Theoretische Informatik – Schreibomat

Florian

September 18, 2012

Tools

- Finite State Machine Designer: `http://madebyevan.com/fsm/`

1 Was ist Informatik?

- Konkrete[1] Konzepte: Algorithmus, Berechnung, Komplexität
- Was will die theoretische Informatik? Formale Bestimmung der Begriffe, unabhängig von Hard- und Software
- Ziel: Grundlegende Eigenschaften von Algorithmen, Berechnungen erkennen. Methoden entwickeln zum Entwurf beweisbar korrekter Hard- und Software (nicht Schwerpunkt dieser Vorlesung)
- Grenzen der automatischen Berechnung aufzeigen
- Typische Fragestellungen: Ist es möglich, ein Programm zu schreiben, das ein anderes Programm als Eingabe erhält und feststellen kann, ob dieses in eine Endlosschleife gerät oder nicht? ⇒ Halteproblem. Nein, es ist natürlich nicht möglich. Eine andere typische Fragestellung ist die Folgende: Wir haben einen Rucksack, und der hat 30 Liter Fassungsvermögen. Und wir haben noch 50 Gegenstände, und jetzt wollen wir wissen: wie lange dauert es, herauszufinden, wieviele Gegenstände in den Rucksack passen? Rucksackproblem, nicht in polynomialer Zeit lösbar ⇒ Es dauert expontiell lange.
- Leider ist es in der Praxis so, dass die Antwort auf ähnliche Fragestellungen nicht immer so offensichtlich ist, und deshalt müssen wir es uns ein bisschen genauer betrachten.

Übung. Gegeben sei das folgende Strassennetz:
(Wildes Gekritzel)

[1] 4. September 2012

(a) Gibt es eine Möglichkeit, alle Strassen genau 1x zu durchlaufen und dann wieder am Ausgangspunkt anzulangen,

(b) Gibt es eine Möglichkeit, alle Kreuzungen genau 1x zu passieren und dann wieder am Ausgangspunkt anzulangen?

Die Antworten sind imfall:

(a) Einfache Aufgabe: Geht immer, wenn die Anzahl der Abzweigungen an jeder Kreuzung gerade ist.

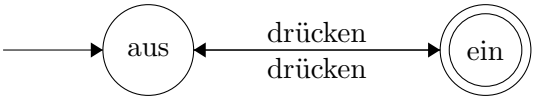
(b) Traveling salesman (TSP). Schwierig! Es ist keine wesentlich bessere Methode bekannt, als einfach alles durchzuprobieren.

Ziel für die Vorlesung: Um solche Fragestellungen systematisch beantworten zu können, brauchen wir ein exaktes mathematisches Modell eines Computers.

2 Automatentheorie

Endliche Automaten, Kontextfreie Grammatiken und Keller-Automaten. Zusätzliche Motivation: Das sind Konzepte, denen man auch häufig in der Praxis begegnet (Compilerbau, Textsuche, Textverarbeitung)

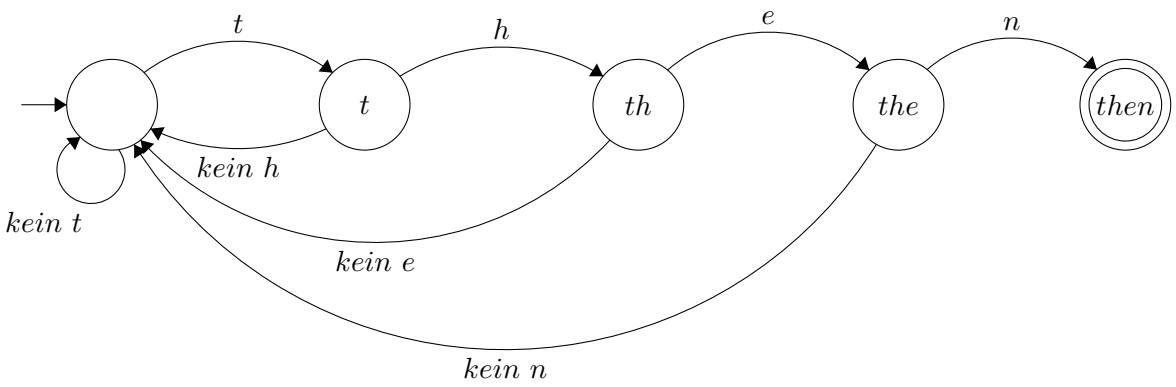
Noch nicht ganz so formal, bisschen intuitiv. Modellierung eines Kippschalters.



