

Kapitel 2

Endliche Automaten

In diesem Kapitel stellen wir besonders einfache abstrakte Rechnermodelle vor, die so genannten *endlichen Automaten*. „Endlichkeit“ bezieht sich dabei auf die Gedächtnisleistung (Speicherkapazität) der Automaten, die im Allgemeinen auf nur endlich viele unterscheidbare Zustände beschränkt ist. Wir werden an diesem Modell eine Reihe typischer Fragestellungen studieren, die von spezifischem, aber auch von allgemeinem Interesse sind; sie lassen sich fast alle in dieser oder ähnlicher Form auch für andere Automatentypen untersuchen. Dies betrifft z.B. Determinismus, Varianten von Automaten oder Redundanzen und deren Beseitigung. Des Weiteren werden wir diese Automaten als Erkennungsmechanismen für eine bestimmte Klasse von Sprachen, nämlich die Klasse der *regulären Sprachen*, kennen lernen.

Neben dieser eher technischen Sicht dessen, was endliche Automaten formal leisten, betrachten wir auch deren *Anwendungen*, z.B. in der Softwaretechnik. Es ist in der Tat so, dass sich viele Realweltgegebenheiten, ja sogar Rechner selbst, mit endlichen Automaten beschreiben lassen, so dass dieses Modell häufig für formale Betrachtungen realer Gegebenheiten bereits völlig ausreicht.

2.1 Deterministische endliche Automaten

Einen *endlichen Automaten* können wir uns als eine „Black Box“ vorstellen, in die wir etwas eingeben können und die sich aufgrund einer Folge von Eingaben in einem entsprechenden Zustand befindet (siehe Bild 2.1). In Abhängigkeit von ihrem jeweiligen Zustand und einer erfolgten Eingabe geht sie in einen Folgezustand über. Befindet sich die Black Box nach der Abarbeitung der Eingabefolge in einem ausgezeichneten Zustand, einem so genannten Endzustand, dann handelt es sich um eine von ihr akzeptierte Folge, falls sie sich nicht in einem Endzustand befindet, hat sie die Eingabe nicht akzeptiert.

Wir wollen endliche Automaten anhand eines konkreten Beispiels zum Systementwurf kennen lernen und aus diesem Beispiel die formalen Definitionen ableiten.

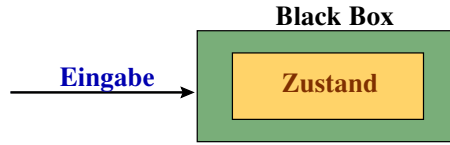
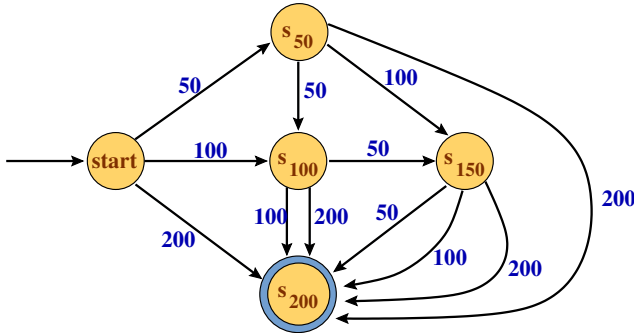


Bild 2.1: Endlicher Automat als Black Box.

Bild 2.2: Zustandsdiagramm des Eintrittsautomaten A_{Eintritt} .

2.1.1 Beispiel: Der Automat A_{Eintritt}

Es soll ein Automat hergestellt werden, der den Eintritt, z.B. in ein Schwimmbad, kontrolliert. Der Eintritt ins Schwimmbad möge € 2.00 kosten, und der Automat soll als Geldstücke 50-Cent-, 1-Euro- und 2-Euro-Stücke akzeptieren. Nach Eingabe einer Geldstückfolge im Gesamtwert von mindestens € 2.00 öffnet der Automat die Eintrittsschranke, vorher nicht.

Ohne an konkrete Hardware (technische Apparate) und Software zu denken, kann man sich den Eintrittsautomaten als Black Box vorstellen, in welche Geldstücke einzuwerfen sind, und die nach Einwurf einer Folge von Geldstücken im Wert von mindestens € 2.00 in einen akzeptierenden Endzustand geht, was in der realen technischen Lösung dann die Öffnung der Schranke bewirken soll.

Die Verarbeitung möglicher Eingabefolgen können wir uns mithilfe eines *Zustandsdiagramms* veranschaulichen (siehe Bild 2.2): Zu Beginn befindet sich der Automat im Zustand *start*. Bei Eingabe von 50 Cents geht der Automat in den Zustand s_{50} über, um sich zu merken, dass bereits 50 Cents eingegeben worden sind. Entsprechend geht er bei Eingabe von 1 Euro in den Zustand s_{100} sowie bei Eingabe von 2 Euro in den Zustand s_{200} über. In analoger Weise gibt das Diagramm die anderen möglichen Zustandsübergänge an. Eine Folge von Geldstücken mit demselben Gesamtwert „landet“ dabei sinnvollerweise im selben Zustand. Die im Zustandsdiagramm veranschaulichten Zustandsübergänge können auch als *Programm* des Automaten verstanden werden.

Die Folgen von Eingaben, die im Endzustand s_{200} enden, repräsentieren Geldstückfolgen, die mindestens den Gesamtwert € 2.00 haben. Dabei ist unser Automat

so angelegt, dass er Überbezahlungen stillschweigend akzeptiert. Wir werden später zeigen, wie mithilfe endlicher Automaten mit Ausgabe, mit so genannten *endlichen Maschinen*, ein „benutzerfreundlicher“ Eintrittsautomat modelliert werden kann, der nicht nur akzeptiert, sondern eine Eintrittskarte und zuviel gezahltes Geld zurückgibt und außerdem das noch einzuwerfende Geld anzeigt.

In einem Zustandsdiagramm werden Zustände durch Kreise und Zustandsübergänge durch Pfeile dargestellt, der Startzustand wird durch einen eingehenden Pfeil, Endzustände werden durch Doppelkreise gekennzeichnet.

Im Sinne abstrakter Rechnermodelle (siehe Kapitel 1.1) besitzt unser Eintrittsautomat zwei Speicher: ein (nach rechts unbeschränktes) *Eingabeband*, auf dem die Eingabefolgen, die Geldstückfolgen, geschrieben werden, und das für jede Eingabefolge von der Verarbeitungseinheit von links nach rechts gelesen wird, sowie einen Zustandsspeicher mit nur einer Speicherzelle, in welcher der jeweils aktuelle Zustand des Automaten geschrieben wird. Auf das Eingabeband hat der Automat nur lesenden Zugriff und zwar ausschließlich von links nach rechts sequentiell, auf den Zustandsspeicher hat er lesenden und schreibenden Zugriff.

In Bild 2.3 ist das Zustandsdiagramm von A_{Eintritt} in Form einer *Zustandstabelle* dargestellt. Der Automat befindet sich im Startzustand und soll die Eingabefolge 50 100 50 verarbeiten. Wir unterstreichen die Eingabesymbole, um zu verdeutlichen, dass es sich um atomare Symbole handelt. Die Eingabefolge 50 100 50 besteht aus den drei Symbolen 50, 100 und noch einmal 50, während z.B. die Folge 5010050 (die unser Automat nicht verarbeiten könnte) aus sieben Symbolen besteht, wobei 0, 1 und 5 die atomaren Symbole sind.

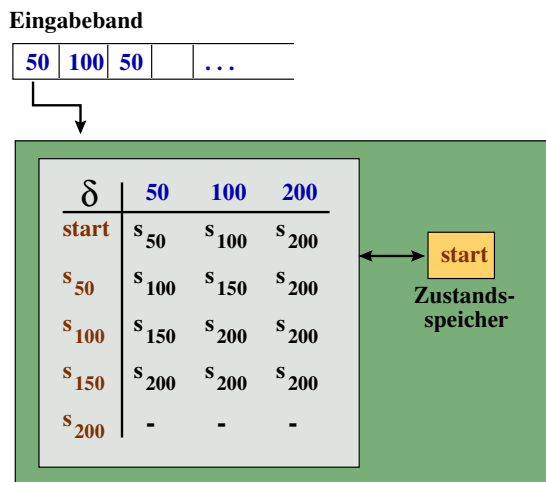


Bild 2.3: Startkonfiguration von A_{Eintritt} .

Die Ausführung eines Programmschrittes, d.h. ein Zustandsübergang, findet folgendermaßen statt: Befindet sich der Automat im Zustand s und der Lesekopf über einer Eingabebandzelle mit Inhalt a , dann geht der Automat in den Zustand über, der

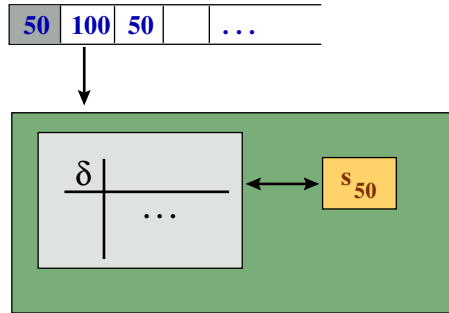


Bild 2.4: Ausführung des Eintrittsprogramms (1).

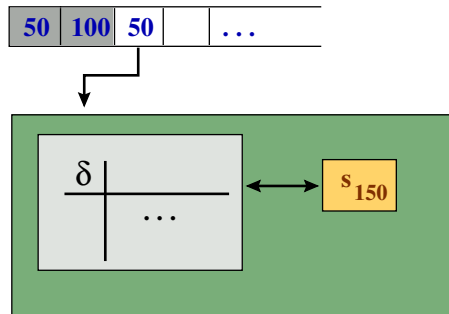


Bild 2.5: Ausführung des Eintrittsprogramms (2).

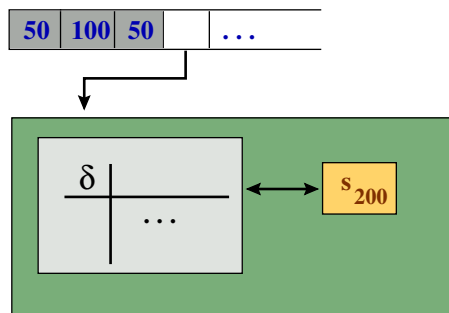


Bild 2.6: Ausführung des Eintrittsprogramms (3).

in der Tabelle in Zeile s und Spalte a steht. Außerdem wird der Lesekopf auf dem Eingabeband auf die nächste Zelle bewegt. Die Bilder 2.4, 2.5 und 2.6 zeigen die weitere Ausführung des Eintrittsprogramms.

Ist die Eingabefolge komplett gelesen, d.h. der Lesekopf befindet sich auf der Speicherzelle hinter der Eingabefolge (siehe Bild 2.6), und beinhaltet der Zustandsspeicher einen Endzustand, dann ist die Eingabefolge vom Automaten *akzeptiert*, in allen anderen Fällen ist die Eingabefolge nicht akzeptiert. So wird von unserem Eintrittsautomat die Folge 50 100 50 akzeptiert, weil nach Verarbeitung dieser Folge der Automat sich im Endzustand s_{200} befindet.

Die Menge aller von einem Automaten akzeptierten Folgen heißt die *Sprache* des Automaten. Die Sprache unseres Eintrittsautomaten ist die folgende Menge von Geldstückfolgen: $\{\underline{50\ 50\ 50\ 50}, \underline{50\ 50\ 50\ 100}, \underline{50\ 50\ 50\ 200}, \underline{50\ 50\ 100}, \underline{50\ 50\ 200}, \underline{50\ 100\ 50}, \underline{50\ 100\ 100}, \underline{50\ 100\ 200}, \underline{50\ 200}, \underline{100\ 50\ 50}, \underline{100\ 50\ 100}, \underline{100\ 50\ 200}, \underline{100\ 100}, \underline{100\ 200}, \underline{200}\}$.

Unser Eintrittsautomat ist ein Beispiel für einen endlichen Automaten. Wir wollen nun mithilfe mathematischer Begriffe Automaten dieser Art sowie ihre Wirkungsweise formal beschreiben.

2.1.2 Alphabete, Wörter, Sprachen

Ein endlicher Automat verarbeitet Wörter, d.h. endliche Zeichenfolgen, die aus atomaren Symbolen gebildet sind, wie z.B. die Folge 100 50 200 von Symbolen für Geldstücke. Die (Eingabe-) Wörter, die ein endlicher Automat verarbeiten soll, stellen wir uns auf einem nach rechts offenen (Eingabe-) Speicher, dem Eingabeband, geschrieben vor. Ein „Lesekopf“ lese das Eingabewort symbolweise von links nach rechts.

Alphabete

Die Symbole, aus denen Eingabewörter bestehen können, fassen wir in einer Menge, dem *Eingabealphabet*, zusammen. Dabei wollen wir nur endliche Eingabealphabete zulassen. Ihre Elemente heißen Symbole oder Buchstaben. Zur Bezeichnung von Eingabealphabeten wird zumeist das Symbol Σ benutzt. Das Eingabealphabet unseres Eintrittsautomaten ist also: $\Sigma = \{\underline{50}, \underline{100}, \underline{200}\}$.

Falls wir kein konkretes, sondern allgemein ein Alphabet Σ betrachten, dann benennen wir dessen Buchstaben in der Regel mit a oder mit a_1, a_2, \dots, a_n :

$$\Sigma = \{a_1, a_2, \dots, a_n\}, n \geq 0$$

$n = 0$ bedeutet, dass Σ leer ist: $\Sigma = \emptyset$. Dabei soll durch die Reihenfolge der Aufzählung der Buchstaben auf Σ eine (lexikografische oder alphabetische) Ordnung festgelegt sein: $a_i < a_{i+1}$, $1 \leq i \leq n - 1$. Zur Bezeichnung von Buchstaben verwenden wir neben a in der Regel weitere Buchstaben vom Anfang des deutschen Alphabetes: b, c, d, \dots

Wörter

Die endlich langen Zeichenfolgen, die über einem Alphabet Σ gebildet werden können, heißen *Wörter über Σ* . Wörter entstehen, indem Symbole oder bereits erzeugte Wörter aneinandergereiht (miteinander verkettet, konkateniert) werden. Σ^* , die Menge aller Wörter, die über dem Alphabet Σ gebildet werden kann, ist wie folgt definiert:

- (1) Jeder Buchstabe $a \in \Sigma$ ist auch ein Wort über Σ , d.h. $a \in \Sigma^*$.
- (2) Werden bereits konstruierte Wörter hintereinandergeschrieben, entstehen neue Wörter, d.h. sind $v, w \in \Sigma^*$, dann ist auch ihre Verkettung (Konkatenation) $vw \in \Sigma^*$.
- (3) ε , das *leere Wort*, ist ein Wort über (jedem Alphabet) Σ , d.h. es gilt immer $\varepsilon \in \Sigma^*$. ε ist ein definiertes Wort ohne „Ausdehnung“. Es hat die Eigenschaft: $\varepsilon w = w\varepsilon = w$ für alle $w \in \Sigma^*$.

Mit Σ^+ bezeichnen wir die Menge aller Wörter über Σ ohne das leere Wort, d.h. $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

Im algebraischen Sinne bildet die Rechenstruktur (Σ^+, \circ) für ein Alphabet Σ und die (Konkatenations-) Operation $\circ : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ definiert durch $v \circ w = vw$ eine *Halbgruppe*, denn die Konkatenation ist eine assoziative Operation: für alle Wörter $u, v, w \in \Sigma^*$ gilt $u \circ (v \circ w) = (u \circ v) \circ w$. Wegen der in der obigen Definition festgelegten Eigenschaft (3) bildet die Struktur (Σ^*, \circ) ein *Monoid* mit dem *Einselement* ε . Die Konkatenation von Wörtern ist über Alphabeten mit mehr als einem Buchstaben nicht kommutativ.

Allgemein notieren wir Wörter in der Regel mit Buchstaben vom Ende des deutschen Alphabetes: u, v, w, x, y, z . Wenn wir im Folgenden ein Wort w buchstabenweise notieren, $w = w_1 \dots w_n$, $n \geq 0$, sind die w_i Buchstaben, also $w_i \in \Sigma$, $1 \leq i \leq n$. $n = 0$ bedeutet, dass w das leere Wort ist: $w = \varepsilon$.

Beispiel 2.1. a) Sei $\Sigma = \{a, b\}$, dann ist

$$\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \dots\}$$

b) Beim Eintrittsautomat ist das Eingabealphabet $\Sigma = \{\underline{50}, \underline{100}, \underline{200}\}$. Σ^* stellt die Menge aller endlichen Geldstückfolgen dar, die mit 50-Cent-, 1-Euro- und 2-Euro-Stücken gebildet werden können einschließlich der leeren Folge. \square

Folgerung 2.1. Ist ein Alphabet Σ nicht leer, dann besitzt Σ^* unendlich viele Wörter; ist dagegen $\Sigma = \emptyset$, dann ist $\Sigma^* = \{\varepsilon\}$. \square

Ist

$$w = \underbrace{v \dots v}_{n\text{-mal}}$$

dann schreiben wir abkürzend $w = v^n$. v^n heißt die *n-te Potenz* von v . Es ist $v^0 = \varepsilon$. In Beispiel 2.1 a) können wir Σ^* also auch wie folgt schreiben:

$$\Sigma^* = \{\varepsilon, a, b, a^2, ab, ba, b^2, a^3, a^2b, aba, ab^2, ba^2, bab, b^2a, b^3, \dots\}$$

Die n -te Potenz eines Wortes v kann auch rekursiv definiert werden: Es ist $v^0 = \varepsilon$ sowie $v^{n+1} = v^n \circ v$ für $n \geq 0$.

Ist $w \in \Sigma^*$ ein Wort der Form $w = xyz$ mit $x, y, z \in \Sigma^*$, dann heißt x *Präfix*, y *Infix* und z *Postfix*, *Suffix* oder *Rest* von w . Man beachte, dass diese Definition die Spezialfälle zulässt, dass x oder y oder z leer sein können. So ist wegen $w = \varepsilon w \varepsilon$ ein Wort w sowohl Präfix als auch Infix als auch Postfix von sich selbst, oder für $w = xy$ mit $x, y \in \Sigma^+$ ist x sowohl Präfix als auch Infix und y ist sowohl Infix als auch Postfix von w .

Falls x ein Präfix von w ist und $x \neq w$ gilt, dann heißt x *echter Präfix* von w . Entsprechende Definitionen gelten für *echter Infix* bzw. *echter Postfix*.

$\text{Pref}(w) = \{x \mid x \text{ ist Präfix von } w\}$ ist die Menge der Präfixe von w ; entsprechend können die Menge $\text{Inf}(w)$ der Infixe bzw. die Menge $\text{Suf}(w)$ der Suffixe von w definiert werden. Es gilt also z.B.

$$\begin{aligned}\text{Pref}(aba) &= \{\varepsilon, a, ab, aba\} \\ \text{Inf}(aba) &= \{\varepsilon, a, b, ab, ba, aba\} \\ \text{Suf}(aba) &= \{\varepsilon, a, ba, aba\}\end{aligned}$$

Wortfunktionen

Auf Wörtern lassen sich eine Reihe von Funktionen definieren, welche sich in Anwendungen z.B. als *Textverarbeitungsfunktionen* wiederfinden. Man denke etwa an einen Editor, welcher es gestattet, nach einem Wort zu suchen. Der Editor fasst eine Textdatei als *einen* (langen) Byte-String auf und sucht darin einen bestimmten Teil- oder Substring. Diese Problemstellung lässt sich formal beschreiben mit der Funktion¹

$$\text{substr} : \Sigma^+ \times \Sigma^+ \rightarrow \{0, 1\}$$

definiert durch

$$\text{substr}(w, v) = \begin{cases} 1, & \text{falls } v \text{ Infix von } w \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

Diese Funktion testet, ob v ein *Teilwort* (englisch: *substring*) von w ist. Es gilt z.B.

$$\text{substr}(ababba, abb) = 1 \text{ sowie } \text{substr}(ababba, aa) = 0$$

In Abschnitt 2.6.2 wird ein Algorithmus angegeben, der mithilfe endlicher Automaten das Problem der Teilworterkennung effizient löst. Dort wird die folgende Variante der Funktion *substr* berechnet:² $\text{substr} : \Sigma^* \times \mathbb{N}_0 \times \mathbb{N}_0 \rightarrow \Sigma^*$ definiert durch

$$\text{substr}(w_1 \dots w_n, i, l) = \begin{cases} w_i \dots w_{i+l-1}, & i + l - 1 \leq n \\ \perp, & \text{sonst} \end{cases}$$

¹ „1“ steht für die Antwort *ja*, „0“ steht für *nein*.

² Wir bezeichnen mit \mathbb{N}_0 die Menge der natürlichen Zahlen einschließlich 0, d.h. $\mathbb{N}_0 = \{0, 1, 2, 3, \dots\}$, und mit \mathbb{N} die Menge der natürlichen Zahlen ohne 0: $\mathbb{N} = \{1, 2, 3, \dots\}$.

