellung $G(A)$) gehörige Programm werden wir mit $P(A)$ bezeichnen.

Sei $\Sigma = \{s_1, \ldots, s_m\}$ das Eingabealphabet eines endlichen Automaten A. **Eine Zeile** von $P(A)$ hat die Form:

erlaubter Befehl (eine Zeile):

if input $= s_1$ **goto** i_1, **if** input $= s_2$ **goto** i_2, ..., **if** input $= s_m$ **goto** i_m; (3.5)

Die Bedeutung dieses Befehls ist, dass man das nächste Symbol der Eingabe liest und mit $s_1, s_2, \ldots, s_m$ vergleicht. Falls dieses Symbol gleich s_j ist, setzt das Programm seine Arbeit in der Zeile i_j fort. Dabei wird das gelesene Symbol gelöscht und in der Zeile i_j wird das nächste Symbol der Eingabe gelesen. Man beachte, dass die Zeilen des Programms nummeriert sind, wobei die Nummerierung bei 0 beginnt.
Diese Programme lösen Entscheidungsprobleme. Die Ausgabe des Programms ist bestimmt durch die Zeilennummer, in der das Programm seine Arbeit beendet. Falls das Programm aus k Zeilen besteht, wählt man eine Teilmenge F von $\{0, 1, \ldots, k-1\}$ aus. Wenn das Programm nach dem vollständigen Lesen der Eingabe in der j-ten Zeile endet, und $j \in F$, dann akzeptiert das Programm die Eingabe. Falls $j \in (\{0, 1, \ldots, k-1\} \setminus F)$, dann akzeptiert das Programm die Eingabe nicht.

Die eins-zu-eins-Korrespondenz zwischen einem endlichen Automaten A und seiner Programmdarstellung $P(A)$ sollte nun offensichtlich sein:

Man nummeriert die Zustände von A. $P(A)$ hat genau so viele Zeilen wie der A Zustände hat, und jedem Zustand von A ist genau eine Zeile in $P(A)$ zugeordnet, welche die Nummer des Zustands trägt. Falls A beim Lesen eines Symbols α von Zustand i in Zustand j übergeht (eine gerichtete Kante mit Markierung α von i nach j hat), dann springt das Programm in Zeile i beim Lesen von α in die Zeile j.

Aufgrund dieser Äquivalenz kann ein endlicher Automat auch durch sein Programm $P(A)$ identifiziert werden.

Mit endlichen Automaten verbindet man oft die schematische Darstellung in Abbildung 3.6. Das Modell besteht aus den drei Hauptkomponenten **Eingabeband**, **Lesekopf** und **Programm**. Das Programm haben wir oben im Detail beschrieben. Das Eingabeband besteht aus einzelnen Feldern. Ein Feld ist eine atomare Speichereinheit, die ein Symbol des betrachteten Alphabets beinhalten kann. Das Eingabeband hat in diesem Modell die Bedeutung des Eingabeworts. Der Lesekopf kann sich entlang des Eingabebandes nur von links nach recht bewegen. Der Lesekopf liest den Inhalt des Feldes, auf das er zeigt.

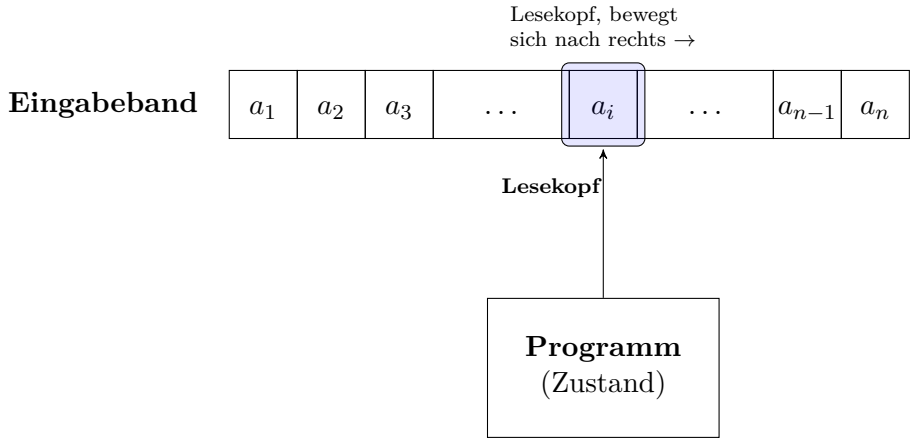


Abbildung 3.6: Schematische Darstellung eines endlichen Automaten.

3.1.3 Formale Definition

Nach den Betrachtungen in Unterabschnitt 3.1.1 und Unterabschnitt 3.1.2 wollen wir eine formale Definition von endlichen Automaten angeben. Diese Definition wird uns sehr natürlich erscheinen.

Definition 3.1 (endlicher Automat)**:**
Ein (deterministischer) ***endlicher Automat*** M ist ein 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$, wobei die einzelnen Komponenten des 5-Tupels wie folgt zu verstehen sind:

(i) Q ist eine endliche Menge von ***Zuständen***.
Die Zustände entsprechen der Menge der Zeilen in der Programmdarstellung und der Knotenmenge in der Graphendarstellung.

(ii) Σ ist ein Alphabet, genannt ***Eingabealphabet***.
Die zulässigen Eingaben für den endlichen Automaten sind alle Wörter über Σ. Die Bedeutung des Eingabealpabets ist bei der Programmdarstellung und Graphendarstellung genau gleich.

(iii) q_0 ist der ***Anfangszustand***.
q_0 entspricht dem mit „start“ markierten Knoten in der Graphendarstellung und der Zeile 0 in der Programmdarstellung.

(iv) $F \subseteq Q$ ist die ***Menge der akzeptierenden Zustände***.
Dies entspricht der Menge der akzeptierenden Knoten in der Graphendarstellung und Menge der akzeptierenden Programmzeilen in der Programmdarstellung.

