Theoretische Informatik
Summary

September 30, 2020

Chapter 1

Alphabete, Wörter, Sprachen und die Darstellung von Problemen

1.1 2.2 Alphabete, Wörter und Sprachen

D2.1: Eine endliche nichtleere Menge $\sum$ heisst **Alphabet**. Die Elemente eines Alphabets werden **Buchstaben (Zeichen, Symbole)** genannt.

Häufig verwendete Alphabete:

- $\sum_{bool} = \{0, 1\}$
- $\sum_{lat} = \{a, b, c, ..., z\}$
- $\sum_{m} = \{0, 1, 2, ..., m-1\}$
- $\sum_{logic} = \{0, 1, x, (,), AND, OR, NOT\}$

D2.2: Ein **Wort** über $\sum$ ist eine endliche (eventuell leere) Folge von Buchstaben aus $\sum$. Das leere Wort λ (manchmal ϵ) ist die leere Buchstabenfolge. Die **Länge** $|w|$ eines Wortes w ist die Anzahl der Vorkommen von Buchstaben in w.

- $\sum^*$: Menge aller Wörter über $\sum$
- $\sum^+ = \sum^* - \{\lambda\}$

Das leere Wort λ ist ein Wort über jedem Alphabet.

Verabredung: Wir werden Wörter ohne Komma schreiben

D2.3: Die **Verkettung (Konkatenation)** für ein Alphabet $\sum$ ist eine Abbildung

$$Kon(x, y) = x \cdot y = xy$$

für alle $x, y \in \sum^*$ Die Verkettung ist eine assoziative Operation und $(\sum^*, Kon)$ ist eine Halbgruppe(Monoid) mit neutralen element λ
Für alle $x, y \in \sum^*$ gilt:

$$|xy| = |x \cdot y| = |x| + |y|$$

D2.4: Für ein Wort $a = a_1a_2a_3...a_n$ mit $a_i \in \sum$ für $i \in \{1, 2, ..., n\}$, bezeichnet $a^R = a_na_{n-1}...a_1$ die **Umkehrung** von a.
D2.5: Sei $\sum$ ein Alphabet. Für alle $x \in \sum^*$ und alle $i \in \mathbb{N}$ definieren wir die i-te **Iteration** x^i von x als

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

Beispiel: aabbaaaaaa $= a^2b^2a^6$

D2.6: Seien v,w $\in \sum^*$ für ein Alphabet $\sum$

- v heisst ein **Teilwort** von w $\iff \exists x, y \in \sum^* : w = xvy$
- v heisst ein **Präfix** von w $\iff \exists y \in \sum^* : w = vy$
- v heisst ein **Suffix** von w $\iff \exists x \in \sum^* : w = xv$

D2.7:

- $|x|_a$ ist die Anzahl der Vorkommen von a in x
- $|A|$ die KArdinalität der Menge A
- $\mathbf{P(A)} = \{S | S \subseteq A\}$ die Potenzmenge von A

D2.8: Sei $\sum = \{s_1, s_2, ..., s_m\}, m \geq 1$ ein Alphabet und sei $s_1 < s_2 < ... < s_m$ eine Ordnung auf $\sum$. Wir definieren die **kanonische Ordnung** auf $\sum^*$ für u,v $\in \sum^*$ wie folgt:

$$u < v \iff |u| < |v| \vee |u| = |v| \wedge u = x \cdot s_i \cdot u' \wedge v = x \cdot s_j \cdot v' \text{ für irgendwelche } x, u', v' \in \sum^* \text{ und } i < j$$

D2.9:

Definition 2.9. *Eine* ***Sprache*** L *über einem Alphabet* Σ *ist eine Teilmenge von* Σ^*. *Das Komplement* $\boldsymbol{L^{\complement}}$ *der Sprache* L *bezüglich* Σ *ist die Sprache* $\Sigma^* - L$.

$\boldsymbol{L_\emptyset} = \emptyset$ *ist die* ***leere Sprache***.

$\boldsymbol{L_\lambda} = \{\lambda\}$ *ist die einelementige Sprache, die nur aus dem leeren Wort besteht.*

Sind L_1 *und* L_2 *Sprachen über* Σ, *so ist*

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

die ***Konkatenation*** *von* L_1 *und* L_2. *Ist* L *eine Sprache über* Σ, *so definieren wir*

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

$$\boldsymbol{L^*} = \bigcup_{i \in \mathbb{N}} L^i \text{ und } \boldsymbol{L^+} = \bigcup_{i \in \mathbb{N}-\{0\}} L^i = L \cdot L^*.$$

L^* *nennt man den* ***Kleene'schen Stern*** *von* L.