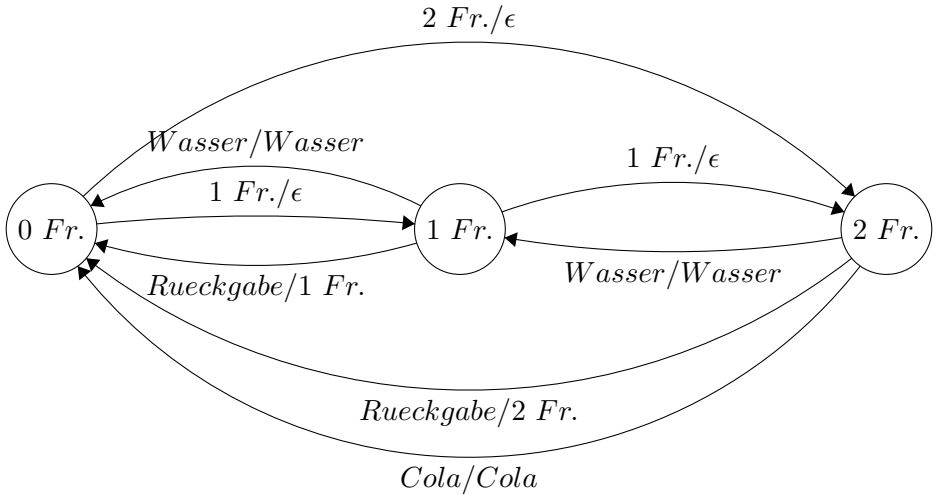
Definition 1 (Endlicher Automat)**.** Wir haben:

- Zwei Zustände, davon ein Startzustand
- und ein akzeptierender Zustand
- Transitionen (Zustandsübergänge): führen den Automaten anhand einer Eingabe von einem Zusand in den nächsten.

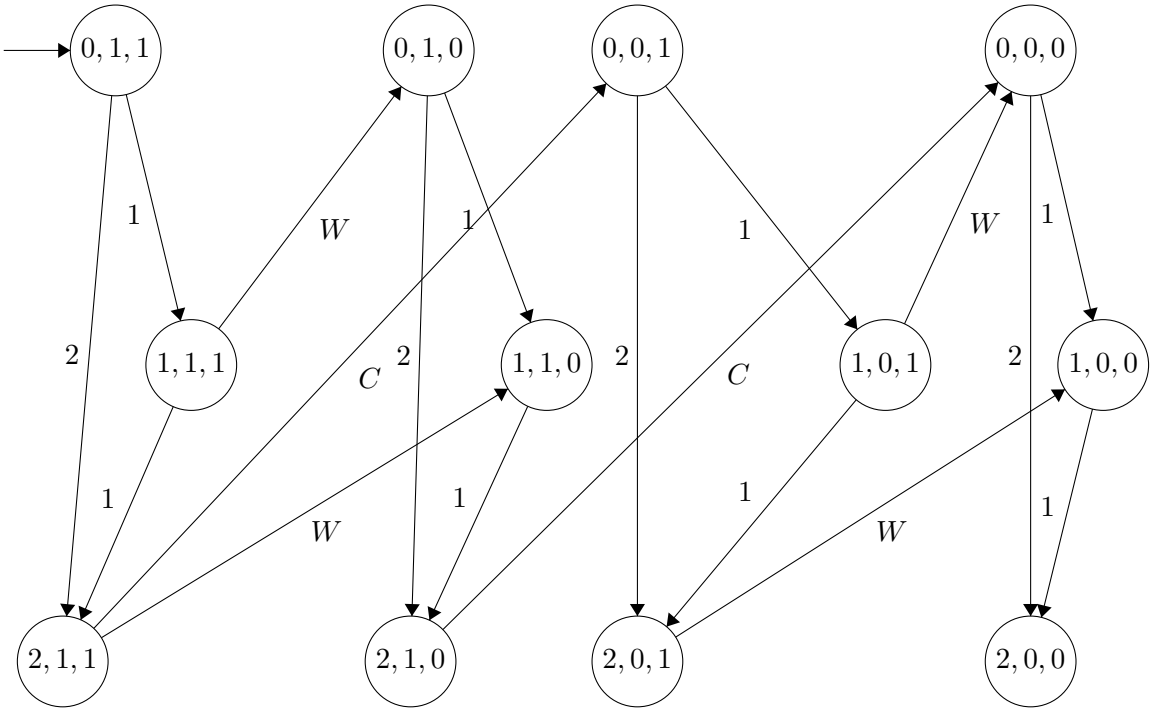
Beispiel: Mustererkennung in Texten: z.B. Suche das Wort "then"



Getränkeautomat (Beispiel für Automat mit Ausgabe): Cola für CHF 2, Wasser für CHF 1. Münzannahme: Münzen zu 1, 2 CHF.