(v) δ ist eine Funktion $Q \times \Sigma \to Q$, die ***Übergangsfunktion*** genannt wird.
Die Übergangsfunktion erhält zwei Argumente. Das erste Argument $q \in Q$ ist der Zustand, in dem sich der endliche Automat M aktuell befindet. Das zweite Argument $a \in \Sigma$ ist ein Symbol (des Eingabeworts), welches der M aktuell liest. $\delta(q, a) = p$ bedeutet, dass M in den Zustand p übergeht, falls M im Zustand q das Symbol a gelesen hat. In der Graphendarstellung von M existiert eine gerichtete Kante (Knoten), welche mit a beschriftet ist und vom Zustand (Knoten) q zum Zustand (Knoten) p führt. In der Programmdarstellung von M existiert eine Programmzeile q^a der Form Gleichung (3.5), welche durch einem `goto` Befehl beim Lesen des Inputs a zur Zeile p (der Zeilennummer, welche p zugeordnet ist).

[a] Genauer gesagt, eine Programmzeile mit der Nummer, welcher dem Zustand q bei der Nummerierung der Zustände zugeordnet wird.

Eine ***Konfiguration*** von M ist ein Element aus $Q \times \Sigma^*$.

Wenn sich M in einer Konfiguration $(q, w) \in (Q \times \Sigma^*)$ befindet, bedeutet dies, dass sich M im Zustand q befindet und noch das Suffix w eines Eingabewortes lesen muss.

Die Konfiguration $(q_0, w) \in (\{q_0\} \times \Sigma)$ ist die ***Startkonfiguration von*** M ***auf*** w.

M beginnt seine Arbeit auf dem Wort w im Zustand q_0.

Jede Konfiguration aus $Q \times \{\lambda\}$ wird ***Endkonfiguration*** genannt. Das gesamte Eingabewort wurde gelesen. Es bleibt nichts mehr (nur noch das leere Wort λ) zu lesen.

Der Ausdruck $(q, w) \xrightarrow{M} (p, v)$ soll die Bedeutung haben, dass die Konfiguration (p, v) in einem ***Schritt*** von M aus der Konfiguration (q, w) erreicht wird:

$$(q, w) \xrightarrow{M} (p, v) \quad \Longleftrightarrow \quad w = av, a \in \Sigma \text{ und } \delta(q, a) = p$$

Da $w = av$, ist a das vorderste Symbol in w und wird deshalb als nächstes gelesen.

Ein Schritt entspricht der Anwendung der Übergangsfunktion auf die Konfiguration, in der sich M im Zustand q befindet und das Symbol a liest.

Eine ***Berechnung*** C von M ist eine endliche Folge $C = C_0, C_1, \ldots, C_n$ von Konfigurationen, mit $C_i \xrightarrow{M} C_{i+1}$ für alle $i \in \{0, 1, \ldots, n-1\}$.

C ist **die Berechnung von** M **auf einem Eingabewort** $x \in \Sigma^*$, falls $C_0 = (q_0, x)$ und $C_n \in (Q \times \{\lambda\})$ (das heisst, C_n ist eine Endkonfiguration).

- Falls $C_n \in (F \times \{\lambda\})$ (zur Erinnerung: F ist die Menge der akzeptierenden Zustände), dann ist C eine **akzeptierende Berechnung** von M auf x. Man sagt dann, dass M **das Wort** x **akzeptiert.**
  M endet seine Arbeit auf dem Eingabewort x in einem akzeptierenden Zustand.
- Falls $C_n \in ((Q \setminus F) \times \{\lambda\})$, dann ist C eine **verwerfende Berechnung** von M auf x. Man sagt dann, dass M **das Wort** x **verwirft** (oder **nicht akzeptiert**). M endet seine Arbeit auf dem Eingabewort x in einem verwerfenden Zustand.

An dieser Stelle möchten wir betonen, dass ein endlicher Automat M auf einem Wort $x \in \Sigma^*$ offensichtlich genau eine Berechnung hat.

Wir möchten eine sehr interessante und wichtige Sprache definieren.

Definition 3.2 (akzeptiere Sprache eines endlichen Automaten)**:**
Sei M ein endlicher Automat. Die **von** M **akzeptierte Sprache** $L(M)$ ist definiert als

$$L(M) = \{w \in \Sigma^* \text{ ; die Berechnung von } M \text{ auf } w \text{ endet in einer Endkonfiguration } (q, \lambda), \text{ mit } q \in F\}$$

Definition 3.3 (Klasse der regulären Sprachen)**:**
$\mathcal{L}_{EA} = \{\ L(M)\ ;\ M$ ist ein endlicher Automat $\}$ ist die Klasse der Sprachen, welche von endlichen Automaten akzeptiert werden. $\mathcal{L}_{EA}$ wird als die **Klasse der regulären Sprachen** bezeichnet. Jede Sprache L aus $\mathcal{L}_{EA}$ wird **regulär** (**reguläre Sprache**) genannt.

Betrachten wir nochmals den endlichen Automaten in Abbildung 3.4. Diesen Automaten wollen wir **formal** beschreiben. Dazu genügt es, die 5 Komponenten des 5-Tupels $(Q, \Sigma, \delta, q_0, F)$ gemäss Definition 3.1, anzugeben. Der zur Graphendarstellung in Abbildung 3.4 äquivalente endliche Automat ist $M = (Q, \Sigma, \delta, q_0, F)$, wobei:

$$\begin{aligned} &Q = \{q, p, r\}, \Sigma = \{a, b, c\}, q_0 = q, F = \{q\} \text{ und} \\ &\delta(q, a) = p, \quad \delta(q, b) = p, \quad \delta(q, c) = r \\ &\delta(p, a) = p, \quad \delta(p, b) = p, \quad \delta(p, c) = p \\ &\delta(r, a) = p, \quad \delta(r, b) = q, \quad \delta(r, c) = p \end{aligned}$$

Man beachte, dass sich die Übergangsfunktion δ vollständig definieren lässt, indem man den Wert der Funktion für alle möglichen Argumente angibt.