Die folgenden Mengen sind Sprachen über dem Alphabet $\Sigma = \{a, b\}$:

- $L_1 = \emptyset$,
- $L_2 = \{\lambda\}$,
- $L_3 = \{\lambda, ab, abab\}$,
- $L_4 = \Sigma^* = \{\lambda, a, b, aa, \ldots\}$,
- $L_5 = \Sigma^+ = \{a, b, aa, \ldots\}$,
- $L_6 = \{a\}^* = \{\lambda, a, aa, aaa, \ldots\} = \{a^i \mid i \in \mathbb{N}\}$,
- $L_7 = \{a^p \mid p \text{ ist eine Primzahl}\}$,
- $L_8 = \{a^i b^{2i} a^i \mid i \in \mathbb{N}\}$,
- $L_9 = \Sigma$,
- $L_{10} = \Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.

L2.1 Seien L_1, L_2, L_3 Sprachen über einem Alphabet $\sum$. Dann gilt:

$$L_1 L_2 \cup L_1 L_3 = L_1 (L_2 \cup L_3)$$

L 2.2 Seien L_1, L_2, L_3 Sprachen über einem Alphabet $\sum$. Dann gilt:

$$L_1 (L_2 \cap L_3) \subseteq L_1 L_2 \cap L_1 L_3)$$

L2.3 Es existieren $U_1, U_2, U_3 \in (\sum_{bool})^*$ so dass

$$U_1 (U_2 \cap U_3) \subset U_1 U_2 \cap U_1 U_3$$

D2.10: Seiene $\sum_1$ und $\sum_2$ zwei beleibige Alphabete. Ein **Homomorphismus** von $\sum_1^*$ nach $\sum_2^*$ ist jede FUnktion $h : \sum_1^* \to \sum_2^*$ mit den folgendend Eigenschaften

- $h(\lambda) = \lambda$
- $h(uv) = h(u) \cdot h(v)$ für alle $u, v \in \sum_1^*$

1.2 2.3 Algorithmische Probleme

D2.11: Das**Entscheidungsproblem ($\sum$,L)** für ein gegebenes Alphabet $\sum$ und eine gegebene Sprache $L \subseteq \sum^*$ ist, für jedes $x \in \sum^*$ zu entscheiden, ob

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

Wenn ein Algorithmus A die Entscheidungsproblem löst sagen wir auch, dass A die sprache L **erkennt**.
Wenn für eine Sprache L ein Algorithmus existiert, der L erkennt sagen wir dass L **rekursiv** ist

Üblicherweise stellen wir ein Entscheidungsproblem (Σ, L) wie folgt dar:

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

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

Beispielsweise ist $(\{a,b\}, \{a^n b^n \mid n \in \mathbb{N}\})$ ein Entscheidungsproblem, das man auch folgendermaßen darstellen kann:

Eingabe: $x \in \{a,b\}^*$.
Ausgabe: Ja, falls $x = a^n b^n$ für ein $n \in \mathbb{N}$.
Nein, sonst.

D2.12: Seien $\sum$ und Γ zwei Alphabete. Wir sagen dass ein Algorithmus A eine **Funktion (Transformation)** $f : \sum^* \to \Gamma^*$ **berechnet (realisiert)** falls

$$A(x) = f(x) \text{ für alle } x \in \sum^*$$

D2.13: Seien $\sum$ und Γ zwei Alphabete und sei $R \subseteq \sum^* \times \Gamma^*$ eine Relation in $\sum^*$ und Γ^*. Ein Algorithmus **A berechnet R (oder löst das Relationsproblem R)** falls für jedes $x \in \sum^*$, für das ein $y \in \Gamma^*$ mit $(x,y) \in R$ existiert gilt:

$$(x, A(x)) \in R$$

Sei $R_{\text{fac}} \subseteq (\Sigma_{\text{bool}})^* \times (\Sigma_{\text{bool}})^*$, wobei $(x,y) \in R_{\text{fac}}$ genau dann, wenn entweder Nummer(y) ein Faktor[6] von Nummer(x) ist, oder $y = 1$, wenn Nummer(x) eine Primzahl ist, oder $y = 0$, wenn $x \in \{0,1\}$. Eine anschauliche Darstellung dieses Relationsproblems könnte wie folgt aussehen.