$\text{substr}(w, i, l)$ liefert das Infix von w der Länge l ab dem i -ten Buchstaben von w , falls ein solches existiert.³

Die *Länge eines Wortes* kann durch die Funktion $\ell : \Sigma^* \rightarrow \mathbb{N}_0$ definiert durch $\ell(\varepsilon) = 0$ und $\ell(wa) = \ell(w) + 1$ für $w \in \Sigma^*$ und $a \in \Sigma$ berechnet werden, die einem Wort über Σ die Anzahl seiner Buchstaben als Länge zuordnet: Das leere Wort hat die Länge 0; die Länge eines Wortes, das mindestens einen Buchstaben a sowie ein – möglicherweise leeres – Präfix w enthält, wird berechnet, indem zur Länge des Präfixes eine 1 addiert wird. Die Länge des Wortes aba über $\Sigma = \{a, b\}$ berechnet sich also rekursiv mit der Funktion ℓ wie folgt:

$$\ell(aba) = \ell(ab) + 1 = \ell(a) + 1 + 1 = \ell(\varepsilon) + 1 + 1 + 1 = 0 + 1 + 1 + 1 = 3$$

Eine andere gängige Bezeichnung für die Länge $\ell(w)$ eines Wortes w ist $|w|$. Offensichtlich gilt für $x, y \in \Sigma^*$:

$$\ell(xy) = \ell(x) + \ell(y) \text{ bzw. } |xy| = |x| + |y|$$

Die Funktion

$$\text{anzahl} : \Sigma^* \times \Sigma \rightarrow \mathbb{N}_0$$

zählt, wie oft ein Buchstabe in einem Wort vorkommt. Sie ist definiert durch

$$\text{anzahl}(\varepsilon, b) = 0 \text{ für alle } b \in \Sigma$$

und

$$\text{anzahl}(wa, b) = \begin{cases} \text{anzahl}(w, b) + 1, & a = b \\ \text{anzahl}(w, b), & a \neq b \end{cases} \quad \text{für } a, b \in \Sigma, w \in \Sigma^*$$

Im leeren Wort kommt kein Buchstabe vor. Stimmt der zu zählende Buchstabe mit dem letzten Buchstaben des zu untersuchenden Wortes überein, wird er gezählt, und die Anzahl muss noch für das Wort ohne den letzten Buchstaben bestimmt werden. Ist der zu zählende Buchstabe verschieden vom letzten Buchstaben, muss nur im Wort ohne den letzten Buchstaben gezählt werden. Es gilt z.B.:

$$\begin{aligned} \text{anzahl}(01010, 1) &= \text{anzahl}(0101, 1) = \text{anzahl}(010, 1) + 1 = \text{anzahl}(01, 1) + 1 \\ &= \text{anzahl}(0, 1) + 1 + 1 = \text{anzahl}(\varepsilon, 1) + 1 + 1 = 0 + 1 + 1 \\ &= 2 \end{aligned}$$

Eine andere Notation für $\text{anzahl}(w, a)$ ist auch $|w|_a$.

Die Anwendung $\text{tausche}(w, a, b)$ der Funktion $\text{tausche} : \Sigma^* \times \Sigma \times \Sigma \rightarrow \Sigma^*$ ersetzt jedes Vorkommen des Buchstabens a im Wort w durch den Buchstaben b . Formal kann tausche definiert werden durch

$$\text{tausche}(\varepsilon, a, b) = \varepsilon$$

³Für eine Funktion f schreiben wir $f(x) = \perp$, falls f für das Argument x nicht definiert ist, d.h. falls x nicht zum Definitionsbereich von f gehört: $x \notin \text{Def}(f)$.

und

$$\text{tausche}(cw, a, b) = \begin{cases} b \circ \text{tausche}(w, a, b), & \text{falls } c = a \\ c \circ \text{tausche}(w, a, b), & \text{falls } c \neq a \end{cases}$$

für $c \in \Sigma$ und $w \in \Sigma^*$. Es gilt z.B. für $\Sigma = \{1, 2, 3\}$:

$$\begin{aligned} \text{tausche}(12322, 2, 1) &= 1 \circ \text{tausche}(2322, 2, 1) = 1 \circ 1 \circ \text{tausche}(322, 2, 1) \\ &= 11 \circ 3 \circ \text{tausche}(22, 2, 1) = 113 \circ 1 \circ \text{tausche}(2, 2, 1) \\ &= 1131 \circ 1 \circ \text{tausche}(\varepsilon, 2, 1) = 11311 \circ \varepsilon \\ &= 11311 \end{aligned}$$

Wir wollen nun noch Ordnungen für Wörter festlegen. Enthält ein geordnetes Alphabet Σ nur einen Buchstaben, ist eine „natürliche“ Ordnung der Wörter von Σ^* durch die Länge der Wörter gegeben. Wenn $|\Sigma| \geq 2$ ist, sind folgende beiden Ordnungen von Interesse:

- Die *lexikografische Ordnung* $\leq \subseteq \Sigma^* \times \Sigma^*$ ist definiert durch: Es gilt $v \leq w$ genau dann, wenn $v \in \text{Pref}(w)$ oder $v = \alpha a \beta$ und $w = \alpha b \gamma$ mit $\alpha, \beta, \gamma \in \Sigma^*$ und $a, b \in \Sigma$ mit $a < b$ ist.
- Die *längenlexikografische Ordnung* $\prec \subseteq \Sigma^* \times \Sigma^*$ ist definiert durch: Es gilt $v \prec w$ genau dann, wenn $|v| < |w|$ oder wenn $|v| = |w|$ und $v \leq w$ ist.

Beispiel 2.2. Sei $\Sigma = \{a, b, c\}$ ein geordnetes Alphabet mit $a < b < c$, dann gilt: $aba \leq ac$, $ac \prec aba$ und $cc \prec aaa$, während $aca \prec aba$, $bb \leq b$ und $bb \prec b$ nicht zutreffen. \square

Formale Sprachen

Wir kommen jetzt auf Sprachen als Ganzes zurück und betrachten *Operationen* auf diesen. Sei Σ ein Alphabet, dann heißt jede Menge $L \subseteq \Sigma^*$ eine (*formale*) *Sprache* über Σ . Sprachen sind also Mengen von Wörtern und können somit mit den üblichen Mengenoperationen wie Vereinigung, Durchschnitt, Differenz miteinander verknüpft werden. Wir wollen für Sprachen eine weitere Verknüpfung einführen: die *Konkatenation*. Seien L_1 und L_2 zwei Sprachen über Σ , dann ist die *Konkatenation* $L_1 \circ L_2$ von L_1 und L_2 definiert durch

$$L_1 \circ L_2 = \{vw \mid v \in L_1, w \in L_2\}$$

Es werden also alle Wörter aus L_1 mit allen Wörtern aus L_2 konkateniert. Gelegentlich lassen wir wie bei der Konkatenation von Wörtern auch bei der Konkatenation von Sprachen das Konkatenationssymbol \circ weg, d.h. anstelle von $L_1 \circ L_2$ schreiben wir $L_1 L_2$. Seien $L_1 = \{\varepsilon, ab, abb\}$ und $L_2 = \{b, ba\}$ zwei Sprachen über dem Alphabet $\Sigma = \{a, b\}$, dann ist

$$L_1 \circ L_2 = \{b, ba, abb, abba, abbb, abbba\}$$

sowie

$$L_2 \circ L_1 = \{b, bab, babb, ba, baab, baabb\}$$

Folgerung 2.2. Allgemein gilt: Falls $\varepsilon \in L_1$ ist, dann ist $L_2 \subseteq L_1 \circ L_2$, bzw. umgekehrt, falls $\varepsilon \in L_2$ ist, dann ist $L_1 \subseteq L_1 \circ L_2$. \square

Die n -te Potenz einer Sprache $L \subseteq \Sigma^*$ ist festgelegt durch: $L^0 = \{\varepsilon\}$ sowie $L^{n+1} = L^n \circ L$ für $n \geq 0$. Sei $L = \{a, ab\}$, dann ist

$$\begin{aligned} L^0 &= \{\varepsilon\} \\ L^1 &= L^0 \circ L = \{\varepsilon\} \circ \{a, ab\} = \{a, ab\} = L \\ L^2 &= L^1 \circ L = \{a, ab\} \circ \{a, ab\} = \{a^2, a^2b, aba, (ab)^2\} \\ L^3 &= L^2 \circ L = \{a^2, a^2b, aba, (ab)^2\} \circ \{a, ab\} \\ &= \{a^3, a^3b, a^2ba, a^2bab, aba^2, aba^2b, (ab)^2a, (ab)^3\} \\ &\vdots \end{aligned}$$

Das Kleene-Stern-Produkt⁴ L^* einer Sprache L ist die Vereinigung aller ihrer Potenzen L^n , $n \geq 0$:

$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \dots$$

Das Kleene-Stern-Produkt von L ohne das leere Wort notieren wir mit L^+ , d.h. es ist

$$L^+ = \bigcup_{n \geq 1} L^n = L^* - L^0$$

Als weitere Operation auf Sprachen führen wir die *Spiegelung* ein. Zunächst definieren wir die Spiegelung von Wörtern: Sei Σ ein Alphabet, dann ist $sp : \Sigma^* \rightarrow \Sigma^*$ definiert durch $sp(\varepsilon) = \varepsilon$ und $sp(wa) = a \circ sp(w)$ für $w \in \Sigma^*$ und $a \in \Sigma$: Die Spiegelung des leeren Wortes ist trivialerweise das leere Wort, die Spiegelung eines nicht leeren Wortes erhält man, indem man den letzten Buchstaben nach vorne schreibt und mit dem Rest genauso verfährt, bis der Rest leer geworden ist. Berechnen wir als Beispiel die Spiegelung des Wortes aab :

$$sp(aab) = b \, sp(aa) = ba \, sp(a) = baa \, sp(\varepsilon) = baa\varepsilon = baa$$

Wir verallgemeinern die Spiegelung von Wörtern auf Sprachen:⁵ Die Funktion

$$SP : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$$

sei definiert durch

$$SP(L) = \{sp(w) \mid w \in L\}$$

$SP(L)$ enthält alle gespiegelten Wörter von L .

Eine Sprache $L \subseteq \Sigma^*$ heißt *präfixfrei* (hat die *Präfixeigenschaft*) genau dann, wenn für alle Wörter $w \in L$ gilt: Ist x ein echter Präfix von w , dann ist $x \notin L$. Von einem Wort $w \in L$ darf also kein echtes Präfix Wort der Sprache sein.

⁴Benannt nach Stephen C. Kleene (1909 – 1998), amerikanischer Mathematiker und Logiker, der fundamentale Beiträge zur Logik und zu theoretischen Grundlagen der Informatik geliefert hat.

⁵ 2^M sei die Potenzmenge der Menge M , d.h. die Menge aller Teilmengen der Menge M . 2^{Σ^*} ist also die Menge aller Sprachen, die über dem Alphabet Σ gebildet werden können.

Beispiel 2.3. Betrachten wir als Beispiele die Sprachen

$$L_1 = \{a^n b^n \mid n \geq 0\} \text{ und } L_2 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$$

L_1 hat die Präfixeigenschaft, denn für jedes Wort $w = a^n b^n$ sind alle echten Präfixe $x = a^n b^i$ mit $n > i$ sowie $x = a^j$ für $j \geq 0$ keine Wörter von L_1 . L_2 ist nicht präfixfrei, denn z.B. ist $w = abab \in L_2$ und $x = ab \in L_2$. \square

Satz 2.1. Eine Sprache $L \subseteq \Sigma^*$ ist präfixfrei genau dann, wenn $L \cap (L \circ \Sigma^+) = \emptyset$ ist.

Beweis „ \Rightarrow “: Wir nehmen an, es sei $L \cap (L \circ \Sigma^+) \neq \emptyset$. Dann gibt es ein Wort $w \in L \cap (L \circ \Sigma^+)$, d.h. es ist $w \in L$ und $w \in (L \circ \Sigma^+)$. Aus der zweiten Eigenschaft folgt, dass es ein $x \in L$ und ein $y \in \Sigma^+$ geben muss mit $w = xy$. Somit gibt es also einen echten Präfix x von w , der in L enthalten ist. Das bedeutet aber einen Widerspruch dazu, dass L präfixfrei ist. Damit ist unsere Annahme falsch, d.h. wenn L präfixfrei ist, dann ist $L \cap (L \circ \Sigma^+) = \emptyset$.

„ \Leftarrow “: Sei nun $L \cap (L \circ \Sigma^+) = \emptyset$. Wir nehmen nun an, dass L nicht präfixfrei ist. Es gibt also ein Wort $w \in L$ mit einem echten Präfix $x \in L$, d.h. es gibt ein $y \in \Sigma^+$ mit $w = xy$. Damit gilt, dass $w \in L \cap (L \circ \Sigma^+)$, d.h. dass $L \cap (L \circ \Sigma^+) \neq \emptyset$ ist, was ein Widerspruch zur Voraussetzung $L \cap (L \circ \Sigma^+) = \emptyset$ ist. Unsere Annahme ist also falsch, L muss also präfixfrei sein.

Damit haben wir insgesamt die Behauptung des Satzes bewiesen. \square

2.1.3 Zustände und Zustandsübergänge

Der andere Speicher eines endlichen Automaten neben dem Eingabeband ist der Zustandsspeicher. Er besteht aus nur einer Speicherzelle, dessen Inhalt der jeweils aktuelle Zustand des Automaten ist. Der aktuelle Zustand kann als Repräsentant oder als Merker für die bisherige Verarbeitung eines Eingabewortes angesehen werden.

Die möglichen Zustände eines Automaten fassen wir in der Zustandsmenge des Automaten zusammen. Zustandsmengen werden allgemein in der Regel mit S bezeichnet. Es werden nur endlich viele Zustände zugelassen, d.h. S ist immer eine endliche Menge. Deswegen heißen die hier betrachteten Automaten *endliche* Automaten. Der Eintrittsautomat hat z.B. die Zustandsmenge

$$S = \{start, s_{50}, s_{100}, s_{150}, s_{200}\}$$

Zu Beginn der Verarbeitung einer Eingabefolge befindet sich ein Automat immer in einem ausgezeichneten Zustand, dem *Startzustand*. Der Eintrittsautomat $A_{Eintritt}$ hat den Startzustand *start*.

Ein Automat akzeptiert eine Eingabefolge, falls er sich, beginnend im Startzustand, nach Verarbeitung der kompletten Eingabefolge in einem Endzustand befindet. Die *Endzustände* eines Automaten werden in der Menge $F \subseteq S$ zusammengefasst. Die Menge der Endzustände des Eintrittsautomaten $A_{Eintritt}$ ist $F = \{s_{200}\}$, der Automat enthält also nur einen Endzustand.

Ein Automat geht in Abhängigkeit seines aktuellen Zustandes sowie des nächsten zu verarbeitenden Symbols der Eingabefolge in einen neuen Zustand über. Diese Verhaltensweise, das „Programm“, kann durch die *Zustandsüberföhrungsfunktion*

$$\delta : S \times \Sigma \rightarrow S$$

festgelegt werden. $\delta(s, a) = s'$ bedeutet, dass der Automat, wenn er sich im Zustand s und der Lesekopf sich unter einer Eingabe-Speicherzelle mit dem Inhalt a befindet, in den Zustand s' übergeht.

Die Zustandsüberföhrungsfunktion wird wegen besserer Lesbarkeit zumeist in Form einer Zustandstabelle (siehe Bild 2.3) oder in Form eines Zustandsdiagramms (siehe Bild 2.2) dargestellt.

2.1.4 Deterministische endliche Automaten und reguläre Sprachen

Wir haben nun alle Komponenten, die einen endlichen Automaten festlegen, formal beschrieben. Die folgende Definition fasst diese Festlegungen zusammen:

Definition 2.1. Ein *deterministischer endlicher (Zustands-) Automat* (auch: *endlicher Akzeptor*) A ist festgelegt durch $A = (\Sigma, S, \delta, s_0, F)$. Dabei ist Σ das Eingabealphabet und S die Zustandsmenge von A , $s_0 \in S$ ist der Startzustand, $F \subseteq S$ die Menge der Endzustände und $\delta : S \times \Sigma \rightarrow S$ die Zustandsüberföhrungsfunktion von A . \square

Es sei bemerkt, dass δ keine totale Funktion sein muss, d.h. δ muss nicht für alle Paare $(s, a) \in S \times \Sigma$ definiert sein. Wir werden später sehen, wie ein partiell definierter endlicher Automat zu einem äquivalenten total definierten erweitert werden kann.

A heißt deterministisch, weil zu einem Zustand und einem Eingabesymbol höchstens ein Folgezustand existieren kann (δ ist eine Funktion). Falls aus dem Zusammenhang klar ist, dass ein endlicher Automat deterministisch ist, lassen wird dieses Adjektiv weg.

Gemäß Definition 2.1 wird unser Eintrittsautomat A_{Eintritt} also formal wie folgt notiert:

$$A_{\text{Eintritt}} = (\{\underline{50}, \underline{100}, \underline{200}\}, \{start, s_{50}, s_{100}, s_{150}, s_{200}\}, \delta, start, \{s_{200}\})$$

wobei δ durch die Zustandstabelle in Bild 2.3 bzw. durch das Zustandsdiagramm in Bild 2.2 gegeben ist. A_{Eintritt} ist nicht total, denn δ ist z.B. für $(s_{200}, \underline{50})$ nicht definiert.

Wir können einen endlichen Automaten A als ein Programm auffassen, das Deklarationen umfasst für mögliche Eingaben (Σ), für Verarbeitungszustände (S), für Programmstart (s_0) und für Programmende (F) sowie Anweisungen (δ) zur Verarbeitung eines Eingabewortes. Die Anweisungen, d.h. die Zustandsübertragungsfunktion, können wir auch als Menge von Tripeln schreiben:

$$\delta = \{(s, a, s') \mid \delta(s, a) = s'\}$$

Das Tripel (s, a, s') kann als „Programmzeile“ des Programms δ von A aufgefasst werden.

Verhalten von Automaten

Wir wollen nun die Verhaltensweise eines Automaten A betrachten, d.h. wir wollen formal beschreiben, wie A das Programm δ auf einem Eingabewort ausführt. Wir werden zwei Ansätze zeigen, um die Programmausführung zu beschreiben: Einer benutzt die von Buchstaben auf Wörter erweiterte Zustandsüberföhrungsfunktion, der andere den Begriff der Konfiguration.

Sei $A = (\Sigma, S, \delta, s_0, F)$ ein endlicher Automat. Wir erweitern die Zustandsüberföhrungsfunktion δ auf

$$\delta^* : S \times \Sigma^* \rightarrow S$$

definiert durch

$$\delta^*(s, \varepsilon) = s \text{ für alle } s \in S$$

sowie durch

$$\delta^*(s, aw) = \delta^*(\delta(s, a), w) \text{ für alle } a \in \Sigma \text{ und } w \in \Sigma^*$$

δ^* beschreibt die schrittweise Abarbeitung eines Wortes v beginnend im Zustand s : Ist $v = \varepsilon$, findet keine Verarbeitung statt, A bleibt im Zustand s . Ist $v = aw$, wird der Folgezustand $\delta(s, a)$ für den ersten Buchstaben a von v berechnet, und es muss dann noch δ^* für diesen Folgezustand und den Rest w von v berechnet werden. Ist der Rest das leere Wort, ist die Berechnung fertig. $\delta^*(s, v) = s'$ bedeutet, dass, wenn A sich im Zustand s befindet, er sich nach Abarbeitung des Wortes v im Zustand s' befindet.

Beispiel 2.4. Wir wollen eine Berechnung des Eintrittsautomaten A_{Eintritt} ausführen, und zwar für den Zustand s_{50} und das Wort 50 50 100:

$$\begin{aligned} \delta^*(s_{50}, \underline{50\ 50\ 100}) &= \delta^*(\delta(s_{50}, \underline{50}), \underline{50\ 100}) = \delta^*(s_{100}, \underline{50\ 100}) \\ &= \delta^*(\delta(s_{100}, \underline{50}), \underline{100}) = \delta^*(s_{150}, \underline{100}) \\ &= \delta^*(s_{150}, \underline{100\varepsilon}) = \delta^*(\delta(s_{150}, \underline{100}), \varepsilon) \\ &= \delta^*(s_{200}, \varepsilon) \\ &= s_{200} \end{aligned}$$

□

Konfigurationen und Konfigurationsübergänge

Eine Konfiguration k_A eines Automaten beschreibt den aktuellen Stand der Verarbeitung einer Eingabefolge durch A . (Falls aus dem Zusammenhang klar ist, auf welchen Automaten A sich eine Konfiguration k_A bezieht, lassen wir im Folgenden den Index A weg und schreiben nur k .) Dieser wird durch zwei Aspekte festgelegt: zum einen durch den aktuellen Zustand von A , zum anderen durch das noch zu verarbeitende Suffix des Eingabewortes. Eine Konfiguration kann also durch ein Paar $k = (s, v)$ mit $s \in S$ und $v \in \Sigma^*$ beschrieben werden: s ist der aktuelle Zustand, v der noch zu verarbeitende Rest des Eingabewortes (der Lesekopf befindet sich unter dem ersten Buchstaben von v). Die Menge K aller prinzipiell möglichen Konfigurationen ist die Menge aller Paare von Zuständen und Wörtern: $K = S \times \Sigma^*$.

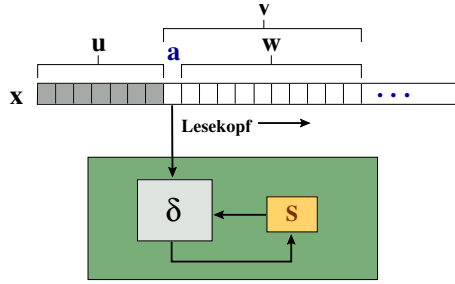


Bild 2.7: Konfiguration eines endlichen Automaten.

Ein Übergang von einer Konfiguration $k = (s, aw)$ zu einer (Folge-) Konfiguration $k' = (s', w)$ kann stattfinden, falls die Zustandsüberführung $\delta(s, a) = s'$ existiert, wobei $s, s' \in S$, $w \in \Sigma^*$ und $a \in \Sigma$ ist (siehe Bild 2.7).