Aufgabe 3.5

Geben Sie die formale Definition des in Abbildung 3.3 dargestellten endlichen Automaten an.

Zum Abschluss dieses Unterabschnitts wollen wir noch zwei nützliche Definitionen vorstellen.

Die Schreibweise

$$(q, w) \xrightarrow[*]{M} (p, u)$$

soll bedeuten, dass eine Berechnung von M existiert, die von der Konfiguration (q, w) zur Konfiguration (p, u) führt. Diese Intuition wollen wir etwas genauer in einer Definition festhalten.

Definition 3.4 (Stern-Operation (transitive Hülle der Schrittrelation))**:**
Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Die Operation $\xrightarrow[*]{M}$ werden wir *Stern-Operation* nennen. Die Stern-Operation ist wie folgt definiert:

$$(q, w) \xrightarrow[*]{M} (p, u) \iff (q = p \text{ und } w = u) \text{ oder es existiert ein } k \in \mathbb{N} \setminus \{0\},$$

sodass

(i) $w = a_1 a_2 \dots a_k u$, $a_i \in \Sigma$, für $i = 1, 2, \dots, k$ und
(ii) es existieren $(k-1)$ Zustände $r_1, r_2, \dots, r_{k-1} \in Q$, sodass
$(q, w) \xrightarrow{M} (r_1, a_2 a_3 \dots a_k u) \xrightarrow{M} (r_2, a_3 a_4 \dots a_k u) \xrightarrow{M} \dots (r_{k-1}, a_k u) \xrightarrow{M} (p, u)$.

Gemäss dieser Definition gilt $(q, w) \xrightarrow[*]{M} (p, u)$ genau dann, wenn entweder $q = p$ und $w = u$ (dann sind die Konfigurationen (q, w) und (p, u) identisch und M hat (p, u) trivialerweise schon erreicht) oder wenn Zwischenzustände existieren, über welche M bei seiner Berechnung auf w zur Konfiguration (p, u) gelangt.

Im engen Zusammenhang mit der Stern-Operation steht die Schreibweise

$$\widehat{\delta}(q, w) = p.$$

Diese soll die Bedeutung haben, dass wenn M im Zustand q das Wort w zu lesen beginnt, dann endet M im Zustand p. Dies ist gleichbedeutend mit der Aussage $(q, w) \xrightarrow[*]{M} (p, \lambda)$. Man kann sich die Operation $\widehat{\delta}(q, w) = p$ so vorstellen, dass ausgehend von der Konfiguration (q, w) so lange (iterativ) die Übergangsfunktion δ angewendet wird, bis die gesamte Eingabe w abgearbeitet ist (auf das leere Wort λ geschrumpft ist). Die Aussage $\widehat{\delta}(q, w) = p$ sagt aus, dass diese iterative Anwendung der

Übergangsfunktion den endlichen Automaten schliesslich in den Endzustand p (Endkonfiguration (p, λ) führen wird.

Definition 3.5 (Dach-Operation)**:**
Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren die Dach-Operation $\widehat{\delta} : Q \times \Sigma^* \to Q$ rekursiv wie folgt:

(i) $\widehat{\delta}(q, \lambda) = q$ für alle $q \in Q$ und
Beim Lesen des leeren Wortes bleibt der endliche Automat in seinem Zustand.

(ii) $\widehat{\delta}(q, wa) = \delta(\widehat{\delta}(q, w), a)$
Der Ausdruck $\widehat{\delta}(q, w)$ entspricht einem Zustand. Wenn wir diesen mit p bezeichnen, also $p := \widehat{\delta}(q, w)$, dann sehen wir sofort, dass $\delta(\widehat{\delta}(q, w), a) = \delta(p, a)$ der normalen Anwendung der Übergangsfunktion δ (ohne Dach) beim Lesen des Symbols a im Zustand p entspricht.

Mithilfe dieser zwei neuen Operationen lässt sich die von einem endlichen Automaten M akzeptierte Sprache $L(M)$ (siehe Definition Gleichung (3.1)) alternativ ausdrücken als:

$$\begin{aligned} L(M) &= \{w \in \Sigma^* \; ; \; (q_0, w) \xrightarrow[*]{M} (p, \lambda) \text{ mit } q \in F\} \\ &= \{w \in \Sigma^* \; ; \; \widehat{\delta}(q_0, w) \in F\}. \end{aligned}$$

Beispiel 3.2:
Wir führen hier nochmals den endlichen Automaten aus Abbildung 3.4 auf:

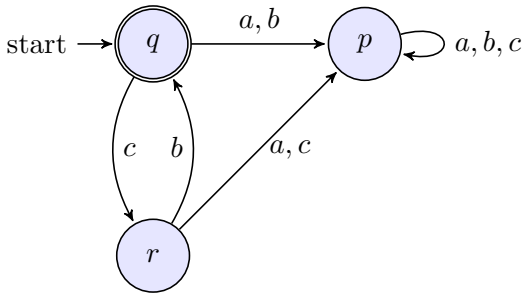


Abbildung 3.7: Endlicher Automat A.