Eingabe: $x \in (\Sigma_{\text{bool}})^*$.
Ausgabe: $y \in (\Sigma_{\text{bool}})^*$, wobei

$$\text{Nummer}(y) = \begin{cases} 0, & \text{falls } x = 0 \text{ oder } x = 1, \\ 1, & \text{falls } x \text{ ist eine Primzahl}, \\ k, & \text{sonst, wobei } k \text{ ein Faktor von Nummer}(x) \text{ ist}. \end{cases}$$

D2.14: Optimierungsproblem:

Definition 2.14. *Ein* ***Optimierungsproblem*** *ist ein 6-Tupel* $\mathcal{U} = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \text{cost}, \text{goal})$, *wobei:*

(i) Σ_I *ist ein Alphabet (genannt* ***Eingabealphabet****),*

(ii) Σ_O *ist ein Alphabet (genannt* ***Ausgabealphabet****),*

(iii) $L \subseteq \Sigma_I^*$ *ist die Sprache der* ***zulässigen Eingaben*** *(als Eingaben kommen nur Wörter in Frage, die eine sinnvolle Bedeutung haben). Ein* $x \in L$ *wird ein* ***Problemfall*** *(****Instanz****)* ***von*** $\mathcal{U}$ *genannt.*

(iv) $\mathcal{M}$ *ist eine Funktion von* L *nach* $\mathcal{P}(\Sigma_O^*)$, *und für jedes* $x \in L$ *ist* $\mathcal{M}(x)$ *die* ***Menge der zulässigen Lösungen für*** x,

(v) cost *ist eine Funktion,* $\text{cost} : \bigcup_{x \in L} (\mathcal{M}(x) \times \{x\}) \to \mathbb{R}^+$, *genannt* ***Kostenfunktion****,*

(vi) goal $\in \{\text{Minimum}, \text{Maximum}\}$ *ist das* ***Optimierungsziel****.*

Eine zulässige Lösung $\alpha \in \mathcal{M}(x)$ *heißt* ***optimal*** *für den Problemfall* x *des Optimierungsproblems* U, *falls*

$$\text{cost}(\alpha, x) = \mathbf{Opt}_{\mathcal{U}}(x) = \text{goal}\{\text{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}.$$

Ein Algorithmus A ***löst*** $\mathcal{U}$, *falls für jedes* $x \in L$

(i) $A(x) \in \mathcal{M}(x)$, $\quad \{A(x)$ ist eine zulässige Lösung des Problemfalls x von $\mathcal{U}.\}$

(ii) $\text{cost}(A(x), x) = \text{goal}\{\text{cost}(\beta, x) \mid \beta \in \mathcal{M}(x)\}$.

Falls goal $=$ *Minimum, ist* $\mathcal{U}$ *ein* ***Minimierungsproblem****; falls* goal $=$ *Maximum, ist* $\mathcal{U}$ *ein* ***Maximierungsproblem****.*

Teilproblem: Ein Optimierungsproblem $U_1 = (\sum_I, \sum_O, L', M, cost, goal)$ ist ein Teilproblem des Optimierungsproblems $U_2 = (\sum_I, \sum_O, L, M, cost, goal)$ falls $L' \subseteq L$

Knotenüberdeckung: Eine Knotenüberdeckung eines Graphen ist jede Knotenmenge $U \subseteq V$, so dass jede Kante aus E zu mindestens einem Knoten aus U inzident ist.

D2.15: Sei $\sum$ ein Alphabet, und sei $x \in \sum^*$. Wir sagen, dass ein Algorithmus A das Wort x **generiert**, falls A für die Eingabe λ die Ausgabe x liefert.

D2.16: Sei $\sum$ ein Alphabetm und sei $L \subseteq \sum^*$. A ist ein **Aufzählungsalgorithmus für L**, falls A für jede Eingabe $n \in \mathbb{N} - \{0\}$ die Wortfolge $x_1, x_2, ..., x_n$ ausgibt, wobei $x_1, x_2, ..., x_n$ die kanonisch n ersten Wörter in L sind.

1.3 Kolmogorov-Komplexität

Komprimierung Die Erzeugung einer kürzeren Darstellung eines Wortes x.

D2.17: Für jedes Wort $x \in (\sum_{bool})^*$ ist die **Kolmogorov-Komplexität K(x)** des Wortes x das Minimum der binären Längen der Pascal-Programme die x generieren.