Ein Konfigurationsübergang von $k = (s, w)$ zu $k' = (s', w')$ wird formal durch

$$(s, w) \vdash (s', w')$$

notiert. Die Abarbeitung eines eingegebenen Wortes $x = x_1x_2 \dots x_r$ durch einen endlichen Automaten A können wir nun durch eine Folge von Konfigurationsübergängen darstellen:

$$(s_0, x_1x_2 \dots x_r) \vdash (s_1, x_2 \dots x_r) \vdash \dots \vdash (s_r, \varepsilon)$$

Dabei muss gelten: $\delta(s_i, x_{i+1}) = s_{i+1}$, $0 \leq i < r$. Ist $s_r \in F$, dann wird das Wort x von A akzeptiert, sonst nicht. Für unseren Eintrittsautomaten A_{Eintritt} gilt also z.B.:

$$(start, \underline{50} \underline{100} \underline{50}) \vdash (s_{50}, \underline{100} \underline{50}) \vdash (s_{150}, \underline{50}) \vdash (s_{200}, \varepsilon)$$

Das Wort 50 100 50 wird vom Eintrittsautomaten akzeptiert, da er sich nach Abarbeitung dieses Wortes in einem Endzustand befindet.

\vdash ist eine Relation über der Menge aller möglichen Konfigurationen $K = S \times \Sigma^*$:

$$\vdash \subseteq (S \times \Sigma^*) \times (S \times \Sigma^*)$$

definiert durch

$$(s, aw) \vdash (s', w) \text{ genau dann, wenn } \delta(s, a) = s', a \in \Sigma, w \in \Sigma^*$$

\vdash^* bezeichnet die reflexiv-transitive Hülle der Relation \vdash . Diese ist rekursiv definiert durch: Es gilt $k \vdash^* k$ für alle $k \in K$, und $k \vdash^* k'$, falls ein $k'' \in K$ existiert mit $k \vdash^* k''$ und $k'' \vdash k'$. Das heißt, wenn es eine Folge von Konfigurationsübergängen

$$k_1 \vdash k_2 \vdash \dots \vdash k_n, n \geq 1$$

gibt, dann gilt:

$$k_1 \vdash^* k_n$$

Oben haben wir gesehen, dass für A_{Eintritt}

$$(start, \underline{50} \underline{100} \underline{50}) \vdash (s_{50}, \underline{100} \underline{50}) \vdash (s_{150}, \underline{50}) \vdash (s_{200}, \varepsilon)$$

gilt, und damit gilt auch

$$(start, \underline{50} \underline{100} \underline{50}) \vdash^* (s_{200}, \varepsilon)$$

δ^* und \vdash^* sind offensichtlich verwandte Beschreibungen der Arbeitsweise eines endlichen Automaten. Es lässt sich sehr leicht begründen, dass

$$\delta^*(s, w) = s' \text{ genau dann gilt, wenn } (s, w) \vdash^* (s', \varepsilon)$$

gilt. So gilt für A_{Eintritt} z.B.:

$$\delta^*(start, \underline{50} \underline{100} \underline{50}) = s_{200}$$

und

$$(start, \underline{50} \underline{100} \underline{50}) \vdash^* (s_{200}, \varepsilon)$$

δ^* beschreibt das Verhalten eines Automaten A : Für den Zustand s und das Wort w ist $\delta^*(s, w)$ der Zustand, in dem sich A nach Abarbeitung von w befindet (falls dieser Zustand existiert). \vdash^* beschreibt den Prozess, d.h. die Konfigurationsfolge, die ein Automat beginnend in einem Zustand bei Anliegen eines Wortes durchläuft. Man kann δ^* als die *funktionale Semantik* eines Automaten bezeichnen, \vdash^* als *operationale Semantik*. Bei endlichen Automaten stimmen diese Semantiken überein.

Reguläre Sprachen

Die Menge aller Eingabewörter w , die einen Automaten A von der Startkonfiguration (s_0, w) in endlich vielen Schritten in eine Endkonfiguration (s, ε) , $s \in F$, überführen, heißt die von A akzeptierte Sprache.

Definition 2.2. a) Sei $A = (\Sigma, S, \delta, s_0, F)$ ein endlicher Automat. Dann heißt die Sprache $L(A) = \{w \in \Sigma^* \mid (s_0, w) \vdash^* (s, \varepsilon), s \in F\}$ die von A akzeptierte Sprache.

b) Eine Sprache $L \subseteq \Sigma^*$ heißt *regulär*, falls es einen endlichen Automaten A gibt, der L akzeptiert, d.h. für den $L = L(A)$ gilt.

c) Sei Σ ein Alphabet. Dann bezeichne DFA_Σ die Klasse der von endlichen Automaten akzeptierten Sprachen über Σ , und REG_Σ bezeichne die Klasse der regulären Sprachen über Σ . (*DFA* ist eine Abkürzung für *Deterministic Finite Automaton*). Wegen b) gilt: $DFA_\Sigma = REG_\Sigma$.

d) Zwei endliche Automaten A und A' heißen äquivalent, falls sie dieselbe Sprache akzeptieren, d.h. falls $L(A) = L(A')$ ist. \square

Nach der obigen Bemerkung über die Verwandtschaft von δ^* und \vdash^* können wir in a) $L(A)$ auch definieren durch:

$$L(A) = \{w \in \Sigma^* \mid \delta^*(s_0, w) \in F\}$$

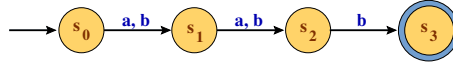


Bild 2.8: Zustandsdiagramm für die Dreiergruppen.

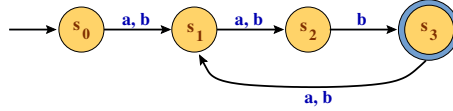


Bild 2.9: Zustandsdiagramm für Folgen von Dreiergruppen.

Beispiel 2.5. a) $L(A_{\text{Eintritt}})$, die Sprache des Eintrittsautomaten, hatten wir bereits am Ende von Abschnitt 2.1.1 angegeben.

b) Wir wollen zur Sprache

$$L_{3b} = \{w \in \{a, b\}^* \mid w = w_1 \dots w_n, w_i \in \{a, b\}, w_{3i} = b, 1 \leq i \leq n, n \geq 0\}$$

einen endlichen Automaten A_{3b} konstruieren, der L_{3b} akzeptiert, d.h. für den $L_{3b} = L(A_{3b})$ gilt. Dazu schauen wir uns zunächst die Bestandteile und die Struktur der Wörter von L an:

- (1) L_{3b} enthält alle Wörter über $\{a, b\}$, bei denen jeder dritte Buchstabe, d.h. deren dritter, sechster, neunter usw. Buchstabe b ist.
- (2) Wir können ein Wort aus L_{3b} also in Dreiergruppen von Buchstaben einteilen, deren jeweils letzter Buchstabe ein b sein muss, die beiden ersten Buchstaben können jeweils a oder b sein.
- (3) Die letzte Gruppe muss nicht aus drei Buchstaben bestehen, da die Gesamtlänge des Wortes kein Vielfaches von drei sein muss.

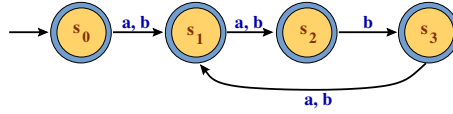
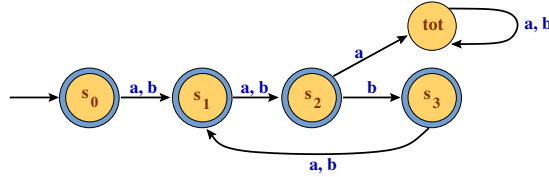
Diese Analyse führt zu folgenden Konstruktionsüberlegungen (d.h. zur Synthese des gesuchten Automaten):

1. Gemäß Beobachtung (2) wird eine Dreiergruppe, deren letzter Buchstabe nicht a ist, von dem Automaten in Bild 2.8 erkannt.
2. Nach dem Akzeptieren einer Dreiergruppe beginnt eine neue. Dazu führen wir einen Zustandsübergang von s_3 nach s_1 ein (siehe Bild 2.9).
3. Beobachtung (3) besagt, dass ein Wort aus L_{3b} jede Länge haben kann (auch die Längen 0, 1 oder 2, denn der dritte Buchstabe dieser Wörter ist nicht a). Wenn alle Zustände Endzustände sind, akzeptiert der Automat Wörter jeder Länge.

Wir erhalten also als Ergebnis unserer Konstruktion den Automaten

$$A_{3b} = (\{a, b\}, \{s_0, s_1, s_2, s_3\}, \delta, s_0, \{s_0, s_1, s_2, s_3\})$$

wobei δ durch das Zustandsüberführungsdiagramm in Bild 2.10 definiert ist. Es gilt offensichtlich: $L_{3b} = L(A_{3b})$. \square

Bild 2.10: Zustandsdiagramm für A_{3b} .Bild 2.11: Zustandsdiagramm für A_{3b}^{total} .

2.1.5 Vollständige Automaten

Die Definition endlicher Automaten (Definition 2.1) erfordert nicht, dass die Zustandsüberführung total definiert sein muss, d.h. δ muss nicht für alle Paare von Zuständen und Eingabebuchstaben definiert sein. So ist zum Beispiel δ aus obigem Beispiel 2.5 b) nicht total definiert, denn für die Eingabe a ist im Zustand s_2 kein Folgezustand definiert. Wir nennen einen endlichen Automaten $A = (\Sigma, S, \delta, s_0, F)$ *vollständig*, falls δ total definiert ist: $\text{Def}(\delta) = S \times \Sigma$.

Wie lässt sich eine *Vervollständigung* des Automaten A_{3b} aus Beispiel 2.5 b) erreichen? Wir führen einen neuen Zustand *tot* ein („toter“ Zustand), der Folgezustand des fehlenden Übergangs vom Zustand s_2 bei Eingabe a wird. Für den neuen Zustand müssen wir ebenfalls alle prinzipiell möglichen Zustandsübergänge definieren, sonst wäre der neue Automat nicht vollständig. Wir nehmen als Folgezustand für *tot* für alle Eingaben natürlich *tot* selbst. Durch diese Transformation erhalten wir den Automaten (siehe Bild 2.11).

$$A_{3b}^{total} = (\{a, b\}, \{s_0, s_1, s_2, s_3, tot\}, \delta_{total}, s_0, \{s_0, s_1, s_2, s_3\})$$

Es gilt offensichtlich, dass A_{3b} und A_{3b}^{total} äquivalent sind, sie akzeptieren dieselbe Sprache.

Die beispielhaft durchgeführte Transformation eines nicht vollständigen in einen vollständigen Automaten lässt sich offensichtlich verallgemeinern: Jeder Automat lässt sich in einen äquivalenten vollständigen Automaten transformieren. Wir geben dazu eine allgemeine Transformation an: Sei $A = (\Sigma, S, \delta, s_0, F)$ ein beliebiger endlicher Automat. Wir konstruieren einen äquivalenten vollständigen Automaten A_{total} zu A wie folgt:

$$A_{total} = (\Sigma, S \cup \{tot\}, \delta_{total}, s_0, F), \quad tot \notin S$$

wobei δ_{total} definiert ist durch:

$$\delta_{total}(s, a) = \begin{cases} \delta(s, a), & \text{falls } \delta \text{ auf } (s, a) \text{ definiert ist} \\ tot, & \text{sonst} \end{cases}$$

tot ist ein neuer Zustand ($tot \notin S$). Zu ihm führen alle Zustandsübergänge, die in A nicht definiert sind. Gelangt A_{total} bei der Abarbeitung eines Eingabewortes einmal in den Zustand tot , bleibt er in diesem bis zum Ende der Verarbeitung, denn es gilt $\delta_{total}(tot, a) = tot$ für alle $a \in \Sigma$.

Offensichtlich gilt für das Ergebnis dieser allgemeinen Transformation: A_{total} ist vollständig und äquivalent zu A .

2.1.6 Zusammenfassung

Endliche Automaten modellieren ein Black-Box-Modell: In Abhängigkeit ihres aktuellen Zustandes und einer Eingabe gehen sie gemäß einer festgelegten, deterministischen Zustandsüberführung in einen neuen Zustand über. Die Abarbeitung eines Eingabewortes, einer endlichen Folge von Eingabesymbolen, geschieht durch einen Prozess, der durch eine Folge von Konfigurationsübergängen formal beschrieben werden kann. Ist der Automat nach vollständiger Abarbeitung des Eingabewortes in einem Endzustand, dann akzeptiert der Automat dieses Wort. Kann er ein Eingabewort nicht komplett abarbeiten, oder ist er nach seiner Abarbeitung nicht in einem Endzustand, akzeptiert er das Wort nicht. Die Menge aller von einem endlichen Automaten akzeptierten Wörter stellt die von ihm akzeptierte Sprache dar. Von endlichen Automaten akzeptierte Sprachen heißen reguläre Sprachen. Jeder endliche Automat kann zu einem äquivalenten totalen transformiert werden.

2.2 Nichtdeterministische endliche Automaten

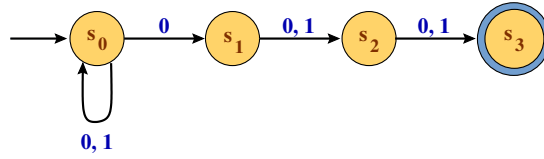
Die im vorherigen Abschnitt betrachteten Automaten sind *deterministisch*, denn in jedem Zustand gibt es für jedes Eingabesymbol höchstens einen (in einem vollständigen Automaten genau einen) Folgezustand. In diesem Abschnitt betrachten wir demgegenüber *nichtdeterministische* endliche Automaten, in denen es zu einem Zustand und einem Eingabesymbol mehrere Folgezustände geben kann, und wir untersuchen, wie sich diese Änderung auf die „Mächtigkeit“ auswirkt: Akzeptieren nichtdeterministische endliche Automaten mehr oder weniger oder andere Sprachen als deterministische? Es wird sich zeigen, dass die Klasse der akzeptierten Sprachen für beide Automatentypen identisch ist; wir werden in nachfolgenden Kapiteln sehen, dass dies bei anderen Automatentypen bzw. Sprachklassen nicht der Fall ist.

2.2.1 Definitionen

Wir betrachten zunächst ein Beispiel: Sei L_1 die Sprache, die alle Bitfolgen enthält, deren drittletztes Bit eine 0 ist:

$$L_1 = \{w \in \{0, 1\}^* \mid w = u0v, u \in \{0, 1\}^* v \in \{00, 01, 10, 11\}\}$$

Bei dem Versuch, einen endlichen Automaten zu konstruieren, der L_1 akzeptiert, stößt man auf das folgende Problem: Falls der Automat eine 0 liest, müsste er wissen, ob es das drittletzte Bit des Eingabewortes ist oder nicht. Falls ja, müsste er in einen Zustand

Bild 2.12: Zustandsdiagramm für A_1 .

verzweigen, von dem aus die beiden restlichen Bits akzeptiert würden, falls nein, in einen Zustand verzweigen, von dem aus weiter Nullen und Einsen akzeptiert würden. Für den Zustand, in dem eine 0 gelesen wird, müssten also zwei Folgezustände existieren. Das Zustandüberführungsdiagramm in Bild 2.12 stellt einen nichtdeterministischen endlichen Automaten A_1 dar, der die Sprache L_1 akzeptiert. Die Zustandsübergänge bilden nun keine Funktion mehr, sondern eine Relation, da der Zustand s_0 für die Eingabe 0 zwei Folgezustände hat, nämlich s_0 und s_1 .

Definition 2.3. Ein *nichtdeterministischer endlicher Automat* $A = (\Sigma, S, \delta, S_0, F)$ besteht aus dem Eingabealphabet Σ , der Zustandsmenge S , der Menge der Startzustände $S_0 \subseteq S$, der Menge der Endzustände $F \subseteq S$ und der Zustandsüberführungsrelation $\delta \subseteq S \times \Sigma \times S$. \square

δ ist eine Menge von Tripeln

(Zustand, Eingabesymbol, Folgezustand)

Das Zustandsdiagramm in Bild 2.12 stellt den nichtdeterministischen Automaten

$$A_1 = (\{0, 1\}, \{s_0, s_1, s_2, s_3\}, \delta, \{s_0\}, \{s_3\})$$

mit

$$\delta = \{(s_0, 0, s_0), (s_0, 0, s_1), (s_0, 1, s_0), (s_1, 0, s_2), (s_1, 1, s_2), (s_2, 0, s_3), (s_2, 1, s_3)\}$$

dar. Alternativ zur Mengendarstellung kann δ analog zum deterministischen Fall auch als Zustandstafel notiert werden:

δ	s_0	s_1	s_2	s_3
0	$\{s_0, s_1\}$	$\{s_2\}$	$\{s_3\}$	-
1	$\{s_0\}$	$\{s_2\}$	$\{s_3\}$	-

Wenn man die Folgezustände eines Zustandes für ein Eingabesymbol zusammenfasst zu einer Menge, dann lässt sich die Zustandsüberführung wieder als Funktion, und zwar als mengenwertige Funktion, definieren:

$$\delta : S \times \Sigma \rightarrow 2^S$$

Die Zustandsüberführung unseres Beispiels wird dann so notiert:

$$\begin{aligned}
 \delta(s_0, 0) &= \{s_0, s_1\} \\
 \delta(s_1, 0) &= \{s_2\} \\
 \delta(s_2, 0) &= \{s_3\} \\
 \delta(s_3, 0) &= \{\} \\
 \delta(s_0, 1) &= \{s_0\} \\
 \delta(s_1, 1) &= \{s_2\} \\
 \delta(s_2, 1) &= \{s_3\} \\
 \delta(s_3, 1) &= \{\}
 \end{aligned}$$

Eine Konfiguration $k = (s, v)$ eines nichtdeterministischen Automaten beschreibt wie im deterministischen Fall den aktuellen Zustand s sowie das noch zu verarbeitende Suffix v des Eingabewortes. Ein Konfigurationsübergang von $k = (s, aw)$ zu $k' = (s', w)$, notiert durch

$$(s, aw) \vdash (s', w)$$

kann stattfinden, falls $(s, a, s') \in \delta$, $a \in \Sigma$, $w \in \Sigma^*$. Die von einem nichtdeterministischen Automaten A akzeptierte Sprache $L(A)$ wird analog zum deterministischen Fall definiert durch:

$$L(A) = \{w \in \Sigma^* \mid (s_0, w) \vdash^* (s, \varepsilon), s_0 \in S_0, s \in F\}$$

Wir wollen mit NFA_Σ die Klasse der von nichtdeterministischen endlichen Automaten akzeptierten Sprachen über dem Alphabet Σ bezeichnen, wobei NFA eine Abkürzung für *Nondeterministic Finite Automata* ist.

Wir wollen nun die Konfigurationsübergänge beim Akzeptieren von Wörtern betrachten. Als Beispiel nehmen wir das Wort 1100011 aus der obigen Sprache L_1 der Bitfolgen, deren drittletztes Bit 0 ist. Wird 1100011 von dem Automaten A_1 (siehe Bild 2.12), den wir zu L_1 konstruiert haben, akzeptiert, d.h. gilt $1100011 \in L(A_1)$? Dazu müssen wir laut Definition überprüfen, ob $(s_0, 1100011) \vdash^* (s_3, \varepsilon)$ gilt:

$$\begin{array}{ll}
 (s_0, 1100011) \vdash (s_0, 100011) & \text{da } (s_0, 1, s_0) \in \delta \\
 \vdash (s_0, 00011) & \text{da } (s_0, 1, s_0) \in \delta \\
 \vdash (s_0, 0011) & \text{da } (s_0, 0, s_0) \in \delta \\
 \vdash (s_0, 011) & \text{da } (s_0, 0, s_0) \in \delta \\
 \vdash (s_0, 11) & \text{da } (s_0, 0, s_0) \in \delta \\
 \vdash (s_0, 1) & \text{da } (s_0, 1, s_0) \in \delta \\
 \vdash (s_0, \varepsilon) & \text{da } (s_0, 1, s_0) \in \delta
 \end{array} \tag{2.1}$$

Das Eingabewort ist abgearbeitet, es ist aber kein Endzustand erreicht. Trotzdem ist die Frage „ $1100011 \in L(A_1)$?“ noch nicht beantwortet, da wir in (2.1) eine nichtdeterministische Wahl getroffen haben, die zur Nichtakzeptanz führt. Wir führen ein so

genanntes *Backtracking* durch, d.h. wir kehren an die letzte Entscheidungsstelle (2.1) zurück und wählen dort (2.2) eine andere Möglichkeit:

$$\begin{array}{ll}
 (s_0, 1100011) \vdash (s_0, 100011) & \text{da } (s_0, 1, s_0) \in \delta \\
 \vdash (s_0, 00011) & \text{da } (s_0, 1, s_0) \in \delta \\
 \vdash (s_0, 0011) & \text{da } (s_0, 0, s_0) \in \delta \\
 \vdash (s_0, 011) & \text{da } (s_0, 0, s_0) \in \delta \\
 \vdash (s_1, 11) & \text{da } (s_0, 0, s_1) \in \delta \\
 \vdash (s_2, 1) & \text{da } (s_1, 1, s_2) \in \delta \\
 \vdash (s_3, \varepsilon), & \text{da } (s_2, 1, s_3) \in \delta
 \end{array} \quad (2.2)$$

Das Wort ist wiederum abgearbeitet, und jetzt ist ein Endzustand erreicht. Es gibt also eine Konfigurationsfolge, die zu einem akzeptierenden Zustand führt, d.h. 1100011 wird von A_1 akzeptiert.

Um festzustellen, ob ein Wort von einem nichtdeterministischen Automaten akzeptiert wird, muss man mindestens eine Konfigurationsfolge finden, bei der das Wort abgearbeitet und ein Endzustand erreicht wird. Nur dann, wenn *alle* Konfigurationsfolgen, bei denen das Wort abgearbeitet wird, *nicht* in einem Endzustand enden, wird das Wort nicht akzeptiert. Im Unterschied dazu reicht im deterministischen Fall in jedem Fall der Test *einer* Konfigurationsfolge, um die Frage nach der Akzeptanz zu beantworten.