Wenn der Automat nur eine Flasche Cola und eine Flasche Wasser enthält:



3 Formale Sprachen

Ziel: Genaue Beschreibung der Ein- und Ausgaben eines Automaten.

Definition 2 (Alphabet)**.** (endliche, nichtleere Menge von Symbolen). Bsp: Binäres Alphabet $\Sigma_{\text{bin}} = \{0, 1\}$, Tastaturalphabet $\Sigma_{\text{tast}} = \{a, \ldots, z, A, \ldots, Z, 0, \ldots 9, \ldots\}$

Definition 3 (Wort (= Zeichenkette, String))**.** Endliche Folge von Symbolen eines gegebenen Alphabets. Beispiel: 01110 ist ein Wort über Σ_{bin}.

Bemerkung (Eigenschaften von Wörtern)**.** Die da wären:

- Leeres Wort: ϵ (manchmal λ) = leere Folge von Symbolen (über einem beliebigen Alphabet)
- Länge eines Wortes: $|w|$ bezeichnet die Anzahl der Symbole im Wort w. $|\epsilon| = 0$.

Bemerkung (Konventionen für Darstellung von Wörtern)**.** Wir sagen:

- $a, b, c, \ldots$ für Buchstaben, Symbole
- $v, w, x, \ldots$ für Wörter

Definition 4 (Potenzen von Wörtern)**.** Es gibt im Angebot:

- Menge aller Wörter einer bestimmten Länge:

Sei Σ Alphabet, so:

$$\begin{aligned} \Sigma^0 &= \{\epsilon\} \\ \Sigma^1 &= \Sigma \\ \Sigma^2 &= \{ab | a, b \in \Sigma\} \\ \Sigma^i &= \{a_1 \dots a_i | a_1 \dots a_i \in \Sigma\} \end{aligned}$$

- Menge aller Wörter über Σ:

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i, \epsilon \in \Sigma^*$$

- Menge aller nichtleeren Wörter über Σ:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i, \epsilon \notin \Sigma^+$$

Beispiel.

Beispiel:

$$\begin{aligned} \Sigma &= \{0, 1\} \\ \Sigma^2 &= \{00, 01, 10, 11\} \\ \Sigma^* &= \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\} \end{aligned}$$

Definition 5 (Konkatenation (Verkettung) von Wörtern)**.**

$$v = a_1, \dots a_k, w = b_1 \dots, b_k \text{ über } \Sigma \Rightarrow v \cdot w = a_1 \dots a_k b_1 \dots b_k = vw$$

Bemerkung (Rechenregeln für Konkatenation von Wörtern)**.**

$$\begin{aligned} v \cdot (v \cdot w) &= (u \cdot v) \cdot w \\ |x \cdot y| &= |x| + |y| \\ x \cdot \epsilon &= \epsilon \cdot x = x \end{aligned}$$

Bemerkung (Infixe, Präfixe, Suffixe)**.** Seien $v, w \in \Sigma^*$ für ein Alphabet Σ, dann ist v ein Teilwort (Infix) von w, falls es $x, y \in \Sigma^*$ gibt, so dass $w = x \cdot v \cdot y$; v ist ein Präfix von w, falls es $y \in \Sigma^*$ gibt, so dass $w = v \cdot y$. Suffix funktionert analog.

Beispiel: abc ist Teilwort von aabcc, aa ist Präfix und Suffix von aabcaa. ϵ ist Teilwort von jedem Wort.

Definition 6 (Sprache)**.** L über Alphabet Σ ist eine Teilmenge von Σ^*, $L \subseteq \Sigma^*$. Sprache ist Menge von Wörtern, kann unendlich gross sein. Leere Sprache: $\emptyset$ enthält keine Wörter, ist über jedem Alphabet definiert. Spezielle Sprache: L_ϵ ist die Sprache, die nur das leeere Wort enthält. $L_\epsilon \neq \emptyset$.