L2.4 Es existiert eine Konstante d, so dass für jede $x \in (\sum_{bool})^*$

$$K(x) \leq |x| + d$$

D2.18: Die Kolmogorov-Komplexität einer natürlichen Zahl n ist

$$K(w_n) = K(Bin(n))$$

L 2.5 Für jede Zahl $n \in \mathbb{N} - \{0\}$ existiert ein Wort $w_n \in (\sum_{bool})^n$ so dass

$$K(w_n) \geq |w_n| = n$$

d.h es existiert für jede Zahl n ein nichtkomprimierbares Wort der Länge n

Satz 2.1 Seien A und B programmiersprachen. Es existiert eine Konstante $c_{A,B}$, die nur von A und B abhängt so dass

$$|K_A(x) - K_B(x)| \leq c_{A,B}$$

für alle $x \in (\sum_{bool})^*$

D2.19: Ein Wort $x \in (\sum_{bool})^*$ heisst **zufällig**, falls $K(x) \geq |x|$. Eine Zahl n heisst **zufällig**, falls

$$K(n) = K(Bin(n)) \geq \lceil log_2(n+1) \rceil - 1$$

Satz 2.2 Sei L eine Sprache über $\sum_{bool}$. Sei für jedes $n \in \mathbb{N} - \{0\}, z_n$ das n-te Wort in L bezüglich der kanonischen Ordnung. Wenn ein Programm A_L existiert, das das Entscheidungsproblem $(\sum_{bool}, L)$ löst, dann gilt für alle $n \in \mathbb{N} - \{0\}$, dass

$$K(z_n) \leq \lceil log_2(n+1) \rceil + c$$

wobei c eine von n unabhängige Konstante ist.

Satz 2.3 (Primzahlsatz) Die Anzahl der Primzahlen wächst so schnell wie die Funktion $\frac{n}{ln(n)}$

$$\lim_{n \to \infty} \frac{Prim(n)}{\frac{n}{ln(n)}} = 1$$

wobei Prim(n) ist die Anzahl der Primzahlen kleiner gleich n.

L2.6: Sei $n_1, n_2, n_3...$ eine steigende unendliche Folge natürlicher Zahlen mit $K(n_i) \geq \frac{\lceil log_2 n_i \rceil}{2}$. Für jedes $i \in \mathbb{N} - \{0\}$ sei q_i die grösste Primzahl, die die Zahl n_i teilt. Dann ist die Menge:

$$Q = \{q_i | i \in \mathbb{N} - \{0\}\}$$

unendlich. L2.6 zeigt dass es unendlich viele Primzahlen gibt und auch dass die Menge der grössten Primzahlfaktoren einer beliebigen unendlichen FOlge natuürlicherZahlen mit nichtrivialer Kolmogorov Komplexität unendlich ist.

Satz 2.4 Für unendlich viele $k \in \mathbb{N}$ gilt

$$Prim(k) \geq \frac{k}{2^{17} log_2(k) \cdot (log_2(log_2(k)))^2}$$

Chapter 2

Endliche Automaten

2.1 Die Darstellungen der endlichen Automaten

Ein Berechnungsmodell muss die folgende Fragen beantworten:

1. Welche elementaren Operationen, aus denen man die Programme zusammenstellen kann, stehen zur Verfügung?
2. Wie viel Speicher steht zur Verfügung und wie geht man mit dem Speicher um?
3. Wie wird die Eingabe eingegeben?
4. Wie wird die Ausgabe bestimmt (ausgegeben)?

Bei endlichen Automaten hat man nur den folgenden Speicher:

- Speicher in dem das Programm gespeichert wird
- Zeiger der auf die angewendete Zeile des Programms zeigt

Daraus folgt, das Programm darf keine Variablen benutzen.
⇒ Die Nummer der aktuellen Programmzeile, ist die einzige wechselnde Information

Wenn $\sum = \{a_1, a_2, ..., a_k\}$ das Alphabet ist über dem die Eingaben dargestellt sind, dann darf der endliche Automoat nur den folgenden Operationstyp benutzen:

$$\begin{aligned} \texttt{select } & input = a_1 \texttt{ goto } i_1 \\ & input = a_2 \texttt{ goto } i_2 \\ & \vdots \\ & input = a_k \texttt{ goto } i_k \end{aligned} \qquad \texttt{if } input = 1 \texttt{ then goto } i \texttt{ else goto } j.$$

Dass Rechte bild ist der Fall wenn $\sum$ nur aus zwei Symbole besteht.Die Bedeutung dieser Operation(Befehl) ist , dass man das nächste Eingabesymbol liest und mit $a_1, a_2, ..., a_k$ vergliecht. Wenn es gleich a_j ist, setzt das Programm die ARbeit in der Zeile i_j fort.
Wir nummerieren die Zeilen mit natürlichen Zahlen $0, 1, 2, 3...$ und die Arbeit des Programms beginnt immer in der Zeile 0.
Solche Programme benutzt man um Entscheidungsprobleme zu lösen. Die antwort is durch die Zeilennummer bestimmt. Wenn nach dem Lesen der gesamten Eingabe des Programm in der j-ten Zeile endet, und $j \in F$ (F eine Teilmenge von den Zeilen der Programm ist) dann akzeptiert das Programm die Eingabe sonst nicht. Bsp:

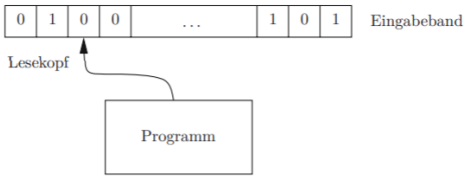
Betrachten wir als Beispiel folgendes Programm A, das Eingaben über dem Alphabet Σ_{bool} bearbeitet.

0: `if` $input = 1$ `then goto 1 else goto 2`
1: `if` $input = 1$ `then goto 0 else goto 3`
2: `if` $input = 0$ `then goto 0 else goto 3`
3: `if` $input = 0$ `then goto 1 else goto 2`

Setzen wir $F = \{0, 3\}$. Das Programm A arbeitet auf einer Eingabe 1011 wie folgt: Es startet in der Zeile 0, und geht in die Zeile 1, nachdem es eine 1 gelesen hat. Es liest eine

Die Berechnung endet in zeile 3, und weil $3 \in F$ wird das Wort 1011 akzeptiert

Endlichen Automaten verbindet man oft mit den Folgenden Modell:



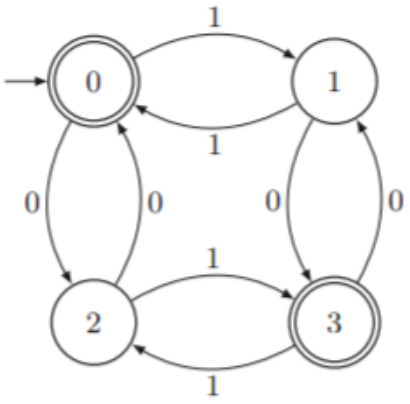
Die 3 Hauptkomponente des Modells sind:

- **Programm:** Gespeichert
- **Band:** Enthält das eingabe wort. Das Band betrachtet man als einen linearen Speicher für die Eingabe und besteht aus Feldern.
- **Lesekopf:** Kann sich auf dem Band nur von links nach rechts bewegen

Feld: Eine elementare Speichereinheit, die ein Symbol aus dem betrachteten Alphabet beinhalten kann

Diese Klasse von Programmen wird heute nicht mehr benutzt um endliche Automaten zu definieren, weil sie keine schöne Struktur haben. Statt dessen wird der folgender Darstellung der Programms verwendet:
Jeder Programm A wird einen gerichteten markierten Graphen $G(A)$ zugeordnet mit folgenden eigenschaften:

- G(A) hat genau so viele Knoten wie das Programm A Zeilen hat
- Jeder Zeile von A ist genau ein Knoten zugeordnet der durch die Nummer der Zeile markiert wird
- Falls das Programm A aus einer Zeile i in die Zeile j beim Lesen eines Symbols b übergeht, dann enthält G(A) eine gerichtete Kante (i,j) mit der Markierung b
- Jeder Knoten von G(A) hat genau den Ausgangsgrad $|\sum|$



Die Graphische darstellung der obigen Programm wobei doppelkreise deuten auf Elemente von F und der Knoten, der der Zeile 0 entspricht wird durch einen zusätzlichen Pfeil markiert.

D 3.1 Ein deterministischer **endlicher Automat (EA)** ist ein Quintupel:

$$M = (Q, \sum, \delta, q_0, F)$$

wobei:

1. Q eine endliche Menge von **Zuständen** ist (die Menge von Zeilen eines Programms ohne Variablen)
2. $\sum$ ein Alphabet, genannt **Eingabealphabet** ist (die zulässigen Eingaben sind all Wörter über $\sum$)
3. $q_0 \in Q$ ist der Anfangszustand (die Zeile 0 des Programms ohne Variablen)
4. $F \subseteq Q$ die **Menge der akzeptierten Zustände** ist
5. δ eine Funktion von $Q \times \sum$ nach Q ist, die **Übergangsfunktion** ($\delta(q, a) = p$ bedeutet, dass M in den Zustand p übergeht, falls M im Zustand q das Symbol a gelesen hat)

textbfKonfiguration: von M ist ein Element aus $Q \times \sum^*$ (Wenn M sich in einer Konfiguration $(q, w) \in Q \times \sum^*$ befindet, bedeutet das, dass M im Zustand q ist und noch das Suffix w eines Eingabewortes lesen soll textbf Startkonfiguration von M auf x: die Konfiguration $(q_0, x) \in \{q_0\} \times \sum^*$ (Die Arbeit (Berechnung) von M auf mus in der startkonfiguration anfangen)

textbfEndkonfiguration: Jede Konfiguration aus $Q \times \{\lambda\}$

textbf Schritt von M ist eine Relation (auf Konfigurationen)

$$\vdash_M \subseteq (Q \times \sum^*) \times (Q \times \sum^*)$$

definiert durch:

$$(q,w) \vdash_M (p,x) \iff w = ax, a \in \sum \text{ und } \delta(q,a) = p$$

Ein Schritt entspricht einer Anwendung der Übergangsfunktion auf die aktuelle Konfiguration in der sich M in einem Zustand q befindet und ein Eingabesymbol a liest.

*Eine **Berechnung** C von M ist eine endliche Folge $C = C_0, C_1, \ldots, C_n$ von Konfigurationen, so dass $C_i \vdash_M C_{i+1}$ für alle $0 \le i \le n-1$. C ist die **Berechnung von M auf einer Eingabe $x \in \Sigma^*$**, falls $C_0 = (q_0, x)$ und $C_n \in Q \times \{\lambda\}$ eine Endkonfiguration ist. Falls $C_n \in F \times \{\lambda\}$, sagen wir, dass C eine **akzeptierende Berechnung** von M auf x ist, und dass **M das Wort x akzeptiert**. Falls $C_n \in (Q-F) \times \{\lambda\}$, sagen wir, dass C eine **verwerfende Berechnung** von M auf x ist, und dass **M das Wort x verwirft** (**nicht akzeptiert**).*

{Man bemerke, dass M für jede Eingabe $x \in \Sigma^*$ genau eine Berechnung hat.}

*Die **von M akzeptierte Sprache $L(M)$** ist definiert als*

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

*$\mathcal{L}_{EA} = \{L(M) \mid M \text{ ist ein EA}\}$ ist die Klasse der Sprachen, die von endlichen Automaten akzeptiert werden. $\mathcal{L}_{EA}$ bezeichnet man auch als die **Klasse der regulären Sprachen**, und jede Sprache L aus $\mathcal{L}_{EA}$ wird **regulär** genannt.*

Benutzen wir noch einmal das Programm A, um die gegebene Definition der endlichen Automaten zu illustrieren. Der zum Programm A äquivalente EA ist $M = (Q, \Sigma, \delta, q_0, F)$ mit

$$Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0,1\}, F = \{q_0, q_3\} \text{ und}$$
$$\delta(q_0,0) = q_2, \delta(q_0,1) = q_1, \delta(q_1,0) = q_3, \delta(q_1,1) = q_0,$$
$$\delta(q_2,0) = q_0, \delta(q_2,1) = q_3, \delta(q_3,0) = q_1, \delta(q_3,1) = q_2.$$

Anschaulicher kann man die Übergangsfunktion δ durch Tabelle 3.1 beschreiben.
Die anschaulichste Darstellung eines EA ist aber die schon angesprochene graphische Form (Abbildung 3.2), die man für den EA in die in Abbildung 3.3 gegebene Form umwandeln kann.

Anschaulicher kann man die Übergangsfunktion δ durch Tabelle 3.1 beschreiben.
Die anschaulichste Darstellung eines EA ist aber die schon angesprochene graphische Form (Abbildung 3.2), die man für den EA in die in Abbildung 3.3 gegebene Form umwandeln kann.