Im deterministischen Fall haben wir zur Klärung der Akzeptanz außer dem Durchlaufen der Konfigurationsfolge

$$(s_0, w) \vdash^* (s, \varepsilon)$$

auch die Möglichkeit, $\delta^*(s_0, w)$ zu berechnen. Dazu haben wir $\delta : S \times \Sigma \rightarrow S$ zu $\delta^* : S \times \Sigma^* \rightarrow S$ erweitert. Wir wollen nun analog dazu im nichtdeterministischen Fall die mengenwertige Zustandsüberführung

$$\delta : S \times \Sigma \rightarrow 2^S$$

auf

$$\delta^* : 2^S \times \Sigma^* \rightarrow 2^S$$

erweitern. Dabei definieren wir δ^* so, dass für ein Wort w alle Zustände bei der Berechnung von $\delta^*(R, w)$ für eine Zustandsmenge $R \subseteq S$ gleichzeitig (parallel) bestimmt werden, die die durch Backtracking entstehenden Konfigurationsfolgen durchlaufen:

$$\delta^*(R, \varepsilon) = R \text{ für alle } R \subseteq S$$

und

$$\delta^*(R, aw) = \delta^* \left(\bigcup_{s \in R} \delta(s, a), w \right) \text{ für } a \in \Sigma \text{ und } w \in \Sigma^*$$

Befinden wir uns aktuell bei der Berechnung von δ^* bei den Zuständen der Menge R , und ist der nächste Buchstabe ein a , bestimmen wir für jeden Zustand $s \in R$ alle möglichen Nachfolgezustände $\delta(s, a)$. Diese bilden die neue aktuelle Zustandsmenge, die Ausgangspunkt für die weitere Berechnung für den noch anstehenden Eingaberest w ist. Ist das Wort abgearbeitet ($w = \varepsilon$), ist die erreichte Zustandsmenge das Endergebnis. Enthält sie mindestens einen Endzustand, gehört das Eingabewort zur Sprache des Automaten, denn dann gibt es mindestens eine akzeptierende Konfigurationsfolge. Enthält die berechnete Zustandsmenge keinen Endzustand, dann gehört das Wort nicht zur Sprache.

Wir wollen mit obigem Beispielwort 1100011 die parallele Berechnung der Folge der Zustandsmengen für A_1 durchführen:

$$\begin{aligned}
& \delta^*(\{s_0\}, 1100011) \\
&= \delta^*\left(\bigcup_{s \in \{s_0\}} \delta(s, 1), 100011\right) = \delta^*(\delta(s_0, 1), 100011) \\
&= \delta^*(\{s_0\}, 100011) \\
&= \delta^*\left(\bigcup_{s \in \{s_0\}} \delta(s, 1), 00011\right) = \delta^*(\delta(s_0, 1), 00011) \\
&= \delta^*(\{s_0\}, 00011) \\
&= \delta^*\left(\bigcup_{s \in \{s_0\}} \delta(s, 0), 0011\right) = \delta^*(\delta(s_0, 0), 0011) \\
&= \delta^*(\{s_0, s_1\}, 0011) \\
&= \delta^*\left(\bigcup_{s \in \{s_0, s_1\}} \delta(s, 0), 011\right) = \delta^*(\delta(s_0, 0) \cup \delta(s_1, 0), 011) \\
&= \delta^*(\{s_0, s_1\} \cup \{s_2\}, 011) \\
&= \delta^*(\{s_0, s_1, s_2\}, 011)
\end{aligned}$$

$$\begin{aligned}
&= \delta^* \left(\bigcup_{s \in \{s_0, s_1, s_2\}} \delta(s, 0), 11 \right) = \delta^*(\delta(s_0, 0) \cup \delta(s_1, 0) \cup \delta(s_2, 0), 11) \\
&= \delta^*(\{s_0, s_1\} \cup \{s_2\} \cup \{s_3\}, 11) \\
&= \delta^*(\{s_0, s_1, s_2, s_3\}, 11) \\
&= \delta^* \left(\bigcup_{s \in \{s_0, s_1, s_2, s_3\}} \delta(s, 1), 1 \right) \\
&= \delta^*(\delta(s_0, 1) \cup \delta(s_1, 1) \cup \delta(s_2, 1) \cup \delta(s_3, 1), 1) \\
&= \delta^*(\{s_0\} \cup \{s_2\} \cup \{s_3\} \cup \{\}, 1) \\
&= \delta^*(\{s_0, s_2, s_3\}, 1) \\
&= \delta^* \left(\bigcup_{s \in \{s_0, s_2, s_3\}} \delta(s, 1), \varepsilon \right) = \delta^*(\delta(s_0, 1) \cup \delta(s_2, 1) \cup \delta(s_3, 1), \varepsilon) \\
&= \delta^*(\{s_0\} \cup \{s_3\} \cup \{\}, \varepsilon) \\
&= \delta^*(\{s_0, s_3\}, \varepsilon) \\
&= \{s_0, s_3\}
\end{aligned}$$

Die berechnete Zustandsmenge enthält mit s_3 einen Endzustand, 1100011 wird also akzeptiert. Alle möglichen Konfigurationsfolgen, die bei der Verarbeitung der Eingabe 1100011 möglich sind, werden auf einmal, parallel, durchlaufen. Als Ergebnis werden alle Zustände berechnet, bei deren Erreichen das Eingabewort komplett abgearbeitet ist. Ist ein Endzustand darunter, wird das Wort akzeptiert, denn dann gibt es mindestens eine akzeptierende Konfigurationsfolge.

Die von einem nichtdeterministischen Automaten A akzeptierte Sprache $L(A)$ können wir somit auch mit Hilfe von δ^* definieren:

$$L(A) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \cap F \neq \emptyset\}$$

2.2.2 Äquivalenz von deterministischen und nichtdeterministischen endlichen Automaten

Nachdem wir für endliche Automaten zwei Varianten, die deterministische und die nichtdeterministische, eingeführt haben, stellt sich – zumindest aus theoretischer Sicht – die Frage nach ihrer Äquivalenz: Gilt $DFA_\Sigma = NFA_\Sigma$, d.h. akzeptieren beide Varianten dieselbe Klasse von Sprachen? Falls nein, gilt dann $DFA_\Sigma \subset NFA_\Sigma$ oder $NFA_\Sigma \subset DFA_\Sigma$, d.h. sind die nichtdeterministischen Automaten mächtiger als die deterministischen oder umgekehrt? Oder sind die Klassen verschieden, enthalten aber Sprachen, die beiden angehören: $DFA_\Sigma \cap NFA_\Sigma \neq \emptyset$, oder sind die Klassen disjunkt, d.h. es gibt keine Sprache, die beiden Klassen angehört: $DFA_\Sigma \cap NFA_\Sigma = \emptyset$.

Sehr schnell kann man einsehen, dass $DFA_\Sigma \subseteq NFA_\Sigma$ gelten muss, denn jeder deterministische Automat kann als nichtdeterministischer aufgefasst werden: In jedem Zustand gibt es für jedes Eingabesymbol höchstens einen Folgezustand. Determinismus kann als „Spezialfall“ von Nichtdeterminismus angesehen werden.

Formal können wir zu einem gegebenen deterministischen Automaten $A = (\Sigma, S, \delta, s_0, F)$ wie folgt einen äquivalenten nichtdeterministischen A_{nd} konstruieren:

$$A_{nd} = (\Sigma, S, \delta_{nd}, \{s_0\}, F)$$

mit

$$\delta_{nd}(s, a) = \begin{cases} \{s'\}, & \text{falls } \delta(s, a) = s' \\ \emptyset, & \text{sonst} \end{cases}$$

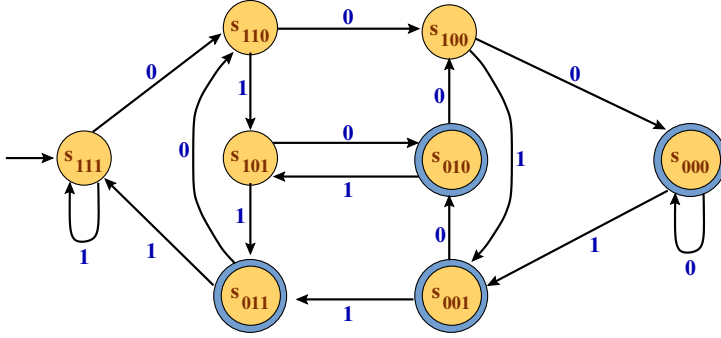
Es ist offensichtlich, dass $L(A_{nd}) = L(A)$ gilt, denn es gilt $\delta_{nd}^*(\{s_0\}, w) \cap F \neq \emptyset$ genau dann, wenn $\delta^*(s_0, w) \in F$ ist.

Wie steht es nun mit der Umkehrung? Gilt auch $NFA_\Sigma \subseteq DFA_\Sigma$, oder gibt es eine Sprache L über Σ mit $L \in NFA_\Sigma$ und $L \notin DFA_\Sigma$?

Betrachten wir zunächst noch einmal die eingangs von Abschnitt 2.2.1 als Beispiel verwendete Sprache L_1 aller Bitfolgen, deren drittletztes Bit 0 ist. Wir haben schnell einen nichtdeterministischen Automaten gefunden, der diese Sprache akzeptiert. Gibt es auch einen deterministischen? Das Problem ist, dass wir, falls wir bei der Verarbeitung einer Bitfolge einer 0 begegnen, erst nach den darauf folgenden beiden Bits wissen, ob die 0 das drittletzte Bit war. In dieser Überlegung liegt bereits der Lösungsansatz: Wir merken uns in den Zuständen des zu konstruierenden Automaten die jeweils zuletzt gelesenen drei Bits. Da es acht Dreiergruppen von Bits gibt, benötigen wir acht Zustände. Das nächste gelesene Bit führt in den entsprechenden Folgezustand über. Zum Merken der Dreiergruppe abc , $a, b, c \in \{0, 1\}$, benutzen wir den Zustand s_{abc} . Alle Zustände s_{0bc} sind Endzustände (das drittletzte Bit war eine 0), s_{111} ist Startzustand (unter den letzten drei Bits war keine Null).

Der folgende deterministische Automat A_{1d} (siehe Bild 2.13) akzeptiert genau die Sprache L_1 :

$$A_{1d} = (\{0, 1\}, \{s_{000}, s_{001}, s_{010}, s_{011}, s_{100}, s_{101}, s_{110}, s_{111}\}, \delta, s_{111}, \{s_{000}, s_{001}, s_{010}, s_{011}\})$$

Bild 2.13: Zustandsdiagramm von A_{1_d} .

Für die Sprache der Bitfolgen, deren drittletztes Bit eine 0 ist, haben wir also auch einen deterministischen Automaten gefunden. Es gilt sogar allgemein: Zu jedem nicht-deterministischen Automaten existiert ein äquivalenter deterministischer. Der folgende Satz fasst die Äquivalenz deterministischer und nichtdeterministischer endlicher Automaten zusammen; sein Beweis ist konstruktiv insofern, als er eine Methode zur Herleitung eines deterministischen Automaten aus bzw. zu einem gegebenen nichtdeterministischen enthält.

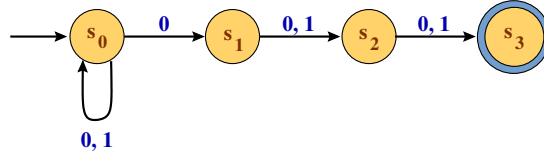
Satz 2.2. Sei Σ ein Alphabet. Dann gilt: $DFA_{\Sigma} = NFA_{\Sigma}$.

Beweis Dass $DFA_{\Sigma} \subseteq NFA_{\Sigma}$ gilt, haben wir bereits oben begründet. Dort haben wir auch ein allgemeines Verfahren zur Transformation eines deterministischen in einen äquivalenten nichtdeterministischen Automaten angegeben.

Wir zeigen nun $NFA_{\Sigma} \subseteq DFA_{\Sigma}$, indem wir ein allgemeines Verfahren angeben, mit dem zu einem nichtdeterministischen Automaten ein äquivalenter deterministischer konstruiert werden kann: Sei also $A = (\Sigma, S, \delta, S_0, F)$ ein nichtdeterministischer Automat und die Zustandsüberführung als mengenwertige Funktion gegeben: $\delta : S \times \Sigma \rightarrow 2^S$.

Die Grundidee ist, dass alle nichtdeterministisch möglichen, mit Backtracking durchlaufbaren, quasi parallelen Konfigurationsübergänge zu einer Berechnung zusammengefasst werden (siehe Abschnitt 2.2.1, Definition und Beispiel zu δ^*). Wir fassen diese Übergänge jeweils zu einem Übergang zusammen. Die Zustände bestehen dann aus Mengen von Zuständen. Die Menge aller möglichen Zustände des deterministischen Automaten ist also die Menge aller Teilmengen von S , d.h. die Potenzmenge 2^S von S ; wegen dieser Überlegung wird die Konstruktion auch *Potenzmengenkonstruktion* genannt. Startzustand des deterministischen Automaten wird die Startzustandsmenge von A , und Endzustände werden die Teilmengen von S , die mindestens einen Endzustand von A enthalten. Insgesamt ergibt sich der deterministische Automat A_d zu A wie folgt:

$$A_d = (\Sigma, 2^S, \delta_d, s_0^d, F_d)$$

Bild 2.14: Zustandsdiagramm von A .

Dabei ist:

$$\delta_d(R, a) = \bigcup_{s \in R} \delta(s, a) \quad (2.3)$$

$$s_0^d = S_0$$

$$F_d = \{R \subseteq S \mid R \cap F \neq \emptyset\}$$

Das angegebene Verfahren ist korrekt: Es liefert zu jedem nichtdeterministischen Automaten A einen äquivalenten deterministischen Automaten A_d . Einen Beweis, dass $L(A) = L(A_d)$ gilt, geben wir hier nicht an, sondern verweisen auf die einschlägige Literatur. \square

Beispiel 2.6. Wir wollen die Potenzmengenkonstruktion am Beispiel des Automaten durchführen, der die Sprache der Bitfolgen, deren drittletztes Bit eine 0 ist, akzeptiert:

$$A = (\{0, 1\}, \{s_0, s_1, s_2, s_3\}, \delta, \{s_0\}, \{s_3\})$$

wobei δ durch das Zustandsdiagramm in Bild 2.14 gegeben ist. Für A_d geben wir eine Konstruktion gemäß dem vorgestellten allgemeinen Verfahren an. Dabei wird δ_d schrittweise gemäß (2.3) festgelegt:

1. Wir beginnen mit dem Startzustand $s_0^d = \{s_0\}$:

$$\begin{aligned} \delta_d(\{s_0\}, 0) &= \bigcup_{s \in \{s_0\}} \delta(s, 0) = \delta(s_0, 0) = \{s_0, s_1\} \\ \delta_d(\{s_0\}, 1) &= \bigcup_{s \in \{s_0\}} \delta(s, 1) = \delta(s_0, 1) = \{s_0\} \end{aligned}$$

Wir erhalten zwei Ergebniszustände $\{s_0, s_1\}$ und $\{s_0\}$, wovon der erste neu ist.

2. Wir machen mit dem in 1. erhaltenen neuen Zustand weiter:

$$\begin{aligned}
 \delta_d(\{s_0, s_1\}, 0) &= \bigcup_{s \in \{s_0, s_1\}} \delta(s, 0) = \delta(s_0, 0) \cup \delta(s_1, 0) = \{s_0, s_1\} \cup \{s_2\} \\
 &= \{s_0, s_1, s_2\} \\
 \delta_d(\{s_0, s_1\}, 1) &= \bigcup_{s \in \{s_0, s_1\}} \delta(s, 1) = \delta(s_0, 1) \cup \delta(s_1, 1) = \{s_0\} \cup \{s_2\} \\
 &= \{s_0, s_2\}
 \end{aligned}$$

Wir erhalten zwei neue Zustände: $\{s_0, s_1, s_2\}$ und $\{s_0, s_2\}$.

3. Wir machen mit diesen beiden Zuständen weiter:

$$\begin{aligned}
 \delta_d(\{s_0, s_1, s_2\}, 0) &= \bigcup_{s \in \{s_0, s_1, s_2\}} \delta(s, 0) \\
 &= \delta(s_0, 0) \cup \delta(s_1, 0) \cup \delta(s_2, 0) = \{s_0, s_1\} \cup \{s_2\} \cup \{s_3\} \\
 &= \{s_0, s_1, s_2, s_3\} \\
 \delta_d(\{s_0, s_1, s_2\}, 1) &= \bigcup_{s \in \{s_0, s_1, s_2\}} \delta(s, 1) = \delta(s_0, 1) \cup \delta(s_1, 1) \cup \delta(s_2, 1) \\
 &= \{s_0\} \cup \{s_2\} \cup \{s_3\} = \{s_0, s_2, s_3\} \\
 \delta_d(\{s_0, s_2\}, 0) &= \bigcup_{s \in \{s_0, s_2\}} \delta(s, 0) = \delta(s_0, 0) \cup \delta(s_2, 0) = \{s_0, s_1\} \cup \{s_3\} \\
 &= \{s_0, s_1, s_3\} \\
 \delta_d(\{s_0, s_2\}, 1) &= \bigcup_{s \in \{s_0, s_2\}} \delta(s, 1) = \delta(s_0, 1) \cup \delta(s_2, 1) = \{s_0\} \cup \{s_3\} \\
 &= \{s_0, s_3\}
 \end{aligned}$$

4. Alle im 3. Schritt berechneten Zustände sind neue Zustände. Wir bestimmen in der entstandenen Reihenfolge ihre Folgezustände.

$$\begin{aligned}
 \delta_d(\{s_0, s_1, s_2, s_3\}, 0) &= \bigcup_{s \in \{s_0, s_1, s_2, s_3\}} \delta(s, 0) \\
 &= \delta(s_0, 0) \cup \delta(s_1, 0) \cup \delta(s_2, 0) \cup \delta(s_3, 0) \\
 &= \{s_0, s_1\} \cup \{s_2\} \cup \{s_3\} \cup \{\} = \{s_0, s_1, s_2, s_3\} \\
 \delta_d(\{s_0, s_1, s_2, s_3\}, 1) &= \bigcup_{s \in \{s_0, s_1, s_2, s_3\}} \delta(s, 1) \\
 &= \delta(s_0, 1) \cup \delta(s_1, 1) \cup \delta(s_2, 1) \cup \delta(s_3, 1) \\
 &= \{s_0\} \cup \{s_2\} \cup \{s_3\} \cup \{\} \\
 &= \{s_0, s_2, s_3\}
 \end{aligned}$$

$$\begin{aligned}
\delta_d(\{s_0, s_2, s_3\}, 0) &= \bigcup_{s \in \{s_0, s_2, s_3\}} \delta(s, 0) = \delta(s_0, 0) \cup \delta(s_2, 0) \cup \delta(s_3, 0) \\
&= \{s_0, s_1\} \cup \{s_3\} \cup \{\} = \{s_0, s_1, s_3\} \\
\delta_d(\{s_0, s_2, s_3\}, 1) &= \bigcup_{s \in \{s_0, s_2, s_3\}} \delta(s, 1) \\
&= \delta(s_0, 1) \cup \delta(s_2, 1) \cup \delta(s_3, 1) = \{s_0\} \cup \{s_3\} \cup \{\} \\
&= \{s_0, s_3\} \\
\delta_d(\{s_0, s_1, s_3\}, 0) &= \bigcup_{s \in \{s_0, s_1, s_3\}} \delta(s, 0) \\
&= \delta(s_0, 0) \cup \delta(s_1, 0) \cup \delta(s_3, 0) = \{s_0, s_1\} \cup \{s_2\} \cup \{\} \\
&= \{s_0, s_1, s_2\} \\
\delta_d(\{s_0, s_1, s_3\}, 1) &= \bigcup_{s \in \{s_0, s_1, s_3\}} \delta(s, 1) \\
&= \delta(s_0, 1) \cup \delta(s_1, 1) \cup \delta(s_3, 1) = \{s_0\} \cup \{s_2\} \cup \{\} \\
&= \{s_0, s_2\} \\
\delta_d(\{s_0, s_3\}, 0) &= \bigcup_{s \in \{s_0, s_3\}} \delta(s, 0) = \delta(s_0, 0) \cup \delta(s_3, 0) = \{s_0, s_1\} \cup \{\} \\
&= \{s_0, s_1\} \\
\delta_d(\{s_0, s_3\}, 1) &= \bigcup_{s \in \{s_0, s_3\}} \delta(s, 1) = \delta(s_0, 1) \cup \delta(s_3, 1) = \{s_0\} \cup \{\} \\
&= \{s_0\}
\end{aligned}$$

5. Es sind keine neuen Zustände entstanden. Die Konstruktion ist fertig. Von den sechzehn Elementen von $2^{\{s_0, s_1, s_2, s_3\}}$ werden tatsächlich nur acht benötigt. Es ergibt sich der zu A äquivalente deterministische Automat

$$\begin{aligned}
A_d = (&\{0, 1\}, \\
&\{\{s_0\}, \{s_0, s_1\}, \{s_0, s_1, s_2\}, \{s_0, s_2\}, \\
&\{s_0, s_1, s_2, s_3\}, \{s_0, s_2, s_3\}, \{s_0, s_1, s_3\}, \{s_0, s_3\}\}, \\
&\delta_d, \{s_0\}, \\
&\{\{s_0, s_1, s_2, s_3\}, \{s_0, s_2, s_3\}, \{s_0, s_1, s_3\}, \{s_0, s_3\}\})
\end{aligned}$$

Man kann zeigen, dass dieser Automat isomorph zum deterministischen Automaten A_{1_d} ist, den wir für die Sprache der Bitfolgen, deren drittletztes Bit 0 ist, konstruiert haben (siehe Bild 2.13). Isomorph bedeutet, dass die Automaten identisch sind bis auf die Bezeichnung der Zustände. Isomorphie von Automaten betrachten wir im Abschnitt 2.5 noch näher. \square

Wie man an dem obigen Beispiel sieht, kann die ausführliche Konstruktion von δ_d eine mühsame Schreibarbeit bedeuten. Man kann die Konstruktion pragmatischer mithilfe

δ_d	0	1
$\{s_0\}$	$\{s_0, s_1\}$	$\{s_0\}$
$\{s_0, s_1\}$	$\{s_0, s_1, s_2\}$	$\{s_0, s_2\}$
$\{s_0, s_1, s_2\}$	$\{s_0, s_1, s_2, s_3\}$	$\{s_0, s_2, s_3\}$
$\{s_0, s_2\}$	$\{s_0, s_1, s_3\}$	$\{s_0, s_3\}$
$\{s_0, s_1, s_2, s_3\}$	$\{s_0, s_1, s_2, s_3\}$	$\{s_0, s_2, s_3\}$
$\{s_0, s_2, s_3\}$	$\{s_0, s_1, s_3\}$	$\{s_0, s_3\}$
$\{s_0, s_1, s_3\}$	$s_0, s_1, s_2\}$	$\{s_0, s_2\}$
$\{s_0, s_3\}$	$\{s_0, s_1\}$	$\{s_0\}$

Tabelle 2.1: Tabellendarstellung für die Konstruktion des Automaten A_d .

einer Zustandstabelle darstellen, wobei die i -te Zeile der Tabelle den i -ten Schritt der Konstruktion abbildet. Dabei schreibt man als Zeilenüberschrift die Zustandsmenge hin, zu der im i -ten Schritt die Folgezustandsmengen konstruiert werden sollen, und als Spaltenüberschriften die Eingabesymbole. Die Folgeszustandsmengen berechnet man dann gemäß (2.3) „im Kopf“ und trägt das Ergebnis an der entsprechenden Stelle in die Tabelle ein. Tabelle 2.1 stellt die Konstruktion aus Beispiel 2.6 in dieser Art und Weise dar.