Definition 7 (Konkatenation von Sprachen)**.** $L_1, L_2 \subseteq \Sigma^* \Rightarrow L_1 \cdot L_2 = L_1 L_2 = \{v \cdot w | v \in L_1, w \in L_2\}$

Definition 8 (Potenzen von Sprachen)**.**

$$\begin{aligned} L^0 &= L_\epsilon = \{\epsilon\} \\ L^{i+1} &= L^i \cdot L \text{ für } i \in \mathbb{N} \end{aligned}$$

Definition 9 (Kleene-Stern)**.**

$$\begin{aligned} L^* &= \bigcup_{i \in \mathbb{N}} L^i \\ L^+ &= \bigcup_{i \in \mathbb{N}} L^i - L_\epsilon \end{aligned}$$

Beispiel. Sei $\Sigma = \{a, b\}, L_1 = \{a^i | i \geq 0\} = \{\epsilon, a, aa, aaa, \dots\}, L_2 = \{b^i | i \geq 0\} = \{\epsilon, b, bb, bbb, \dots\}$:

$L_1 \cdot L_2 = \{\epsilon, a, b, ab, aab, abb, aaab, \dots\} = \{a^i b^j | i \geq 0, j \geq 0\}$ ist die Menge aller Wörter über $\{a, b\}$, in dem alle as vor allen bs vorkommen.

Übung. Seien $L_1, L_2, L_3 \subseteq \Sigma^*$.

(a) Gilt $(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3)$? Ja.

(b) Gilt $(L_1 \cup L_2) \cdot L_3 = L_1 \cdot L_3 \cup L_2 \cdot L_3$? Ja.

(c) Gilt $(L_1 \cdot L_2) \cup L_3 = (L_1 \cup L_3) \cdot (L_2 \cup L_3)$? Quatsch.

(d) Gilt $L_1 \cdot L_2 = L_2 \cdot L_1$? Quatsch.

Beweis. (a)

$$\begin{aligned} (L_1 \cdot L_2) \cdot L_3 &= \{vw | v \in L_1 \cdot L_2, w \in L_3\} \\ &= \{xyw | x \in L_1, y \in L_2, w \in L_3\} \\ &= \{xz | x \in L_1, z \in L_2 L_3\} \\ &= L_1 \cdot (L_2 \cdot L_3) \end{aligned}$$

(b)

$$\begin{aligned} (L_1 \cup L_2) \cdot L_3 &= \{vw | v \in L_1 \cup L_2, w \in L_3\} \\ &= \{vw | v \in L_1, w \in L_3\} \cup \{vw | v \in L_2, w \in L_3\} \\ &= (L_1 \cdot L_3) \cup (L_2 \cdot L_3) \end{aligned}$$

(c) Gegenbeispiel: $L_1 = L_2 = \{a\}, L_3 = \{b\}$. Daraus folgt:

$$\begin{aligned} L_1 L_2 = \{aa\} &\Rightarrow (L_1 L_2) \cup L_3 = \{aa, b\} \\ L_1 \cup L_3 = \{a, b\} = L_2 \cup L_3 &= \Rightarrow \dots \end{aligned}$$

□

Bemerkung. (d) gilt aber für $|\Sigma| = 1$.

Definition 10 (Entscheidungsproblem)**.** Die Eingabe ist eine Sprache L über einem Alphabet Σ sowie ein Wort $w \in \Sigma^*$. Die Ausgabe soll lauten: JA falls $w \in L$ oder NEIN falls $w \notin L$.

Die Modellierung von vielen alltäglichen Berechnungsproblemen im Formalismus der formalen Sprachen.

Beispiel (Primzahltest)**.** Das Alphabet $\Sigma = \{0, 1\}$. Die Sprache

$$L = \{w \in \Sigma^* | w \text{ ist Binärdarstellung einer Primzahl}\}$$

Eine Zahl $p \in N$ ist Primzahl genau dann, wenn $Bin(p) \in L$.
Identifizierung von Sprachen als Probleme.

4 Kurze Einführung in formale Beweise

Behauptungen[2] mit Unanfechtbarkeitsanspruch wollen bewiesen werden. Insbesondere negative Aussagen der Form "Dieses Rechenmodell kann diese Aufgabe nicht lösen" brauchen eine präzise Formulierung und Begründung, um glaubwürdig zu sein[3]. Unkontrollierte Hausaufgabe

Als Beweismethoden haben wir im Angebot:

Deduktion (zum Beweis einer Implikation) Zum Beweis von Wenn-Dann-Aussagen, z.B.

$$\text{Wenn } \underbrace{x \geq 4}_{\text{Hypothese}} \text{ , dann } \underbrace{x^2 \geq 16}_{\text{Konklusion}}$$

Vorgehen: Hypothese als wahr annehmen, dann Folgerungen darauf anwenden, bis die Konklusion folgt.