Tabelle 3.1

Zustand	Eingabe 0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

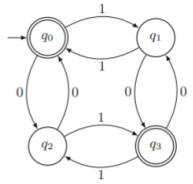


Abbildung 3.3

Die Berechnung von M auf der Eingabe 1011 ist

$$(q_0, 1011) \vdash_M (q_1, 011) \vdash_M (q_3, 11) \vdash_M (q_2, 1) \vdash_M (q_3, \lambda).$$

Weil $q_3 \in F$, ist $1011 \in L(M)$.
Die folgende Definition führt Bezeichnungen ein, die den formalen Umgang mit endlichen Automaten erleichtern.

D 3.2

Definition 3.2. *Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Wir definieren $\vdash_M^*$ als die reflexive und transitive Hülle der Schrittrelation $\vdash_M$ von M; daher ist*

$$(q,w) \vdash_M^* (p,u) \iff (q = p \text{ und } w = u) \text{ oder } \exists k \in \mathbb{N} - \{0\},$$

so dass

(i) $w = a_1 a_2 \ldots a_k u$, $a_i \in \Sigma$ *für* $i = 1, 2, \ldots, k$, *und*

(ii) $\exists r_1, r_2, \ldots, r_{k-1} \in Q$, *so dass*
$(q,w) \vdash_M (r_1, a_2 \ldots a_k u) \vdash_M (r_2, a_3 \ldots a_k u) \vdash_M \cdots \vdash_M (r_{k-1}, a_k u) \vdash_M (p,u)$.

Wir definieren $\hat{\delta}: Q \times \Sigma^* \to Q$ durch:

(i) $\hat{\delta}(q,\lambda) = q$ *für alle* $q \in Q$ *und*

(ii) $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w),a)$ *für alle* $a \in \Sigma$, $w \in \Sigma^*$, $q \in Q$.

Die Bedeutung von $(q,w) \vdash_M^* (p,u)$ ist, dass es eine Berechnung von M gibt, die ausgehend von der Konfiguration (q,w) zu der Konfiguration (p,u) führt. Die Aussage $\hat{\delta}(q,w) = p$ bedeutet, dass, wenn M im Zustand q das Wort w zu lesen beginnt, dann endet M im Zustand p (also $(q,w) \vdash_M^* (p,\lambda)$). Daher können wir schreiben:

$$L(M) = \{w \in \Sigma^* \mid (q_0,w) \vdash_M^* (p,\lambda) \text{ mit } p \in F\}$$
$$= \{w \in \Sigma^* \mid \hat{\delta}(q_0,w) \in F\}.$$

L 3.1

$$L(M) = \{w \in \{0,1\}^* || w|_0 + |w|_1 \equiv 0 mod 2\}$$

Jeder EA teilt die Menge $\sum^*$ in $|Q|$ Klassen auf

$$K1[p] = \{w \in \sum{}^{"} \mid \hat{\delta}(q_0,w) = p\} = \{w \in \sum{}^* \mid (q_0,w) \vdash_M^* (p,\lambda)\}$$

es gilt:

- $\bigcup_{p \in Q} K1[p] = \sum^*$
- $K1[p] \cap K1[q] = \emptyset \forall p, q \in Q, p \neq q$
- $L(M) = \bigcup_{p \in F} K1[p]$

Die Relation R_{delta} die durch

$$x R_\delta y \iff \hat{\delta}(q_0,x) = \hat{\delta}(q_0,y)$$

wird eine Äquivalenzrelation auf $\sum^*$, die die endlich vielen Klassen $K1[p]$ bestimmt

2.2 Simulationen

Jeder elementare Schritt der simulierten Berechnung wird durch einen Schritter der simulierenden Berechnung nachgemacht.

L 3.2 Sei $\sum$ ein Alphabet und seien $M_1 = (Q_1, \sum, \delta_1, q_{01}, F_1)$ und $M2 = (Q_2, \sum, \delta_2, q_{02}, F_2)$ zwei EA. Für jede Mengenoperation $\odot \in \{\cup, \cap, --\}$ existiert ein EA M, so dass

$$L(M) = L(M_1) \odot L(M_2)$$

Beweis Idee: Wir konstruieren M so, dass es die Arbeit der beiden Automaten M_1, M_2 simulieren kann. Die Zustände von M werden Paare (q,p) zugeordnet, wobei dass erste Element von (q,p) soll q sein genau dann, wenn sich M_1 gerade im Zustand q befindet. Analog das zweite Element für M_2