Wie wir bereits festgestellt haben, ist der zu dieser Graphendarstellung äquivalente endliche Automat $A = (Q, \Sigma, \delta, q_0, F)$, mit:

$$\begin{aligned} &Q = \{q, p, r\}, \Sigma = \{a, b, c\}, q_0 = q, F = \{q\} \text{ und} \\ &\delta(q, a) = p, \quad \delta(q, b) = p, \quad \delta(q, c) = r \\ &\delta(p, a) = p, \quad \delta(p, b) = p, \quad \delta(p, c) = p \\ &\delta(r, a) = p, \quad \delta(r, b) = q, \quad \delta(r, c) = p \end{aligned}$$

Für diesen endlichen Automaten gilt zum Beispiel die Aussage

$$(q, cbcbcabac) \xrightarrow[*]{A} (r, abac),$$

da eine Berechnung von A auf dem Wort $cbcbcabac$ existiert, welche die Konfiguration $(q, cbcbcabac)$ zur Konfiguration $(r, abac)$ führt. Diese Berechnung lautet:

$$(q, cbcbcabac) \xrightarrow{A} (r, bcbcabac) \xrightarrow{A} (q, cbcabac) \xrightarrow{A} (r, bcabac) \xrightarrow{A} (q, cabac) \xrightarrow{A} (r, abac).$$

Für M gilt auch

$$\widehat{\delta}(r, bcac) = p.$$

Dies sehen wir ein, indem wir Punkt (ii) der Definition 3.5 wiederholt anwenden:

$$\widehat{\delta}(r, bcac) = \delta(\widehat{\delta}(r, bca), c) = \delta(\delta(\widehat{\delta}(r, bc), a), c) = \delta(\delta(\delta(\widehat{\delta}(r, bc), c), a), c) =$$
$$= \delta(\delta(\delta(\delta(\widehat{\delta}(r, \lambda), b), c), a), c)$$

Gemäss Punkt (i) der Definition 3.5 ist $\widehat{\delta}(r, \lambda) = r$ und somit verschwinden alle Dach-Operationen aus dem Ausdruck (wir haben den Basis-Fall der Rekusion erreicht). Die Übergangsfunktionen δ lassen sich schrittweise von innen nach aussen auflösen:

$$\widehat{\delta}(r, bcac) = \delta(\widehat{\delta}(r, bca), c) = \delta(\delta(\widehat{\delta}(r, bc), a), c) = \delta(\delta(\delta(\widehat{\delta}(r, bc), c), a), c) =$$
$$= \delta(\delta(\delta(\delta(\widehat{\delta}(r, \lambda), b), c), a), c) = \quad /\!\to \widehat{\delta}(r, \lambda) = r$$
$$= \delta(\delta(\delta(\delta(r, b), c), a), c) = \quad /\!\to \delta(r, b) = q$$
$$= \delta(\delta(\delta(q, c), a), c) = \quad /\!\to \delta(q, c) = r$$
$$= \delta(\delta(r, a), c) = \quad /\!\to \delta(r, a) = p$$
$$= \delta(p, c) = \quad /\!\to \delta(p, c) = p$$
$$= p$$

3.2 Beweise der Nichtexistenz

In Definition 3.3 haben wir die Klasse $\mathcal{L}_{EA}$ der regulären Sprachen kennengelernt. Eine Sprache L ist regulär ($L \in \mathcal{L}_{EA}$), genau dann, wenn ein endlicher Automat A existiert, welcher die Sprache L akzeptiert ($L(A) = L$).

Um zu beweisen, dass eine gegebene Sprache L regulär ist, genügt es also einen endlichen Automaten A zu konstruieren und zu begründen, dass L die von A akzeptierte Sprache ist. Dieses Vorgehen haben wir bereits mehrfach demonstriert.

Nun stellt sich natürlich die Frage, ob jede Sprache regulär ist, oder existieren Sprachen, die nicht regulär sind? Wie könnte man beweisen, dass eine Sprache nicht regulär ist?

Nur weil wir nicht in der Lage sind, einen endlichen Automaten, für eine Sprache zu finden, bedeutet dies nicht, dass die Sprache nicht regulär ist. Es könnte natürlich sein, dass die Sprache zwar regulär ist, wir jedoch nicht die passende Idee für die Konstruktion eines entsprechenden endlichen Automaten gefunden haben. Unsere Unzulänglichkeit einen endlichen Automaten konstruieren zu können, ist kein Beweis dafür, dass die Sprache nicht regulär ist. Um korrekt zu beweisen, dass eine Sprache nicht regulär ist, müssen wir begründen, warum es keinen endlichen Automaten für diese Sprache geben kann.

Im Unterschied zu konstruktiven Beweisen, bei denen man die Existenz eines Objekts mit gewissen Eigenschaften direkt durch eine Konstruktion eines solchen Objekts beweist (wir konstruieren zum Beispiel einen endlichen Automaten M mit vier Zuständen, der eine gegebene Sprache akzeptiert), kann man bei den Beweisen der Nichtexistenz mit einer unendlichen Menge von Kandidaten (zum Beispiel allen endlichen Automaten) nicht so vorgehen, dass man alle Kandidaten einen nach dem anderen betrachtet und überprüft, dass keiner die gewünschten Eigenschaften hat. Um die Nichtexistenz eines Objekts mit gegebenen Eigenschaften in einer unendlichen Klasse von Kandidaten zu beweisen, muss man für gewöhnlich eine tiefgreifende Kenntnis über diese Klasse haben, die im

Widerspruch zu den gewünschten Eigenschaften steht.

Beweise der Nichtexistenz gehören zu den schwierigsten Aufgaben der Informatik und der Mathematik überhaupt. Das P vs NP Problem, welches eines der bekanntesten offenen Probleme der Informatik ist, befasst sich ebenfalls mit der Problematik des Beweisens von Existenz oder Nichtexistenz[3].

Da endliche Automaten sehr stark eingeschränkte Programme sind (siehe Unterabschnitt 3.1.2), ist der Beweis der Nichtexistenz eines endlichen Automaten für eine gegebene Sprache eine relativ einfache Aufgabe. Wir nutzen diese Tatsache, um eine einfache Einführung in die Methodik der Erstellung von Beweisen der Nichtexistenz zu geben.

Das charakteristische Merkmal endlicher Automaten ist, dass sie zu jeder Zeit nur abgespeichert haben, in welchem Zustand sie sich aktuell befinden und welches Eingabesymbol sie als Nächstes lesen werden. Ansonsten verfügen sie über keinen Speicher. Dies bedeutet insbesondere, dass ein endlicher Automat nicht abgespeichert hat (sich nicht erinnern kann), auf welchem Wege er in seinen aktuellen Zustand gelangt ist. Betrachten wir nochmal den endlichen Automaten in Abbildung 3.7. Die zwei verschiedenen Wörter $x := cbcbcba$ und $y := b$ führen beide in den Zustand p. In Zukunft wird A nicht mehr zwischen x und y unterscheiden können. Auf welchem Wege der endliche Automat in einen Zustand gekommen ist, spielt also keine Rolle. Der endliche Automat hat kein „Gedächtnis“. Wenn A also nach dem Lesen zweier Wörter x und y jeweils in demselben Zustand q_0 endet (also $\widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y)$), dann gilt für alle Wörter $z \in \Sigma^*$ (alle zukünftigen Eingaben), dass

$$\widehat{\delta}(q_0, xz) = \widehat{\delta}(q_0, yz).$$

In diesem Sinne, können somit bereits gelesene Eingaben, welche in denselben Zustand führen, als gleichwertig angesehen werden.

Diese wichtige Eigenschaft wollen wir in einem Satz formulieren und formal beweisen.

Theorem 3.1 (Pfad-Invarianz endlicher Automaten)**:**
Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Seien $x, y \in \Sigma^*$ unterschiedliche Wörter ($x \neq y$) über Σ , mit der Eigenschaft:

$$(q_0, x) \xrightarrow[*]{A} (p, \lambda) \quad \text{und} \quad (q_0, y) \xrightarrow[*]{A} (p, \lambda)$$

für einen Zustand $p \in Q$ (dies ist gleichbedeutend mit $\widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y) = p$). Dann existiert für jedes Wort $z \in \Sigma^*$ ein Zustand $r \in Q$, sodass A seine Arbeit auf xz und yz jeweils im Zustand r beendet (A landet für xz und yz in demselben Zustand). Damit gilt insbesondere, dass

$$xz \in L(A) \Longleftrightarrow yz \in L(A)$$

Entweder liegen beide Wörter xz und yz in der von A akzeptierten Sprache $L(M)$, oder keines der beiden Wörter liegt in $L(A)$. Es kann nicht sein, dass eines der beiden Wörter in der Sprache liegt und das andere nicht.

[3] https://www.claymath.org/millennium-problems/p-vs-np-problem

Beweis 3.1:
Aus der Existenz der zwei Berechnungen

$$(q_0, x) \xrightarrow[*]{A} (p, \lambda) \quad \text{und} \quad (q_0, y) \xrightarrow[*]{A} (p, \lambda)$$

von A folgt sofort die Existenz folgender zwei Berechnungen auf xz und yz:

$$(q_0, xz) \xrightarrow[*]{A} (p, z) \quad \text{und} \quad (q_0, yz) \xrightarrow[*]{A} (p, z)$$

für alle Wörter $z \in \Sigma^*$. Wenn wir den Zustand, in den die Berechnung von A auf z ausgehende vom Zustand p führt, mit r bezeichnen (also $r := \widehat{\delta}(p, z)$), dann ist die Berechnung von A auf xz

$$(q_0, xz) \xrightarrow[*]{A} (p, z) \xrightarrow[*]{A} (r, \lambda)$$

und die Berechnung von A auf yz

$$(q_0, yz) \xrightarrow[*]{A} (p, z) \xrightarrow[*]{A} (r, \lambda).$$

Falls r ein akzeptierender Zustand ist ($r \in F$), dann sind beide Wörter xz und yz in $L(A)$. Falls $r \notin F$, dann sind beide Wörter xz und yz nicht in $L(A)$.

Wir zeigen jetzt, wie Theorem 3.1 benutzt werden kann um, indirekt (durch Widerspruch) zu beweisen, dass eine Sprache nicht regulär ist. Dieses indirekte Vorgehen wollen wir auch hier anwenden. Betrachten wir die Sprache

$$L_{count} := \{\ w \in \{0,1\}^*\ ;\ w = 0^n 1^n \text{ für ein } n \in \mathbb{N}\ \}.$$

Die Sprache L ist die Menge aller binären Wörter, welche gleich viele Nullen und Einsen enthalten und in denen alle Nullen vor dem ersten Auftreten einer Eins stehen. Das kürzeste Wort in L_{count} ist das leere Wort λ. Die weiteren Wörter in L_{count} sind $01, 0011, 000111, 00001111, \ldots$ (aufsteigende Längen). Wir werden beweisen, dass L_{count} keine reguläre Sprache ist, also $L_{count} \notin \mathcal{L}_{EA}$ gilt. Intuitiv scheint es so, dass jeder endliche Automat, welcher L_{count} akzeptieren würde, die Anzahl der Nullen abspeichern müsste, um diese Anzahl später mit der Anzahl Einsen abgleichen zu können. Ein endlicher Automat hat jedoch keinen Speicher, um die Anzahl der Nullen speicher zu können. Wir werden formal beweisen, dass diese Intuition genau richtig ist.

Theorem 3.2:
Die Sprache L_{count} ist nicht regulär ($L_{count} \notin \mathcal{L}_{EA}$).

Beweis 3.2:
Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an, L_{count} sei regulär. Dann existiert ein endlicher Automat $A = (Q, \{0,1\}, \delta, q_0, F)$ mit $L(A) = L_{count}$. Die Anzahl der Zustände Q von A ist $|Q|$. Wir betrachten die $|Q| + 1$ Wörter (ein Wort mehr als A Zustände hat):

$$0^1, 0^2, 0^3, \ldots, 0^{|Q|}, 0^{|Q|+1}. \tag{3.6}$$

Da in der Auflistung 3.6 mehr Wörter stehen, als A Zustände hat, müssen sich in der Liste (gemäss des Taubenschlagprinzips[a]) mindesten zwei unterschiedliche Wörter finden lassen, welche den Automaten in denselben Zustand führen. Damit existieren also zwei Zahlen

$i, j \in \{1, 2, \ldots, |Q|, |Q|+1\}$ mit $i < j$, sodass

$$\widehat{\delta}(q_0, 0^i) = \widehat{\delta}(q_0, 0^j).$$

Gemäss Theorem 3.1 muss für jedes Wort $z \in \{0,1\}^*$ gelten, dass

$$0^i z \in L_{count} \iff 0^j z \in L_{count}.$$

Doch dies ist ein Widerspruch, denn für die Wahl $z := 1^j$ liegt das Wort $0^j z = 0^j 1^j$ in der Sprache L_{count} ($0^j 1^j$ hat gleich viele Nullen wie Einsen), doch das Wort $0^i z = 0^i 1^j$ liegt nicht in L_{count}, da $i < j$ gilt und $0^i 1^j$ somit mehr Einsen als Nullen enthält. Da wir einen Widerspruch erhalten haben, muss unsere Annahme, L_{count} sei regulär verworfen werden und wir haben $L_{count} \notin \mathcal{L}_{EA}$ bewiesen.