Beispiel. Sei $x \geq 4$. Es gilt $4^2 = 16$ und die Quadratfunktion ist steigend. Daraus folgt: $x^2 \geq 4^2 = 16$

Hierfür ist es oft hilfreich, Definitionen von Begriffen einzusetzen.

Beispiel. Wenn $L = \{w \in \{a, b\}^* | |w| \text{ gerade}\}$, dann gilt für alle $x \in L^2$, dass $|x|$ gerade ist.

Beweis. Sei $x \in L^2$.

Definition von L^2 einsetzen. Es existieren $u, v \in L$, so dass $x = uv$.

Definition von L einsetzen. $|u|$ ist gerade, $|v|$ ist gerade.

$\Rightarrow |x| = |u| + |v| \Rightarrow |x|$ ist gerade. □

[2] 11. September 2012
[3] Hopcraft et al., Abschnitt 1.2–1.4

Doppelte Deduktion (zum Beweis einer Äquivalenz) Zerlegung in zwei Wenn-Dann-Aussagen, wird getrennt bewiesen.

Beispiel (Gleichheit von Mengen)**.**

$$A = B \Rightarrow A \subseteq B \wedge B \subseteq A \Rightarrow A = B$$

Beispiel. Sprachen sind ja auch Mengen. Also: seien $L_1, L_2 \in \Sigma^*$. Dann gilt:

$$L_1 \cdot (L_1 \cup L_2) = L_1^2 \cup L_1 \cdot L_2$$

Beweis. Zweimal:

"$\subseteq$":
Sei $x \in L_1 \cdot (L_1 \cup L_2) \Rightarrow \exists x = uv, u \in L_1, v \in L_1 \cup L_2$.
Fallunterscheidung: (a) $v \in L_1 \Rightarrow x = uv, u \in L_1, v \in L_1 \Rightarrow x \in L_1^2$.
(b) $v \in L_2 \Rightarrow x = uv, u \in L_1, v \in L_2 \Rightarrow x \in L_1 \cdot L_2$.

- "$\supseteq$":
Sei $x \in L_1^2 \cup L_1 \cdot L_2$.
Fallunterscheidung: (a) $x \in L_1^2 \Rightarrow x = uv, u, v \in L_1$
$\Rightarrow v \in L_1 \cup L_2$
$\Rightarrow x \in L_1 \cdot (L_1 \cup L_2)$.
(b) $x \in L_1 \cdot L_2 \Rightarrow x = uv, u \in L_1, v \in L_2$
$\Rightarrow v \in (L_1 \cup L_2)$
$\Rightarrow x \in L_1 \cdot (L_1 \cup L_2)$

□

Widerspruchsbeweis Es wird eine Annahme getroffen und dann solange gefolgert, bis Quatsch herauskommt. Daraus folgt, dass die Annahme auch falsch sein muss.

Beispiel. Seien $a, b \in \mathbb{N}$ und $a, b \geq 2$. Dann gilt: a teilt nicht $ab + 1$.

Beweis. Seien $a, b \in \mathbb{N}$. Angenommen, a teilt $ab + 1$.

Ziel: Annahme zum Widerspruch führen, dann folgt daraus Negation, also die gewünschte Konklusion.

a teilt $ab \Rightarrow a$ teilt ab und $ab + 1$.

$\Rightarrow a$ teilt $(ab + 1) - (ab) = 1$

$\Rightarrow a = 1$. Bonk! Widerspruch zu $a \geq 2$. □

Beispiel. Es gibt unendlich viele Primzahl (äquivalente Formulierung: Es gibt keine grösste Primzahl).

Beweis. Angenommen, es gäbe eine grösste Primzahl. Wir nennen die Anzahl der Primzahlen k.

Seien $p_1 < p_2 < \cdots < p_k$ die Primzahlen. Betrachten wir $x = p_1 \cdot p_2 \cdot \cdots \cdot p_k + 1$. Da p_k die grösste Primzahl ist und $x > p_k$, kann x keine Primzahl sein.