Konstruktion von M.
Sei $M = (Q, \Sigma, \delta, q_0, F_\odot)$, wobei

(i) $Q = Q_1 \times Q_2$,

(ii) $q_0 = (q_{01}, q_{02})$,

(iii) für alle $q \in Q_1$, $p \in Q_2$ und $a \in \Sigma$, $\delta((q,p),a) = (\delta_1(q,a), \delta_2(p,a))$,

(iv) falls $\odot = \cup$, dann ist $F = F_1 \times Q_2 \cup Q_1 \times F_2$,
{Mindestens einer von M_1 und M_2 endet in einem akzeptierenden Zustand.}
falls $\odot = \cap$, dann ist $F = F_1 \times F_2$, und
{M_1 und M_2 müssen beide akzeptieren.}
falls $\odot = -$, dann ist $F = F_1 \times (Q_2 - F_2)$.
{M_1 muss akzeptieren und M_2 darf nicht akzeptieren.}

Um die Behauptung $L(M) = L(M_1) \odot L(M_2)$ für jedes $\odot \in \{\cup, \cap, --\}$ zu beweisen, reicht es, die folgende Gleichheit zu zeigen:

$$\tilde{\delta}((q_{01}, q_{02}, x) = (\tilde{\delta_1}(q_{01}, x), \tilde{\delta_2}(q_{02}, x)) \text{ für alle } x \in \sum^*$$

Diese Gleichheit können wir mit induktion zeigen.

(a) *Induktionsanfang.*
Falls $x = \lambda$, ist (3.1) offenbar erfüllt.

(b) *Induktionsschritt.*
Wir beweisen für jedes $i \in \mathbb{N}$, dass, wenn (3.1) erfüllt ist für jedes $x \in \Sigma^*$ mit $|x| \leq i$, dann ist (3.1) erfüllt für jedes $w \in \Sigma^{i+1}$.
Sei w ein beliebiges Wort aus Σ^{i+1}. Dann ist $w = za$ für irgendwelche $z \in \Sigma^i$ und $a \in \Sigma$. Aus der Definition der Funktion $\hat{\delta}$ erhalten wir

$$\begin{aligned}
\hat{\delta}((q_{01}, q_{02}), w) &= \hat{\delta}((q_{01}, q_{02}), za) \\
&= \delta(\hat{\delta}((q_{01}, q_{02}), z), a) \\
&= \delta((\hat{\delta_1}(q_{01}, z), \hat{\delta_2}(q_{02}, z)), a) \\
&\overset{(3.1)}{=} (\delta_1(\hat{\delta_1}(q_{01}, z), a), \delta_2(\hat{\delta_2}(q_{02}, z), a)) \\
&\overset{\text{Def. } \delta}{=} (\hat{\delta_1}(q_{01}, za), \hat{\delta_2}(q_{02}, za)) \\
&= (\hat{\delta_1}(q_{01}, w), \hat{\delta_2}(q_{02}, w)).
\end{aligned}$$

□

2.3 Beweise der Nichtexistenz

Um zu zeigen, dass eine Sprache L nicht regulär ist ($L \notin \mathcal{L}_{EA}$), genügt es zu beweisen dass es keinen EA gibt, der die Sprache akzeptiert. Solcher Beweise nennt man **Beweise der Nichtexistenz**.
Wir wissen, dass endliche Automaten keine andere Speichermöglichkeit als den aktuellen Zustand besitzen. Das bedeutet für einen EA A, der nach dem Lesen zweier unterschiedlicher Wörter x und y im gleichen Zustand endet (also $\tilde{\delta}(q_0, x) = \tilde{\delta}(q_0, y)$), dass A in Zukunft nicht mehr zwischen x und y unterscheiden kann. D.h für all $z \in \sum^*$ gilt, dass

$$\tilde{\delta}_A(q_0, xz) = \tilde{\delta}_A(q_0, yz)$$

L 3.3 Sei $A = (Q, \sum, \delta_A, q_0, F)$ ein EA. Seien $x, y \in \sum^{"}, x \neq y$ so dass

$$(q_0, x) \vdash_A^* (p, \lambda) \text{ und } (q_0, y) \vdash_A^* (p, \lambda)$$

für ein $p \in Q$ (also $\tilde{\delta}_A(q_0, x) = \tilde{\delta}_A(q_0, y) = p(x, y \in K1[p])$) Dann existiert für jedes $z \in \sum^*$ ein $r \in Q$, so dass xz und yz $\in K1[r]$ also gilt insbesondere:

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