[a] https://de.wikipedia.org/wiki/Schubfachprinzip

Aufgabe 3.6

Beweisen Sie mithilfe des Theorem 3.1, dass die Sprache

$$L := \{\, a^n b^m c^n \;;\; n, m \in \mathbb{N} \,\} \tag{3.7}$$

nicht regulär ist.

Aufgabe 3.7

Beweisen Sie mithilfe des Theorem 3.1, dass die Sprache

$$L := \left\{\, a^n b^m c^h \;;\; n, m, h \in \mathbb{N} \text{ und } m > n + 2h \,\right\} \tag{3.8}$$

nicht regulär ist.

Aufgabe 3.8

Beweisen Sie mithilfe des Theorem 3.1, dass die Sprache

$$L := \{\, w\#w \;;\; w \in \{a,b\}^* \,\}. \tag{3.9}$$

nicht regulär ist.

Aufgabe 3.9

Beweisen Sie, dass die Sprache

$$L := \left\{\, 0^i 1^j \;;\; i, j \in \mathbb{N} \,\right\} \tag{3.10}$$

regulär ist.

Aufgabe (Challenge) 3.10

Entwerfen Sie einen endlichen Automaten für die Sprache

$$L := \{ \text{Dec}(n) \; ; \; n \in \mathbb{N}, n > 0 \text{ und } n \text{ ist durch 11 teilbar} \}.$$

Dabei steht $\text{Dec}(n)$ für die Dezimaldarstellung von n ohne führende Nullen.

Tipp: Eine ganze Zahl x mit Dezimaldarstellung $\text{Dec}(x) = x_1 x_2 \dots x_m$ ist genau dann durch 11 teilbar, wenn die alternierende Quersumme $+x_1 - x_2 + x_3 - x_4 + x_5 - \dots \pm x_m$ dieser Zahl durch 11 teilbar ist. Beispiel: Die Zahl 4752 ist durch 11 teilbar, da ihre alternierende Quersumme $+4 - 7 + 5 - 2 = 0$ durch 11 teilbar ist.

3.3 Lösungen der Aufgaben

✔ Lösungsvorschlag zu Aufgabe 3.1 ✔

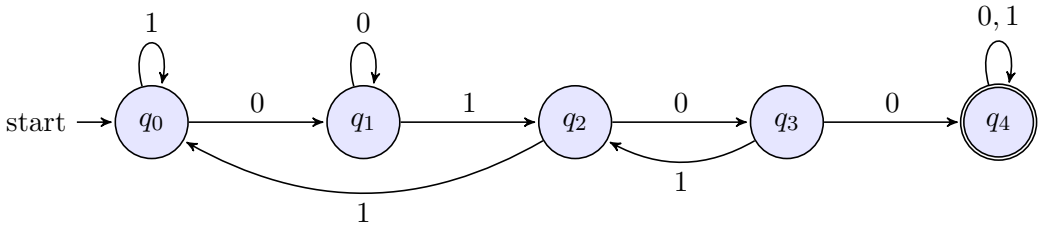


Abbildung 3.8: Endlicher Automat für die Sprache L_1 in Ausdruck Gleichung (3.4).

✔ Lösungsvorschlag zu Aufgabe 3.2 ✔

Der endliche Automat in Abbildung 3.3 akzeptiert genau die Eingabewörter, für die der endliche Automat seine Arbeit in einem der beiden akzeptierenden Zuständen p_0 **oder** p_3 beendet. Man kann den Graphen mental wie folgt aufteilen:

- obere Hälfte: Zustände p_0, p_1
- untere Hälfte: Zustände p_2, p_3
- linke Hälfte: Zustände p_0, p_2
- rechte Hälfte: Zustände p_1, p_3

In die *untere Hälfte* gelangt man nach dem erstmaligen Lesen eines Symbols a. Wenn ein weiteres a gelesen wird, gelangt man wieder zurück in die *obere Hälfte*. Man von der *oberen Hälfte* in die *untere Hälfte* (und umgekehrt) nur durch Lesen des Symbols a gelangen. Da der endliche Automat in der *oberen Hälfte* im Zustand p_0 beginnt, ist klar, dass sich der endliche Automat genau dann in einem der Zustände der *oberen Hälfte* (p_0, p_1) befindet, wenn er eine **gerade Anzahl** von a's gelesen hat.

Mit einer völlig analogen Argumentation begründet man, dass sich der endliche Automat genau dann in einem der Zustände der *linken Hälfte* (p_0, p_2) befinden kann, wenn er eine **gerade Anzahl** von b's gelesen hat.

Der endliche Automat befindet sich somit genau dann im Zustand p_0, wenn er eine gerade Anzahl von a's **und** eine gerade Anzahl b's gelesen hat, wenn also für das Eingabewort w gilt: $|w|_a$ ist gerade **und** $|w|_b$ ist gerade.

Der endliche Automat befindet sich somit genau dann im Zustand p_3, wenn er eine ungerade Anzahl von a's **und** eine ungerade Anzahl b's gelesen hat, wenn also für das Eingabewort w gilt: $|w|_a$ ist ungerade **und** $|w|_b$ ist ungerade.

Zusammengefasst, entspricht die vom endlichen Automaten akzeptierte Sprache L_2 der Menge

$$L_2 = \{\ w \in \{a,b\}^*\ ;\ \text{Das Wort } w \text{ hat eine gerade Länge } (|w| \text{ ist eine gerade Zahl}).\ \}$$

✔ Lösungsvorschlag zu Aufgabe 3.3 ✔

Intuitiv gesprochen, ist M_{ab} die Menge aller Wörter aus $\{a,b,c\}^*$ welche zwar auf b enden, aber nicht auf bb. Wir erhalten M_{ab} durch das (mengentheoretische) Subtrahieren der Menge L_{abb} von der Menge aller Wörter M_{axb}, die mit a beginnen und auf b enden:

$$M_{axb} := \{\ w \in \{a,b,c\}^*\ ;\ w = axb,\ \text{für ein } x \in \{a,b,c\}^*\ \}$$
$$M_{ab} = M_{axb} \setminus L_{abb}$$

Alternative Lösung:

Die Wörter w in M_{ab} haben die Form $w = axyb$, mit $x \in \{a,b,c\}^*$ und $y \in \{a,c\}$. Dies sind genau die Wörter, welche mit a beginnen, an letzter Stelle das Symbol b steht, aber an der zweitletzten Stelle nicht das Symbol b. Folglich gilt auch:

$$M_{ab} := \{\ w \in \{a,b,c\}^*\ ;\ w = axyb,\ \text{für ein } x \in \{a,b,c\}^*\ \text{ und ein } y \in \{a,c\}\ \}$$

✔ Lösungsvorschlag zu Aufgabe 3.4 ✔

```
if input = a goto 2, if input = b goto 2, if input = c goto 1;
if input = a goto 2, if input = b goto 0, if input = c goto 2;
if input = a goto 2, if input = b goto 2, if input = c goto 2;
```

✔ Lösungsvorschlag zu Aufgabe 3.5 ✔

Der zur Graphendarstellung in Abbildung 3.3 äquivalente endliche Automat ist $M = (Q, \Sigma, \delta, q_0, F)$, wobei:

$$\begin{aligned}
&Q = \{p_0, p_1, p_2, p_3\}, \Sigma = \{a,b\}, q_0 = p_0, F = \{p_0, p_3\}\ \text{ und}\\
&\delta(p_0, a) = p_2, \quad \delta(p_0, b) = p_1\\
&\delta(p_1, a) = p_3, \quad \delta(p_1, b) = p_0\\
&\delta(p_2, a) = p_0, \quad \delta(p_2, b) = p_3\\
&\delta(p_3, a) = p_1, \quad \delta(p_3, b) = p_2
\end{aligned}$$

✔ Lösungsvorschlag zu Aufgabe 3.6 ✔

Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an, $L = \{\ a^n b^m c^n\ ;\ n, m \in \mathbb{N}\ \}$ sei regulär. Dann existiert ein endlicher Automat $A = (Q, \{a,b,c\}, \delta, q_0, F)$ mit $L(A) = L$. Die Anzahl der Zustände Q von A ist $|Q|$. Wir betrachten die $|Q| + 1$ Wörter (ein Wort mehr als A Zustände hat):

$$a^1, a^2, a^3, \ldots, a^{|Q|}, a^{|Q|+1}. \tag{3.11}$$

Da in der Auflistung 3.11 mehr Wörter stehen, als A Zustände hat, müssen sich in der Liste (gemäss des Taubenschlagprinzips) mindesten zwei unterschiedliche Wörter finden lassen, welche den Automaten in denselben Zustand führen. Damit existieren also zwei Zahlen $i, j \in \{1, 2, \ldots, |Q|, |Q|+1\}$ mit $i < j$, sodass

$$\widehat{\delta}(q_0, a^i) = \widehat{\delta}(q_0, a^j).$$

Gemäss Theorem 3.1 muss für jedes Wort $z \in \{a,b,c\}^*$ gelten, dass

$$a^i z \in L \iff a^j z \in L.$$

Doch dies ist ein Widerspruch, denn für die Wahl $z := c^j$ liegt das Wort $a^j z = a^j c^j$ in der Sprache L ($a^j c^j$ hat gleich viele a's wie c's), doch das Wort $a^i z = a^i c^j$ liegt nicht in L, da $i < j$ gilt und $a^i c^j$ somit mehr c's als a's enthält. Da wir einen Widerspruch erhalten haben, muss unsere Annahme, L sei regulär verworfen werden und wir haben $L \notin \mathcal{L}_{EA}$ bewiesen.

✔ Lösungsvorschlag zu Aufgabe 3.7 ✔

Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an,

$$L = \left\{ a^n b^m c^h \ ; \ n, m, h \in \mathbb{N} \text{ und } m > n + 2h \right\}$$

sei regulär. Dann existiert ein endlicher Automat $A = (Q, \{a, b, c\}, \delta, q_0, F)$ mit $L(A) = L$. Die Anzahl der Zustände Q von A ist $|Q|$. Wir betrachten die $|Q| + 1$ Wörter (ein Wort mehr als A Zustände hat):

$$a^1, a^2, a^3, \ldots, a^{|Q|}, a^{|Q|+1}. \tag{3.12}$$

Da in der Auflistung 3.12 mehr Wörter stehen, als A Zustände hat, müssen sich in der Liste (gemäss des Taubenschlagprinzips) mindesten zwei unterschiedliche Wörter finden lassen, welche den Automaten in denselben Zustand führen. Damit existieren also zwei Zahlen $i, j \in \{1, 2, \ldots, |Q|, |Q| + 1\}$ mit $i < j$, sodass

$$\widehat{\delta}(q_0, a^i) = \widehat{\delta}(q_0, a^j).$$

Gemäss Theorem 3.1 muss für jedes Wort $z \in \{a, b, c\}^*$ gelten, dass

$$a^i z \in L \iff a^j z \in L.$$

Doch dies ist ein Widerspruch, denn für die Wahl $z := b^j$ liegt das Wort $a^i z = a^i b^j$ in der Sprache L $(j > i)$, doch das Wort $a^j z = a^j b^j$ liegt nicht in L. Da wir einen Widerspruch erhalten haben, muss unsere Annahme, L sei regulär verworfen werden und wir haben $L \notin \mathcal{L}_{EA}$ bewiesen.

✔ Lösungsvorschlag zu Aufgabe 3.8 ✔

Wir beweisen die Behauptung durch Widerspruch. Wir nehmen also an, $L = \{ \ w\#w \ ; \ w \in \{a, b\}^* \ \}$ sei regulär. Dann existiert ein endlicher Automat $A = (Q, \{a, b, \#\}, \delta, q_0, F)$ mit $L(A) = L$. Die Anzahl der Zustände Q von A ist $|Q|$. Wir betrachten die $|Q| + 1$ Wörter (ein Wort mehr als A Zustände hat):

$$a^1, a^2, a^3, \ldots, a^{|Q|}, a^{|Q|+1}. \tag{3.13}$$

Da in der Auflistung 3.13 mehr Wörter stehen, als A Zustände hat, müssen sich in der Liste (gemäss des Taubenschlagprinzips) mindesten zwei unterschiedliche Wörter finden lassen, welche den Automaten in denselben Zustand führen. Damit existieren also zwei Zahlen $i, j \in \{1, 2, \ldots, |Q|, |Q| + 1\}$ mit $i < j$, sodass

$$\widehat{\delta}(q_0, a^i) = \widehat{\delta}(q_0, a^j).$$

Gemäss Theorem 3.1 muss für jedes Wort $z \in \{a, b, \#\}^*$ gelten, dass

$$a^i z \in L \iff a^j z \in L.$$

Doch dies ist ein Widerspruch, denn für die Wahl $z := \#a^i$ liegt das Wort $a^i z = a^i \# a^i$ in der Sprache L $(j > i)$, doch das Wort $a^j z = a^j \# a^i$ liegt nicht in L. Da wir einen Widerspruch erhalten haben, muss unsere Annahme, L sei regulär verworfen werden und wir haben $L \notin \mathcal{L}_{EA}$ bewiesen.

✔ Lösungsvorschlag zu Aufgabe 3.9 ✔

Wir zeigen, dass die gegebene Sprache regulär ist, indem wir einen endlichen Automaten für diese Sprache angeben. Der endliche Automat ist gegeben durch:

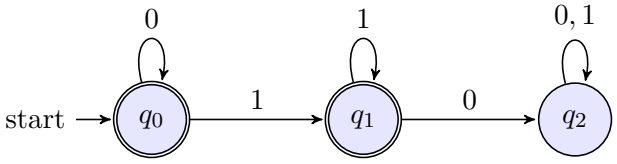


Abbildung 3.9: Endlicher Automat für die Sprache $L := \{ \ 0^i 1^j \ ; \ i, j \in \mathbb{N} \ \}$.

✔ Lösungsvorschlag zu Challenge 3.10 ✔

Wir geben einen endlichen Automaten $M = (Q, \Sigma, \delta, q0, F)$ mit $L(M) = L$ an.

- $Q = (q_0', q_t, q_0, q_1, \ldots, q_{10}, p_0, p_1, \ldots, p_{10})$ (dies sind 24 Zustände)
- $\Sigma = \Sigma_{10} = \{0, 1, \ldots, 9\}$
- Der Anfangszustand $q0$ ist gegeben durch q_0'.
- $F = \{q_0, p_0\}$
- Die Übergangsfunktion δ ist vollständig beschrieben durch:

$$\begin{aligned}
\delta(q_0', 0) &= q_t, \\
\delta(q_t, j) &= q_t, \quad j \in \{0, 1, \ldots, 9\}, \\
\delta(q_0', j) &= q_j, \quad j \in \{1, 2, \ldots, 9\}, \\
\delta(q_i, j) &= p_{(i-j) \bmod 11}, \quad i, j \in \{0, 1, \ldots, 9\}, \\
\delta(p_i, j) &= q_{(i+j) \bmod 11}, \quad i, j \in \{0, 1, \ldots, 9\}.
\end{aligned}$$

Kapitel 4

Turingmaschinen

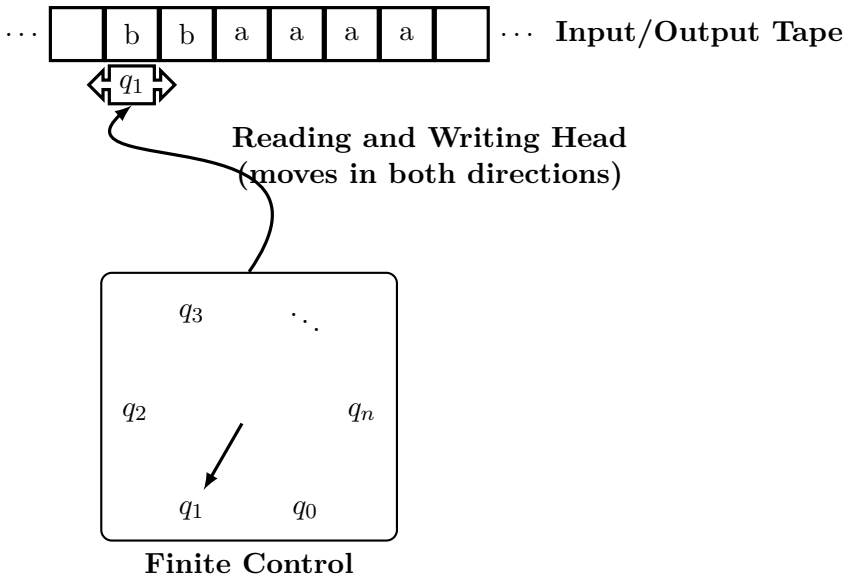


Abbildung 4.1: Turing-Maschine