Also gibt es eine Primfaktorzerlegung von x, und p_l sei die kleinste Primzahl in dieser Zerlegung.

Dann gilt: p_l teilt $p_1 \cdot p_2 \cdot dots \cdot p_k$ und p_l teilt darum auch x.

Daraus folgt: p_l teilt $p_1 \cdot p_2 \cdot dots \cdot p_k + 1$. Aus dem vorhergehenden Beweis (a teilt $ab + 1$) folgt, dass das ein Widerspruch ist. □

Induktion Vor allem verwendet für Beweise über natürliche Zahlen. Ziel: Zeige, dass die Aussage $S(n)$ für alle $n \geq n_0$ gilt.

Methode:

1. Induktionsanfang (Anker). Zeige $S(n_0)$.
2. Induktionsschritt. Zeige $S(i) \Rightarrow S(i+1)$.

Idee: Wenn $S(n_0)$ gilt und $S(i) \Rightarrow S(i+1)$ für alle $i \geq n_0$, dann folgt aus $S(n_0)$ und $S(n_0) \Rightarrow S(n_0+1)$, dann gilt $S(n_0+1)$. Undsoweiter, undsofort.

Beispiel (Der kleine Gauss)**.** . Für alle $n \geq 1$ gilt: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Beweis. Anker: $\sum_{i=1}^{1} = \frac{1+(1+1)}{2}$

Schritt: es gelte $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ (Induktionsvoraussetzung).

Zeige: $\sum_{i=1}^{k+1} i = \frac{(k+1)(k+1+1)}{2}$ (Induktionsbehauptung).

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) \\
&= \frac{k(k+1)}{2} + (k+1) \\
&= \frac{k(k+1) + 2(k+1)}{2} \\
&= \frac{(k+2)(k+1)}{2} \\
&= \frac{(k+1+1)(k+1)}{2}
\end{aligned}$$

□

Rekursive Definitionen und strukturelle Induktion

Beispiel (Rekursive Definition von arithmetischen Ausdrücken)**.** Basis: Jede Zahl und jede Variable ist ein arithmetischer Ausdruck: $(a+3)\cdot 5$ ist ein arithmetischer Ausdruck, $(a+3\cdot 5$ [sic] ist keiner (Klammerfehler).

Rekursion: Wenn E und F Ausdrücke sind, dann definieren wir, dass dann auch $E+F$, $E\cdot F$, (E) arithmetische Ausdrücke sind.

Behauptung: Jeder Ausdruck enthält gleichviele linke wie rechte Klammern.

Beweis. Mit struktureller Induktion:

1. I.A.: Basisausdrücke haben keine Klammern. Daraus folgt: gleichviele linke wie rechte Klammern (0).
2. Sei G ein arithmetischer Ausdruck ($G = E+F, E\cdot F, (G)$). Fallunterscheidung:
   (a) $G = E + F$: Induktionsvoraussetzung: E und F haben gleich viele linke wie rechte Klammern. Gilt also auch für G, da keine Klammern hinzukommen.
   (b) $G = E \cdot F$: analog.
   (c) (G): Induktionsvoraussetzung: E hat gleich viele linke wie rechte Klammern. Gilt also auch für G, da 1 linke und 1 rechte Klammer hinzukommt.

□

5 Unendlichkeit

Beweis für 3.7. Behauptung: $|A| < |B| \Leftarrow |A| = |C| \wedge C \subset B$
$A = B = \mathbb{N} = \{0, 1, 2, 3, \dots\}$ $C = \{1, 2, 3, 4, \dots\}$ $(0,1), (1,2), \dots, (k, k+1)$ □

Aufgaben: 3.7, 3.8, 3.11, 3.17, 3.18, 3.19

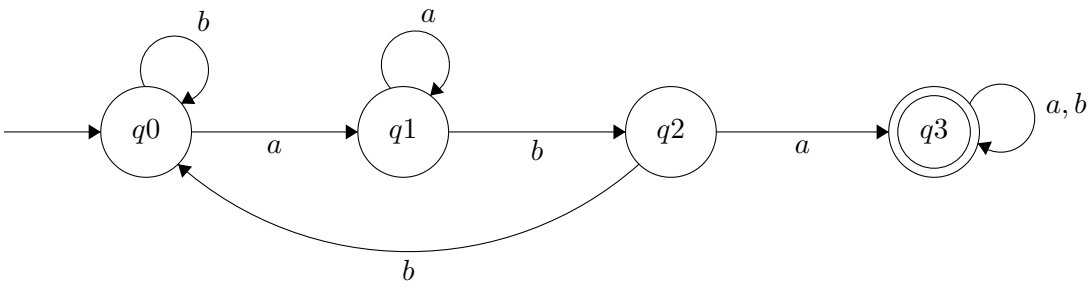
6 Endliche Automaten

Unterscheidung:

Deterministisch Automat kann zu einem Zeitpunkt nur einen Zustand annehmen.