2.2.3 Zusammenfassung

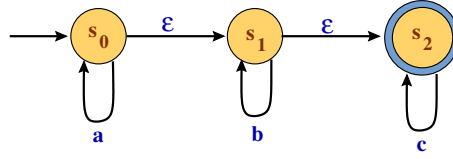
Nichtdeterministische endliche Automaten lassen in einem Zustand für eine Eingabe mehrere Folgezustände zu. Bei der sequentiellen Abarbeitung von Eingabewörtern zur Feststellung, ob sie akzeptiert werden oder nicht, kann Backtracking nötig sein. Die parallele Abarbeitung eines Wortes führt alle möglichen sequentiellen Konfigurationsfolgen gleichzeitig aus. Mithilfe der Potenzmengenkonstruktion können nichtdeterministische in äquivalente deterministische Automaten transformiert werden. Nichtdeterministische und deterministische endliche Automaten akzeptieren dieselbe Klasse von Sprachen, die Klasse der regulären Sprachen.

2.3 Endliche Automaten mit ε -Übergängen

In diesem Abschnitt betrachten wir eine weitere Variante endlicher Automaten: Es werden ε -Übergänge zugelassen, d.h. Zustandswechsel, ohne dass ein Buchstabe gelesen wird. Der Lesekopf bleibt bei einem solchen Übergang also stehen.

2.3.1 Definitionen

Dass die Möglichkeit von ε -Übergängen die Konstruktion eines Automaten erleichtern kann, soll folgendes Beispiel zeigen: Sei L_{abc} die Sprache, deren Wörter aus einer beliebig langen Folge von a 's, gefolgt von einer beliebig langen Folge von b 's, gefolgt von einer beliebig langen Folge von c 's besteht. Formal können wir diese Sprache wie

Bild 2.15: Zustandsdiagramm von A_{abc} .

folgt notieren:

$$L_{abc} = \{a^i b^j c^k \mid i, j, k \geq 0\}$$

Der Automat (siehe Bild 2.15)

$$A_{abc} = (\{a, b, c\}, \{s_0, s_1, s_2\}, \delta, \{s_0\}, \{s_2\})$$

akzeptiert L_{abc} . Im Startzustand s_0 werden die beliebig vielen a 's erkannt, in s_1 die b 's und in s_2 die c 's. Zwischen den Zuständen sind „Spontanübergänge“ möglich, bei denen kein Buchstabe akzeptiert wird. Insbesondere dadurch berücksichtigt dieses Zustandsübergangsdiagramm in einfacher Weise, dass ein Wort aus L_{abc} nicht notwendigerweise a 's, b 's und c 's enthalten muss.

Definition 2.4. Ein endlicher ε -Automat ist definiert durch $A = (\Sigma, S, \delta, S_0, F)$. Dabei sind Σ, S, S_0 und F wie bei nichtdeterministischen Automaten festgelegt, und δ ist eine Zustandsüberführungsrelation, die auch ε -Übergänge zulässt:

$$\delta \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S \quad \square$$

Eine Konfiguration $k = (s, w)$ zeigt wie bisher den aktuellen Stand der Bearbeitung eines Eingabewortes an: s ist der aktuelle Zustand und w das noch zu bearbeitende Suffix des Eingabewortes. Die Konfigurationsübergangsrelation \vdash muss allerdings die ε -Übergänge berücksichtigen:

$$(s, aw) \vdash (s', w) \text{ genau dann, wenn } (s, a, s') \in \delta, a \in \Sigma \cup \{\varepsilon\}, w \in \Sigma^*$$

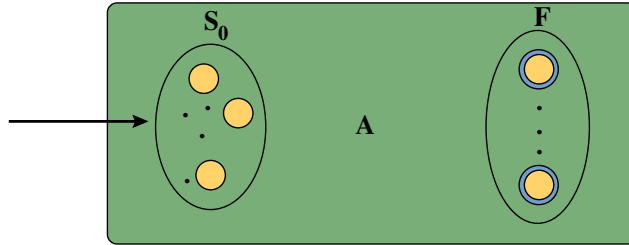
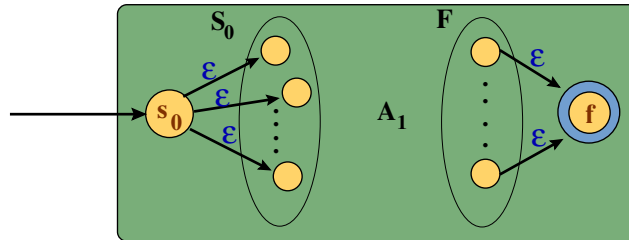
Die von einem ε -Automaten A akzeptierte Sprache $L(A)$ ist analog zum nichtdeterministischen Fall definiert durch:

$$L(A) = \{w \in \Sigma^* \mid (s_0, w) \vdash^* (s, \varepsilon), s_0 \in S_0, s \in F\}$$

Mit εFA_Σ bezeichnen wir die Klasse der Sprachen über Σ , die von ε -Automaten akzeptiert werden.

2.3.2 Äquivalenz von ε -Automaten zu nichtdeterministischen endlichen Automaten

Auch für die ε -Variante stellt sich die Frage ihrer Mächtigkeit im Vergleich zu den beiden anderen vorher betrachteten Varianten. Gilt $\varepsilon FA_\Sigma = NFA_\Sigma$ und damit $\varepsilon FA_\Sigma = DFA_\Sigma$? Der folgende Satz beantwortet diese Frage.

Bild 2.16: Schematische Darstellung eines endlichen Automaten A .Bild 2.17: Ergebnis der Transformation nach dem ersten Schritt: A_1 .

Satz 2.3. Sei Σ ein Alphabet. Dann gilt: $\varepsilon FA_\Sigma = NFA_\Sigma$.

Beweis Jeder nichtdeterministische Automat ist auch ein (spezieller) ε -Automat, nämlich ein ε -Automat ohne ε -Übergänge. Es gilt also: $NFA_\Sigma \subseteq \varepsilon FA_\Sigma$.

Um zu zeigen, dass auch $\varepsilon FA_\Sigma \subseteq NFA_\Sigma$ gilt, geben wir ein allgemeines Verfahren an, mit dem ein ε -Automat in einen äquivalenten nichtdeterministischen Automaten transformiert werden kann. Sei also $A = (\Sigma, S, \delta, S_0, F)$ ein ε -Automat. Die Transformation erfolgt in vier Schritten:

1. Wir transformieren A in einen Automaten mit genau einem Startzustand s_0 und mit genau einem Endzustand f , wobei s_0 und f zwei neue Zustände sind. Von s_0 führen wir zu jedem bisherigen Startzustand $s \in S_0$ einen ε -Übergang ein. Analog führen wir von jedem bisherigen Endzustand $s \in F$ einen ε -Übergang zum neuen Endzustand f ein.

Die Bilder 2.16 und 2.17 stellen diese Transformation schematisch dar. Formal erhalten wir als Zwischenergebnis den offensichtlich zu A äquivalenten Automaten

$$A_1 = (\Sigma, S_1, \delta_1, \{s_0\}, \{f\})$$

mit $S_1 = S \cup \{s_0, f\}$, $s_0 \notin S$, $f \notin S$ und

$$\delta_1 = \delta \cup \{(s_0, \varepsilon, s) \mid s \in S_0\} \cup \{(s, \varepsilon, f) \mid s \in F\}$$

Es werden also zunächst ε -Übergänge hinzugefügt.

2. Als Nächstes eliminieren wir alle ε -Zykel aus A_1 . Ein ε -Zykel ist eine Folge von Zuständen s_1, s_2, \dots, s_m , $m > 1$, so dass von s_1 nach s_2 , von s_2 nach s_3 , ..., von s_{m-1} nach s_m sowie von s_m zurück nach s_1 jeweils nur ein ε -Übergang führt. Alle diese Zustände nehmen wir aus S_1 heraus und fügen dafür einen neuen Zustand s_ε hinzu. Alle Übergänge, die vorher in die s_i , $1 \leq i \leq m$, geführt haben bzw. von ihnen ausgegangen sind, werden auf s_ε gerichtet bzw. von s_ε ausgeführt. Bild 2.18 stellt diese Transformation schematisch dar.

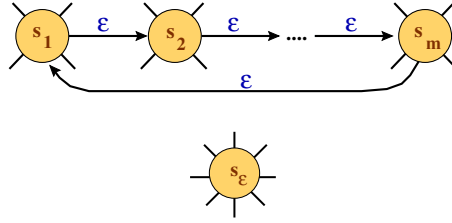


Bild 2.18: ε -Zykel und seine Reduktion auf einen Zustand.

Falls alle ε -Zykel eliminiert sind, löschen wir noch alle Übergänge (s, ε, s) , d. h. ε -Übergänge, die von einem Zustand in sich selbst führen. Das Ergebnis der bisherigen ε -Elimination sei der Automat A_2 .

3. In diesem Schritt fügen wir für alle weiteren ε -Übergänge einen „echten“ Übergang ein: Gibt es einen ε -Übergang von einem Zustand s' zu einem Zustand s , d. h. es gilt $(s', \varepsilon, s) \in \delta_2$, und geht von s ein a -Übergang zu einem Zustand t , d. h. es gilt $(s, a, t) \in \delta_2$, dann fügen wir in δ_2 den neuen Übergang (s', a, t) ein. Dies führen wir so lange durch, bis keine neuen Übergänge hinzukommen.

Die Bilder 2.19 und 2.20 veranschaulichen diese Transformation. Der resultierende Automat heiße A_3 .

4. Zum Schluss bestimmen wir die Endzustandsmenge und eliminieren alle noch verbliebenen ε -Übergänge.

Die Endzustandsmenge besteht bisher aus dem im 1. Schritt neu eingeführten Zustand f . Die endgültige Zustandsmenge F' besteht aus allen Zuständen, von denen aus f über eine Folge von ε -Übergängen erreichbar ist, d. h.

$$F' = \{t \in S_3 \mid (t, \varepsilon) \vdash^* (f, \varepsilon)\}$$

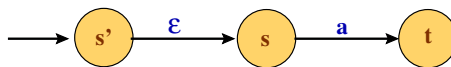
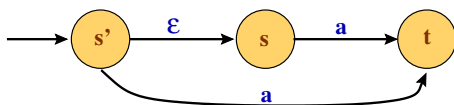
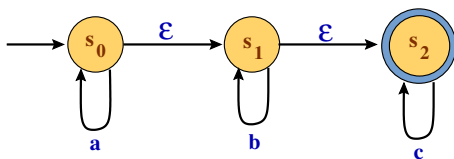
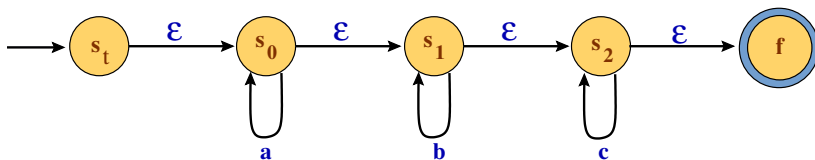


Bild 2.19: ε -Übergang.

Bild 2.20: Einfügen eines direkten Übergangs für den ε -Übergang.Bild 2.21: Zustandsdiagramm von A_{abc} .Bild 2.22: Zustandsdiagramm von A_1 .

Nach der Bestimmung von F' eliminieren wir den Zustand f aus S_3 sowie alle noch existierenden ε -Übergänge. Der resultierende Automat sei mit A_4 bezeichnet.

Es gilt: $L(A_4) = L(A)$ (ohne Beweis). □

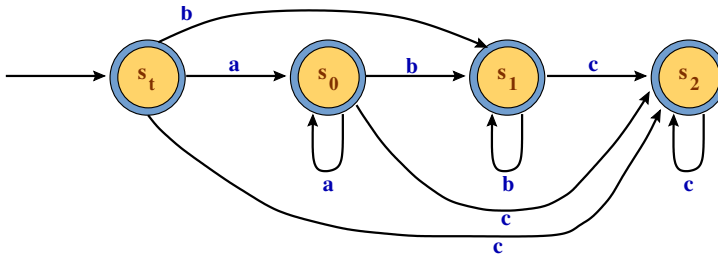
Als Beispiel transformieren wir den zu Beginn des Abschnitts konstruierten ε -Automaten A_{abc} (siehe Bild 2.21), der die Sprache L_{abc} akzeptiert, gemäß dem vorgestellten Verfahren in einen äquivalenten Automaten ohne ε -Übergänge. Im 1. Schritt fügen wir den Startzustand s_t , den Endzustand f sowie einen ε -Übergang von s_t zum vorherigen Startzustand s_0 und einen ε -Übergang vom vorherigen Endzustand s_2 nach f ein. Das Zustandsdiagramm des Ergebnisautomaten A_1 zeigt Bild 2.22. Der 2. Schritt entfällt, weil A_1 keine ε -Zykel enthält, d.h. es ist $A_2 = A_1$. Im 3. Schritt fügen wir für ε -Übergänge „echte“ Übergänge ein. Wir initialisieren δ_3 mit δ_2 ($\delta_3 := \delta_2$) und erzeugen die weiteren Elemente von δ_3 schrittweise: Ist $(s', \varepsilon, s) \in \delta_3$ und $(s, a, t) \in \delta_3$, dann wird (s', a, t) zu δ_3 hinzugefügt: $\delta_3 := \delta_3 \cup \{(s', a, t)\}$. Tabelle 2.2 zeigt die verschiedenen Teilschritte.

Im 4. Schritt bestimmen wir die Endzustände und eliminieren alle ε -Übergänge. Endzustände werden die Zustände, von denen es eine Folge von ε -Übergängen zum Endzustand f gibt. In unserem Beispiel trifft das für jeden Zustand zu, d.h. s_t, s_0, s_1 und s_2 sind Endzustände.

Falls $(s', \varepsilon, s) \in \delta_3$	und $(s, a, t) \in \delta_3$	füge (s', a, t) in δ_3 ein: $\delta_3 := \delta_3 \cup \{(s', a, t)\}$
$(s_t, \varepsilon, s_0) \in \delta_3$	$(s_0, a, s_0) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_t, a, s_0)\}$
$(s_0, \varepsilon, s_1) \in \delta_3$	$(s_1, b, s_1) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_0, b, s_1)\}$
$(s_1, \varepsilon, s_2) \in \delta_3$	$(s_2, c, s_2) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_1, c, s_2)\}$
$(s_t, \varepsilon, s_0) \in \delta_3$	$(s_0, b, s_1) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_t, b, s_1)\}$
$(s_0, \varepsilon, s_1) \in \delta_3$	$(s_1, c, s_2) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_0, c, s_2)\}$
$(s_t, \varepsilon, s_0) \in \delta_3$	$(s_0, c, s_2) \in \delta_3$	$\delta_3 := \delta_3 \cup \{(s_t, c, s_2)\}$

Tabelle 2.2: Schritt 3 bei der ε -Elimination

Nach Elimination des Zustands f und aller ε -Übergänge ist die Transformation fertig. Das Ergebnis A_4 stellt das Zustandsdiagramm in Bild 2.23 dar. Es gilt: $L(A_4) = L(A_{abc}) = L_{abc}$.

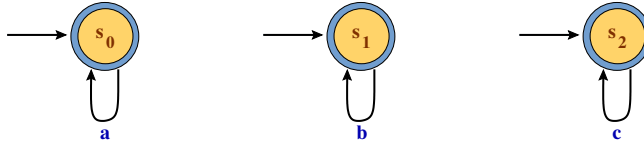
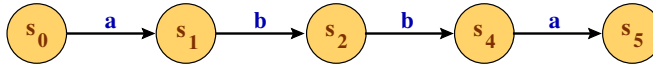
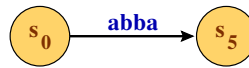
Bild 2.23: Zustandsdiagramm von A_4 .

Wir werden im Abschnitt 3.1.3 sehen, dass ε -Übergänge hilfreich bei der modularen Zusammensetzung von endlichen Automaten sind. An unserer Beispielsprache L_{abc} können wir bereits die Grundidee nachvollziehen. L_{abc} lässt sich darstellen als Konkatenation von drei einfachen Sprachen:

$$L_{abc} = \{a^i \mid i \geq 0\} \circ \{b^j \mid j \geq 0\} \circ \{c^k \mid k \geq 0\} = \{a\}^* \circ \{b\}^* \circ \{c\}^*$$

Die Sprachen $\{a\}^*$, $\{b\}^*$ und $\{c\}^*$ werden von Automaten mit den in Bild 2.24 dargestellten Zustandsdiagrammen akzeptiert. Das „Hintereinanderschalten“ dieser drei Automaten mithilfe von ε -Übergängen ergibt unseren Automaten A_{abc} (siehe Bild 2.21).

Folgerung 2.3. Aus dem Schritt 1 des Beweises von Satz 2.3 folgt unmittelbar, dass jeder endliche Automat in einen äquivalenten transformiert werden kann, der genau einen Start- und genau einen Endzustand hat. \square

Bild 2.24: Zustandsdiagramme für $\{a\}^*$, $\{b\}^*$ und $\{c\}^*$.Bild 2.25: Zustandsdiagramm, das das Wort *abba* erkennt.Bild 2.26: Ein Übergang, der das Wort *abba* erkennt.

2.3.3 Zusammenfassung

Endliche Automaten mit ε -Übergängen erlauben Zustandsübergänge ohne Verarbeiten (Lesen) eines Eingabesymbols. Sie sind äquivalent zu endlichen Automaten ohne ε -Übergänge. ε -Übergänge eignen sich zum modularen Zusammenschalten von endlichen Automaten.

2.4 Verallgemeinerte endliche Automaten

Wir wollen endliche Automaten dahingehend erweitern, dass Zustandsübergänge nicht nur für Symbole, sondern auch für Wörter definiert werden können. Betrachten wir als Beispiel einen Ausschnitt eines Zustandsdiagramms wie etwa in Bild 2.25 dargestellt. Dieses Teildiagramm ist sicherlich äquivalent zu dem in Bild 2.26 gezeigten Übergang.

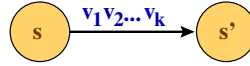
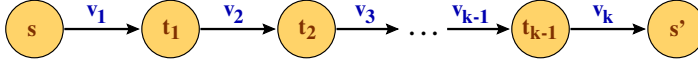
2.4.1 Definitionen

Definition 2.5. Sei Σ ein Alphabet. Dann heißt $A = (\Sigma, S, \delta_*, s_0, F)$ mit $s_0 \in S$, $F \subseteq S$ sowie $\delta_* \subseteq S \times \Sigma^* \times S$ mit $|\delta_*| < \infty$ *verallgemeinerter endlicher Automat* über Σ . \square

Da Σ^* für $|\Sigma| \neq \emptyset$ unendlich ist, müssen wir für die Zustandsüberführungen bei verallgemeinerten endlichen Automaten im Gegensatz zu den bisher betrachteten fordern, dass es nur endliche viele Übergänge geben darf.

Die Definitionen von Konfigurationsübergängen muss ebenfalls verallgemeinert werden:

$$(s, vw) \vdash^* (s'w) \text{ genau dann, wenn } (s, v, s') \in \delta_*, v, w \in \Sigma^*$$

Bild 2.27: Zustandsübergang für das Wort $v = v_1v_2 \dots v_k$.Bild 2.28: Zustandsübergangsfolge für das Wort $v = v_1v_2 \dots v_k$.

Analog zu den anderen Varianten ist $L(A) = \{w \in \Sigma^* \mid (s_0, w) \vdash^* (s, \varepsilon), s \in F\}$ die von A akzeptierte Sprache.

Mit GFA_Σ (GFA steht für *Generalized Finite Automata*) bezeichnen wir die Klasse der Sprachen über Σ , die von verallgemeinerten endlichen Automaten akzeptiert werden.

2.4.2 Äquivalenz von verallgemeinerten und endlichen Automaten

Die Frage, die wir uns jetzt wieder stellen, ist natürlich die nach der Äquivalenz: Gilt $GFA_\Sigma = \varepsilon FA_\Sigma$, d.h. sind die verallgemeinerten Automaten äquivalent zu den anderen Varianten? Jeder endliche ε -Automat ist ein spezieller verallgemeinerter Automat. Es gilt also $\varepsilon FA_\Sigma \subseteq GFA_\Sigma$. Der folgende Satz besagt, dass auch die Umkehrung $GFA_\Sigma \subseteq \varepsilon FA_\Sigma$ gilt.

Satz 2.4. Zu jedem verallgemeinerten endlichen Automaten existiert ein äquivalenter endlicher Automat.

Beweis Wir geben ein allgemeines Verfahren an, mit dem zu jedem verallgemeinerten Automaten ein äquivalenter endlicher Automat konstruiert werden kann. Sei $A = (\Sigma, S, \delta_*, s_0, F)$ ein verallgemeinerter Automat. Für jedes Wort $v = v_1v_2 \dots v_k \in \Sigma^+$, $k \geq 2$, mit $\delta_*(s, v) = s'$, führen wir $k - 1$ neue Zustände t_1, \dots, t_{k-1} , $k \geq 2$, ein sowie die Zustandsüberführungen $\delta(s, v_1) = t_1$, $\delta(t_i, v_{i+1}) = t_{i+1}$, $1 \leq i \leq k - 2$, und $\delta(t_{k-1}, v_k) = s'$. Der Übergang in Bild 2.27 wird also transformiert in die Übergangsfolge in Bild 2.28. \square

Verallgemeinerte endliche Automaten lassen Zustandsübergänge nicht nur für Symbole, sondern auch für Wörter zu. Diese Möglichkeit erlaubt in manchen Fällen eine effizientere Darstellung der Automaten. Verallgemeinerte endliche Automaten sind äquivalent zu den anderen Varianten endlicher Automaten.

2.4.3 Weitere Varianten endlicher Automaten

Wir haben bisher vier Konzepte zur Charakterisierung regulärer Sprachen kennen gelernt: deterministische, nichtdeterministische endliche Automaten, endliche Automaten mit ε -Übergängen sowie verallgemeinerte endliche Automaten. Wir haben gezeigt,

dass alle diese Konzepte äquivalent sind:

$$REG_{\Sigma} = DFA_{\Sigma} = NFA_{\Sigma} = \varepsilon FA_{\Sigma} = GFA_{\Sigma}$$

Wenn wir einen Automaten eines bestimmten Typs konstruieren sollen, können wir also zunächst – möglicherweise abhängig von der Aufgabenstellung – einen Automaten eines „leichter“ konstruierbaren Typs erstellen und diesen dann in die gewünschte Variante transformieren. So ist es z.B. nahe liegend, zu den Bitfolgen, deren drittletztes Bit 0 ist, den nichtdeterministischen Automaten in Bild 2.12 zu konstruieren und nicht den deterministischen in Bild 2.13.

Für die Transformationen zwischen den Varianten stehen Programme zur Verfügung, welche die vorgestellten Verfahren implementieren, so dass die Transformationen nicht per Hand vorgenommen werden müssen.

Es gibt noch weitere Typen endlicher Automaten, z.B. stochastische Automaten, Fuzzy-Automaten und endliche Zweiweg-Automaten. Bei stochastischen Automaten sind den Zustandsübergängen Wahrscheinlichkeiten zugeordnet. Bei Fuzzy-Automaten sind die Übergänge mit einer „unscharfen“ Bewertung versehen. Zweiweg-Automaten können den Lesekopf auch nach links bewegen.

Zustände können auch durch n -Tupel dargestellt werden. Diese Möglichkeit ist von Nutzen, wenn Zustände durch n unterschiedliche Merkmale gekennzeichnet sind. So kann z.B. ein Angestellter durch Merkmale wie Name, Anschrift, Familienstand, Qualifikation, Gehaltsgruppe etc. beschrieben sein. Eine Heirat führt dann durch Änderungen der Merkmale Name, Familienstand, Gehaltsgruppe zu einem neuen Zustand.

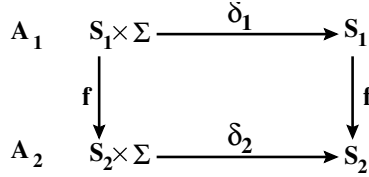
Sind solche Merkmalsbeschreibungen sogar Vektoren, und bildet die Menge der Merkmalsvektoren einen Vektorraum, können Zustandänderungen durch lineare Abbildungen beschrieben sein. Dabei kann zwischen diskreten und stetigen Abbildungen unterschieden werden. Anwendungen findet man in vielen Ingenieurbereichen.

Mit all diesen Typen werden wir uns hier nicht weiter beschäftigen, sondern verweisen auf die Literatur. Es sollte nur auf die Vielfältigkeit der Theorie und der Anwendung von Automaten verwiesen werden, die weit über den hier dargestellten Standardstoff im Informatik-Grundstudium hinausgeht.

2.5 Minimierung endlicher Automaten

In diesem Abschnitt lernen wir Verfahren zur Minimierung von endlichen Automaten kennen. Wir werden sehen, dass man zu jedem endlichen Automaten einen äquivalenten minimalen Automaten konstruieren kann. Dabei heißt minimal: minimale Anzahl von Zuständen.

Definition 2.6. Sei L eine reguläre Sprache über dem Alphabet Σ ($L \in REG_{\Sigma}$) und A ein endlicher Automat über Σ , der L akzeptiert, d.h. $L = L(A)$. A heißt *minimaler Automat* (für L), falls es keinen Automaten A' mit weniger Zuständen ($|S'| < |S|$) gibt, der L akzeptiert. \square

Bild 2.29: Isomorphie von A_1 und A_2 .

2.5.1 Isomorphie endlicher Automaten

Der minimale Automat ist sogar eindeutig in dem Sinne, dass sich minimale Automaten untereinander höchstens in der Bezeichnung ihrer Zustände unterscheiden, d.h. ihre Eingabesymbole sind gleich, die Anzahl ihrer Zustände ist gleich, die Anzahl der Endzustände ist gleich und die Zustandsüberführungen sind gleich, wenn bei deren Vergleich die unterschiedlichen Benennungen der Zustände berücksichtigt werden.

Diese Art der Gleichheit, die so genannte *Strukturgleichheit*, wollen wir formal definieren.

Definition 2.7. Seien $A_1 = (\Sigma, S_1, \delta_1, s_{0_1}, F_1)$ und $A_2 = (\Sigma, S_2, \delta_2, s_{0_2}, F_2)$ deterministische endliche Automaten über Σ mit $|S_1| = |S_2|$. A_1 und A_2 heißen *isomorph* genau dann, wenn es eine bijektive Abbildung $f : S_1 \rightarrow S_2$ gibt, für die

$$f(\delta_1(s, a)) = \delta_2(f(s), a), \quad s \in S_1, \quad a \in \Sigma \quad (2.4)$$

sowie

$$f(F_1) = F_2 \quad (2.5)$$

gilt. Wir notieren die *Isomorphie* von A_1 und A_2 mit: $A_1 \cong A_2$. \square

Die Bedingung (2.4) legt fest, dass die Zustandsübergänge in beiden Automaten gleich sind, dabei können die sich entsprechenden Zustände unterschiedlich benannt sein: Die Bijektion f ordnet jedem Zustand aus S_1 eineindeutig einen Zustand aus S_2 zu, beschreibt also die Umbenennung der Zustände. Die Gleichung (2.4) fordert, dass, wenn in A_1 ein Zustandsübergang vom Zustand s mit Eingabe a ausgeführt und der Folgezustand umbenannt wird, derselbe Zustand erreicht wird, als wenn s umbenannt und in A_2 der a -Übergang von dieser Umbenennung aus durchgeführt wird. Es gilt also:

$$\delta_1(s, a) = s' \text{ genau dann, wenn } \delta_2(f(s), a) = f(s')$$

Das Funktionsdiagramm in Bild 2.29 veranschaulicht die mathematische Beschreibung der Strukturgleichheit. Die Bedingung (2.4) wird auch dadurch ausgedrückt, dass man sagt, dass *das Diagramm kommutiert*: Der Weg im Uhrzeigersinn, d.h. Zustandsübergang in A_1 und anschließend Umbenennung, liefert dasselbe Ergebnis wie der Weg entgegen dem Uhrzeigersinn, d.h. zuerst Umbenennung und dann Übergang in A_2 .

Die Bedingung (2.5) fordert, dass genau die Endzustände von A_1 in die Endzustände von A_2 umbenannt werden.

2.5.2 Der Satz von Myhill und Nerode

Wir lernen zunächst eine weitere – eine algebraische – Charakterisierung der Klasse der regulären Sprachen kennen. Diese dient dann als Grundlage für ein weiteres Verfahren zur Minimierung von endlichen Automaten.

Es sei Σ ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache. Wir definieren die Relation $R_L \subseteq \Sigma^* \times \Sigma^*$ durch

$$x R_L y \text{ genau dann, wenn für alle } z \in \Sigma^* \text{ gilt } xz \in L \text{ gdw. } yz \in L \quad (2.6)$$

Diese Relation ist eine Äquivalenzrelation, denn sie ist

- reflexiv: Offensichtlich gilt $xz \in L$ gdw. $xz \in L$ für jedes $x \in \Sigma^*$ und jedes $z \in \Sigma^*$, d.h. für alle $x \in \Sigma^*$ gilt $x R_L x$;
- symmetrisch: Es sei $x R_L y$, dann ist $xz \in L$ gdw. $yz \in L$ für alle $z \in \Sigma^*$ und damit $yz \in L$ gdw. $xz \in L$ für alle $z \in \Sigma^*$, somit gilt $y R_L x$;
- transitiv: Es sei $u R_L v$ und $v R_L w$, d.h. $uz \in L$ gdw. $vz \in L$ und $vz \in L$ gdw. $wz \in L$ für alle $z \in \Sigma^*$, woraus $uz \in L$ gdw. $wz \in L$ für alle $z \in \Sigma^*$ folgt und damit $u R_L w$.

Es sei $A = (\Sigma, S, \delta, s_0, F)$ ein deterministischer endlicher Automat. Wir definieren die Relation R_A durch

$$x R_A y \text{ genau dann, wenn } \delta^*(s_0, x) = \delta^*(s_0, y) \quad (2.7)$$

Es ist leicht einzusehen, dass auch die Relation R_A eine Äquivalenzrelation ist. $x R_A y$ bedeutet, dass der Automat A beim Abarbeiten der Wörter x und y denselben Zustand erreicht. Daraus folgt, dass zu jedem vom Startzustand s_0 aus erreichbaren Zustand von A genau eine Äquivalenzklasse existiert. Die Anzahl der Äquivalenzklassen von R_A , der so genannte *Index* von R_A , ist also endlich.

R_A hat zudem die definierende Eigenschaft (2.6) von R_L , denn es gilt: Ist $x R_A y$, dann ist $xz R_A yz$ für alle $z \in \Sigma^*$, weil

$$\delta^*(s_0, xz) = \delta^*(\delta^*(s_0, x), z) = \delta^*(\delta^*(s_0, y), z) = \delta^*(s_0, yz)$$

und damit die Bedingung (2.7) für xz und yz erfüllt ist, also $xz R_A yz$ für alle $z \in \Sigma^*$ gilt.

Die Relationen R_L und R_A sind so genannte *Rechtskongruenzen*. Das bedeutet, dass sie Äquivalenzrelationen sind, die verträglich mit der Konkatenation von Wörtern von rechts sind. Jeder deterministische endliche Automat A über dem Alphabet Σ induziert also die Rechtskongruenz R_A über Σ^* .

Der folgende *Satz von Myhill und Nerode* stellt den Zusammenhang zwischen R_L und R_A für $L = L(A)$ her.

Satz 2.5. Σ sei ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache. Dann sind die folgenden drei Aussagen äquivalent:

a) $L \in REG_\Sigma$;

b) L ist die Vereinigung von Äquivalenzklassen einer Rechtskongruenz auf Σ^* mit endlichem Index;

c) Der Index von R_L ist endlich.

Beweis „a) \Rightarrow b)“: Da $L \in REG_\Sigma$ ist, gibt es einen deterministischen endlichen Automaten A mit $L = L(A)$. Wir haben schon festgestellt, dass R_A eine Rechtskongruenz auf Σ^* mit endlichem Index darstellt. Die Äquivalenzklassen von R_A sind durch die von s_0 erreichbaren Zustände festgelegt; jeder dieser Zustände legt genau eine Äquivalenzklasse fest. Die Endzustände von A legen die Klassen fest, die genau die Wörter von L enthalten.

„b) \Rightarrow c)“: Sei R irgendeine Rechtskongruenz auf Σ^* mit endlichem Index, deren Äquivalenzklassen vereinigt L ergeben. Falls $x R y$ gilt, dann gilt auch $xz R yz$ für alle $z \in \Sigma^*$, weil R rechtskongruent ist. Daraus folgt, dass $xz \in L$ ist genau dann, wenn $yz \in L$ ist. Somit gilt $x R_L y$. Wir haben also gezeigt, dass aus $x R y$ folgt $x R_L y$. Daraus folgt, dass die Äquivalenzklasse von x bezüglich R in der Äquivalenzklasse von x bezüglich R_L enthalten ist: $[x]_R \subseteq [x]_{R_L}$. Jede Äquivalenzklasse von R ist also in einer Äquivalenzklasse von R_L enthalten: R ist eine Verfeinerung von R_L . Es folgt, dass der Index von R_L nicht größer sein kann als der von R , und da der Index von R endlich ist, ist somit der Index von R_L ebenfalls endlich.

„c) \Rightarrow a)“: Die Rechtskongruenz R_L auf Σ^* habe einen endlichen Index $k \geq 1$. Es gibt also endlich viele, nämlich k Repräsentanten $x_1, x_2, \dots, x_k \in \Sigma^*$ mit

$$\Sigma^* = [x_1]_{R_L} \cup [x_2]_{R_L} \cup \dots \cup [x_k]_{R_L}$$

und

$$[x_i]_{R_L} \cap [x_j]_{R_L} = \emptyset \text{ für } 1 \leq i, j \leq k \text{ und } i \neq j$$

Wir konstruieren den Automaten

$$A_{R_L} = (\Sigma, S, \delta, s_0, F)$$

mit

$$\begin{aligned} S &= \{[x_1]_{R_L}, [x_2]_{R_L}, \dots, [x_k]_{R_L}\} \\ \delta([x]_{R_L}, a) &= [xa]_{R_L} \\ s_0 &= [\varepsilon]_{R_L} \\ F &= \{[x]_{R_L} \mid x \in L\} \end{aligned}$$

Wir zeigen nun, dass für diesen Automaten A_{R_L} gilt $L = L(A_{R_L})$:

$$\begin{aligned} x \in L(A_{R_L}) & \text{ genau dann, wenn } \delta^*(s_0, x) \in F \\ & \text{ genau dann, wenn } \delta^*([\varepsilon]_{R_L}, x) \in F \\ & \text{ genau dann, wenn } [x]_{R_L} \in F \\ & \text{ genau dann, wenn } x \in L \end{aligned}$$

Damit haben wir im Ringschluss „a) \Rightarrow b) \Rightarrow c) \Rightarrow a)“ die Äquivalenz der Aussagen a), b) und c) gezeigt. \square

Aus der Äquivalenz der Aussagen a) und b) folgt unmittelbar eine weitere Charakterisierung der Klasse der regulären Sprachen.

Folgerung 2.4. Es sei Σ ein Alphabet. Dann ist $L \in REG_\Sigma$ genau dann, wenn R_L endlichen Index hat. \square

Beispiel 2.7. a) Wir betrachten die Sprache aller Bitfolgen, die mit 1 enden und mindestens die Länge 2 haben:

$$L = \{ w1 \mid w \in \{0, 1\}^+ \}$$

Wir betrachten die Relation R_L und stellen fest, dass $x R_L y$ genau dann gilt, wenn

- x und y die Länge 0 haben, also gleich dem leeren Wort sind, oder
- x und y die Länge 1 haben oder mindestens die Länge 2 haben und auf 0 enden, oder
- x und y mindestens die Länge 2 haben und auf 1 enden.

Es gilt also

$$\begin{aligned} [\varepsilon]_{R_L} &= \{ \varepsilon \} \\ [0]_{R_L} &= \{ 0, 1, 00, 10, 000, 010, 100, 110, \\ &\quad 0000, 0010, 0100, 0110, 1000, 1010, 1100, 1110, \dots \} \\ [01]_{R_L} &= \{ 01, 11, 001, 011, 101, 111, \\ &\quad 0001, 0011, 0101, 0111, 1001, 1011, 1101, 1111, \dots \} \end{aligned}$$

sowie

$$\{0, 1\}^* = [\varepsilon]_{R_L} \cup [0]_{R_L} \cup [01]_{R_L}$$

Der Index von R_L ist gleich 3, also endlich. Damit wissen wir, dass L regulär ist.

Mit der Konstruktion aus dem Beweis von Satz 2.5 erhalten wir folgenden deterministischen endlichen Automaten für die Sprache L :

$$A_{R_L} = (\{0, 1\}, \{ [\varepsilon]_{R_L}, [0]_{R_L}, [01]_{R_L} \}, \delta, [\varepsilon]_{R_L}, \{ [01]_{R_L} \})$$

mit der Zustandsüberführung

$$\begin{aligned}
 \delta([\varepsilon]_{R_L}, 0) &= [0]_{R_L} \\
 \delta([\varepsilon]_{R_L}, 1) &= [0]_{R_L} \\
 \delta([0]_{R_L}, 0) &= [0]_{R_L} \\
 \delta([0]_{R_L}, 1) &= [01]_{R_L} \\
 \delta([01]_{R_L}, 0) &= [0]_{R_L} \\
 \delta([01]_{R_L}, 1) &= [01]_{R_L}
 \end{aligned}$$

b) Wir betrachten die Sprache $L = \{ a^n b^n \mid n \geq 1 \}$ und bestimmen die Äquivalenzklassen von R_L :

$$\begin{aligned}
 [ab]_{R_L} &= L \\
 [a^2b]_{R_L} &= \{ a^2b, a^3b^2, a^4b^3, \dots \} \\
 [a^3b]_{R_L} &= \{ a^3b, a^4b^2, a^5b^3, \dots \} \\
 &\vdots
 \end{aligned}$$

Allgemein gilt für $k \geq 1$:

$$[a^i b]_{R_L} = \{ a^{i+n-1} b^n \mid i \geq 1 \}$$

Für $r \neq s$ stehen die Wörter $a^r b$ und $a^s b$ nicht in Relation, denn für $z = b^{r-1}$ gilt $a^r b z \in L$, aber $a^s b z \notin L$. Damit sind die Äquivalenzklassen $[a^i b]_{R_L}$ für alle $i \geq 1$ voneinander verschieden, d.h. es gibt unendlich viele Äquivalenzklassen, der Index von R_L ist also unendlich. Es folgt, dass L keine reguläre Sprache ist.

Im Abschnitt 3.3.2 werden wir mit einer weiteren Methode feststellen, dass die Sprache L nicht regulär ist. \square

2.5.3 Verfahren zur Minimierung endlicher Automaten

Der folgende Satz ist eine Folgerung aus dem Satz von Myhill und Nerode. Er besagt, dass der Automat A_{R_L} ein minimaler Automat für die Sprache L ist.

Satz 2.6. Sei Σ ein Alphabet und $L \in REG_\Sigma$. Dann ist der mit dem Verfahren im Beweis von Satz 2.5 für L konstruierte Automat A_{R_L} der bis auf Isomorphie minimale Automat für L .

Beweis Aus dem Beweis von Satz 2.5 folgt, dass für jeden deterministischen endlichen Automaten A für L gilt: $R_A \subseteq R_L = R_{A_{R_L}}$. Die Anzahl der Zustände von A ist also größer oder gleich der Anzahl der Zustände von A_{R_L} . Somit ist die Anzahl der Zustände von A_{R_L} minimal in Bezug auf L .

Jetzt überlegen wir noch, dass es zu A_{R_L} keinen strukturell verschiedenen Automaten geben kann, der ebenfalls minimal für L ist. Wenn nämlich A schon minimal

ist, dann folgt $R_A = R_L$ und damit $R_A = R_{A_{R_L}}$, woraus sich die Strukturgleichheit von A und A_{R_L} ergibt. \square

Wir gehen im Folgenden davon aus, dass ein zu minimierender deterministischer endlicher Automat $A = (\Sigma, S, \delta, s_0, F)$ nur erreichbare Zustände enthält. Ein Zustand $s \in S$ ist erreichbar, wenn es ein Wort $w \in \Sigma^*$ gibt mit $\delta^*(s_0, w) = s$. Nicht erreichbare Zustände sind überflüssig, sie haben keine Bedeutung für die vom Automaten A akzeptierte Sprache $L(A)$. Ein minimaler Automat kann niemals unerreichbare Zustände enthalten. Nicht erreichbare Zustände können also eliminiert werden.

Die Äquivalenzklassenkonstruktion aus dem Satz von Myhill und Nerode ist die Grundlage für das folgende Verfahren zur Bestimmung des Minimalautomaten zu einem gegebenen deterministischen endlichen Automaten $A = (\Sigma, S, \delta, s_0, F)$.

Dieses Verfahren wird auch *Markierungsalgorithmus* genannt:

1. Bilde eine Tabelle für alle Zustandspaare $\{s, t\}$, $s, t \in S$, $s \neq t$.
2. Markiere alle Paare $\{s, t\}$ mit $s \notin F$ und $t \in F$.
3. Teste für jedes noch nicht markierte Paar $\{s, t\}$ und jedes $a \in \Sigma$, ob das Paar $\{\delta(s, a), \delta(t, a)\}$ schon markiert ist. Falls ja, dann markiere das Paar $\{s, t\}$.
4. Führe Schritt 3. so lange aus, bis die Markierungen sich nicht mehr ändern.
5. Bilde für jeden Zustand s die Menge $S = \{s\} \cup \{t \mid \{s, t\} \text{ ist unmarkiert}\}$, d.h. der Zustand s wird mit allen Zuständen t zu einem Zustand zusammengefasst, für die $\{s, t\}$ nicht markiert ist. Wir nennen diese Mengen Blöcke, Π sei die Menge dieser Blöcke. Wir erhalten als minimalen Automaten

$$A_{min} = (\Sigma, \Pi, \delta_{min}, S_0, \{S \in \Pi \mid S \cap F \neq \emptyset\})$$

mit

$$\delta_{min}(S, a) = \bigcup_{s \in S} \delta(s, a)$$

Beispiel 2.8. Wir wenden das Verfahren auf den Automaten

$$A = (\{0, 1\}, \{s_0, s_1, s_2, s_3, s_4\}, \delta, s_0, \{s_3, s_4\})$$

an, dessen Zustandsdiagramm in Bild 2.30 dargestellt ist. Dieser Automat akzeptiert die Sprache

$$L = \{w1 \mid w \in \{0, 1\}^+\}$$

aus Beispiel 2.7 a).

Im ersten Schritt stellen wir die Tabelle

s_1				
s_2				
s_3				
s_4				
	s_0	s_1	s_2	s_3

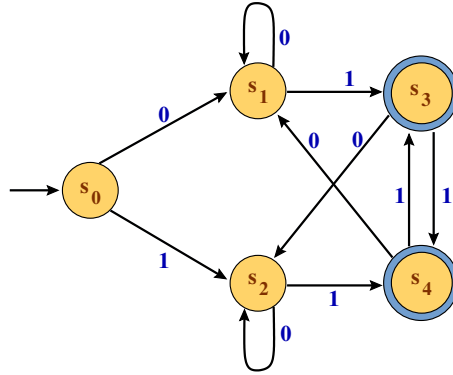


Bild 2.30: Ein zu minimierender Automat.

auf.

Der zweite Schritte führt zu den Markierungen

s_1				
s_2				
s_3	*	*	*	
s_4	*	*	*	
	s_0	s_1	s_2	s_3

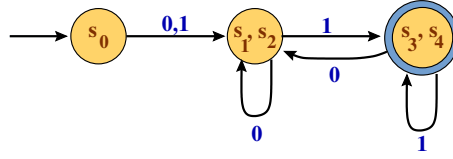
Jetzt führen wir die Schritte 3. und 4. durch. Betrachten wir zuerst das unmarkierte Paar $\{s_0, s_2\}$, dann ist das Paar $\{\delta(s_0, 0), \delta(s_2, 0)\} = \{s_1, s_2\}$ nicht markiert, aber das Paar $\{\delta(s_0, 1), \delta(s_2, 1)\} = \{s_2, s_4\}$ ist markiert, also wird $\{s_0, s_2\}$ markiert. Die Tabelle hat somit folgendes Aussehen:

s_1				
s_2	*			
s_3	*	*	*	
s_4	*	*	*	
	s_0	s_1	s_2	s_3

Als nächstes betrachten wir das Paar $\{s_0, s_1\}$. Das Paar $\{\delta(s_0, 0), \delta(s_1, 0)\} = \{s_1, s_2\}$ ist nicht markiert, das Paar $\{\delta(s_0, 1), \delta(s_1, 1)\} = \{s_2, s_3\}$ ist hingegen markiert. Damit wird auch $\{s_0, s_1\}$ markiert, und wir erhalten die folgende Tabelle:

s_1	*			
s_2	*			
s_3	*	*	*	
s_4	*	*	*	
	s_0	s_1	s_2	s_3

Jetzt betrachten wir das Paar $\{s_1, s_2\}$. Die Paare $\{\delta(s_1, 0), \delta(s_2, 0)\} = \{s_1, s_2\}$ und $\{\delta(s_1, 1), \delta(s_2, 1)\} = \{s_3, s_4\}$ sind nicht markiert, also wird auch $\{s_1, s_2\}$ nicht markiert.

Bild 2.31: Zustandsdiagramm des zu A minimalen Automaten A_{min} .

Es bleibt noch das Paar $\{s_3, s_4\}$. Sowohl $\{\delta(s_3, 0), \delta(s_4, 0)\} = \{s_1, s_2\}$ als auch $\{\delta(s_3, 1), \delta(s_4, 1)\} = \{s_3, s_4\}$ sind nicht markiert, also wird auch $\{s_3, s_4\}$ nicht markiert.

Damit ergibt sich keine Änderung der Markierung. Die beiden Paare $\{s_1, s_2\}$ und $\{s_3, s_4\}$ bleiben unmarkiert und werden jeweils zu einem Zustand verschmolzen.

Im fünften Schritt erhalten wir so die Zustandsmengen

$$S_0 = \{s_0\}, \quad S_1 = \{s_1, s_2\}, \quad S_3 = \{s_3, s_4\}$$

und damit den Automaten

$$A_{min} = (\{0, 1\}, \{S_0, S_1, S_3\}, \delta_{min}, S_0, \{S_3\})$$

mit

$$\begin{array}{lll} \delta_{min}(S_0, 0) = S_1 & \delta_{min}(S_1, 0) = S_1 & \delta_{min}(S_3, 0) = S_1 \\ \delta_{min}(S_0, 1) = S_1 & \delta_{min}(S_1, 1) = S_3 & \delta_{min}(S_3, 1) = S_3 \end{array}$$

Bild 2.31 zeigt das Zustandsdiagramm dieses Automaten. Dieser Automat ist isomorph zu dem Automaten A_{RL} aus Beispiel 2.7 a). \square

Falls am Ende des Markierungsalgorithmus alle Paare markiert sind, bedeutet dies, dass der ursprüngliche Automat bereits minimal ist.

Wir wollen noch eine Variante des Markierungsalgorithmus betrachten, welche schrittweise die Zustandsmenge und die Überföhrungsfunktion des minimalen Automaten A_{min} zu einem gegebenen vollständigen deterministischen Automaten A konstruiert. Dabei wird in jedem i -ten Schritt eine disjunkte Zerlegung

$$\Pi_i = \{S_{i1}, S_{i2}, \dots, S_{ik_i}\}, \quad k_i \geq 1$$

von S erzeugt. Wir nennen die Elemente S_{ij} , $1 \leq j \leq k_i$, von Π_i Blöcke. Dabei ist Π_{i+1} feiner als Π_i , d.h. es ist $k_{i+1} \geq k_i$ und zu jedem Block $S \in \Pi_{i+1}$ existiert ein Block $S' \in \Pi_i$ mit $S \subseteq S'$. Die Elemente von Π_i sind die Zustände im i -ten Schritt, auf denen die i -te Zustandsüberföhrung δ_i definiert wird.

Das Verfahren endet, wenn im Schritt $i + 1$ keine neue Partition mehr entsteht, d.h. falls $\Pi_{i+1} = \Pi_i$ ist. Das Verfahren endet auf jeden Fall, wenn die feinste Zerlegung, die überhaupt möglich ist, erreicht wird. Die feinste Zerlegung ist die, die als Blöcke alle einelementigen Teilmengen von S enthält. In diesem Fall ist der Automat A bereits minimal.

Wir stellen jetzt das Verfahren vor:

1. Wir zerlegen die Zustandsmenge S in zwei disjunkte Teilmengen: $S_{11} = F$ und $S_{12} = S - F$. Die erste Partition ist also: $\Pi_1 = \{S_{11}, S_{12}\} = \{F, S - F\}$.
2. Sei $\Pi_i = \{S_{i1}, S_{i2}, \dots, S_{ik_i}\}$, $k_i \geq 0$. Bilde die Partition Π_{i+1} gemäß folgender Bedingung: Zwei Zustände s und s' gehören genau dann zum selben Block in Π_{i+1} , falls für jedes $a \in \Sigma$ gilt: $\delta_i(s, a)$ und $\delta_i(s', a)$ gehören zum selben Block in Π_i .
3. Falls $\Pi_{i+1} = \Pi_i$ ist, dann ist Π_i die Zustandsmenge des minimalen Automaten. Falls $\Pi_{i+1} \neq \Pi_i$ ist, wird die Konstruktion bei Schritt 2 fortgesetzt.

Wenn das Verfahren mit Erreichen der Endbedingung $\Pi_{i+1} = \Pi_i$ endet, dann ist Π_i die Zustandsmenge von A_{min} . δ_{min} ist durch δ_i gegeben, der Startzustand von A_{min} ist der Block von Π_i , der den Startzustand s_0 von A enthält, und die Endzustandsmenge von A_{min} besteht aus allen Blöcken von Π_i , die mindestens einen Endzustand von A enthalten.

Wir wollen als Beispiel wieder den in Bild 2.30 gezeigten Automaten

$$A = (\{0, 1\}, \{s_0, s_1, s_2, s_3, s_4\}, \delta, s_0, \{s_3, s_4\})$$

minimieren und führen die obige Konstruktion schrittweise aus:

- Die Endzustandsmenge ist $F = \{s_3, s_4\}$. Damit ergeben sich die Blöcke der ersten Partition: $S_{11} = \{s_3, s_4\}$ und $S_{12} = S - F = \{s_0, s_1, s_2\}$, also

$$\Pi_1 = \{\{s_3, s_4\}, \{s_0, s_1, s_2\}\}$$

Zur Überprüfung der Bedingung, die im nächsten Schritt die Zerlegung bestimmt, benutzen wir die folgende Zustandstabelle, deren Einträge sich wie folgt ergeben: Für Zustand s und Eingabe a tragen wir den Block S_{ij} ein, falls $\delta(s, a) \in S_{ij}$ ist.

	S_{11}		S_{12}		
	s_3	s_4	s_0	s_1	s_2
0	S_{12}	S_{12}	S_{12}	S_{12}	S_{12}
1	S_{11}	S_{11}	S_{12}	S_{11}	S_{11}

- Wenn wir die Zustandstabelle betrachten, sehen wir, dass für den Block S_{11} gilt: Alle Zustände aus S_{11} haben bei Eingabe von 0 Folgezustände, die in demselben Block, nämlich in S_{12} , liegen. Ebenso haben alle Zustände aus S_{11} bei Eingabe von 1 Folgezustände aus demselben Block, nämlich aus S_{11} . S_{11} braucht also nicht weiter zerlegt zu werden und bleibt als Block in Π_2 erhalten, wir bezeichnen ihn dort mit S_{21} .

Während alle Zustände aus S_{12} bei Eingabe von 0 Folgezustände aus demselben Block, nämlich aus S_{12} , haben, ist das bei Eingabe von 1 nicht der Fall: Der

Folgezustand von s_0 liegt in S_{12} , die Folgezustände von s_1 und s_2 liegen im Block S_{11} . Deshalb teilen wir S_{12} auf in die Blöcke $S_{22} = \{s_0\}$ und $S_{23} = \{s_1, s_2\}$. Wir erhalten also die neue Zerlegung

$$\Pi_2 = \{\{s_3, s_4\}, \{s_0\}, \{s_1, s_2\}\}$$

- Π_2 ist verschieden von Π_1 , wir müssen also mit Schritt 2 des Verfahrens fortfahren. Dazu benutzen wir analog zu oben die Zustandstabelle bezüglich Π_2 .

	S_{21}		S_{22}	S_{23}	
	s_3	s_4	s_0	s_1	s_2
0	S_{23}	S_{23}	S_{23}	S_{23}	S_{23}
1	S_{21}	S_{21}	S_{23}	S_{21}	S_{21}

- Wir stellen fest, dass für jeden Block gilt, dass für jeden seiner Zustände für jede Eingabe der Folgezustand jeweils in demselben Block liegt. Es entstehen also keine neuen Blöcke, d.h. es ist $\Pi_3 = \Pi_2$.
- Das Verfahren endet. Der zu A minimale Automat A_{min} ergibt sich als

$$A_{min} = (\{0, 1\}, \{S_{21}, S_{22}, S_{23}\}, \delta_{min}, S_{22}, \{S_{21}\})$$

wobei δ_{min} durch die obige Zustandstabelle gegeben ist. Dieser Automat ist derselbe, den wir durch den Markierungsalgorithmus erhalten haben und dessen Zustandsdiagramm in Bild 2.31 dargestellt ist.

Zu jedem deterministischen endlichen Automaten A existiert also ein äquivalenter minimaler Automat A_{min} . Minimal heißt: Es gibt keinen zu A äquivalenten deterministischen endlichen Automaten, der weniger Zustände als A_{min} hat. A_{min} ist eindeutig, wenn man von der Bezeichnung seiner Zustände absieht, denn alle minimalen Automaten zu A sind isomorph zueinander.

Bei nichtdeterministischen endlichen Automaten gilt diese Eigenschaft nicht. Als Beispiel betrachten wir die beiden nichtdeterministischen endlichen Automaten in Bild 2.32 für die Sprache L aus Beispiel 2.7 a). Beide Automaten sind minimal für L , aber nicht isomorph.

2.6 Anwendungen endlicher Automaten

In diesem Abschnitt stellen wir einige Anwendungen endlicher Automaten vor. Wir tun dies exemplarisch anhand typischer Bereiche aus der Informatik, wobei unsere Auswahl nicht vollständig sein kann, allerdings die Spannweite erkennen lassen soll, welche in der Verwendung endlicher Automaten gegeben ist: Sie reicht z.B. von der Beschreibung von Rechnersystemen und deren Systemprogrammierung über die Teilworterkennung bis hin zur Spezifikation und Verifikation von objektorientierten Entwürfen sowie der Modellierung von Kommunikationsabläufen zwischen Systemen oder zwischen Menschen und Maschinen.

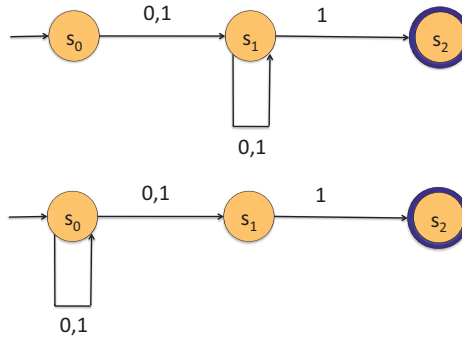


Bild 2.32: Zwei nicht isomorphe nicht deterministische endliche Automaten für die Sprache $L = \{ w1 \mid w \in \{0,1\}^+ \}$ mit minimaler Anzahl von Zuständen.

2.6.1 Rechnersysteme und Systemprogrammierung

Prinzipiell lässt sich jedes (sequentielle) Rechnersystem als endlicher Automat modellieren, sofern man gewisse Abstraktionen vornimmt: Wir sehen ab von Möglichkeiten zur Ein- bzw. Ausgabe (von Daten oder Programmen); auch mache man sich klar, dass ein endlicher Automat in der bisher betrachteten Form *genau eine* Aufgabe ausführen kann, die quasi fest verdrahtet wird. Er ist also *nicht programmierbar*, d. h. durch Austausch eines Programms zur Ausführung einer neuen Aufgabe zu bewegen. Auch ein parallel arbeitendes System geht offensichtlich über die Modellierungsmöglichkeiten eines endlichen Automaten hinaus. Ein sequentieller Rechner ist auf der Ebene der reinen Hardware als ein synchron (also getaktet) arbeitendes *Schaltwerk* beschreibbar, welches aus einem Schaltnetz (der CPU) zur Berechnung von Schalt- oder Booleschen Funktionen sowie aus Speicherelementen besteht. Jeder mögliche Speicherinhalt, alle Speicher eines Rechnersystems (CPU-Register, Haupt-, Cache-, Sekundärspeicher) umfassend, stellt einen möglichen Zustand dar; mögliche Veränderungen des Inhaltes, welche sich aus Berechnungen oder Schaltvorgängen im Rechner ergeben, sind die Zustandsübergänge. Eingabe- sowie Ausgabedaten werden im Allgemeinen so codiert, dass sie in dualer Form vorliegen; ein Startzustand ist z.B. dadurch gekennzeichnet, dass er alle Speicherinhalte bis auf die Eingabedaten löscht; ein Endzustand ist erreicht, wenn das Schaltwerk eine aktuelle Rechnung beendet hat. Wir werden in Kapitel 4 auf diese Analogie genauer eingehen, wenn uns nämlich Automaten *mit Ausgabe* zur Verfügung stehen.

Das Betriebssystem eines Rechners organisiert *Programmausführungen* im Allgemeinen als *Prozesse*, welche unter anderem abwechselnd die CPU des betreffenden Rechners belegen. Es lassen sich dann verschiedene (aber nur endliche viele) *Prozesszustände* unterscheiden in Abhängigkeit davon, ob ein Prozess aktuell Zugriff auf die CPU hat, also de facto in Ausführung („running“) ist, ob er auf die Zuteilung der

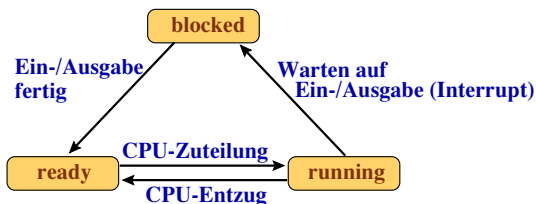


Bild 2.33: Prozesszustände und -übergänge.

CPU wartet („ready“) oder auf Ein- oder Ausgabe von bzw. nach außen (etwa Lesen vom oder Schreiben in den Hauptspeicher) wartet („blocked“). Derartige Prozesszustände und mögliche Zustandsübergänge sind in Bild 2.33 gezeigt; offensichtlich kann dieses Diagramm je nach Bedarf der Modellierung verfeinert oder erweitert und sogar an andere Situationen innerhalb eines Betriebssystems angepasst werden.

2.6.2 Teilworterkennung

Wenn man, z.B. zum Schreiben eines Programmtextes, ein Textverarbeitungssystem oder einen Editor benutzt, macht man meist von den Funktionen dieses Systems zum Suchen (*search*) oder Ändern (*replace*, *replace all*) von Wörtern Gebrauch. Um ein Wort zu ändern, muss es zunächst gefunden werden; wir wollen uns hier auf das Suchproblem beschränken. Konkret wollen wir das folgende Problem untersuchen: Gegeben sei ein Text w , gesucht sind die Stelle oder die Stellen, an der bzw. an denen ein Text v in w vorkommt. Beide Texte seien Wörter über einem Alphabet Σ (z.B. die Menge der ASCII-Symbole).

Das Problem kann auf Bilder (Muster) verallgemeinert werden. Das Problem, ein Teilbild in einem Bild zu suchen, ist ein grundlegendes Problem der Mustererkennung und der Bildverarbeitung. Das Problem wird auch *Pattern-Matching-Problem* genannt. Wir wollen das Pattern-Matching-Problem nur für den linearen Fall, d.h. für Wörter über einem Alphabet Σ betrachten. Es lässt sich durch die Funktion

$$pm : \Sigma^* \times \Sigma^* \rightarrow 2^{\mathbb{N}_0}$$

definiert durch

$$pm(w, v) = \{i \mid substr(w, i, i + |v| - 1) = v\}$$

formal beschreiben. $pm(w, v)$ gibt die Stellen (Indizes) von w an, an denen v in w vorkommt. Es gilt also z.B.

$$pm(aabaaa, aa) = \{1, 4, 5\}$$

Wir wollen uns ein Verfahren überlegen, welches für den Fall, dass v ein Teilwort von w ist, die Stelle des ersten Vorkommens bestimmt. Sei dazu $w = w_1 \dots w_k$, $v = v_1 \dots v_l$, $w_i, v_j \in \Sigma$, $1 \leq i \leq k$, $1 \leq j \leq l$, $k, l \geq 0$. Ein naives Verfahren ist das folgende:

- Durchlaufe das Wort w von links nach rechts so weit, dass v aufgrund seiner Länge prinzipiell noch Teilwort von w sein kann, d.h. durchlaufe w von w_1 bis maximal w_{k-l+1} .
- Prüfe bei jedem w_i , $1 \leq i \leq k-l+1$, ob

$$w_i \dots w_{i+l-1} = v_1 \dots v_l \quad (2.8)$$

ist. Falls ja, kommt v in w erstmalig ab der Stelle i vor, d.h.

$$\text{substr}(w, i, i+l-1) = v$$

und i ist die kleinste Zahl mit dieser Eigenschaft. Falls (2.8) nicht gilt, d.h. es gibt ein j mit $w_{i+j-1} \neq v_j$, $1 \leq j \leq l$, muss die Prüfung (2.8) ab dem nächsten Buchstaben w_{i+1} von w durchgeführt werden.

Folgende Darstellung veranschaulicht das Verfahren:

$$\begin{array}{ccc} w_1 \dots w_{i-1} & v_1 \dots v_l & w_{i+l} \dots w_k \\ & w_i \dots w_{i+l-1} & \end{array}$$

Eine programmiersprachliche Formulierung (PASCAL-ähnlich) des Verfahrens lautet wie folgt:

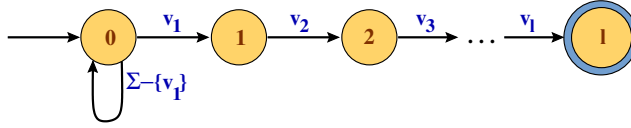
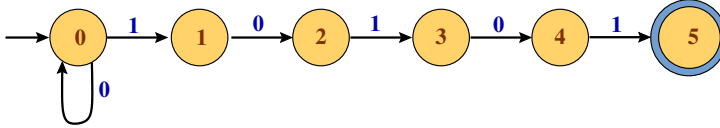
```

for i := 1 to k - l + 1 do begin
  nochgleich := true;
  j := 1;
  while (j ≤ l) and nochgleich do
    if wi+j-1 ≠ vj then nochgleich := false
    else j := j + 1;
  if nochgleich then write('v kommt an der Stelle', i, 'in w vor')
end;
```

Welchen Aufwand (Anzahl der Vergleiche) hat dieses Verfahren im schlimmsten Fall (englisch: worst case)? Der schlimmste Fall liegt dann vor, wenn v nicht in w vorkommt. Dann muss für alle w_i , $1 \leq i \leq k-l+1$, also $k-l+1$ -mal, der Vergleich (2.8) durchgeführt werden. Dieser Vergleich bedeutet für alle j , $1 \leq j \leq l$, w_{i+j-1} mit v_j zu vergleichen. Das sind l Vergleiche. Insgesamt müssen also $(k-l+1) \cdot l$ Vergleiche durchgeführt werden. In der Regel wird l , die Länge des gesuchten Wortes, im Verhältnis zu k , der Länge des zu durchsuchenden Wortes, sehr klein sein, so dass $(k-l+1) \cdot l$ ungefähr gleich $k \cdot l = |w| \cdot |v|$ ist.

Im Fall der erfolgreichen Suche wird im Mittel (englisch: average case) die Hälfte der Buchstaben $w_1 \dots, w_{k-l+1}$ durchlaufen werden, d.h. die Anzahl der Vergleiche ist dann ungefähr gleich $\frac{k \cdot l}{2} = \frac{1}{2} |w| \cdot |v|$.

Es gibt wesentlich effizientere Verfahren, das Pattern-Matching-Problem mit weniger Vergleichen zu lösen. Wir stellen eines vor, welches das Problem mithilfe endlicher Automaten löst.

Bild 2.34: Automat, der $v = (\Sigma - \{v_1\})^* \{v_1 \dots v_l\}$ akzeptiert.Bild 2.35: Automat A_v , der $\{0\}^* \{10101\}$ akzeptiert.

Zunächst konstruieren wir zum gesuchten Wort $v = v_1 \dots v_l$ den deterministischen endlichen Automaten A_v mit dem Zustandsdiagramm in Bild 2.34. Es gilt $L(A_v) = (\Sigma - \{v_1\})^* \circ \{v\}$.

Im nächsten Schritt vervollständigen wir A_v so, dass gilt: Verarbeitet A_v das Wort $w = w_1 \dots w_k$, dann ist A_v im Zustand j , $1 \leq j \leq l$, genau dann, wenn die letzten j von A_v gelesenen Buchstaben von w mit den ersten j Buchstaben von v übereinstimmen und es keinen Index kleiner als j mit dieser Eigenschaft gibt. Es liegt also folgende Situation vor:

$$w = w_1 \dots w_{i-1} v_1 \dots v_j w_{i+j} \dots w_k$$

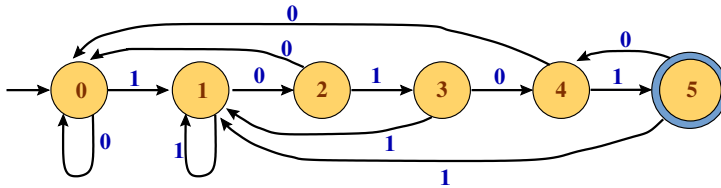
Es ist also $w_i \dots w_{i+j-1} = v_1 \dots v_j$. Liest A_v den nächsten Buchstaben w_{i+j} von w , gibt es zwei Möglichkeiten:

1. $w_{i+j} = v_{j+1}$, d.h. der nächste Buchstabe von w stimmt mit dem nächsten Buchstaben von v überein: A_v geht in den Zustand $j + 1$ über;
2. $w_{i+j} \neq v_{j+1}$: A_v geht in den Zustand $r \leq j$ über, wobei r die größte Zahl ist, so dass $v_1 \dots v_r$ Suffix von $v_1 \dots w_{i+j}$ ist. Denn offensichtlich stimmen dann die letzten r von w gelesenen Buchstaben mit den ersten r von v überein, und r ist der größte Index mit dieser Eigenschaft.

Wir wollen das vorgestellte Verfahren an einem Beispiel durchführen. Sei $\Sigma = \{0, 1\}$ ein Alphabet und das zu suchende Wort $v = 10101$.

Im ersten Schritt erhalten wir A_v , der $\{0\}^* \{10101\}$ akzeptiert (siehe Bild 2.35). Jetzt müssen wir A_v vervollständigen: Zustand 0 ist bereits vervollständigt. Wird im Zustand 1 eine 1 gelesen, wurde bis dahin 11 gelesen. Das größte Suffix von 11, das mit v am Anfang übereinstimmt, ist 1. Bei Eingabe von 1 geht A_v in den Zustand 1 über. Also fügen wir zu A_v einen 1-Übergang vom Zustand 1 zu sich selbst hinzu.

Wird im Zustand 2 eine 0 gelesen, wurde bis dahin 100 gelesen. Kein Suffix von 100 stimmt mit irgendeinem Anfangsstück von v überein (anders ausgedrückt: Nur

Bild 2.36: Vervollständigter Automat A_v .

das Suffix ε von 100 stimmt mit dem Präfix ε von v überein). Deshalb führen wir in A_v einen 0-Übergang vom Zustand 2 zum Zustand 0 ein.

Das Endergebnis der Vervollständigung von A_v zeigt Bild 2.36. Die Vervollständigung von Zustand 5 geschieht wie folgt: Wird eine 1 gelesen, ist das bis dahin gelesene Wort 101011. Nur die letzte 1 davon stimmt mit einem Präfix von v , nämlich mit 1, überein. 1 wird im Zustand 1 erkannt, also fügen wir einen 1-Übergang von Zustand 5 zum Zustand 1 ein. Wird im Zustand 5 eine 0 gelesen, wurde bis dahin 101010 gelesen. Das größte Suffix davon, das mit einem Anfangsstück von v übereinstimmt, ist 1010. Dieses Wort wird im Zustand 4 erkannt, also enthält A_v einen 0-Übergang vom Zustand 5 zum Zustand 4.

Mithilfe des zu einem Muster v konstruierten Automaten A_v wird nicht nur die erste, sondern jede Stelle im Wort w gefunden, an der v vorkommt. Dazu wird ein Zähler vor Beginn der Eingabe von w in A_v auf den Wert $-|v| + 1$ gesetzt. Der Zähler wird beim Abarbeiten von w bei jedem Zustandsübergang um eins erhöht. Immer dann, wenn der Endzustand erreicht wird, gibt der Zähler eine Stelle in w an, an der v vorkommt, und alle diese Stellen werden aufsteigend angegeben.

Wie hoch ist der Aufwand für dieses Verfahren? Das Verfahren besteht aus zwei Teilen: die Konstruktion von A_v und das Abarbeiten von Wörtern w durch A_v . Zu einem gegebenen Muster v kann der Automat A_v in $c \cdot |v|$ Schritten konstruiert werden. Dabei ist c eine positive reellwertige Konstante, die weder von der Länge von v noch von den Längen der Wörter w abhängt. Das Abarbeiten eines Wortes w durch A_v benötigt genau $|w|$ Schritte. Damit benötigt das vorgestellte Verfahren $c \cdot |v| + |w|$ Schritte, was in der Regel wesentlich weniger als beim eingangs betrachteten naiveren Verfahren mit dem Aufwand $\frac{1}{2} \cdot |v| \cdot |w|$ ist.

Das Ausgangsproblem der Betrachtungen dieses Abschnitts, ein Wort oder einen Text v in einem gegebenen Text w zu suchen, lässt sich nicht nur auf die Mustererkennung und Bildverarbeitung verallgemeinern, sondern auch auf das *Suchen im Web*. Auch hier geht es darum, ein vorgegebenes Wort in einer Menge von HTML-Dokumenten zu finden (bzw. sämtliche Vorkommen aufzufinden), um die gefundenen Dokumente (oder zumindest deren Web-Adressen) dann in einer bestimmten Reihenfolge auszugeben.

Eine weitere Anwendung, die sich formal in völlig analoger Weise formulieren lassen, ist die lexikalische Analyse eines Compilers für eine Programmiersprache.

2.6.3 Weitere Anwendungen

In allen Bereichen der Softwaretechnik, z.B. in der Anforderungsanalyse, beim Softwareentwurf, bei der Spezifikation, bei der Implementierung, finden ebenfalls Methoden und Techniken Anwendung, die mittelbar oder unmittelbar endliche Automaten als Hilfsmittel verwenden.

Bei der dynamischen Modellierung verwenden gängige objektorientierte Methoden wie UML (Unified Modeling Language) Zustandsdiagramme. Zustandsübergänge repräsentieren dabei Ereignisse, die – möglicherweise abhängig von einer Bedingung – einen Zustandswechsel bewirken.

Unser Eingangsbeispiel in diesem Kapitel, der Eintrittsautomat A_{Eintritt} , ist ein Beispiel für die Modellierung eines Mensch-Maschine-Dialogs. Viele Softwareprojekte beinhalten die Gestaltung von Mensch-Computer-Schnittstellen. Zur Modellierung von Dialogabläufen eignen sich so genannte *Interaktionsdiagramme* (kurz: IAD). Interaktionsdiagramme sind Erweiterungen von endlichen Automaten. Sie bestehen aus Zuständen, die die Situationen beschreiben, in denen das System auf Eingaben wartet, Zustandsübergängen, welche so genannte virtuelle Tasten repräsentieren, die den weiteren Ablauf eines Dialogs steuern, und Aktionen, die bei Zustandsübergängen ausgeführt werden. Bild 2.37 beschreibt den Dialog mit dem Eintrittsautomaten mithilfe eines Interaktionsdiagramms, wobei Zustände durch Kreise und Aktionen durch Rechtecke dargestellt sind.

Das Konzept der endlichen Automaten findet also vielfältige Anwendungen bei Problemlösungen in unterschiedlichen Bereichen der Praktischen, der Angewandten und der Technischen Informatik. Beispiele sind: Das Pattern-Matching-Problem in der Textverarbeitung (allgemeiner: Mustererkennung und -verarbeitung), die Modellierung von Zuständen und Zustandsüberführungen in der Systemprogrammierung, die dynamische Modellierung in der objektorientierten Softwareentwicklung, die Beschreibung von Mensch-Computer-Dialogen.

2.7 Bibliographische Hinweise und Ergänzungen

Wie am Ende von Kapitel 1 bereits erwähnt, findet man weitere Darstellungen zu endlichen Automaten in den dort genannten Lehrbüchern zur Automatentheorie; diese generellen Literaturhinweise werden daher an dieser Stelle nicht mehr wiederholt.

Oberschelp und Vossen (2006) verwenden endliche Automaten zur Beschreibung von Rechnern mit endlichem Speicher und zeigen damit, dass das Konzept der endlichen Automaten zu diesem Zweck (wenigstens im Prinzip) ausreicht; man vergleiche hierzu auch Kapitel 4. Die bekanntesten Verfahren zur Teilworterkennung bzw. zum Pattern Matching stammen von Boyer und Moore (1977) sowie von Knuth et al. (1977); man vergleiche hierzu auch die Darstellung bei Ottmann und Widmayer (2011). Hitz et al. (2005) behandelt die heute weit verbreitete Modellierungssprache UML.

Neben den hier beschriebenen endlichen Automaten werden in der Literatur auch Automaten auf *unendlichen* Wörtern sowie *Baumautomaten* untersucht. Bei der ers-

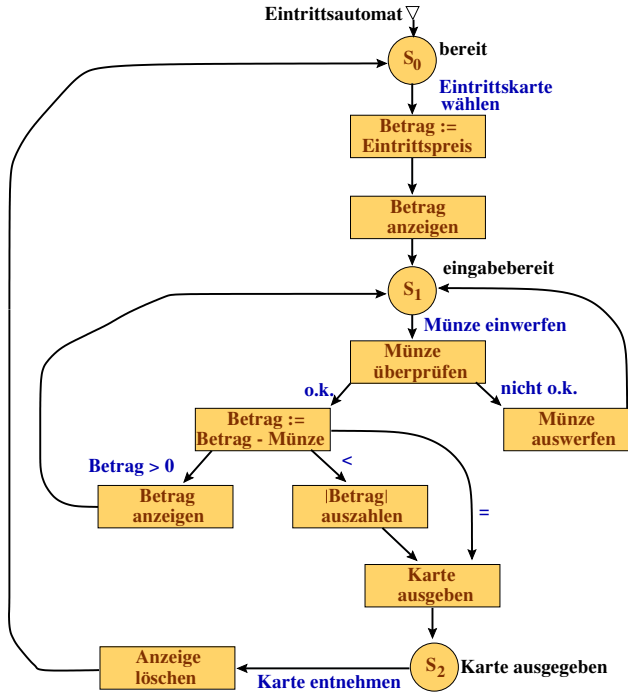


Bild 2.37: Interaktionsdiagramm für den Eintrittsautomaten.

teren Kategorie, auch *Büchi-Automaten* genannt, handelt es sich um nichtdeterministische endliche Automaten, die unendlich lange Wörter, so genannte ω -Wörter, erkennen können. Ein solcher Automat *erkennt* ein ω -Wort, falls der Automat beim Abarbeiten des Wortes (im üblichen Sinne) eine Folge von Zuständen durchläuft, in welcher ein Endzustand unendlich oft vorkommt. Baumautomaten operieren auf (binären) Bäumen, deren sämtliche Knoten mit Buchstaben aus oder Wörtern über einem gegebenen Alphabet beschriftet sind. Sie verallgemeinern sequentielle Automaten der in diesem Kapitel betrachteten Art dadurch, dass sie ausgehend von der Wurzel des Baumes die Pfade zu den Blättern *parallel* durchlaufen. Sowohl für Büchi- wie für Baumautomaten sind eine Reihe der Resultate über (gewöhnliche) endliche Automaten ebenfalls nachweisbar; das gleiche gilt für die von diesen Automaten erkannten Sprachen im Vergleich zu den regulären Sprachen. Des Weiteren gibt es tiefer gehende Beziehungen zwischen Aspekten der mathematischen Logik und endlichen Automaten sowie zwischen endlichen Automaten, (unendlichen) Spielen und der Verifikation reaktiver Systeme. Einen Einstieg in alle diese Thematiken findet man etwa bei Thomas (1990, 1997, 1999, 2002) und Grädel et al (2002). Man vergleiche hierzu auch die Einführung von Thomas (2010) in das Themenheft zur Theoretischen Informatik des Informatik-Spektrums sowie die Übersicht von Björklund et al. (2010) in die Zusammenhänge zwischen Automaten und mathematischer Logik.

2.8 Übungen

- 2.1 Erweitern Sie den Eintrittsautomaten A_{Eintritt} aus Abschnitt 2.1.1 so, dass dieser neben Münzen auch EC-Karten akzeptiert.
- 2.2 Konstruieren Sie in Analogie zum Automaten A_{Eintritt} einen Automaten A_{Geld} , der die Funktionsweise moderner Bankautomaten auf abstrakter Ebene nachbildet (d.h. Ausgabe von Geldscheinen in Stückelung 5, 10, 20, 50, 100 und 200 Euro bis zu einem Limit von 1.000 bzw. 200 Euro pro Tag bei Karten des eigenen oder eines fremden Geldinstituts, ferner Karteneinzug nach dreimaliger Eingabe einer falschen PIN).
- 2.3 a) *integer*-Zahlen in einer Programmiersprache müssen syntaktisch eindeutig definiert werden, damit der Compiler überprüfen kann, ob eine Zeichenfolge eine syntaktisch korrekt gebildete *integer*-Zahl ist. In der Programmiersprache PASCAL bestehen *integer*-Zahlen aus Ziffernfolgen ohne führende Nullen, mit oder ohne Vorzeichen. Geben Sie einen endlichen Automaten an, der die Menge der syntaktisch korrekten PASCAL-*integer*-Zahlen akzeptiert.
- b) Erweitern Sie Ihren Automaten aus a) zu einem Automaten, der *real*-Zahlen akzeptiert. Eine *real*-Zahl besteht aus einer *integer*-Zahl, die einen Dezimal- und einen Exponentialanteil haben kann (beide oder einer von beiden kann fehlen). Der Dezimalanteil beginnt mit einem Punkt, dem eine beliebige nicht leere Ziffernfolge folgen muss. Der Exponentialanteil beginnt mit dem Symbol E, dem eine *integer*-Zahl folgen muss.
- 2.4 Konstruieren Sie endliche Automaten, die die folgende Sprachen L_1, L_2, L_3 über dem Alphabet $\{a, b, c\}$ akzeptieren:
- a) L_1 enthält alle Wörter, die mindestens einmal das Suffix *abc* besitzen;
 - b) L_2 enthält alle Wörter, die genau einmal das Suffix *abc* enthalten;
 - c) L_3 enthält alle Wörter, die das Suffix *abc* nicht enthalten.
- 2.5 Konstruieren Sie einen endlichen Automaten, der die Sprache

$$L = \{w \in \{a, b\}^* \mid |w|_a = 2i, |w|_b = 3j, i, j \geq 0\}$$

akzeptiert.

- 2.6 a) Für $c \in \mathbb{N}$ sei $L_c = \{w \in \{a, b\}^* \mid a^c \text{ ist nicht Infix von } w\}$. Geben Sie eine formale Definition für eine Familie von vollständigen Automaten A_c an mit $L_c = L(A_c)$.
- b) Für $c \in \mathbb{N}$ sei $L_c = \{w \in \{a, b\}^* \mid a^c \text{ ist Infix von } w\}$. Geben Sie eine formale Definition für eine Familie von vollständigen Automaten A_c an mit $L_c = L(A_c)$.

2.7 *ISBN 10*-Nummern⁶ sind alle Folgen $x_1 \dots x_{10}$ der Länge 10 über dem Alphabet $\{0, 1, \dots, 9, X\}$, die genau die folgenden Eigenschaften erfüllen:

- (1) $x_i \in \{0, 1, \dots, 9\}$, $1 \leq i \leq 9$: Die ersten 9 Ziffern dürfen nicht X sein.
- (2) $x_{10} \in \{0, 1, \dots, 9, X\}$: x_{10} kann auch X sein. X steht für die „Ziffer“ 10.
- (3) Es muss $\sum_{i=1}^{10} i \cdot x_i = 0 \pmod{11}$ gelten: Die Summe der mit der Stelle gewichteten Ziffern muss durch 11 teilbar sein.

Konstruieren Sie einen endlichen Automaten, der gerade die *ISBN 10*-Nummern akzeptiert.

2.8 Zeichnen Sie das Zustandsdiagramm des gemäß der allgemeinen Konstruktion von Satz 2.2 konstruierten Automaten A_d . Vergleichen Sie dieses mit dem in Bild 2.13 dargestellten Zustandsdiagramm des unmittelbar vor dem Satz 2.2 konstruierten deterministischen Automaten A_{1_d} für die Sprache der Bitfolgen mit drittletztem Bit gleich 0.

2.9 Konstruieren Sie zu dem nichtdeterministischen endlichen Automaten

$$A = (\{a, b\}, \{p, q, r\}, \delta, p, \{r\})$$

mit

δ	p	q	r
a	$\{p, q\}$	$\{p, q, r\}$	$\{\}$
b	$\{r\}$	$\{\}$	$\{q, r\}$

einen äquivalenten deterministischen.

2.10 Konstruieren Sie zu dem nichtdeterministischen endlichen Automaten

$$A = (\{a\}, \{0, 1, 2, 3, 4, 5\}, \delta, 0, \{0, 2, 5\})$$

mit

δ	0	1	2	3	4	5
a	$\{1, 3\}$	$\{2\}$	$\{1, 3\}$	$\{4\}$	$\{5\}$	$\{1, 3\}$

einen äquivalenten deterministischen.

2.11 Minimieren Sie den Automaten

$$A = (\{0, 1\}, \{s_0, s_1, s_2, s_3, s_4\}, \delta, s_0, \{s_4\})$$

mit dem in Bild 2.38 dargestellten Zustandsdiagramm. Beschreiben Sie $L(A)$ umgangssprachlich, und geben Sie $L(A)$ formal an.

⁶*ISBN* ist eine Abkürzung für „International Standard Book Number“.

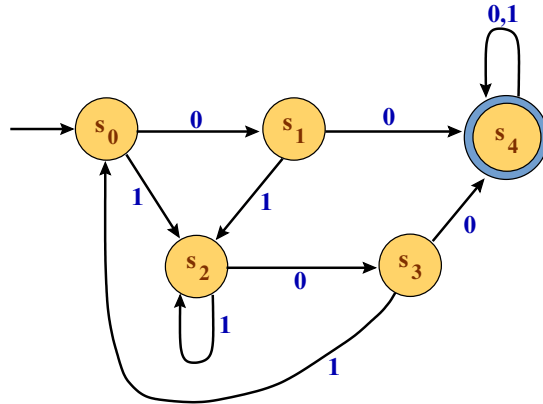


Bild 2.38: Zustandsdiagramm von A.

2.12 Minimieren Sie den endlichen Automaten

$$A = (\{a, b, c\}, \{s_0, s_a, s_c, s_{aa}, s_{cc}, g_{ac}, u_{ac}\}, \delta, s_0, \{s_0, s_{aa}, s_{cc}, g_{ac}\})$$

mit

δ	s_0	s_a	s_c	s_{aa}	s_{cc}	g_{ac}	u_{ac}
a	s_a	s_{aa}	g_{ac}	s_a	u_{ac}	u_{ac}	g_{ac}
b	s_0	s_a	s_c	s_{aa}	s_{cc}	g_{ac}	u_{ac}
c	s_c	g_{ac}	s_{cc}	u_{ac}	s_c	u_{ac}	g_{ac}

Grundkurs Theoretische Informatik

Eine anwendungsbezogene Einführung - Für
Studierende in allen Informatik-Studiengängen

Vossen, G.; Witt, K.-U.

2016, XVIII, 485 S. 142 Abb., Softcover

ISBN: 978-3-8348-1770-9