Nichtdeterministisch Automat kann gleichzeitig mehrere Zustände besitzen.

Beispiel. DEA zum Erkennen des Musters aba:



Zustände speichern das bereits gelesene Teilmuster:

- q_0: noch nichts vom Teilwort aba gelesen.
- q_1: zumindest schon das Wort a gelesen.
- q_2: schon ab gelesen.
- q_3: das ganze Muster gelesen, es kann also akzeptiert werden.

Transitionen (Zustandsübergänge£) beschreiben die Änderung beim Lesen eines Zeichens des Wortes.

Definition 11 (Formale Definition (nochmal))**.** Ein DEA A wird beschrieben durch $A = (Q, \Sigma, \delta, q_0, F)$, wobei:

- Q ist eine endliche Menge von Zuständen,
- Σ ist das Alphabet für die Eingabe,
- q_0 ist der Startzustand, $q_0 \in Q$,
- F ist die Menge der akzeptierenden Zustände (Endzustände), $F \subseteq Q$,
- $\delta : Q \times \Sigma \mapsto Q$ ist die Übergangsfunktion (Transitionsfunktion), die für jeden Zustand und jedes Eingabesymbol einen eindeutigen Folgezustand definiert.

Beispiel (von vorher)**.**

$$A = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\}),$$

wobei δ gegeben ist durch:

$$\begin{aligned}
\delta(q_0, a) &= q_1,\\
\delta(q_0, b) &= q_0,\\
\delta(q_1, a) &= q_1,\\
\delta(q_1, b) &= q_2,\\
\delta(q_2, a) &= q_3,\\
\delta(q_2, b) &= q_0,\\
\delta(q_3, a) &= q_3,\\
\delta(q_3, b) &= q_3
\end{aligned}$$

(oder als Transitionstabelle aufgeschrieben):

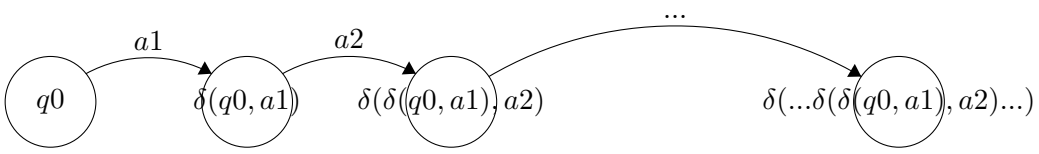
	a	b
$\mapsto q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

Die Transitionstabelle enthält alle Informationen über einen DEA, meistens ist aber die graphische Darstellung übersichtlicher.

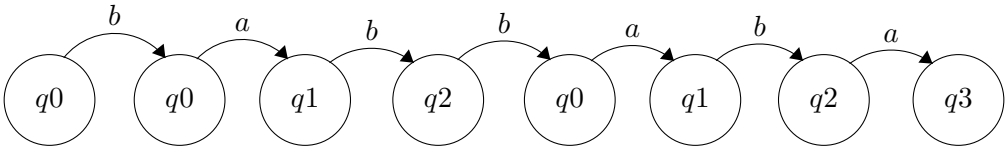
Ziel: Automaten zur Beschreibung von Sprachen verwenden.

Definition 12 (Berechnung)**.** Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA, $w \in \Sigma^*$ ein Wort.

Die Berechnung von A auf w ist die Folge von Zuständen, die A beim Lesen von w durchläuft, also, gilt für $w = a_1 a_2 \dots a_k$:



Beispiel. Berechnung von A auf $w = babbaba$:



Sinnvollerweise: Erweiterung von δ af Wörter (erweiterte Übergangsfunktion).

Definition 13 (Erweiterte Übergangsfunktion)**.** $\hat{\delta} : Q \times \Sigma^* \mapsto Q$ ist rekursiv definiert wie folgt:

- $\hat{\delta}(q, a) = \delta(q, a)$ für alle $q \in Q, a \in \Sigma$

- $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ für alle $q \in Q, x \in \Sigma^*, a \in \Sigma$.

Bemerkung. Anschaulich: $\hat{\delta}(q, w)$ betstimmt, in welchem Zustand A endet, wenn er vom Zustand q aus das Wort w liest.

Beispiel. $\hat{\delta}(q_0, babbaba)$

Definition 14. Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DEA. Die von A akzeptierte Sprache $L(A) \subseteq \Sigma^*$ ist die Menge aller $w \in \Sigma^*$, so dass $\hat{\delta}(q_0, w) \in F$.

Bemerkung. Anschaulich: Die Menge aller Wörter, die den DEA vom Anfangszustand aus in einen Endzustand führen.

Beispiel. $\hat{\delta}(q_0, babbaba) = q_3 \in F \Rightarrow w \in L(A)$ (w wird von A akzeptiert.)

Definition 15 (Reguläre Sprachen)**.** $\mathcal{L}(\text{DEA}) = \{L(A) | A \text{ ist ein DEA}\}$ ist die Klasse der Sprachen, die von DEAs akzeptiert werden. Man nennt sie die Klasse der regulären Sprachen und $L \in \mathcal{L}(\text{DEA})$ wird regulär genannt.

Beispiel. DEA A entwerfen, der die Sprache

$L = \{w \in \{0,1\}^* | w \text{ enthält eine gerade Anzahl Nullen und eine gerade Anzahl Einsen}\}$

akzeptiert.

Idee: 4 Zustände, die speichern, wieviele Nullen und Einseln modulo 2 bereits gelesen wurden.

- q_{00}: gerade Anzahl Nullen, gerade Anzahl Einsen
- q_{01}: gerade Anzahl Nullen, ungerade Anzahl Einsen
- q_{10}: ungerade Anzahl Nullen, gerade Anzahl Einsen
- q_{11}: ungerade Anzahl Nullen, ungerade Anzahl Einsen

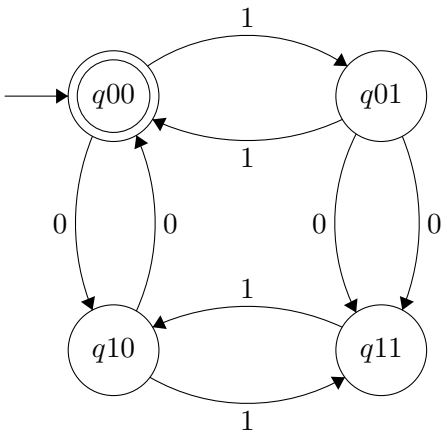
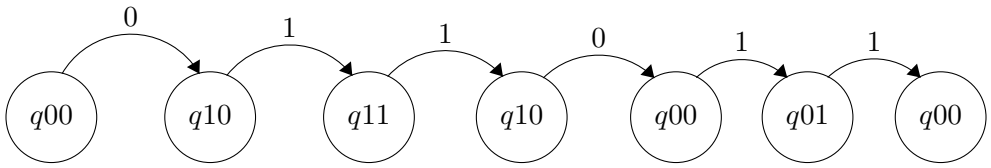


Tabelle:

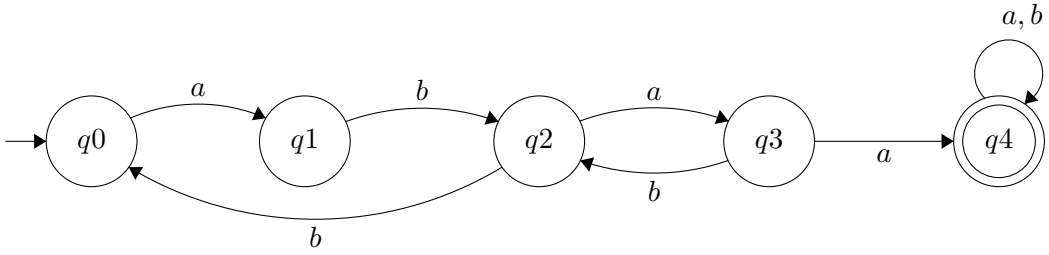
	0	1
$\mapsto *q_{00}$	q_{10}	q_{01}
q_{01}	q_{11}	q_{00}
q_{10}	q_{00}	q_{11}
q_{11}	q_{01}	q_{10}

Berechnung von A auf $w = 011011$:



$\Rightarrow w \in L(A), \hat{\delta}(q_{00}, 011011) = q_0$

Lösung der Aufgaben
1a:



1b: