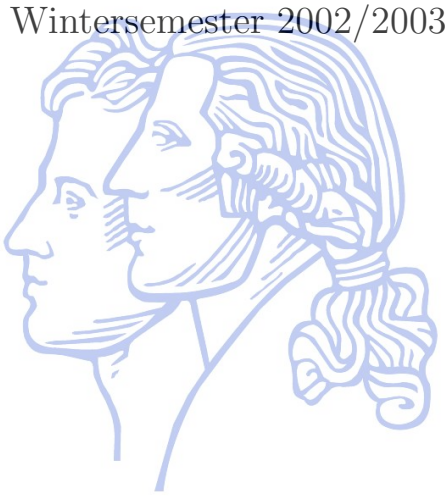
Mitschrift zur Vorlesung von Dr. Till Nierhoff

Theoretische Informatik 2

Niels Lohmann

Wintersemester 2002/2003



nlohmann@informatik.hu-berlin.de
http://www.informatik.hu-berlin.de/~nlohmann

Vorwort

Bei diesem Dokument handelt es sich um die Mitschrift aus der Vorlesung „Theoretische Informatik II“, gehalten im Wintersemester 2002/2003 von Dr. Till Nierhoff, angereichert mit ein paar Beispielen aus der Übung und Ergänzungen aus Sekundärliteratur.

Da ich ein Naturtalent in theoretischer Informatik und allen damit verbundenen Disziplinen bin, mein ganzes Leben diesem Fach und dieser Mitschrift widme, jede Vorlesung auf Video mitschneide und absolut gewissenhaft mitschreibe, übernehme ich selbstverständlich jedwede Garantie auf Fehlerfreiheit und Vollständigkeit. . .

. . . nicht!

Leider ist nach der Klausur sämtlicher Elan, diese Mitschrift zur Perfektion zu treiben, von mir gewichen, weswegen dies die vorerst finale Version ist. Wer noch (kritische) Fehler findet, darf sie entweder behalten oder noch besser mir eine kurze E-Mail an nlohmann@informatik.hu-berlin.de schicken.

Tschö,

Niels Lohmann

http://www.informatik.hu-berlin.de/~nlohmann/arbeit

Inhaltsverzeichnis

Teil I

Entscheidbarkeit und Chomsky-Hierarchie

1 Turingmaschinen

➜ Literatur: [Köbler, Kapitel 3.2] und [Wegener, Kapitel 2.1]

Definition (k-Band-Turingmaschine, k-TM):
*$M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ ist eine **k-Band-Turingmaschine**, oder kurz „k-TM“, wenn*

- *Z eine endliche **Zustandsmenge**,*
- *$\square \notin \Sigma$ ein endliches **Eingabealphabet**,*
- *$\Gamma \supseteq \Sigma \cup \{\square\}$ ein endliches **Arbeitsalphabet**,*
- *$\delta : Z \times \Gamma^k \to \mathfrak{P}\left(Z \times \Gamma^k \times \{R, L, N\}^k\right)$ eine **Überführungsfunktion**,*
- *$q_0 \in Z$ der **Startzustand** und*
- *$E \subseteq Z$ die Menge der **Endzustände** ist.*

*M ist **deterministisch** oder **k-DTM** wenn zusätzlich gilt:*
$\forall z \in Z, a \in \Gamma^k : \underbrace{|\delta(z,a)|}_{card} \leq 1$, d.h. die Maschine zu keiner Zeit aus mehreren Folgezuständen wählen kann.

Beispiel 1:
$(q, a_1, \ldots, a_k) \to (q', a'_1, \ldots, a'_k, D_1, \ldots, D_k)$, d.h.
$(q', a'_1, \ldots, a'_k, D_1, \ldots, D_k) \in \delta_($q, a_1, \ldots, a_k) mit z.B. $D_1 = L$ und $D_2 = R$.

1.1 Konfigurationen

Definition (Konfiguration):
*Eine **Konfiguration** wird durch $(q, u_1, v_1, \ldots, u_k, v_k) \in Z \times (\Gamma^* \times \Gamma^+)^k$ beschrieben. Dabei kann $u_i \in \Gamma^*$ auch das leere Wort ε sein, wobei $v_i \in \Gamma^+$ stets ein Zeichen sein muss.*

Definition (Startkonfiguration):
*Die **Startkonfiguration** ist $K_x = (q_0, \varepsilon, x, \varepsilon, \Box, \ldots, \varepsilon, \Box)$, wobei $x \in \Sigma^*$ die Eingabe ist. Der Kopf des ersten Bandes steht auf der Eingabe x, die anderen Köpfe auf dem leeren Band ($\Box$).*

Definition (Folgekonfiguration):
$(q, u_1, a_1v_1, \ldots, u_k, a_kv_k) \vdash (q', u_1', v_1', \ldots, u_k', v_k')$. *Dabei beschreibt u_1 die Bandbeschriftung links vom Kopf, a_1 das Zeichen unter dem Kopf i und v_1 die Bandbeschriftung rechts vom Kopf.*
$(q, a_1, \ldots, a_k) \to (q', a_1', \ldots, a_k', D_1, \ldots, D_k)$ *und*

- $u_i' = u_i, v_i' = a_i'v_i$, *falls* $D_i = N$ *(keine Kopfbewegung)*
- $u_i = u_ia_i', v_i' = v_i$, *falls* $D_i = R$ *(Kopfbewegung nach rechts)*
- $u_i' = u_i'', v_i' = b_ia_i'v_i$, *falls* $D_i = L$ *(Kopfbewegung nach links)*
  ($u_i = u_i''b_i$, d.h. b_i ist das letzte Zeichen von u_i)

Definition (akzeptierende Konfiguration):
$K \vdash^* K'$ *falls* $K = K_1$ *und* $\exists l \exists K_1, \ldots, K_l : K_1 \vdash \ldots \vdash K_l \vdash K'$, *d.h. die Konfiguration K' wird durch endlich viele (l) Zwischenkonfigurationen erreicht.*

Definition (akzeptierte Sprache):
M ***akzeptiert*** $x \in \Sigma^*$, *falls von der Startkonfiguration K_x aus eine **akzeptierte Konfiguration**, d.h. eine mit Endzustand erreicht ist:* $\exists K : K_x \vdash^* K$ *mit* $K \in E \times (\Gamma^* \times \Gamma^+)^k$. *Diese akzeptierte Sprache wird mit* $L(M) = \{x \in \Sigma^* \mid M \text{ akzeptiert } x\}$ *bezeichnet.*

Definition (Berechenbarkeit, rekursive Funktionen):
Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-DTM. $f : \Sigma^* \to \Gamma^*$ ***wird von M berechnet**:*
$f(x) = y \Leftrightarrow K_x \vdash^* (q_e, u_1v_1, \ldots, u_{k-1}v_{k-1}, y, \Box)$, $q_e \in E$. *O.B.d.A. gibt es keine Nachfolgekonfiguration für $q_e \in E$.* ***f ist (Turing-) berechenbar**. Ist f außerdem total, nennt man f **rekursiv**.*

1.2 Entscheidbarkeit, Semi-Entscheidbarkeit

Definition (entscheidbar, semi-entscheidbar, rekursiv aufzählbar):
Sei $L \subseteq \Sigma^*$ eine Sprache. Dann gilt:

- *L ist **entscheidbar*** $:\Leftrightarrow \chi_L(x) = \begin{cases} 1, & \text{falls } x \in L \\ 0, & \text{falls } x \notin L \end{cases}$ *ist berechenbar.*
- *L ist **semi-entscheidbar*** $:\Leftrightarrow \widehat{\chi}_L(x) = \begin{cases} 1, & \text{falls } x \in L \\ \uparrow, & \text{falls } x \notin L \end{cases}$ *ist berechenbar.*
  ($\uparrow$ heißt „nicht definiert“)
- *L ist **rekursiv aufzählbar*** $:\Leftrightarrow L = \emptyset$ *oder* $\exists f : \Gamma^* \to \Sigma^*$ *(Turing-) berechenbar.*
  $L = W(f) = \{f(x) \mid x \in \Gamma^*\} = f(\Gamma^*)$
  *f heißt **Aufzählung** und hat L als Bildbereich.*
- $\mathcal{REC} := \{A \mid A$ *entscheidbar*$\}$
- $\mathcal{RE} := \{A \mid A$ *rekursiv aufzählbar*$\}$
- *A entscheidbar $\Rightarrow$ A semi-entscheidbar*

Beispiel 2 (entscheidbare und semi-entscheidbare Sprachen):
Gegeben sind folgende Sprachen:

1. $L_1 = \{x \mid x$ ist gültige arithmetische Gleichung$\} \subseteq \{0, \ldots, 9, +, -, \cdot, :, =\}^*$
2. $L_2 = \{x \mid x$ ist zusammenhängender Graph z.B. Baum$)\}$
3. $L_3 = \{x \mid x$ ist syntaktisch korrektes C-Programm$\} \subseteq \text{ASCII}^*$
4. $L_4 = \{x \mid x$ ist C-Programm, das immer terminiert$\}$

Die Sprachen L_1, L_2 und L_3 sind entscheidbar, L_4 semi-entscheidbar. Also $L_1 \in \mathcal{REC}$, $L_2 \in \mathcal{REC}$, $L_3 \in \mathcal{REC}$ und $L_4 \in \mathcal{RE}$.

Satz 1.1 (Äquivalenzen):
Sei $A \subseteq \Sigma^*$ eine Sprache. Dann sind folgende Aussagen äquivalent:

(i) A ist semi-entscheidbar

(ii) $A = L(M)$, M ist k-TM

(iii) $A = L(M)$, M ist 1-TM

(iv) A ist rekursiv aufzählbar

(v) $A = L(M)$, M ist k-DTM

(vi) $A = L(M)$, M ist 1-DTM

Beweis (von Satz 1.1):
(i) $\Rightarrow$ (ii):

M berechnet $\widehat{\chi}_A \Rightarrow L(M) = A$

(ii) $\Rightarrow$ (iii):

M k-TM mit $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$, $L(M) = A$

1. Konstruiere 1-TM $M' = (Z' \cup Z, \Sigma, \Gamma', \delta', q_0, E)$

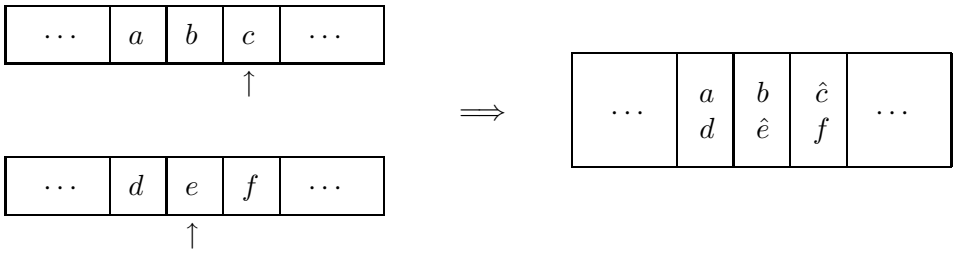


Abbildung 1.1: Illustration: Übergang 2-DTM in 1-DTM mit 2 Spuren

   Dazu ist das neue Alphabet entsprechend ein Vektor.
2. Dann läuft M' nach rechts über das Band und merkt sich im Zustand auf welchen Zeichen die Köpfe stehen: $\hat{a}_1, \ldots, \hat{a}_k$ (endlich viele).
3. M' wählt eine Anweisung aus $\delta_M(q, a_1, \ldots, a_k)$.
4. M' läuft wieder nach links und realisiert die Anweisungen (Zeichen mit Hut austauschen, Hut auf jeweiliges Nachbarzeichen bzw. kein Verschieben des Hutes).
5. M' geht in den neuen Zustand der Anweisung ($\in Z$).

Es gilt: $L(M') = L(M)$, da die Konfiguration, in denen der Kopf ganz links steht, einer Konfigurationsfolge von M entspricht.

(iii) $\Rightarrow$ (iv):

Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $L(M) = A$ gegeben.

$$f(y) = \begin{cases} x, & \text{falls } y = K_x \#\# K_1 \#\# \ldots \#\# K_t \\ x_0, & \text{sonst} \end{cases}$$

wobei $K_i = z\#u\#v$ mit $z \in Z; u, v \in \Gamma^*$ und $K_x \vdash^M K_1 \vdash^M \ldots \vdash^M K_t$ mit $K_t \in E \times \Gamma^* \times \Gamma^*$. O.B.d.A sei außerdem $x_0 \in A$.

Die Konfigurationen von M werden mit den Trennzeichen # kodiert.

$f : (\Gamma \cup Z \cup \{\#\})^* \to \Sigma^*$, $A = W(f)$, f ist berechenbar und gibt alle Eingaben x aus, die M akzeptiert. Wenn $z \notin \Gamma^*$, so ist z auch nicht Bild von f.

(iv) $\Rightarrow$ (v):

Gegeben sei f, berechenbar mit $W(f) = A$, $f : \Gamma^* \to \Sigma^*$

$\to \exists$ k-DTM, die f berechnet

$\to \exists$ konstruiere $(k+1)$-DTM, die A akzeptiert:

1. zähle auf Band 2 Γ^*, z.B. in lexikografischer Reihenfolge, auf
2. simuliere M auf Band $2 \ldots k+1$
3. vergleiche Ausgabe von M mit x
   - Gleichheit $\to$ akzeptiere, Ende
   - Ungleichheit $\to$ nächstes Wort aus Γ^*

(v) $\Rightarrow$ (vi):

siehe Beweis (ii) $\Rightarrow$ (iii)

(vi) $\Rightarrow$ (i):

Die 1-DTM schreibt im ursprünglichen Endzustand noch eine 1 aufs Band (als Ergebnis für χ_L) und terminiert erst dann.

□

Beispiel 3 (anhand einer 2-DTM):
Gegeben sei eine 2-DTM $M_1 = (\{q_0, q_1, q_H\}, \{0,1\}, \{0,1,\square\}, \delta, q_0, \{q_H\})$ mit folgender Überführungsfunktion δ:

δ	$\square, \square$	$0, \square$	$1, \square$
q_0	$q_H, \square, \square, N, N$	$q_1, 0, 1, R, R$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$q_0, 1, \square, R, N$
q_H	$\emptyset$	$\emptyset$	$\emptyset$

Dabei bedeutet in der ersten Zeile „$0, \square$“ eine 0 unter dem oberen Kopf und eine leere Zelle unter dem unteren Kopf.

Aus der Eingabe $x = 0101$ ergibt sich die Startkonfiguration $K_x = (q_0, \varepsilon, 0101, \varepsilon, \square)$, die wie folgt in den Endzustand $q_H \in E$ überführt wird:

$$\begin{aligned} K_x = (q_0, \varepsilon, 0101, \varepsilon, \square) &\vdash (q_1, 0, 101, 1, \square) \\ &\vdash (q_0, 01, 01, 1, \square) \\ &\vdash (q_1, 010, 1, 11, \square) \\ &\vdash (q_0, 0101, \square, 11, \square) \\ &\vdash (q_H, 0101, \square, 11, \square) \end{aligned}$$

$\Rightarrow$ Die Eingabe $x = 0101$ wird offensichtlich von M_1 akzeptiert: $x \in L(M_1)$.

Behauptung:
$L(M_1) = \{(01)^n \mid n \in \mathbb{N}\}$

Beweis:
Sei $x = x_1 \dots x_m$.

- $x_1 = 1$: keine Folgekonfiguration für $K_x \Rightarrow x$ wird nicht akzeptiert
- $m = 0$ (leeres Band): M_1 akzeptiert
- $x_1 = 0$: M_1 geht in q_1 (und schreibt 1 auf 2. Band)
  - $x_2 \neq 1 \Rightarrow M_1$ akzeptiert nicht.
  - $x_2 = 1 \Rightarrow M_1$ geht in q_0, und $K_{x_3 \cdots x_m}$, wobei $x_3 \cdots x_m$ die restliche Eingabe ist (Kopf auf 2. Band wieder auf $\square$)

  Induktion nach m: M_1 akzeptiert $\Leftrightarrow x_3 \dots x_m = (01)^{\frac{m-2}{2}}$

$\Rightarrow x = (01)^{\frac{m}{2}}$ $\square$

Beispiel 4 (Addierer):
Sei $M = (\{z_0, z_1, z_2, z_e\}, \{0, 1, \}, \{0, 1, \square\}, \delta, q_0, \{z_e\})$ eine Turingmaschine mit der Überführungsfunktion

δ	0	1	$\square$
z_0	$z_0, 0, R$	$z_0, 1, R$	$z_1, \square, L$
z_1	$z_2, 1, L$	$z_1, 0, L$	$z_e, 1, N$
z_2	$z_2, 0, L$	$z_2, 1, L$	$z_e, \square, R$

Bei der Maschine M handelt es sich um einen Addierer, der zur binär kodierten Eingabe eine 1 addiert.

Satz 1.2 (Sprache, Komplement der Sprache):
A entscheidbar $\Leftrightarrow A, \bar{A} = \Sigma^* \backslash A$ semi-entscheidbar

Beweis (von Satz 1.2):
Es gilt, beide Richtungen der Äquivalenz zu zeigen:

1. $\Rightarrow$ Klar: Wenn A entscheidbar ist, ist auch $\Sigma^* \backslash A$ entscheidbar ($\chi_{\bar{A}}(x) = 1 - \chi_A(x)$). Da alle entscheidbaren Sprachen auch semi-entscheidbar sind, sind auch A und $\bar{A}$ semi-entscheidbar.

2. $\Leftarrow$ Seien $f_1 : \Sigma^* \to \Sigma^* : W(f_1) = A$ und $f_2 : \Sigma^* \to \Sigma^* : W(f_2) = \bar{A}$ Funktionen.

   a) zähle Σ^* z.B. in lexikografischer Reihenfolge auf

   b) berechne $f_1(y)$ und $f_2(y)$

      - falls $x = f_1(y) \Rightarrow x \in W(f_1) \to x \in A \Rightarrow$ Ausgabe 1 und Stopp
      - falls $x = f_2(y) \Rightarrow x \in W(f_2) \to x \in \bar{A} \Rightarrow$ Ausgabe 0 und Stopp

   c) sonst nächstes y

   Die berechnende Funktion ist total, da $W(f_1) \cup W(f_2) = \Sigma^*$. □

2 Entscheidbarkeit

2.1 Kodierung von Turingmaschinen und universelle TM (UTM)

Sei M eine Turingmaschine mit $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$ mit $Z = \{z_0, \ldots, z_m\}$, $E = \{z_m\}$ und $\Gamma = \{a_0 = \square, a_1, \ldots, a_l\}$

- **Kodierung über** $\{0, 1, \#\}$

Objekt	kodiert als
$i \in \mathbb{N}$	$\text{bin}(i)$
$\text{anw} := z_i a_j \to z_k, a_l, D$	$\text{code}(\text{anw}) = \text{bin}(i)\#\text{bin}(j)\#\text{bin}(k)\#\text{bin}(l)\#\text{bin}(D)$
$\delta : \{\text{anw}_1, \ldots, \text{anw}_n\}$	$\text{code}(\delta) = \text{code}(\text{anw}_1)\#\# \ldots \#\#\text{code}(\text{anw}_n)$
ganze TM M	$\text{bin}(l)\#\#\#\text{bin}(m)\#\#\#\text{code}(\delta)$

  Wobei $\text{bin}(i)$ Binärdarstellung der Zahl i, und D die möglichen Kopfbewegungen mit $D \in \{1, 2, 3\} \mathrel{\widehat{=}} \{L, N, R\}$ bedeuten. Zusammen mit der Zustandstabelle und den einzelnen Anweisungen wird insbesondere die Anzahl l der Zeichen im Alphabet, sowie die Anzahl der Zustände m kodiert.

- **Kodierung über** $\{0, 1\}$

  Es wird das Alphabet $\{0, 1, \#\}$ neu kodiert:

Zeichen	kodiert als
0	00
1	01
#	10

Definition:
Sei M_w wie folgt definiert:

$$M_w = \begin{cases} M, & \text{falls } w = w_M \\ M_1, & \text{sonst} \end{cases}$$

Dabei ist w die Gödelnummer (binär, siehe oben) einer Turingmaschine und M_1 eine beliebige aber feste Turingmaschine ist.

Satz (universelle TM):
Es existiert eine DTM M^*, die die universelle Turingmaschine (UTM), die M_w für gegebenes w, auch M_w bei Eingabe x für zusätzlich gegebenes x (gegeben als $w\#x$) simulieren kann.

2.2 Halteproblem

Definition (Halteproblem):
*Das **Halteproblem** H ist die Sprache*

$$H = \{w\#x \mid M_w \text{ ist 1-DTM, } M_w(x) \neq\uparrow; \quad w, x \in \{0,1\}^*\}$$

Dabei ist x eine binär kodierte Eingabe und w eine binär kodierte Gödelzahl einer 1-DTM. „$M_w(x) \neq\uparrow$“ bedeutet, dass die Turingmaschine M_w die Eingabe x akzeptiert, also terminiert.

Satz 2.1:
Die Sprache H ist semi-entscheidbar.

Beweis (von Satz 2.1):
Angabe einer Turingmaschine U', die H akzeptiert. Dabei sei U' eine k-DTM mit $k \geq 3$ Bändern:

Band	Beschriftung	Kodierung
1	Eingabe	$w\#x$
2	Zustand aktuell von M_w	$\text{bin}(i)$, entspricht z_i
3	Bandinschrift aktuell von M_w	$\text{bin}(i_1)\#\ldots\#\text{bin}(i_e)$, entspricht $a_{i_1}\ldots a_{i_e}$

Der Kopf zeigt auf das 1. Zeichen des Buchstabens, auf dem der Kopf von M_w steht.

- Initialisierung
  - Prüfe, ob Eingabe $w\#x$ und M_w 1-DTM ist.
    Falls nicht: $\nearrow$ (Maschine stoppt nicht).
  - Schreibe $0 = \text{bin}(0)$ auf Band 2.
  - Schreibe x auf Band 3 und bewege Kopf auf 1. Zeichen.
- Schleife
  - Prüfe, ob auf Band 2 $\text{bin}(m)$ steht ($z_m \in E$). Falls ja: Endzustand.
  - Sonst finde Anweisung w, die auf Zustand (von M_w) und aktuelles Zeichen passt. Falls keine vorhanden: $\nearrow$ (daher Problem semi-entscheidbar)
  - Führe Anweisung aus.

Die UTM U sei die 1-Band-Version von U'. Es gilt:

$$\begin{aligned} U' \text{ akzeptiert} &\Leftrightarrow M_w \text{ bei Eingabe } x \text{ den Endzustand erreicht} \\ &\Leftrightarrow M_w(x) \neq\uparrow \text{ (an Stelle } x \text{ definiert)} \\ &\Leftrightarrow w\#x \in H \end{aligned}$$

□

Definition (spezielles Halteproblem):
*Das **spezielle Halteproblem** ist die Sprache*

$$K = \{w \in \{0,1\}^* \mid M_w \textit{ 1-DTM}, M_w(w) \neq\uparrow\}$$

d.h. M_w stoppt bei der eigenen Kodierung w.

Satz 2.2:
$K \notin \mathcal{REC}$

Beweis (von Satz 2.2, indirekt):
These: $\exists M : M$ 1-DTM, die χ_K berechnet.

Modifiziere M, sodass M bei Ausgabe 1 stattdessen endlos nach rechts läuft ($\nearrow$).

Sei M' diese modifizierte Maschine.

$$\begin{aligned} w_{M'} \in K &\Leftrightarrow M(w_{M'}) = 1 \\ &\Leftrightarrow M'(w_{M'}) =\uparrow \\ &\Leftrightarrow w_{M'} \notin K \text{ (Widerspruch!)} \end{aligned}$$

□

Satz:
Seien $f : \{0,1\}^* \to \{0,1\}$ Funktionen. Es gibt nur eine abzählbare Anzahl von berechenbaren Funktionen (da es nur abzählbar viele Turingmaschinen gibt, die f berechnen).

$\Rightarrow$ Es gibt viele nicht berechenbare Funktionen

➜ Literatur: [Köbler, Kapitel 4 und 4.1]

Definition (Reduzierung):
Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. ***A reduzierbar auf B*** *($A \leq B$), wenn*

$$\exists f : \Sigma^* \to \Gamma^* : x \in A \Leftrightarrow f(x) \in B$$

Dabei sei die ***Reduktion*** *f berechenbar und total.*

Notation: $\mathfrak{K} \subseteq \mathfrak{P}(\Sigma^*)$ (unter $\leq$ abgeschlossen), d.h.

$$B \in \mathfrak{K}, A \leq B \Rightarrow A \in \mathfrak{K}$$

co-$\mathfrak{K}$ $= \{\bar{A} \mid A \in \mathfrak{K}\}$

Beispiel 5 (Reduktion):
$K \leq H$, dann $w \mapsto w\#w$ ist berechenbar.

Satz 2.3:
$\mathcal{RE}$ und $\mathcal{REC}$ sind unter $\leq$ abgeschlossen.

Beweis (von Satz 2.3):
Annahme: $A \leq B$ mit Reduktion f.

$x \in A \Leftrightarrow f(x) = B \Leftrightarrow \chi_B(f(x)) = \hat{\chi}_B(f(x)) = 1$
$\Downarrow$
$\chi_A(x) = \hat{\chi}_A(x) = 1$

$\rightarrow$ $\chi_A = \chi_B \circ f$ $\rightarrow$ χ_A berechenbar, falls χ_B es ist
$B \in \mathcal{REC} \Rightarrow A \in \mathcal{REC}$

$\hat{\chi}_A = \hat{\chi}_B \circ f$ $\rightarrow$ $\hat{\chi}_A$ berechenbar, falls $\hat{\chi}_B$ es ist
$B \in \mathcal{RE} \Rightarrow A \in \mathcal{RE}$

□

Korollar:
$K \in \mathcal{RE}$ (aus Satz 2.1)
$H \notin \mathcal{REC}$ (aus Satz 2.2)
$\bar{K} \notin \mathcal{RE}$ (aus Satz 1.2 und 2.2)

Außerdem gilt: $\mathcal{RE} \cap \text{co-}\mathcal{RE} = \mathcal{REC}$

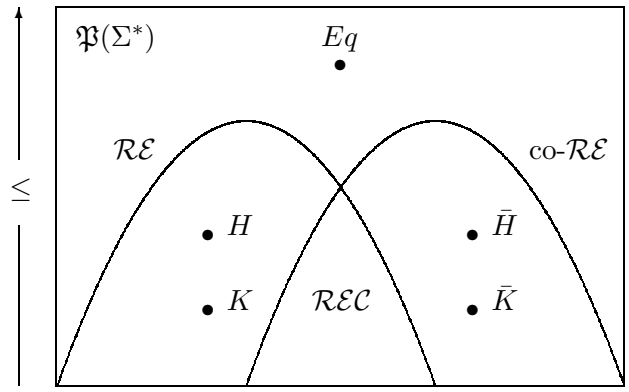


Abbildung 2.1: Sprachen und Sprachklassen

Für Beweise bezüglich Eq siehe Übungsaufgabe 12

2.3 weitere Unentscheidbarkeit durch Reduktion von H

Beispiel 6 (Halteproblem bei leerem Band):
Sei $H_0 = \{w \mid M_w \text{ 1-DTM}, M_w(\varepsilon) \neq \uparrow\}$ das Halteproblem bei leerem Band.

Zu zeigen: $H \leq H_0$ via

$$f_0 : y \longmapsto \begin{cases} w' & \text{falls } y = w\#x,\ x \text{ beliebig} \\ w_0 \notin H_0 & \text{sonst} \end{cases}$$

Dabei sei $M_{w'}$ eine Modifikation von M_w, sodass $M_{w'}$ erst prüft, ob das Band leer ist, dann x schreibt und anschließend wie M_w weiterarbeitet. Es gilt: f_0 ist berechenbar.

$\Rightarrow M_{w'}(\varepsilon) = M_w(x)$

$$\begin{aligned} w' = f_0(y) \in H_0 &\Leftrightarrow M_{w'}(\varepsilon) \neq \uparrow \\ &\Leftrightarrow M_w(x) \neq \uparrow \\ &\Leftrightarrow w\#x = y \in H \end{aligned}$$

$\longrightarrow f_0(y) \in H_0 \Leftrightarrow y \in H$ □

Beispiel 7:
Sei $H_{\text{all}} = \{w \mid M_w \text{ 1-DTM}, M_w \text{ hält für alle Eingaben}\}$

Zu zeigen: $H \leq H_{\text{all}}$ via

$$f_{\text{all}} : y \longmapsto \begin{cases} w' & \text{falls } y = w\#x,\ x \text{ beliebig} \\ w_{\text{all}} \notin H_{\text{all}} & \text{sonst} \end{cases}$$

Dabei sei $M_{w'}$ eine Modifikation von M_w, sodass $M_{w'}$ die Eingabe zunächst löscht (bzw. sie durch x ersetzt) und anschließend wie M_w weiterarbeitet. Es gilt: f_{all} ist berechenbar.

$$\begin{aligned} w' = f_{\text{all}}(y) \in H_{\text{all}} &\Leftrightarrow M_{w'}(z) \neq \uparrow \quad \forall z \in \Sigma^* \\ &\Leftrightarrow M_w(x) \neq \uparrow \\ &\Leftrightarrow y = w\#x \in H \end{aligned}$$

2.4 Satz von Rice

Satz 2.4 (Satz von Rice):
Sei $\emptyset \neq \mathcal{F} \subsetneq \text{PREK} = \{f \mid f \text{ berechenbar, partiell}\}$

Dann ist $K(\mathcal{F}) = \{w \mid M_w(\cdot) \in \mathcal{F}\} \notin \mathcal{REC}$.

Dabei beschreibt $K(\mathcal{F})$ die Menge aller Turingmaschinen, deren berechnete Funktion (bezeichnet durch $M_w(\cdot)$) in $\mathcal{F}$ liegt.

Beispiel 8 (für Satz von Rice – $\mathcal{F}_0$):
Sei $\mathcal{F}_0 = \{f \in \text{PREK} \mid f(\varepsilon) \neq \uparrow\}$ die Menge der Funktionen, die für die leere Eingabe definiert sind.

$\Rightarrow K(\mathcal{F}_0) = \{w \mid M_w \text{ 1-DTM}, M_w(\varepsilon) \neq \uparrow\} \notin \mathcal{REC}$

Beispiel 9 (für Satz von Rice – $\mathcal{F}_{\text{all}}$):
Sei $\mathcal{F}_{\text{all}} = \{f \in \text{PREK} \mid f(z) \neq \uparrow \quad \forall z \in \Sigma^*\}$ die Menge der Funktionen, die stets definiert sind.

$\Rightarrow K(\mathcal{F}_{\text{all}}) = \{w \mid M_w(z) \neq \uparrow \quad \forall z \in \Sigma^*\} \notin \mathcal{REC}$

Bemerkung:
Kein Beispiel für den Satz von Rice ist allerdings die Menge aller (Turing-) berechenbaren Funktionen: $\mathcal{F}^* = \{f \mid \exists w : M_w(\cdot) = f\} = \text{PREK}$, da $K(\mathcal{F}^*) \in \mathcal{REC}$.

Beweis (von Satz 2.4):
Definition: Sei $\uparrow(x) = \uparrow \forall x \in \Sigma^*$.

Bemerkung:
O.B.d.A. sei $\uparrow(\cdot) \notin \mathcal{F}$. Wäre $\uparrow(\cdot) \in \mathcal{F}$, würde für diesen Beweis $\overline{\mathcal{F}}$ mit $\overline{\mathcal{F}} = \{f \in \text{PREK} \mid f \notin \mathcal{F}\}$ betrachtet, denn es gilt: $K(\overline{\mathcal{F}}) = \overline{K(\mathcal{F})}$. Diese Menge ist nicht trivial, denn wenn $\overline{\mathcal{F}}$ trivial wäre, wäre auch $\mathcal{F}$ trivial. Außerdem gilt $K(\mathcal{F}) \in \mathcal{REC} \Leftrightarrow \overline{K(\mathcal{F})} \in \mathcal{REC}$.

Sei $l \in \{0,1\}^* : M_l(\cdot) \in \mathcal{F}$. Idee zur Reduktion $H \leq K(\mathcal{F})$:

$$f : w\#x \begin{cases} w' : M_{w'}(\cdot) = \uparrow(\cdot), & \text{falls } w\#x \notin H \\ w' : M_{w'}(\cdot) = M_l(\cdot), & \text{sonst} \end{cases}$$

Dazu: $u \in \{0,1\}^*$ Kodierung der UTM

Gegeben: $w\#x$. Konstruiere w' mit folgendem Ablauf von $M_{w'}$

1. M_u auf Eingabe $w\#x$ simuliert (dabei Eingabe unberührt)
2. Falls Endzustand (von M_u) erreicht: M_l auf der Eingabe simulieren.

Es gilt: w' ist berechenbar (aus $w\#x$)

$$\begin{aligned} w\#x \in H &\Rightarrow M_{w'}(\cdot) = M_l(\cdot) \in K(\mathcal{F}) \\ w\#x \notin H &\Rightarrow M_{w'}(\cdot) = \uparrow(\cdot) \notin K(\mathcal{F}) \end{aligned}$$

$\longrightarrow$ Reduktion formal: $f_{\text{Rice}} : y \longmapsto \begin{cases} w', & y = w\#x \\ w_0 \notin K(\mathcal{F}), & \text{sonst} \end{cases}$

□

2.5 Post'sches Korrespondenzproblem

➜ Literatur: [Köbler, Kapitel 4.2] und [Wegener, Kapitel 2.8]

Definition (Post'sches Korrespondenzproblem):
Gegeben: $\begin{pmatrix} x_1 & x_2 & \cdots & x_k \\ y_1 & y_2 & \cdots & y_k \end{pmatrix}$ *Jede Spalte z.B.* $\begin{matrix} x_1 \\ y_1 \end{matrix}$ *Regel.*

Gesucht: $i_1, \ldots, i_n : x_{i_1} x_{i_2} \cdots x_{i_n} = y_{i_1} y_{i_2} \cdots y_{i_n}$

Diese Problem nennt sich ***Post'sches Korrespondenzproblem*** *oder kurz* ***PKP****.*

Falls gefordert wird, dass $i_1 = 1$ *handelt es sich um das* ***modifizierte PKP*** *oder kurz* ***MPKP****. Es hat gegenüber dem PKP eine eingeschränkte Lösungsmenge.*

Beispiel 10 (PKP/MPKP):

$\begin{pmatrix} 1 & 10 & 011 \\ 101 & 00 & 11 \end{pmatrix}$ hat die Lösung $1,3,2,3$: $\underbrace{\begin{matrix} 1 \\ 101 \end{matrix}}_{1} \underbrace{\begin{matrix} 011 \\ 11 \end{matrix}}_{3} \underbrace{\begin{matrix} 10 \\ 00 \end{matrix}}_{2} \underbrace{\begin{matrix} 011 \\ 11 \end{matrix}}_{3}$.

Die Lösung ist auch MKPK, da $i_1 = 1$.

Satz 2.5:
H $\leq$ MPKP

Satz 2.6:
MPKP $\leq$ PKP

Korollar:
PKP und MPKP $\notin \mathcal{REC}$ (wegen Transitivität)

Beweis (von Satz 2.5):
Idee: $H = L(U)$ mit U ist UTM $U = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $\Sigma = \{0, 1, \#\}$, \$ $\notin \Gamma$.

Konstruiere aus $w\#z$ MPKP-Instanz, die Lösung hat $\Leftrightarrow$ U erreicht bei Eingabe $w\#z$ Endzustand, d.h. $w\#z \in H$

Dazu gibt es fünf Klassen von Regeln:

1. **Startregel**

x_1	=	\$
y_1	=	$\$K_{w\#z}\$$

   Dabei ist K eine Konfiguration der Form $K = (u, q, v) \mathrel{\hat{=}} uqv$ mit u, v Strings, q Zustand. $K_{w\#z}$ ist hierbei der Startkonfiguration.

   Das Alphabet für MPKP ist $\Gamma \dot\cup Z \dot\cup \{\$\}$. Dabei ist $Z \neq \Gamma$, \$ $\in Z$, \$ $\notin \Gamma$.

   Die Anwendung dieser Regel (als erstes) erzwingt die Lösung in der Form:

   $$\$K_{w\#z}\$K_1\$ \ldots \$K_t\$L_1\$ \ldots \$L_{|K_t|}\$\$$$

   Dabei ist K_t der akzeptierende Endzustand und L_i Regeln, die den Vorsprung in der unteren Zeile aufholen.

2. **Kopierregeln** ($\forall a \in \Gamma$ oder $a =$ \$)

x	=	a
y	=	a

   Es gibt $|\Gamma| + 1$ Kopierregeln (eine für jedes Zeichen im Alphabet Γ), sowie eine für \$.

3. **Überführungsregeln** ($\forall a, a', b \in \Gamma, \forall q, q' \in Z$)

   a) Keine Kopfbewegung: $\delta : (q, a) \to (q', a', N)$ (von U)

x	=	qa
y	=	$q'a'$

   und, falls $a = \square$:

x	=	$q\$$
y	=	$q'a'\$$

b) Kopfbewegung nach rechts: $\delta : (q, a) \to (q', a', R)$:

$$\boxed{\begin{array}{lcl} x & = & qa \\ y & = & a'q' \end{array}} \text{ und, falls } a = \square: \boxed{\begin{array}{lcl} x & = & q\$ \\ y & = & a'q'\$ \end{array}}$$

c) Kopfbewegung nach links: $\delta : (q, a) \to (q', a', L)$:

$$\boxed{\begin{array}{lcl} x & = & bqa \\ y & = & q'ba' \end{array}} \text{ und, falls } a = \square:$$

$$\boxed{\begin{array}{lcl} x & = & \$qa \\ y & = & \$q'\square a' \end{array}} \text{ (linker Rand), bzw. } \boxed{\begin{array}{lcl} x & = & bq\$ \\ y & = & q'ba'\$ \end{array}} \text{ (rechter Rand)}$$

Dabei bedeuten die Fälle $a = \square$ ein Erweitern des Universums der TM bei Erreichen des Bandrandes.

Nun gibt es folgende Situation:

$$\$ \cdots\cdots qa \cdots \$$$
$$\underbrace{\$ \cdots\cdots qa \cdots \$}_{\text{Konfiguration}} \underbrace{\cdots\cdots q'a' \cdots \$}_{\text{Nachfolgekonf.}}$$

Es gibt $3 \cdot |Z|$ Überführungsregeln (für jeden Zustand drei Regeln für die Kopfbewegungen L, R, N).

4. **Löschregeln** ($\forall a \in \Gamma, \forall q \in E$)

$$\boxed{\begin{array}{lcl} x & = & aq \\ y & = & q \end{array}} \text{ und } \boxed{\begin{array}{lcl} x & = & qa \\ y & = & q \end{array}}$$

Danach bleibt nur noch der Vorsprung: $q\$$, wobei $q \in E$.

Es gibt $2 \cdot |\Gamma| \cdot |E|$ Löschregeln (je 2 für Endzustand und Alphabetzeichen).

5. **Abschlussregeln** ($\forall q \in E$)

$$\boxed{\begin{array}{lcl} x & = & q\$\$ \\ y & = & \$ \end{array}}$$

Nun ist der Vorsprung aufgeholt.

Es gibt eine Abschlussregel.

Mit diesen Regelklassen kann nun der eigentliche Beweis geführt werden. Dabei wird nach folgendem Prinzip vorgegangen:

1. Startregel. Dadurch hat y-Folge eine Konfiguration Vorsprung.
2. Kopier-/Überführungsregeln schreiben jeweils eine Konfiguration in die x-Folge und deren Nachfolgekonfiguration in die y-Folge.
3. Löschregeln. Wenn Endzustand erreicht ist, schrumpft der Vorsprung der y-Folge jeweils um ein Zeichen.

4. Abschlussregel. Der letzte Vorsprung (Endzustand und \$\$) wird aufgeholt. → fertig

MPKP hat eine Lösung, wenn ein Endzustand erreicht wird, ansonsten ist $w \notin$ MPKP

Beweis (Richtung $\Rightarrow$)

zu zeigen: $w\#z \in H \Rightarrow \exists i_1, \ldots, i_l : x_{i_1} \cdots x_{i_l} = y_{i_1} \cdots y_{i_l}$, sowie $i_1 = 1$ (wegen MPKP).

Aus $w\#z \in H$ folgt: $\exists K_1, \ldots, K_t : K_{w\#z} \vdash_U \cdots \vdash_U K_t$, wobei $K_t \in E \times \Gamma^* \times \Gamma^*$ und $K_{w\#z}$ die Startkonfiguration ist.

- $i_1 = 1$

  $$\begin{aligned} x(= x_{i_1}) &= \$ \\ y(= y_{i_1}) &= \$K_{w\#z}\$ \quad = \$q_0 w\#z\$ \end{aligned}$$

  Annahme: $\delta(q_0, w_1) = (q, a', R)$ (dabei ist R nur ein Beispiel)

- i_2 Überführungsregel

  $$\begin{aligned} x = x_{i_1}x_{i_2} &= \$q_0 w_1 \\ y = y_{i_1}y_{i_2} &= \$q_0 w\#z\$a'q \end{aligned}$$

- $i_3, \ldots, i_{|w\#x|+2}$ Kopierregeln, die letzte ist $\genfrac{}{}{0pt}{}{\$}{\$}$

  $$\begin{aligned} x &= \$K_{w\#z}\$ \\ y &= \$K_{w\#z}\$\underbrace{a'q\ w_2 \cdots w_{|w|}\ \#z}_{=K_1}\$ \end{aligned}$$

  K_1 ist nun überführt/verarbeitet.

- $i_{|w\#z|+3}, \ldots, i_r$: Kopier- und Überführungsregeln

  $$\begin{aligned} x &= \$K_{w\#z}\$K_1\$\ldots\$K_{t-1}\$ \\ y &= \$K_{w\#z}\$K_1\$\ldots\$K_{t-1}\$K_t\$ \end{aligned}$$

  ↑ ab hier: kopieren/löschen/kopieren...

  K_t ist akzeptierende Konfiguration mit Endzustand.

  Durch das Erreichen der Endkonfiguration K_t können nun die Kopier- und die Löschregeln angewendet werden:

  $$\begin{array}{ll} \cdots\ \$K_{t-1}\$K_t\$ & \cdots\ \$q_e\ a\$q_e\$\$ \\ \cdots\ \$K_{t-1}\$K_t\$K_t\ (\text{–1 Zeichen}) & \cdots\ \$q_e\ a\$q_e\$\$ \end{array}$$

  Im oberen Beispiel wurde bereits die Abschlussregel angewandt, die an das Ende der Folgen $\genfrac{}{}{0pt}{}{q_e\$\$}{\$}$ anhängt.

Nur die Länge von $w\#z$ unterscheidet das beschriebene Verfahren. Die Regeln (bis auf die Anfangsregel) hängen jedoch nur von der UTM U ab, sind also konstant.

Beweis (Richtung $\Leftarrow$)

Angenommen, $i_1, \ldots, i_l$ ist Lösung für die MPKP-Instanz.

- $i_1 = 1$

  $x_{i_1} = \$$
  $y_{i_1} = \$K_{w\#z}\$$

- Angenommen $i_1, \ldots, i_k$: $\quad y_{i_1} y_{i_2} \ldots y_{i_k} = x_{i_1} x_{i_2} \ldots x_{i_k} K\$$

  Dabei ist $K = uqv$ $(q \notin E)$ eine Konfiguration, die an die x-Folge gehangen werden muss, um die Länge der y-Folge zu haben. K ist also der Vorsprung der y-Folge.

  Dann müssen $i_{k+1}, \ldots, i_r$ Kopier- und Überführungsregeln sein, da die Lösch- und Abschlussregeln einen Endzustand voraussetzen.

  Dies geschieht, bis $y_{i_1}, \ldots, y_{i_r} = x_{i_1}, \ldots, x_{i_r} K'\$$ (nach Wahl der Überführungsregeln $K \vdash_U K'$.

Daher ist von $K_{w\#z}$ keine akzeptierende Konfiguration erreichbar.

$\rightarrow$ Instanz hat keine Lösung.

□

Beweis (von Satz 2.6):
Zu zeigen: MPKP $\leq$ PKP

Annahme: $\$ \notin \Sigma$

Es gilt: $\begin{pmatrix} x_1 & \cdots & x_k \\ y_1 & \cdots & y_k \end{pmatrix} \in \text{MPKP} \Leftrightarrow \begin{pmatrix} \overleftrightarrow{x_1} & \overrightarrow{x_1} & \cdots & \overrightarrow{x_k} & \S \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \cdots & \overleftarrow{y_k} & \S\S \end{pmatrix} \in \text{PKP}.$

Die letzte Regel hat die Zahl $k+2$, die vorletzte die Zahl $k+1$.

Es gilt: $w = w_1 \ldots w_n \rightarrow \begin{cases} \overrightarrow{w} & w_1\$w_2\$\ldots w_n\$ & \text{nach den Zeichen steht ein \$} \\ \overleftarrow{w} & \$w_1\$w_2\$\ldots\$w_n & \text{vor den Zeichen steht ein \$} \\ \overleftrightarrow{w} & \$w_1\$w_2\$\ldots\$w_n\$ & \text{vor und nach den Zeichen steht ein \$} \end{cases}$

- Richtung $\Rightarrow$

  MPKP-Lösung $i_1 \ldots i_l$ ergibt PKP-Lösung: $1, i_2 + 1, \ldots, 1_l + 1, k + 2$

- Richtung $\Leftarrow$

  PKP-Lösung $i_1 \ldots i_l$ ergibt MPKP-Lösung:

  - $i_1 = 1$, weil keine andere Regel mit dem gleichen Zeichen oben und unten (\$) beginnt
  - $i_2, \ldots, i_l > 1$, weil x immer ein \$ Vorsprung hat. Deshalb auch $i_l = k + 1$, da dies die einzige Regel ist, einen \$ aufzuholen.

  $\rightarrow 1, i_2, \ldots, i_{l-1}$ MPKP-Lösung nach Definition von $\overrightarrow{\cdot}$, $\overleftarrow{\cdot}$ und $\overleftrightarrow{\cdot}$.

3 Grammatiken

➜ Literatur: [Köbler, Kapitel 1.6]

Definition (Grammatik):
Eine Grammatik ist ein 4-Tupel der Art $G = (V, \Sigma, P, S)$, wobei

- V *eine endliche Menge (**Variablen**)*
- Σ *ein Alphabet (**Terminale**)*
- P ***Produktionen***
- $S \in V$ ***Startvariable***

$P \subseteq (\Sigma \cup V)^+ \times (\Sigma \cup V)^*$ sind Paare von **Satzformen**, wobei die linke durch die rechte ersetzt werden kann.

Notation:

$(u_1 v_1), (u_1 v_2), \ldots, (u_1 v_k) \in P$ schreiben wir als $u \Rightarrow_G v_1, \ldots, v_k$ bzw. $u \Rightarrow v_1, \ldots v_k$

Beispiel 11 (reguläre Ausdrücke):
Sei Σ ein Alphabet. $\text{RegExp}_\Sigma = (\{R\}, \Sigma \cup \{(,), |, *, \emptyset, \varepsilon\}, P, R)$ mit

$$\begin{aligned} P: \quad & R \to \emptyset, \varepsilon \\ & R \to a, a \in \Sigma \\ & R \to (R)^*, (R\,|R), RR \end{aligned}$$

Definition (Ableitung):
Seien u, v Satzformen.

- $u \Rightarrow_G v \Leftrightarrow \exists x, y, y', z$ *Satzformen mit* $u = xyz$, $v = xy'z$, $y \to_G y'$
- $u \Rightarrow_G^k v \Leftrightarrow \exists u_1, \ldots, u_{k-1}$ *Satzformen:* $\underbrace{u \Rightarrow_G u_1 \Rightarrow_G u_2 \Rightarrow_G \ldots \Rightarrow_G u_{k-1} \Rightarrow_G v}_{\text{Ableitung der Länge } k}$
- $u \Rightarrow_G^* v \Leftrightarrow \exists k : u \Rightarrow_G^k v$

Man schreibt $\Rightarrow$, $\Rightarrow^k$ und $\Rightarrow^*$, falls G klar ist.

Beispiel 12 (Ableitung):
Sei $G = \text{RegExp}_{\{0,1\}}$
$R \Rightarrow (R)^* \Rightarrow (RR)^* \Rightarrow (0R)^* \Rightarrow (01)^* \in (\Sigma \cup \{(,),\ |, *, \emptyset, \varepsilon\})^*$

Definition (erzeugte Sprache):
$L(G) = \{x \in \Sigma_G^* \mid S_G \Rightarrow_G^* x\}$

Beispiel 13 (Sprache der regulären Ausdrücke):
Sei $L \subseteq (\Sigma \cup \{(,),\ |, *, \emptyset, \varepsilon\})^*$ die Sprache der regulären Ausdrücke, die wie folgt induktiv definiert ist:

1. $\Sigma \cup \{\emptyset, \varepsilon\} \subseteq L$
2. $x, y \in L \Rightarrow (x)^*, (x|y), xy \in L$

Behauptung:
$L = L(\text{RegExp}_\Sigma)$, d.h. L ist die Sprache, die durch die Grammatik RegExp_Σ erzeugt wird.

Beweis:
Richtung $\subseteq$

Voraussetzung: $x \in L$
Zu zeigen: $x \in L(\text{RegExp}_\Sigma)$

Induktion nach Wortlänge $n = |x|$:

Induktionsanfang $n = 1$**:**

$\rightarrow$ Fall 1. $R \rightarrow \varepsilon, \emptyset, a\ (a \in \Sigma)$

Induktionsschritt

Induktionsvoraussetzung: $n > 1 : \forall y, z \in L,\ |y|, |z| < n$

Induktionsbehauptung: $R \Rightarrow^* y,\ R \Rightarrow^* z$
Induktionsbeweis: $n > 1$: Fall 2:
2a: $x = (y)^*$
2b: $x = (y|z)$
2c: $x = yz$

- 2a: $R \Rightarrow (R)^* \Rightarrow^* (y)^*$

- 2a: $R \Rightarrow (R|R)^* \Rightarrow^* (y|R) \Rightarrow^* (y|z)$
- 2a: $R \Rightarrow RR \Rightarrow^* yR \Rightarrow^* yz$

Richtung $\subseteq$

Voraussetzung: $x \in L(\text{RegExp}_\Sigma)$, d.h.
$R \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow x$

Zu zeigen: $x \in L$

Induktion nach n:

Induktionsanfang $n = 1$:

$x \in \{\emptyset, \varepsilon\} \cup \Sigma \subseteq L$

Induktionsschritt

Induktionsvoraussetzung: Länge der Ableitung $< n \Rightarrow x \in L$

Induktionsbeweis: $n > 1 : \alpha_1 \in \{(R)^*, (R|R), RR\})$, da sonst keine weiteren Ableitungsschritte mehr möglich sind.

Jeder weitere Ableitungsschritt entspricht einer Produktion, die eines der R's ableitet. Reihenfolge sei o.B.d.A. so, dass immer das am weitesten links stehende Nichtterminal als nächstes abgeleitet wird. (Dies ist möglich, weil $n \rightarrow v \rightarrow u = R$) Man spricht dann von *Linksableitung*.

$\alpha_1 = (R)^* : x = (v)^*, R \Rightarrow^{n-1} v$
$\rightarrow$ Induktionsvoraussetzung: $v \in L \rightarrow x \in L$

$\alpha_1 = (R|R)$: wegen Linksableitung: $\exists k : \alpha_k = (y|R)$ mit $k < n$
$\rightarrow R \Rightarrow^{k-1} y, R \Rightarrow^{n-k} z$, wobei $x = (y|z)$
$\rightarrow$ Induktionsvoraussetzung: $y \in L, z \in L \rightarrow (y|z) = x \in L$

Analoges Vorgehen für $\alpha_1 = RR$

Beispiel 14 (Links- und Rechtsableitung):
Rechtsableitung: $R \Rightarrow RR \Rightarrow Ri \Rightarrow hi$

Linksableitung: $R \Rightarrow RR \Rightarrow hR \Rightarrow hi$

Es gibt eine Analogie zwischen Maschinen und Grammatiken: Grammatiken erzeugen Wörter, während Maschinen Wörter akzeptieren oder nicht.

Definition (Grammatik-Typen):
Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

- *G und $L(G)$ heißen vom **Typ 0**, da hier den Regeln keinerlei Einschränkungen auferlegt sind.*
- *Falls $u \to v \in P \Rightarrow |u| \leq |v|$, dann heißen G und $L(G)$ vom **Typ 1** oder **kontextsensitiv**. Es sind also nur verlängernder oder längenerhaltende Produktionen erlaubt.*
- *Falls $u \to v \in P \Rightarrow u \in V$, dann heißen G und $L(G)$ vom **Typ 2** oder **kontextfrei**. Die Variable u, die alleine auf der linken Seite steht, kann unabhängig ihres Auftretens stets durch v ersetzt werden.*
- *Falls $u \to v \in P \Rightarrow u \in V, v \in \Sigma V$ oder $v \in \Sigma$ oder $v = \varepsilon$, dann heißen G und $L(G)$ vom **Typ 3** oder **regulär**. Wenn $|v| \geq 2$, dann besteht v aus mindestens einem Zeichen aus Σ, gefolgt von höchstens einem Zeichen aus V. Rechts stehen also entweder nur Terminalzeichen oder Terminalzeichen gefolgt von einer Variablen.*

ε-Sonderregel für Typ 1:
auch erlaubt: $u \to v : |u| \leq |v|$ oder $u = S$, $v = \varepsilon$, aber dann S niemals in der rechten Seite enthalten. Bei Typ 2 und Typ 3-Grammatiken ist das leere Wort ε jedoch erlaubt.

Bemerkung:
Typ 2 ist auch Typ 1:

- falls $u \to xAz$, $A \to \varepsilon$, dann $u \to xAz, xz$
- falls $S \to \varepsilon$, $y \to xSz$, dann neues Startsymbol S' mit $S' \to \varepsilon, S$

Dazu beachte man insbesondere Satz 5.2 in Kapitel 5 über kontextfreie Sprachen.

Definition (Sprachklassen):
$\mathcal{RE} = \{L \mid L \text{ vom Typ } 0\}$
$\mathcal{CSL} = \{L \mid L \text{ vom Typ } 1\}$
$\mathcal{CFL} = \{L \mid L \text{ vom Typ } 2\}$
$\mathcal{REG} = \{L \mid L \text{ vom Typ } 3\}$

Es gilt: $\mathcal{REG} \subseteq \mathcal{CFL} \subseteq \mathcal{CSL} \subseteq \mathcal{RE}$

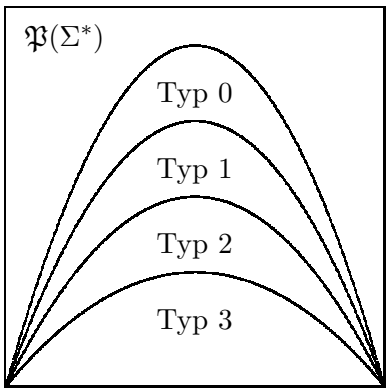


Abbildung 3.1: Chomsky-Hierarchie

3.1 Typ 0 und Typ 1

Definition (linear beschränkter Automat, LBA):
Sei Σ ein Alphabet, $\Sigma' := \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$, $\Gamma \supset \Sigma' \cup \{\square\}$.

*Die 1-TM M mit $M = (Z, \Sigma', \Gamma, \delta, q_0, E)$ heißt **linear beschränkter Automat** oder **LBA** $:\Leftrightarrow$ der Platzbedarf jeder Rechnung bei Eingabe $x_1 x_2 \dots x_{n-1}\hat{x}_n$ höchstens n ist. Dabei darf M nicht-deterministisch sein.*

$x = x_1 \dots x_{n-1}x_n \in L(M) :\Leftrightarrow \exists$ akzeptierte Rechnung. Dabei ist $L(M) \in \Sigma^*$, d.h. ohne Hut.

Beispiel 15 (LBA):
Sei M_2 ein LBA mit $\Sigma = \{a, b, c\}$, $E = \{q_H\}$, $\Gamma = \Sigma' \cup \{A, B, C, \square\}$ und der Überführungsfunktion

δ	a	b	c	$\square$	A	B	C	$\hat{c}$
q_0	q_1AR			$q_H\square N$				
q_1	q_1aR	q_2BR				q_1BR		
q_2		q_2bR	q_3CL				q_2CR	q_4CL
q_3	q_3aL	q_3bL			q_0AR	q_3BL	q_3CL	
q_4					q_HAN	q_4BL	q_4CL	
q_H								

$q_0aabbc\hat{c} \vdash Aq_1abbc\hat{c} \vdash Aaq_1bbc\hat{c} \vdash AaBq_2bc\hat{c} \vdash AaBbq_2c\hat{c} \vdash AaBq_3bC\hat{c} \vdash Aaq_3BbC\hat{c} \vdash Aq_3aBbC\hat{c} \vdash q_3AaBbC\hat{c} \vdash Aq_0aBbC\hat{c} \vdash AAq_1BbC\hat{c} \vdash AABq_1bC\hat{c} \vdash AABBq_2C\hat{c} \vdash AABBCq_2\hat{c} \vdash AABBq_4CC \vdash AABq_4BCC \vdash AAq_4BBCC \vdash Aq_4ABBCC \vdash Aq_HABBCC$

also $a^2b^2c^2 \in L(M)$. Es gilt: $L(M) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$

Satz 3.1 (Zusammenhänge TM/LBA und Sprachen):
sei L eine Sprache

1. L vom Typ 0 $\Leftrightarrow L \in \mathcal{RE}$
2. L vom Typ 1 $\Leftrightarrow \exists M$ LBA : $L = L(M)$

Beweis (von Satz 3.1 (2.)):
Beweis von 2., Richtung $\Rightarrow$

Sei $G = (V, \Sigma, P, S)$ Typ 1-Grammatik. Daraus LBA M, die wie folgt arbeitet:

1. Auswahl einer Produktion $\alpha \to \beta \in P$ (nicht deterministisch)
2. Auswahl eines Vorkommens von β auf Band (nicht deterministisch)
3. ersetze Vorkommen β durch α (linksbündig)
4. nicht benötigter Platz ($|\beta| - |\alpha|$ Zellen) durch Verschieben von δ anhängen
5. ist Bandinschrift $= \hat{S}$ (Startsymbol mit Hut)
   - ja: akzeptieren
   - nein: weiter bei 1.

Es gilt: $x_1 \dots x_n \in L(M) \Leftrightarrow \exists n\ \alpha_i\beta_i\gamma_i\delta_i\ (0 \le i \le n)$:
$\gamma_0\beta_0\delta_0 = x_1 \dots \hat{x}_n$ (Startkonfiguration), $\alpha_i \to \beta_i \in P\ \forall i$
$\gamma_{i+1}\beta_{i+1}\delta_{i+1} = \gamma_i\alpha_i\delta_i, \delta_n\alpha_n\delta_n = \hat{S}$
$\Leftrightarrow S \Rightarrow_G^n x_1 \dots x_n \Leftrightarrow x \in L(G)$

Beweis von 2., Richtung $\Leftarrow$

$L = L(M), M$ ist LBA mit $M = (Z, \Sigma', \Gamma, \delta, q_0, E)$.
Daraus erstelle $G = (V, \Sigma, P, S)$ Typ 1-Grammatik.
Dabei $V = \{S, A', (c, a), ((q, c), a) \mid a, b \in \Sigma, c, d \in \Gamma, q \in Z\}$ und P mit folgenden Regeln:

1. **Startregeln**
   $S \to A(\hat{a}, a),\ ((q_0, \hat{a}), a)$
   $A \to A(a, a),\ ((q_0, a), a)$
   $S \to \varepsilon$, falls $\varepsilon \in L(M)$

($a \in \Sigma, q_0$ Startzustand)
Dabei werden zunächst Paare erzeugt, wobei das letzte das Zeichen mit Hut enthält.
z.B. $S \Rightarrow^n ((q_0, x_1), x_1)(x_2, x_2) \dots (x_{n-1}, x_{n-1})(\hat{x}_n, x_n)$

2. **Überführungsregeln**
   - keine Kopfbewegung:
     $((q, c), a) \to ((q', c'), a)$ $(a \in \Sigma, ((q', c', N) \in \delta(q, c))$
   - Kopfbewegung nach rechts:
     $((q, c), a)(b, d) \to (c', a)((q', b), d)$ $(a, d \in \Sigma, (q', c', R) \in \delta(q, c))$
   - Kopfbewegung nach links:
     $(d, b)((q, c), a) \to ((q', d), b)(c', a)$ $(a, d \in \Sigma, (q', c', L) \in \delta(q, c))$

   Um aus einem Zustand eine Regel zu erzeugen, wird beispielsweise wie folgt vorgegangen:
   $(q_1, A, R) \in \delta(q_0, a) \rightsquigarrow ((q_0, a), a)(b, b) \to (A, a)((q_1, b), b) \in P$

3. **Abschlussregeln**
   $((q, c), a) \to a$
   $(c, a) \to a$

   Dabei wird bei Endzuständen der erste Teil des Paares (Nichtterminale) gelöscht.

Beispiel 16 (für Start-, Überführungs- und Abschlussregeln):
Sei M_2 der LBA aus Beispiel 15 und $L = L(M_2)$ die von ihm akzeptierte Sprache. Sei $abc \in L$ (für $a^n b^n c^n$ mit $n = 1$) ein zu überprüfendes Wort:

$$
\begin{aligned}
S &\Rightarrow A(\hat{c}, c) \Rightarrow A(b, b)(\hat{c}, c) \Rightarrow ((q_0), a), a)(b, b)(\hat{c}, c) && (3.1)\\
&\Rightarrow (A, a)((q_1, b), b)(\hat{c}, c) \Rightarrow (A, a)(B, b)((q_2), \hat{c}), c) && (3.2)\\
&\Rightarrow (A, a)((q_4, B), b)(C, c) \Rightarrow ((q_4, A), a)(B, b)(C, c) && (3.3)\\
&\Rightarrow ((q_H, A), a)(B, b)(C, c) && (3.4)\\
&\Rightarrow a(B, b)(C, c) \Rightarrow ab(C, c) \Rightarrow abc && (3.5)
\end{aligned}
$$

Dabei werden zunächst (3.1) die Startregeln angewendet, die das Wort abc, zusammen mit dem Startzustand auf das Band schreibt. Anschließend werden die Überführungsregeln angewendet (3.2, 3.3). Dabei ist (3.2) als Konfigurationsübergang $Aq_1 b\hat{c} \vdash ABq_2\hat{c}$ zu verstehen. In (3.4) ist ein Endzustand q_H erreicht worden, sodass in (3.5) die Abschlussregeln angewendet werden, also die ersten Teile der Paare gelöscht werden.

O.B.d.A. wird die Reihenfolge 1. Startregeln, 2. Überführungsregeln, 3. Abschlussregeln benutzt.

Zu zeigen: $L(G) = L(M)$

$x \in L(G)$

$\Leftrightarrow \exists m : S \Rightarrow^m x$

$\Leftrightarrow S \Rightarrow^n \underbrace{((q_0, x_1), x_1)(x_2, x_2) \dots (x_{n-1})(x_{n-1})(\hat{x}_n, x_n)}_{=: \alpha} \Rightarrow^{m-n} x$

$\Leftrightarrow S \underbrace{\Rightarrow^n}_{(1)} \alpha \underbrace{\Rightarrow^{m-2n}}_{(2)} (c_1, x_1) \dots (c_l, x_l)((q, c_{l+1}), x_{l+1}) \dots (c_n, x_n) \underbrace{\Rightarrow^n}_{(3)} x$

Dabei ist α die 1. Satzform. die weder A noch S enthält und $x = x_1 x_2 \dots x_n$.

Es werden o.B.d.A. erst die Überführungsregeln und erst dann die Abschlussregeln benutzt, da in allen Überführungsregeln auf der linken Seite kein Terminal vorkommt. Reihenfolge der Regeln:

1. n Startregeln, für ein Wort der Länge n (1)
2. $m-2n$ Überführungsregeln für $m-2n$ Konfigurationsübergänge (längenerhaltend) (2)
3. n Abschlussregeln, für ein Wort der Länge n (längenerhaltend) (3)

Damit $((q, c_{l+1}), x_{l+1})$ abgeleitet werden kann, muss $q \in E$ (Endzustand) sein

$\Leftrightarrow K_x \vdash_M^{m-2n} c_1 \dots c_l q c_{l+1} \dots c_n \Leftrightarrow x \in L(M)$

□

Beweis (von Satz 3.1, (1.)):
Analog zum Beweis von Satz 3.1, (2.):

$\Rightarrow$ der Automat kann (bei verkürzenden Produktionen) den durch die Eingabe benutzten Bereich verlassen

$\Leftarrow$ es müssen zusätzliche Überführungsregeln für den Fall, dass der Automat die Bandinschrift erweitert hinzugefügt werden, sowie eine Abschlussregel für „□“

3.2 Kontextsensitive Sprachen

Definition (Wortproblem):
*Das **Wortproblem** besteht, bei einer gegebenen Sprache L und einem Wort x darin, zu entscheiden ob $x \in L$.*

Bemerkung:
Eine Sprache kann durch eine Turingmaschine, ein LBA oder eine Grammatik angegeben werden. Diese Formen sind nach Satz 3.1 äquivalent.

Bemerkung:
Das Wortproblem ist für die Klasse $\mathcal{RE}$ nicht entscheidbar (es handelt sich um das Halteproblem).

Satz 3.2:
Das Wortproblem ist für die Klasse $\mathcal{CSL}$ entscheidbar.

Beweis (von Satz 3.2):
Sei M LBA und $L = L(M)$, sowie x ein Wort. Der LBA kann bei Eingabe $x = x_1 \dots x_n$ höchstens $|Z| \cdot n \cdot |\Gamma|^n$ verschiedene Konfigurationen haben.

Ein LBA akzeptiert entweder ein Wort oder gerät in eine Endlosschleife. Einer dieser Fälle tritt nach $\leq |Z| \cdot n \cdot |\Gamma|^n$ Konfigurationen ein.

Sei nun $G = (V, A)$ ein gerichteter Graph, wobei

- $V = \{K \mid$ Konfiguration der Länge $n\}$
- $A = \{(K, K') \mid K \vdash K'\}$

Dann gilt: $K \vdash^* K' \Leftrightarrow \exists K - K'$ Weg in $G = (V, A)$. Damit entscheidet man das Wortproblem zu $x = x_1 \dots x_n$ wie folgt:

- Aufbau von $G = (V, A)$
- feststellen, ob von K_x aus ein $K' \in E \times \Gamma^* \times \Gamma^*$ erreichbar ist (z.B. mit Tiefensuche)

Korollar:
$\mathcal{CSL} \subseteq \mathcal{REC} \subsetneq \mathcal{RE}$ (z.B. $H \in \mathcal{RE} \backslash \mathcal{CSL}$)

Bemerkung:
Das Verfahren hat exponentialen Zeitbedarf und ist deswegen „nicht effektiv“.

4 Reguläre Sprachen

4.1 endliche Automaten

Definition (endlicher Automat, DFA, NFA):
Sei $M = (Z, \Sigma, \delta, q_0, E)$ *mit*

- Z *endliche Zustandsmenge*
- Σ *Alphabet*
- $q_0 \in Z$ *Startzustand*
- $E \subseteq Z$ *Endzustände*
- $\delta : Z \times \Sigma \to Z$ *Überführungsfunktion*

M heißt ***endlicher Automat****, oder kurz* ***DFA****.*
Sei $M = (Z, \Sigma, \delta, Q_0, E)$ *mit*

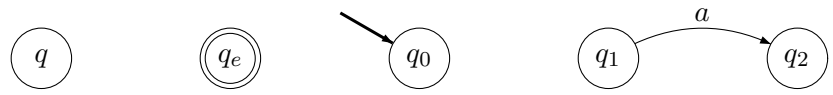
- Z *endliche Zustandsmenge*
- Σ *Alphabet*
- $Q_0 \subseteq Z$ *Startzustände*
- $E \subseteq Z$ *Endzustände*
- $\delta : Z \times \Sigma \to \mathfrak{P}(Z)$ *Überführungsfunktion*

M heißt ***nicht-deterministischer endlicher Automat****, oder kurz* ***NFA****.*

Definition (akzeptierte Sprache):
Sei M DFA und $x = x_1 \dots x_n$.
$L(M) = \{x \mid \exists q_1, \dots, q_{n-1} \in Z \; \exists q_n \in E \; \forall i \in \{0, \dots, n-1\} : \delta(q_i, x_{i+1}) = q_{i+1}\}$

Sei N NFA und $x = x_1 \dots x_n$.
$L(N) = \{x \mid \exists q_0 \in Q_0, \exists q_1 \dots, q_{n-1} \in Z \; \exists q_n \in E \; \forall i \in \{0, \dots, n-1\} : q_{i+1} \in \delta(q_i, x_{i+1})\}$

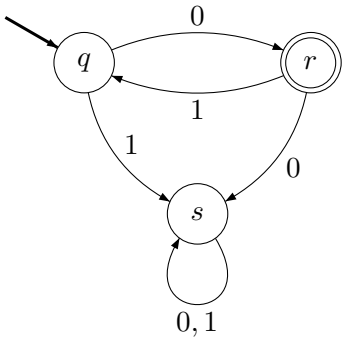
Darstellung von DFAs/NFAs als gerichteter Graph:



Zustand Endzustand Startzustand Zustandsübergang

Beispiel 17 (Graphische Darstellung eines DFA):
Sei M_3 ein DFA mit $M_3 = (\{q, r, s\}, \{0, 1\}, \delta, q, \{r\})$ mit der Überführungsfunktion δ :

δ	0	1
q	r	s
r	s	q
s	s	s



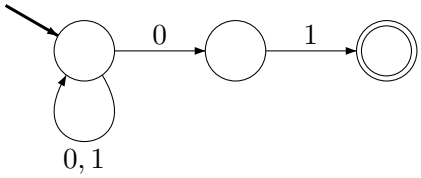
Es gilt: $L(M_3) = \{0, 010, 01010, \ldots\} = 0(10)^*$

Beweis:
Wenn M_3 beim Lesen eines Wortes x in den Zustand s übergeht, gilt $x \notin L(M_3)$, da von dort kein Endzustand erreicht werden kann.

x wird akzeptiert $\Leftrightarrow$ M_3 pendelt beim Lesen zwischen q und r und hört bei r auf.

$\Rightarrow L = 0(01)^*$

Beispiel 18 (Graphische Darstellung eines NFA):
Sei M_4 ein NFA mit folgendem Graphen:



$L(M_4) = \{0, 1\}^*01$

➜ Literatur: [Köbler, Kapitel 1.2, 1.6]

Satz 4.1 (Äquivalenz DFA, NFA, Typ-3-Grammatik):
Sei $L \subseteq \Sigma^*$ eine Sprache. Dann sind folgende Aussagen äquivalent:

(i) $\exists$ DFA $M : L = L(M)$

(ii) L ist vom Typ 3

(iii) $\exists$ NFA $N : L = L(N)$

Beweis (von Satz 4.1):
(i) $\Rightarrow$ (ii):

Aus gegebenem DFA $M = (Z, \Sigma, \delta, q_0, E)$ wird Grammatik $G = (V, \Sigma, P, S)$ gebildet, wobei

- $V = Z$ (jeder Zustand entspricht einer Variable)
- $S = q_0$ (der Startzustand entspricht dem Startsymbol)
- $P = \{q \to ar \mid a \in \Sigma, q, r \in V, r = \delta(q, a)\} \cup \{q \to \varepsilon \mid q \in E\}$

Sei $x = x_1 \dots x_n$ ein Wort der Länge n.

$$
\begin{aligned}
& x \in L(M) \\
\Leftrightarrow\ & \exists q_1, \dots, q_{n-1} \in Z\ \exists q_n \in E\ \forall i \in \{0, \dots, n-1\} : q_{i+1} = \delta(q_i, x_{i+1}) \\
\Leftrightarrow\ & \exists q_0, \dots, q_n \in V : \forall i \in \{0, \dots, n-1\} : q_i \to x_{i+1} q_{i+1} \in P, q_n \to \varepsilon \in P \\
\Leftrightarrow\ & q_0 \Rightarrow_G^* x_1 \dots x_i q_i \Rightarrow_G^* x_1 \dots x_n q_n \Rightarrow_G x_1 \dots x_n \\
\Leftrightarrow\ & x \in L(G)
\end{aligned}
$$

(ii) $\Rightarrow$ (iii):

Aus $G = (V, \Sigma, P, S)$ wird zunächst $G' = (V \dot\cup \{X\}, \Sigma, P', S)$ gebildet. Dabei enthält P' alle Produktionen von P, außer solchen der Form $A \to a$. Für diese enthält P' die Produktion $A \to aX$ und außerdem $X \to \varepsilon \in P'$.

Satz:
$x \in L(G) \Leftrightarrow x \in L(G')$

Beweis (Richtung „$\Rightarrow$“):
$x \in L(G)$, also $S \Rightarrow_G^n x$. Dann gilt entweder $S \Rightarrow_G^{n-1} x_1 \dots x_n A \Rightarrow_G x$ (also $A \to \varepsilon$) oder $S \Rightarrow_G^{n-1} x_1 \dots x_{n-1} A \Rightarrow_G x$ (also $A \to x_n$).

- Fall 1: Alle Produktionen sind aus $P \cap P'$ und damit $x \in L(G')$
- Fall 2: In diesem Fall sind die Produktionen von $S \Rightarrow_G^{n-1} x_1 \dots x_{n-1} A$ aus $P \cap P'$, also gilt: $S \Rightarrow_{G'}^{n-1} x_1$. Die letzte Produktion ist $A \to x_n$. Deshalb $A \to x_n X, X \to \varepsilon \in P'$. Daher gilt: $S \Rightarrow_{G'}^{n-1} x_1 \dots x_n A \Rightarrow_{G'} x$.

Beweis (Richtung „$\Leftarrow$“):
$S \Rightarrow_{G'}^{n+1} x$, dann $S \Rightarrow_{G'}^{n} xA$.

- Falls $A \neq X$, $S \Rightarrow_G^n xA \Rightarrow_G x$. Dann wurden nur Produktionen aus G angewendet, also $x \in L(G)$.
- Falls $A = X$, dann war die vorletzte Produktion $A \to x_n X, A \in V$ mit $X \notin V$ und damit laut Voraussetzung $A \to x_n \in P$. Daher $S \Rightarrow_G^n S$ und damit $x \in L(G)$. □

Aus G' wird nun NFA N gebildet mit $N = (Z, \Sigma, \delta, Q_0, E)$ mit

- $Z = V$
- $Q_0 = \{S\}$
- $E = \{A \in V \mid A \to \varepsilon \in P'\}$
- $\delta(A, a) = \{B \in V \mid A \to aB \in P'\}$

Sei $x = x_1 \dots x_n$ ein Wort der Länge n.

$$
\begin{aligned}
& x \in L(N) \\
\Leftrightarrow\ & \exists q_1 \dots q_{n-1} \in Z\ \exists q_n \in E : \forall i \in \{0, \dots, n-1\} : q_{i+1} \in \delta(q_i, x_{i+1}) \\
\Leftrightarrow\ & \exists q_1 \dots q_n \in V : q_n \to \varepsilon \in P'\ \forall i \in \{0, \dots, n-1) : q_i \to x_{i+1} q_{i+1} \in P' \\
\Leftrightarrow\ & \exists q_1 \dots q_n \in V : S \Rightarrow_{G'}^i x_1 \dots x_i q_i \Rightarrow_{G'}^{n-i} x q_n \Rightarrow_{G'} x\ \forall i \in \{0, \dots, n-1\} \\
\Leftrightarrow\ & x \in L(G')
\end{aligned}
$$

(iii) ⇒ (i):

Aus $N = (Z, \Sigma, \delta, Q_0, E)$ wird $M = (Z', \Sigma, \delta', q_0, E')$ mit

- $Z' = \mathfrak{P}(Z)$
- $q_0 = Q_0$
- $E' = \{Q \subseteq Z \mid E \cap Q \neq \emptyset\}$
- $\delta'(Q, a) = \bigcup_{q \in Q} \delta(q, a)$ M merkt sich die jeweils erreichbaren Zustände.

Sei $x = x_1 \dots x_n$ ein Wort der Länge n.

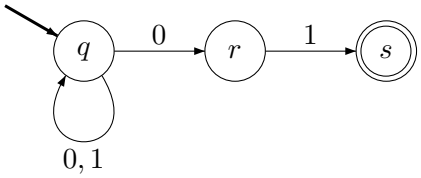
$$
\begin{aligned}
& x \in L(M) \\
\Leftrightarrow\ & \exists q_1 \dots q_{n-1} \in Z'\ \exists Q_n \in E'\ \forall i \in \{0,\dots,n-1\} : Q_{i+1} = \delta'(Q_i, x_{i+1}) \\
\Leftrightarrow\ & \exists Q_1 \dots Q_n \subseteq Z, Q_n \cap E \neq \emptyset, \forall i \in \{0,\dots,n-1\} : Q_{i+1} = \bigcup_{q \in Q_i} \delta(q, x_{i+1}) \\
\Leftrightarrow\ & \exists q_n \in E : \exists Q_1 \dots Q_{n-1}\ \forall i \in \{0,\dots,n-2\} : Q_{i+1} = \bigcup_{q \in Q_i} \delta(q, x_{x+1}) \\
& \text{und } q_n \in \bigcup_{q \in Q_{n-1}} \delta(q, x_n) \\
\Leftrightarrow\ & \exists q_{n-1} \in Z\ \exists q_n \in E\ \exists Q_1 \dots Q_{n-2} \subseteq Z : \forall i \in \{0,\dots,n-3\} : Q_{i+1} = \bigcup_{q \in Q_i} \delta(q, x_{i+1}) \\
& \text{und } q_n \in \delta(q_{n-1}, x_n), q_{n-1} \in \bigcup_{q \in Q_{n-2}} \delta(q, x_{n-1}) \\
\Leftrightarrow\ & \exists q_0 \in Q_0\ \exists q_1 \dots q_{n-1} \in Z\ \exists q_n \in E : \forall i \in \{0,\dots,n-1\} : q_{i+1} \in \delta(q_i, x_{i+1}) \\
\Leftrightarrow\ & x \in L(N)
\end{aligned}
$$

Zu dieser Konstruktion beachte man Beispiel 19.

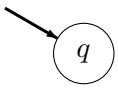
□

Beispiel 19 (Konstruktion eines DFA aus NFA):
Es wird aus dem NFA M_4 aus Beispiel 18 ein DFA M konstruiert. Dabei wird o.B.d.A. bei der Tiefensuche zunächst der Fall für die Eingabe 0 überprüft.

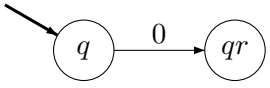
Der NFA M_4:



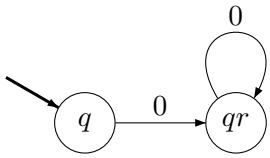
1. Der Startzustand von M ist q.



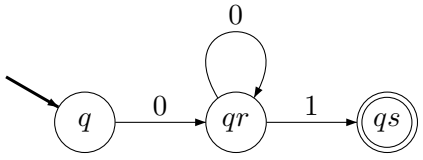
2. Bei der Eingabe 0 im Zustand q können die Zustände q und r erreicht werden ⇒ neuer Zustand qr in M.



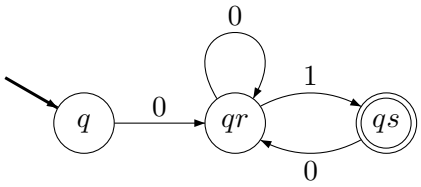
3. Die Eingabe 0 im Zustand q führt wieder in den Zustand q.



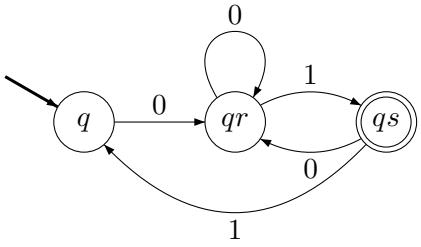
4. Bei der Eingabe 1 im Zustand qr können die Zustände q und s (Endzustand) erreicht werden ⇒ neuer Endzustand qs in M.



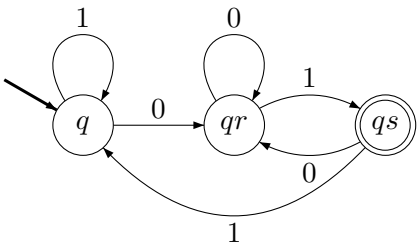
5. Die Eingabe 0 im Zustand qs führt wieder in den Zustand qr.



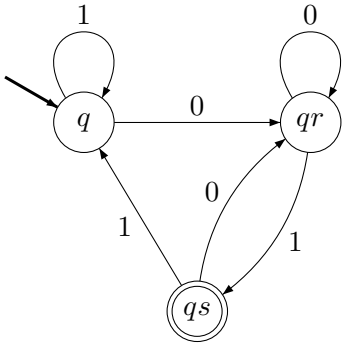
6. Die Eingabe 1 im Zustand qs führt wieder in den Zustand q.



7. Die Eingabe 1 im Zustand q führt wieder in den Zustand q.



Der fertige DFA M:



Korollar:
Ein DFA kann ohne überflüssige, d.h. nicht vom Startzustand erreichbare Zustände direkt aus einem NFA konstruiert werden.

Korollar (aus Satz 4.1):
Seien $M_1 = \{Z_1, \Sigma, \delta_1, q_1, E\}$ und $M_2 = \{Z_2, \Sigma, \delta_2, q_2, E_2\}$ DFAs. Dann gilt:

(i) $L(M_1)^* \in \mathcal{REG}$

(ii) $L(M_1)L(M_2) \in \mathcal{REG}$

Beweis:
(i): $L(M_1)^* \in \mathcal{REG}$

Sei $N = (Z_1 \dot\cup \{q_{\text{neu}}\}, \Sigma, \delta, Q_0, E\}$ ein NFA mit

- $Q_0 = \{q_1, q_{\text{neu}}\}$
- $E = E \cup \{q_{\text{neu}}\}$
- $\delta(q,a) = \begin{cases} \{\delta_1(q,a)\} & q \notin E_1 \\ \{\delta_1(q,a), \delta_1(q_1,a)\} & \text{sonst} \end{cases}$

$x \in L(N)$
$\Leftrightarrow x = \varepsilon\ (q_{\text{neu}} \in Q_0) \vee x = x_1 \dots x_n$ ($x_i \in \Sigma^*$ Wörter, bei denen beim Lesen des 1. Zeichens von x_i ($i \geq 2$) einer der neuen Übergänge verwendet wird)
$\Leftrightarrow x = \varepsilon \vee x = x_1 \dots x_i$ und $\forall i\ x_i \in L(M_1)$
$\Leftrightarrow x \in L(M_1)^*$

(ii): $L(M_1)L(M_2) \in \mathcal{REG}$

Analog sei $N' = (Z_1 \dot\cup Z_2, \Sigma, \delta', Q_0', E')$ NFA mit

- $Q_0 = \begin{cases} \{q_1\} & q_1 \notin E' \\ \{q_1, q_2\} & \text{sonst} \end{cases}$
- $E' = \begin{cases} E_2 & q_2 \notin E_2 \\ E_1 \cup E_2 & \text{sonst} \end{cases}$
- $\delta'(q,a) = \begin{cases} \{\delta_1(q,a)\} & q \in Z_1 \backslash E_1 \\ \{\delta_1(q,a), \delta_2(q_2,a)\} & q \in E_1 \\ \{\delta_2(q,a)\} & \text{sonst} \end{cases}$

□

4.2 reguläre Ausdrücke

➜ Literatur: [Köbler, Kapitel 1.3, 1.2]

Erinnerung: Die reguläre Ausdrücke waren definiert durch $\text{RegExp}_\Sigma = (\{R\}, \Sigma \cup \{\emptyset, \varepsilon, (,), |, *\}, P, R)$ mit $P : R \to \emptyset, \varepsilon, a \ (a \in \Sigma),\ R \to RR, (R|R), (R)^*$.

γ regulärer Ausdruck $\Leftrightarrow \gamma \in L(\text{RegExp}_\Sigma)$

Definition (dargestellte Sprache $L(\gamma)$):

$L(\emptyset) = \emptyset$
$L('\varepsilon') = \{\varepsilon\}$
$L(a) = \{a\}$

$\gamma = \alpha\beta$	$L(\gamma) = L(\alpha)L(\beta)$
$\gamma = (\alpha\|\beta)$	$L(\gamma) = L(\alpha) \cup L(\beta)$
$\gamma = \alpha^*$	$L(\gamma) = L(\alpha)^*$

Satz 4.2:
$\{L(\gamma) \mid \gamma \text{ regulärer Ausdruck } \} = \mathcal{REG}$

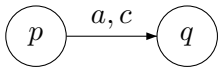
Beweis:
Richtung $\subseteq$:

siehe oben, sowie Übungsaufgabe 28 ($A \cup B = \overline{\overline{A} \cap \overline{B}}$).

Richtung $\supseteq$:

Sei M DFA mit $M = (Z, \Sigma, \delta, q_0, E)$. Zu zeigen: $\exists \gamma : L(\gamma) = L(M)$

Notation: $\hat{\delta}(p, \varepsilon) = p$, $\hat{\delta}(p, xa) = \delta(\hat{\delta}(p, x), a)$



$\hat{=} \quad \gamma^0_{p,q} = (a|c)$

O.B.d.A: $Z = \{1, \ldots, l\}$. Sei außerdem $x = x_1 \ldots x_n$

Dann: $L^r_{pq} = \{x \in \Sigma^* \mid \hat{\delta}(p, x) = q \; \forall i \in \{1, \ldots, n-1\} : \hat{\delta}(p, x_1 \ldots x_i) \leq r\}$

1. Wir wissen: $\exists \gamma^0_{pq} : L^0_{pq} = \begin{cases} \{a \mid \delta(p,a) = q\} & \text{falls } p \neq q \\ \{a \mid \delta(p,a) = q\} \cup \{\varepsilon\} & \text{sonst} \end{cases} = L(\gamma^0_{pq})$
2. $L^{r+1}_{pq} = L^r_{pq} \cup L^r_{p,r+1}(L^r_{r+1,r+1})^* L^r_{r+1,q} \overset{\text{induktiv}}{=} \gamma^{r+1}_{pq} = \gamma^r_{pq} \mid \gamma^r_{p,r+1}(\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}$
3. $E = \{i_1 \ldots i_k\} \Rightarrow \gamma = (\gamma^l_{q_o,i_1} | \gamma^l_{q_o,i_2} | \ldots | \gamma^l_{q_o,i_k})$ mit
   $L(\gamma) = \bigcup_{i_j \in E} L(\gamma^l_{q_o,i_j}) = \bigcup_{i_j \in E} L(L^l_{q_o,i_j}) = L(M)$

□

4.3 Pumping-Lemma

➜ Literatur: [Köbler, Kapitel 1.4]

Satz 4.3 (Pumping-Lemma für reguläre Sprachen):
$L \in \mathcal{REG} \Rightarrow \exists l \in \mathbb{N} : \forall x \in L : |x| \geq l \Rightarrow \exists u, v, w : x = uvw$

(i) $v \neq \varepsilon$

(ii) $|uv| \leq l$

(iii) $uv^i w \in L \; \forall i \in \mathbb{N}$

Beispiel 20:
Sei $L = \{a^n b^n \mid n \in \mathbb{N}\} \notin \mathcal{REG}$

Annahme: $L \in \mathcal{REG}$.

Dann gibt es nach Satz 4.3 $l \in \mathbb{N}$, sodass alle Worte x mit $|x| \geq l$ in Satz 4.3 beschrieben zerlegt werden können: z.B. $x = a^l b^l$ ($|x| = 2l \geq l$).

Dann $\exists u, v, w$ wie in Satz 4.3 und wegen (ii) $uv = a^k$, $k \leq l$ und wegen (i) $v = a^m$, $m \geq 1$. Wegen (iii) $uv^2w = a^{l+m}b^l \in L$. Dies ist ein Widerspruch.

Beweis (von Satz 4.3):
Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA, sodass $L(M) = L$. Wähle $l = |Z|$. Beim Lesen eines Wortes x mit $|x| \geq l$ durchläuft M mindestens einen Zustand mindestens zweimal (Zustand q).

- u Teilwort bis 1. Mal Erreichen des Zustandes q
- v Teilwort bis 2. Mal Erreichen des Zustandes q
- w Rest von x

Dann $\hat{\delta}(q, v) = q$ und deshalb $\hat{\delta}(q, v^i) = q \; \forall i \in \mathbb{N}$ und damit
$\hat{\delta}(q_0, uv^iw) = \hat{\delta}(\hat{\delta}(\hat{\delta}(q_0, u), v^i), w) = \hat{\delta}(\hat{\delta}(q, v^i), w) = \hat{\delta}(q, w) \in E$

Lemma:
Die Umkehrung des Pumping-Lemmas für reguläre Sprachen gilt nicht.

Beweis:
Durch Gegenbeispiel in Übungsaufgabe 31.

4.4 Anzahl der Zustände eines DFA

➜ Literatur: [Köbler, Kapitel 1.5] und [Wegener, Kapitel 4.2]

Definition (Sprache von q, Äquivalenzklassen von Zuständen):
Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA ohne überflüssige Zustände, d.h. Zustände, die nicht vom Startzustand aus erreicht werden können.

$\forall q \in Z : L_q = \{x \mid \hat{\delta}(q, x) \in E\}$ *die* ***Sprache von q****, die alle Teilworte x enthält, mit denen von q aus ein Endzustand erreicht werden kann.*

$\tilde{q} = \{p \mid L_p = L_q\}$ *und* $\forall Q \subseteq Z : \tilde{Q} = \{\tilde{q} \mid q \in Q\}$ *sind* ***Äquivalenzklassen****. Von den Zuständen q und p aus kann mit dem gleichen Restwort ein Endzustand erreicht werden.*

$\forall p, q \in Z : L_p = L_q \Leftrightarrow \tilde{p} = \tilde{q}$

Definition:
$L_x := \{z \mid xz \in L = L(M)\} = L_{\hat{\delta}(q_0,x)}$ *ist die Menge aller Teilworte* z*, mit denen* xz *zur Sprache* L *gehört.*

Satz (Minimale Anzahl von Zuständen):
Der DFA $\tilde{M} = (\tilde{Z}, \Sigma, \delta', \tilde{q_0}, \tilde{E})$ mit $\delta'(\tilde{p}, a) = \widetilde{\delta(p,a)}$ und $L(\tilde{M}) = L(M)$ hat die minimale Anzahl von Zuständen.

Bemerkung:
Es werden die Zustände aus M in Äquivalenzklassen zusammengefasst und δ' ist der Übergang von einer Äquivalenzklasse in eine andere. Dabei ist p ein Repräsentant der jeweiligen Äquivalenzklasse $\tilde{p}$.

Beweis:
Der vorherige Satz enthält folgende Aussagen:

(i) δ' ist wohldefiniert

(ii) $L(\tilde{M}) = L(M)$

(iii) $k = ||\tilde{Z}||$ ist minimal

Diese Aussagen werden nun einzeln bewiesen:

(i) $p, q \in Z$ mit $\widetilde{\delta(p,a)} \neq \widetilde{\delta(q,a)} \Rightarrow \exists L_{\delta(p,a)} \;\Delta\; L_{\delta(q,a)}$ (also Worte, die nur in einer Sprache sind)

$\Rightarrow \exists x : ax \in L_p \;\Delta\; L_q \Rightarrow L_p \neq L_q \Rightarrow \tilde{p} \neq \tilde{q}$

Also: $\tilde{p} = \tilde{q} \Rightarrow \widetilde{\delta(p,a)} = \widetilde{\delta(q,a)}$ und deshalb δ' wohldefiniert.

(ii) Sei $x = x_1 \dots x_n \in \Sigma^*$ ein Wort. M durchläuft beim Lesen von x die Zustände $q_0, \dots, q_n$ und $\tilde{M}$ die Zustände $\tilde{q_0} \dots \tilde{q_n}$

$x \in L(M) \Leftrightarrow q_n \in E \Leftrightarrow \tilde{q_n} \in \tilde{E} \Leftrightarrow x \in L(\tilde{M})$

(iii) $k := ||\tilde{Z}||$ minimal

$p, q \in Z$ mit $\tilde{p} \neq \tilde{q} \Rightarrow \exists x, y \in \Sigma^* : \hat{\delta}(q_0, x) = p, \hat{\delta}(q_0, y) = q$ (keine überflüssigen Zustände)

$L_x = L_p \neq L_q = L_y$

$k \leq ||\{L_x \mid x \in \Sigma^*\}|| =: l$ (alle L_x verschieden)

Sei $M' = (Z_{M'}, \Sigma, \delta_{M'}, q_{M'}, E_{M'})$ ein DFA mit weniger als l Zuständen. Dann gibt es $x, y \in \Sigma^*$, sodass $L_x \neq L_y$, aber $\hat{\delta}_{M'}(q_{M'}, x) = \hat{\delta}_{M'}(q_{M'}, y) = q^*$.

Sei o.B.d.A. $u \in L_x \backslash L_y$. Dann ist $xu \in L$, aber $yu \notin L$.

$xu \in L \Rightarrow \hat{\delta}_{M'}(q^*, u) \in E_{M'}$ und $yu \notin L \Rightarrow \hat{\delta}_{M'}(q^*, u) \notin E_{M'}$

Dies ist ein Widerspruch. Kein DFA mit weniger als k Zuständen akzeptiert L.

□

Definition (Index einer Sprache):
Sei $L \subseteq \Sigma^*$ eine Sprache. Man nennt $||\{L_x \mid x \in \Sigma^*\}||$ ***Index der Sprache L****. Dabei ist stets $L_\varepsilon = L$.*

Korollar:
Sei $L \subseteq \Sigma^*$ eine Sprache. $||\{L_x \mid x \in \Sigma^*\}|| = \infty \Rightarrow L \notin \mathcal{REG}$. Sprachen mit einem unendlichen Index sind also nicht regulär.

Beispiel 21 (unendlicher Index einer Sprache):
Sei $L = \{a^n b^n \mid n \in \mathbb{N}\}$ eine Sprache.

Für $i < j$ gilt: $b^i \in L_{a^i} \backslash L_{a^j}$, denn $a^i b^i \in L$, aber $a^i b^j \notin L$.

Der Index von L ist $||\{L_{a^i} \mid i \in \mathbb{N}\}|| = \infty$. Da der Index von L ist unendlich ist, ist L nicht regulär.

Beispiel 22 (endlicher Index einer Sprache):
Sei $L = \{a^i \mid i \in \mathbb{N}\} \subseteq \{a, b\}^*$ eine Sprache.

$L_{a^i} = L$. Wenn x ein b enthält, so ist $L_x = \emptyset$.

Der Index der Sprache L ist 2.

Definition (Zeuge, kürzester Zeuge):
$\tilde{p} \neq \tilde{q} \Rightarrow \exists x \in L_p \ \Delta \ L_q$. Dieses Wort x wird ***Zeuge*** *für $\tilde{p} \neq \tilde{q}$ in einer Sprache genannt.*

Das Wort x ist der ***kürzeste Zeuge*** *dafür, falls $|x|$ minimal.*

ax ist der kürzeste Zeuge für $\tilde{p} \neq \tilde{q}$ ($a \in \Sigma, x \in \Sigma^*$)
$\Leftrightarrow$ x kürzester Zeuge für $\delta(p, a) \neq \delta(q, a)$ (da x der Nachfolger von a ist)

Es gilt: Gibt es keine kürzesten Zeugen der Länge $m \in \mathbb{N}$, dann auch keinen der Länge m', $m' < m$.

Algorithmus, M in $\tilde{M}$ zu überführen

Im folgenden sind U, U' Listen, in denen die Zustände gesammelt werden, die nicht zusammengefasst werden können.

- **Initialisierung:**

  $U \leftarrow \{\{p,q\} \mid p \in E, q \in Z \backslash E\}$ (Endzustände können nicht mit „Nichtendzuständen“ zusammengefasst werden.)

- **Schleife:**

  Wiederhole

  - $U' \leftarrow \{\{p,q\} \mid \exists a \in \Sigma : \{\delta(p,a), \delta(q,a)\} \in U\} \backslash U$
  - $U \leftarrow U \cup U'$

  bis $U' = \emptyset$

Satz 4.5:
Der Algorithmus überführt M in $\tilde{M}$.

Beweis (von Satz 4.5):
Der vorherige Satz enthält folgende Aussagen:

(i) der Algorithmus terminiert

(ii) nach der Schleife gilt: $\{p,q\} \in U \Leftrightarrow \tilde{p} \neq \tilde{q}$

Diese Aussagen werden nun einzeln bewiesen:

(i) In jedem Schleifendurchlauf kommt ein Paar zu U. Es gibt weniger als $|Z|^2$ Paare, nämlich genau $\binom{|Z|}{2}$. Also ist die Anzahl der Durchläufe kleiner oder gleich $|Z|^2$.

(ii) Ist $\tilde{p} \neq \tilde{q}$ und x ein kürzester Zeuge dafür, dann kommt $\{p,q\}$ im k. Durchlauf zu U ($k = |x|$). Dabei ist der 0. Durchlauf die Initialisierung.

Induktion über k:

Induktionsanfang ($k = 0$):
$\varepsilon \in L_p \,\Delta\, L_q \Leftrightarrow p \in E \wedge q \in Z \backslash E \vee q \in E \wedge p \in Z \backslash E$

Induktionsschritt ($k \geq 1$):
$\Rightarrow x = aq, a \in \Sigma, y \in \Sigma^*$ ist kürzester Zeuge für $\widetilde{\delta(p,a)} \neq \widetilde{\delta(q,a)}$, d.h. die Nachfolgezustände von a liegen auseinander.

Induktionsvoraussetzung:
$\Rightarrow \{\delta(p,a), \delta(q,a)\}$ kam $(k-1)$. Durchlauf zu U.
$\Rightarrow \{p,q\}$ kommt im k. Durchlauf zu U.

Bricht die Schleife nach dem n. Durchlauf ab, so gibt es keinen kürzesten Zeugen der Länge n oder größer.

Daher gilt: Ist $\{p, q\} \notin U$ zu diesem Zeitpunkt, dann gibt es keinen Zeugen für $\tilde{p} \neq \tilde{q}$, also $\tilde{p} = \tilde{q}$. □

Bemerkung:
Der Algorithmus hat die Laufzeit $\mathcal{O}\left(|Z|^4|\Sigma|\right)$. Er kann auch mit der Komplexität $\mathcal{O}\left(|Z|^2|\Sigma|\right)$ berechnet werden.

5 Kontextfreie Sprachen

➜ Literatur: [Köbler, Kapitel 2.1]

5.1 Kellerautomaten

Definition (Kellerautomat, Kelleralphabet, Kelleranfangszeichen):
Ein Automat $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ *mit*

- Z, Σ, z_0 *wie bisher*
- Γ ***Kelleralphabet***
- $\# \in \Gamma$ ***Kelleranfangszeichen***
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathfrak{P}_e(Z \times \Gamma^*)$

nennt man ***Kellerautomat*** *oder* ***pushdown automaton/PDA****.*

$\mathfrak{P}_e$ *ist dabei die endliche Potenzmenge (M kann nichtdeterministisch sein) mit* $\mathfrak{P}_e(A) = \{B \subseteq A \mid B \text{ endlich}\}$.

Definition (Übergang, Konfiguration, Nachfolgekonfiguration):
Übergang: $quA \to pB_1 \dots B_k$, *falls* $(p, B_1 \dots B_k) \in \delta(q, u, A)$
$(p, q \in Z, u \in \Sigma \cup \{\varepsilon\}, A, B_1, \dots, B_k \in \Gamma)$

Konfiguration: $K \in Z \times \Sigma^* \times \Gamma^*$

Nachfolgekonfiguration: $K \vdash K'$, *falls* $K = (q, x_i \dots x_n, A_1 \dots A_l)$
$qx_iA_1 \to pB_1 \dots B_k$ *und* $K' = (p, x_{i+1} \dots x_n, B_1 \dots B_k A_2 \dots A_l)$
oder $q\varepsilon A_1 \to pB_1 \dots B_k$ *und* $K' = (p, x_i \dots x_n, B_1 \dots B_k A_2 \dots A_l)$ *(*ε*-Übergang, also 1. Zeichen nicht gelesen)*

Definition (akzeptiertes Wort):
$x \in L(M) \Leftrightarrow (q_0, x, \#) \vdash^* (p, \varepsilon, \varepsilon)$ *($p \in Z$).*

Achtung: *Bei leerem Keller ist kein Übergang möglich. δ ist total.*

Beispiel 23 (Kellerautomat):
Sei $M = (\{p, q\}, \{a, b\}, \{A, \#\}, \delta, q, \#)$ ein Kellerautomat mit $L(M) = \{a^n b^n \mid n \in \mathbb{N}\}$ und der Überführungsfunktion δ wie folgt:

$$\begin{array}{lcll} q, \varepsilon, \# & \rightarrow & p & (1) \\ q, a, \# & \rightarrow & q, A & (2) \\ q, a, A & \rightarrow & q, AA & (3) \\ q, b, A & \rightarrow & p & (4) \\ p, b, A & \rightarrow & p & (5) \end{array}$$

Das Wort $ab \in L(M)$ wird wie folgt akzeptiert: $(q, ab, \#) \overset{(2)}{\vdash} (q, b, A) \overset{(4)}{\vdash} (q, \varepsilon, \varepsilon)$

Satz 5.1:
L vom Typ 2 $\Leftrightarrow \exists$ PDA $M : L(M) = L$

Beweis (von Satz 5.1):
Richtung „$\Rightarrow$“:

Gegeben Grammatik $G = (V, \Sigma, P, S)$. Daraus wird PDA M mit $M = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$ gebildet (S ist das Kelleranfangszeichen):

$$\begin{array}{llll} (1) & q\varepsilon A & \rightarrow & q\alpha \quad (\forall A \rightarrow \alpha \in P) \\ (2) & qaa & \rightarrow & q \quad (\forall a \in \Sigma) \end{array}$$

Es gilt: $x = x_1 \dots x_n \in L(G) \Leftrightarrow \exists$ Linksableitung:

$$\begin{array}{ll} & S \Rightarrow a_0 \Rightarrow^* x_1 a_1 \Rightarrow^* x_1 x_2 a_2 \Rightarrow^* x_1 \dots x_{n-1} a_{n-1} \Rightarrow^* x \\ \Leftrightarrow & (q, x, S) \vdash_{(1)} (q, x, a_0) \vdash^*_{(1)} (q, x, x_1 a_1) \vdash_{(2)} (q, x_2 \dots x_n, a_1) \vdash^*_{(1)} (q, x_2 \dots x_n, x_2 a_2) \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ & \vdash^*_{(1)} (q, x_{n-1} x_n, x_{n-1} a_{n-1}) \vdash_{(2)} (q, x_n, a_{n-1}) \vdash^*_{(1)} (q, x_n, x_n) \vdash_{(2)} (q, \varepsilon, \varepsilon) \\ \Leftrightarrow & x \in L(M) \end{array}$$

Richtung „$\Leftarrow$“:

Gegeben $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ Daraus wird Grammatik G mit $G = (V, \Sigma, P, S)$ mit $V = \{S\} \cup \{X_{qAp} \mid q, p \in Z, A \in \Gamma\}$ gebildet.

Idee: $X_{qAp} \Rightarrow_G^m x \Leftrightarrow (q, x, A) \vdash_M^m (p, \varepsilon, \varepsilon)$ und $P' = \{S \to X_{q_0 \# p} \mid p \in Z\} \subseteq P$

Denn dann gilt:

$$\begin{array}{rcl} x \in L(M) & \Leftrightarrow & \exists m, p : (q_0, x, \#) \vdash^m (p, \varepsilon, \varepsilon) \\ & \Leftrightarrow & \exists m, p : S \Rightarrow X_{q_0 \# p} \Rightarrow^m x \\ & \Leftrightarrow & x \in L(G) \end{array}$$

Idee: $X_{qAp} \Rightarrow^{m'} \Delta x$

$$P = P' \cup \{X_{qAp} \to u X_{p_1 B_1 p_2} X_{p_2 B_2 p_3} \dots X_{p_k B_k p} \mid \begin{array}{l} u \in \Sigma \cup \{\varepsilon\}, \\ P_2 \dots P_k \in Z, \\ quA \to_M p_1 B_1 \dots B_k\} \end{array}$$

Zu zeigen: $X_{qAp} \Rightarrow^m x \Leftrightarrow (q, x, A) \vdash^m (p, \varepsilon, \varepsilon)$

Induktion nach m:

Induktionsanfang:
$m = 0$. Die Ableitung ist mit 0 Schritten nicht möglich.

Induktionsvoraussetzung:
$m \geq 1, \forall k < m : X_{qAp} \Rightarrow^k x \Leftrightarrow (q, x, A) \vdash^k (p, \varepsilon, \varepsilon)$

Induktionsschritt:
$X_{qAp} \Rightarrow \alpha_1 \Rightarrow \alpha_2 \dots \Rightarrow \alpha_m = x$ Linksableitung nach Definition von P:
$\alpha_1 = u X_{p_1 B_1 p_2} \dots X_{p_k B_k p}$, wo $\begin{cases} u \in \Sigma \cup \{\varepsilon\}, p_2 \dots p_k \in Z, \\ quA \to p_1 B_1 \dots B_k \in \delta \end{cases}$

Aus Linksableitung folgt:

$$x = u v_1 \dots v_k, \text{ wo } X_{p_i B_i p_{i+1}} \Rightarrow^{m_i} v_i \text{ und } \sum_{i=1}^{k} m_i = m - 1$$

Nach Induktionsvoraussetzung gilt: $(p_i, v_i, B_i) \vdash^{m_i} (p_{i+1}, \varepsilon, \varepsilon)$, da $m_i < m \; \forall i$ also

$$\begin{array}{lll} (q, x = u v_1 \dots v_k, A) & \vdash & (p_1, v_1 \dots v_k, B_1 \dots B_k) \\ & \vdash^{m_1} & (p_2, v_2 \dots v_k, B_2 \dots B_k) \\ & \vdash^{m_2} & \dots \\ & \vdots & \\ & \vdash^{m_{k-1}} & (p_k, v_k, B_k) \\ & \vdash^{m_k} & (p, \varepsilon, \varepsilon) \end{array}$$

□

5.2 Chomsky-Normalform

Satz 5.2 (Chomsky Normalform):
L vom Typ 2 $\Rightarrow \exists$ Grammatik G, die nur Produktionen der Form $A \to BC$ und $A \to a$ enthält mit $L(G) = L\backslash\{\varepsilon\}$

Beweis (von Satz 5.2):
Um den Satz zu beweisen, wird die Grammatik nach und nach manipuliert, bis sie die gewünschte Form (CNF, Chomsky-Normalform) hat:

1. Sei G''' Typ 2-Grammatik mit $L(G''') = L$. Dann existiert eine Typ 2-Grammatik G'' mit $L(G'') = L(G''')\backslash\{\varepsilon\}$, die keine Produktionen der Form $A \to \varepsilon$ enthält.

   Beweis: siehe Übungsaufgabe 36a

2. Es existiert eine Typ 2-Grammatik G' mit $L(G') = L(G'')$ ohne Produktionen der Form $A \to \varepsilon$ (ε-Produktionen) oder $A \to B$ (Variablenumbenennungen).

   Beweis:

   a) Annahme: Es gibt Regel $A \to B$, jedoch nicht $B \to D$
   Entferne $A \to B$ und nehme $A \to \alpha$ für alle $B \to \alpha$ hinzu.
   Dadurch ändert sich die Sprache nicht:
   - Enthält eine (alte) Ableitung $A \to B$, dann auch $B \to \alpha$ (und zwar später). Falls beide nicht direkt aufeinander folgen, verschiebe $A \to B$ nach hinten. Dann ersetze beide durch $A \to \alpha$.
   - Enthält eine (neue) Ableitung $A \to \alpha$, dann ersetze durch $A \to B$, $B \to \alpha$.

   b) Annahme: G'' enthält Zyklus $A_1 \to A_2$ $A_2 \to A_3$, …, $A_{k-1} \to A_k$, $A_k \to A_1$.
   Entferne diese Produktionen und ersetze alle Vorkommen von $A_2, \ldots, A_k$ durch A_1. Falls $S \in \{A_2, \ldots, A_k\}$, so ist A_1 das neue Startsymbol.
   Dadurch ändert sich die Sprache nicht:
   - Kommt $A_2, \ldots, A_k$ in einer alten Ableitung vor, so kann dies durch A_1 ersetzt werden.
   - Kommt A_1 in einer neuen Ableitung vor, dann sind die beiden beteiligten Produktionen aus Produktionen entstanden, in denen A_i und A_j vorkamen. Ersetze die neue durch die alte Produktion und gegebenfalls Umbenennung des Zyklus.

   Kann weder a) noch b) angewandt werden, enthält die Grammatik keine Umbenennung mehr.

3. Es existiert eine CNF-Grammatik $G : L(G) = L(G')$

   Beweis: siehe Übungsaufgabe 36b

□

5.3 Pumping Lemma

Satz 5.3 (Pumping-Lemma für kontextfreie Sprachen):
$\forall L \in \mathcal{CFL} \exists l \in \mathbb{N} \forall z \in L : |z| \geq l \Rightarrow u, v, w, x, y : z = uvwxy$ und

(i) $vx \neq \varepsilon$

(ii) $|vwx| \leq l$

(iii) $uv^i wx^i y \in L \forall i \in \mathbb{N}$

Beispiel 24:
Sei $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Behauptung:
$L \notin \mathcal{CFL}$

Beweis:
Annahme: $L \in \mathcal{CFL}$. Nach Satz 5.3 $\Rightarrow \exists$ Pumpingzahl l. Außerdem gilt nach Satz 5.3: $a^l b^l c^l = uvwxy$ mit (i), (ii) und (iii).

$(i) \Rightarrow vx$ enthält ein Zeichen $(*)$
$(ii) \Rightarrow vwx$ und damit vx enthält ein anderes Zeichen nicht $(**)$

$\Rightarrow uv^2wx^2y$ enthält mehr Zeichen nach $(*)$ als Zeichen nach $(**)$. Widerspruch zu (iii)!

Also gilt: $L \notin \mathcal{CFL}$. □

Definition (Syntaxbaum, Wurzel, Nachfolger):
Sei $G = (V, \Sigma, P, S)$ Grammatik für Sprache $L\backslash\{\varepsilon\}$ in CNF. Sei T ein Baum mit $T = (V, E)$, $S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_m = z$, wo $z \in L$ mit:

*(i) Ein innerer Knoten w heißt **Wurzel** und an ihm steht S. Alle Kanten sind von w weggerichtet:*

Für alle Kanten $e = \{u, v\}$ gilt: entweder liegt u auf dem $w - v$-Pfad in T ($h(e) = v, t(e) = u$), oder v liegt auf dem $w - u$-Pfad ($h(e) = u, t(e) = v$). Dabei ist $h(e)$ die Pfeilspitze und $t(e)$ das Pfeilende der Kante e.

(ii) An den Blättern stehen die Terminale, an den inneren Knoten (Nicht-Blätter) stehen Variablen.

*(iii) Die **Nachfolger** $\Gamma^+(u) = \{v \mid \{u, v\} \in E, h(\{u, v\}) = v\}$ eines Knotens u sind von links nach rechts sortiert.*

(iv) Steht an u die Variable A und an $\Gamma^+(u)$ von links nach rechts die Zeichen $\beta_1 \ldots \beta_r$, dann ist $A \to \beta_1 \ldots \beta_r \in P$.

Definition (T_n):
$T = T_n$, wobei T_n wie folgt induktiv definiert ist:

- *T_0 ist ein Baum mit einem Knoten (der Wurzel), an dem S steht*
- *T_i ($i \geq 1$) ergibt sich aus T_{i-1} wie folgt:*

  Wird im Schritt $\alpha_{i-1} \Rightarrow \alpha_i$ ein Vorkommen der Variable A mittels der Regel $A \to \beta_1 \dots \beta_r$ abgeleitet, dann werden an das Blatt von T_{i-1}, an dem dieses Vorkommen steht, Nachfolger mit $\beta_1, \dots, \beta_r$ angehängt.

Beweis (von Satz 5.3):
Sei $G = (V, \Sigma, P, S)$ Grammatik für $L \backslash \{\varepsilon\}$ in CNF. Sei T Syntaxbaum zu $S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_m \Rightarrow z$ wo $z = z_1 \dots z_n$, $n \geq l = 2^k$, wobei k die Anzahl der Variablen von G ist. Der Syntaxbaum ist ein Binärbaum, da G in CNF gegeben ist.

O.B.d.A. kommen in der Ableitung von z aus S zunächst Ableitungen der Form $A \to BC$ und dann solche der Form $A \to a$ vor.

$\Rightarrow T_n$ hat n Blätter: in jedem Schritt werden an ein Blatt zwei Nachfolger gehängt.

Proposition:
Hat ein Binärbaum mindestens $2^{k-1} + 1$ Blätter, dann auch ein Pfad von der Wurzel zu einem Blatt der Länge k.

Beweis durch Induktion nach k:

Induktionsanfang ($k = 0$):
Jeder Binärbaum hat einen Wurzel-Blatt-Pfad (der Länge ≥ 0).

Induktionsschritt ($k \to k + 1$):
Jeder der beiden Nachfolger der Wurzel ist Wurzel eines Teilbaumes. Einer davon hat mindestens $2^{k-1} + 1$ Blätter und damit nach Induktionsvoraussetzung einen Pfad der Länge k. Damit hat der gesamte Baum einen Pfad der Länge $k + 1$. □

Beweis (von Satz 5.3 – Fortsetzung):
Korollar: T_n hat einen Pfad von der Wurzel bis zu einem Blatt der Länge $\geq k$

Auf diesem Pfad liegen $k + 1$ Knoten, an denen je eine Variable steht. Eine Variable kommt also zweimal vor (unter den letzten $k + 1$ Knoten).

Sei U der Teilbaum von T_m von Nachfolgern des ersten Vorkommens. Sei U' der Teilbaum von T_m von Nachfolgern des zweiten Vorkommens. U' ist in U enthalten.

Sei u das Teilwort von z links von U
v das Teilwort von z in U links von U'
w das Teilwort von z in U'
x das Teilwort von z in U rechts von U'
y das Teilwort von z rechts von U

$\Rightarrow vx \neq \varepsilon$ (Wurzel von U und U' unterschiedlich)

$|vwx| \leq 2^k = l$

Induktionsanfang ($i \leq 1$):
$i = 0$: Ersetze U durch U': Syntaxbaum für uv^0wx^0y

$i = 1$: siehe oben

Induktionsschritt ($i \to i + 1$):
Ersetze U' durch Kopie von U: Syntaxbaum für $uv^{i+1}wx^{i+1}y$

□

Lemma:
Die Umkehrung des Pumping-Lemmas für kontextfreie Sprachen gilt nicht.

Beweis:
Durch Gegenbeispiel in Übungsaufgabe 39.

5.4 (Un)Entscheidbare Probleme für kontextfreie Grammatiken

Satz 5.4:
Gegeben G_1, G_2 kontextfreie Grammatiken. Dann sind folgende Fragen nicht entscheidbar:

(i) $L(G_1) \cap L(G_2) = \emptyset$?

(ii) $L(G_1) = \Sigma^*$?

(iii) $L(G_1) = L(G_2)$?

(iv) G_1 mehrdeutig ?

Beweis (von Satz 5.4):
Beweisidee: Reduzieren von PKP auf (i) - (iv).

Beweis von (i):

zu zeigen: PKP $\leq$ SKF (Schnittproblem für kontextfreie Grammatiken) mit

$$\text{SKF} = \{(G_1, G_2), G_1, G_2 \text{ kontextfrei}, L(G_1) \cap L(G_2) = \emptyset\}$$

Sei $f((x_1, x_2) \dots (x_k, y_k)) = (G_1, G_2)$. G_1 erzeugt x und G_2 erzeugt y mit folgenden Regeln:

$$G_1 : S \to 0^i 1 S x_i, 0^i 11 x_i \quad G_2 : S \to 0^i 1 S y_i, 0^i 11 y_i$$

Dabei ist S die einzige Variable.

Die Funktion f muss total (d.h. auch für Mülleingaben definiert) sein:
$f(x) = (S \to \varepsilon, S \to \varepsilon)$ wenn x ungültige Eingabe ist, z.B. $x = (x_1)$.

- $L(G_1) = \{0^{i_1} 10^{i_2} 1 \dots \underbrace{0^{i_n} 11 x_{i_n}}_{\text{letzte Regel}} x_{i_{n-1}} \dots x_{i_2} x_{i_1}, n \geq 1, 1 \leq i_j \leq k\}$
- $L(G_2) = \{0^{i_1} 10^{i_2} 1 \dots 0^{i_n} 11 y_{i_n} y_{i_{n-1}} \dots y_{i_2} y_{i_1}, n \geq 1, 1 \leq i_j \leq k\}$

$L(G_1) \cap L(G_2) = \{0^{i_1} 1 \dots 0^{i_n} 11 z \mid z = x_{i_n} \dots x_{i_1} = y_{i_n} \dots y_{i_1}\} \neq \emptyset$
$\Leftrightarrow \alpha = \{i_1 \dots i_n\}$ ist Lösung für PKP
$\Leftrightarrow ((x_1, y_1) \dots (x_k, y_k)) \in$ PKP □

Beweis von (iv):

zu zeigen: PKP $\leq$ {Grammatik ist kontextfrei und mehrdeutig}

$f((x_1, y_1) \dots (x_k, y_k)) = G$, wobei G hat Regeln

$$S \to S_1, S_2 \quad S_1 \to 0^j 1 S_1 x_j, 0^j 1 1 x_j \quad S_2 \to 0^j 1 S_2 y_j, 0^j 1 1 y_j$$

Dabei erzeugt S_1 und S_2 jeweils eine Sprache. Mittels S wird ausgewählt, welche Sprache erzeugt werden soll. Sowohl von S_1, als auch von S_2 ausgehend ist die Ableitung eindeutig.

Beweis von (i) $\Rightarrow$ G mehrdeutig $\Leftrightarrow$ $((x_1, y_1) \dots (x_k, y_k)) \in$ PKP. □

Beweisidee für (ii):

Reduktion von $\overline{\text{PKP}}$:

$(x_1, y_1) \dots (x_k, y_k) \notin$ PKP $\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset \Leftrightarrow L(\overline{G_1}) \cup L(\overline{G_2}) = \Sigma^*$

$f((x_1, y_1) \dots (x_k, y_k)) = G$, wobei $L(G) = L(\overline{G_1}) \cup L(\overline{G_2})$

Problem: Es ist nicht klar, ob G kontextfrei ist, weil kontextfreie Grammatiken im Allgemeinen nicht unter Schnitt- und Komplementbildung, jedoch unter Vereinigung abgeschlossen sind.

Beispiel: Die Sprache $L = \{a^i b^i c^i \mid i \geq 0\}$ ist nicht kontextfrei, aber $A \cap B$ mit $A = \{a^i b^i c^j \mid i \geq 0, j \geq 0\}$ und $B = \{a^i b^j c^j \mid i \geq 0, j \geq 0\}$

Satz 5.5:
Das Leerheitsproblem für kontextfreie Sprachen ist entscheidbar, d.h. gegeben eine kontextfreie Grammatik G: ist $L(G) = \emptyset$?

Beweis (von Satz 5.5):
zwei Beweisansätze:

- Algorithmisch

  `EINGABE:` $G = (V, \Sigma, P, S)$ kontextfrei
  $U_0 \leftarrow \emptyset$;
  $i \leftarrow 0$;
  `REPEAT`
  $U_i \leftarrow \{A \mid A \in V, A \to \alpha, \alpha \in (\Sigma \cup U_{i-1})^*\} \cup U_{i-1}$;
  $i \leftarrow i + 1$;
  `UNTIL` $U_i = U_{i-1}$;
  `IF` $S \in U_i$ `AUSGABE` $L(G) \neq \emptyset$
  `ELSE AUSGABE` $L(G) = \emptyset$;

- Induktion

  Induktionsanfang:
  $\forall A \in U_0 \; \exists w \in \Sigma^* \; A \to w$

Induktionsvoraussetzung:
$\forall A \in U_j \; \exists w \in \Sigma^* \; A \rightarrow^* w$

Induktionsschritt:
Sei $A \in U_{j+1}$ Algorithmus $\Rightarrow \exists \alpha$ mit $A \rightarrow \alpha$ und $A' \in U_j \; \forall A' \in \alpha$

Induktionsvoraussetzung auf $A' \Rightarrow A \rightarrow \alpha \Rightarrow^* w'$

□

Satz 5.6:
Das Leerheitsproblem für kontextsensitive Sprachen ist nicht entscheidbar.

Beweis (von Satz 5.6):
zu zeigen: SKF $\leq$ Leerheitsproblem $\mathcal{CFL}$

gesucht: $f(G_1, G_2) = G$

$L(G_1) \cap L(G_2) = \emptyset \Leftrightarrow L(G) = \emptyset$

Definieren f durch $L(G) = L(G_1) \cap L(G_2)$

zu zeigen: G kontextsensitiv und f berechenbar

gesucht: LBA M mit $L(M) = L(G)$

Seien M_1, M_2 LBA's mit $L(M_1) = L(G_1)$ und $L(M_2) = L(G_2)$.

Beschreibung von M: M simuliert erst M_1 auf Eingabe x ohne x zu verändern. Dazu werden vor der Simulation von M_1 die Bandzeichen y durch (y, y) ersetzt. Mit den 1. Koordinaten wird gerechnet und falls M_1 akzeptiert wird die 1. Koordinate gelöscht. Wenn M_1 akzeptiert wird M_2 auf Eingabe x simuliert. □

Zusammenfassung

	Wortproblem $x \in L$?	Leerheit $L = \emptyset$?	Äquivalenz $L_1 = L_2$?	Inklusion $L_1 \subseteq L_2$?	Schnittproblem $L_1 \cap L_2 \neq \emptyset$?
$\mathcal{REG}$	ja	ja	ja	ja	ja
$\mathcal{DCFL}$	ja	ja	ja	nein	nein
$\mathcal{CFL}$	ja	ja	nein	nein	nein
$\mathcal{CSL}$	ja	nein	nein	nein	nein
$\mathcal{RE}$	nein	nein	nein	nein	nein

Tabelle 5.1: Entscheidbarkeit

	Schnitt	Vereinigung	Komplement	Produkt	Stern
$\mathcal{REG}$	ja	ja	ja	ja	ja
$\mathcal{DCFL}$	nein	nein	ja	nein	nein
$\mathcal{CFL}$	nein	ja	nein	ja	ja
$\mathcal{CSL}$	ja	ja	ja	ja	ja
$\mathcal{RE}$	ja	ja	nein	ja	ja

Tabelle 5.2: Abschlusseigenschaften

Die Klasse der deterministisch-kontextfreien Sprachen $\mathcal{DCFL}$ wurde in der Vorlesung nicht behandelt und ist nur zur Vollständigkeit aufgeführt.

Teil II

Elementare Strukturen und Algorithmen

6 Graphen

➜ Literatur: [Hougardy]

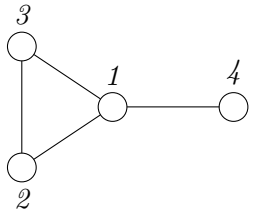
Zur Graphentheorie beachte man auch Kapitel A auf Seite 88.

6.1 Grundlagen

Definition (Graph, Knoten, Kanten):
$G = (V, E)$ *ist ein* ***ungerichteter Graph****, wobei*

- V *endliche Menge von* ***Knoten*** *($V \neq \emptyset$)*
- $E \subseteq \binom{V}{2}$ ***Kanten***

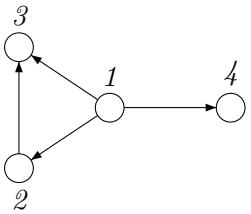
Graphische Darstellung: $G = (\{1,2,3,4\}, \{\{1,2\},\{1,3\},\{2,3\},\{1,4\}\})$



$D = (V, E)$ *ist ein* ***gerichteter Graph/Digraph****, wobei*

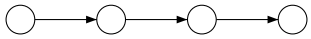
- V *endliche Menge von* ***Knoten*** *($V \neq \emptyset$)*
- $A \subseteq V \times V$ ***Kanten***

Graphische Darstellung: $G = (\{1,2,3,4\}, \{(1,2),(1,3),(2,3),(1,4)\})$

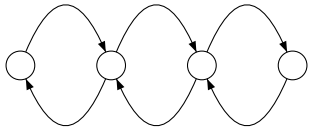


Bemerkung:
Graphen können modellieren:

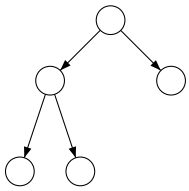
- Datenstrukturen
  - Listen



  - doppelte Listen



  - Heaps



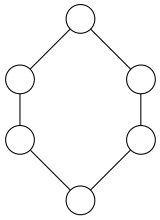
- Rechnungen/Rechnermodelle
  - endliche Automaten
  - Syntaxbäume
  - Konfigurationsgraph für LBAs
- Realweltobjekte
  - Autobahnen
  - Rechner/Telefonnetzwerke
  - soziale Netzwerke

Definition (spezielle Graphen):
Sei $[n] := \{1, \ldots, n\}$ $(n \in \mathbb{N})$

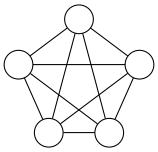
1. ***vollständige Graphen/Cliquen:*** $K_n = ([n], \binom{[n]}{2})$
2. ***leerer Graph:*** $E_n = ([n], \emptyset)$
3. ***Pfad:*** $P_n = ([n+1], \{\{i, i+1\} \mid i \in [n]\})$
4. ***Kreise:*** $C_n = ([n], \{\{i, i+1\} \mid i \in [n-1]\} \cup \{n, 1\})$, $n \geq 3$
5. ***Hyperwürfel:*** $Q_n = (\{0,1\}^n, \{\{u, v\} \mid \mathit{Hammingabstand} = 1\})$

Beispiel 25 (spezielle Graphen):

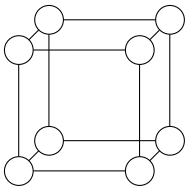
Kreis C_6



Clique K_5

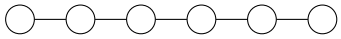


Hyperwürfel Q_3



leerer Graph E_2

Pfad P_5



Bemerkung:
Übergang gerichteter/ungerichteter Graph

- Kanten eines ungerichteten Graphen richten:

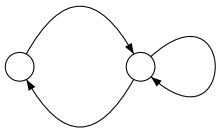
  $G = (V, E)$, $h : E \to V$, $t : E \to V$ mit $\{h(e), t(e)\} = e \; \forall e \in E$

  $D = (V, A)$ mit $A = \{\{t(e), h(e)\} \mid e \in E\}$: Pfeilspitzen an $h(e)$ malen.

- Graph, der gerichtetem Graphen unterliegt:

  $D = (V, A) \to G = (V, E)$ mit $E = \{\{u, v\} \mid (u, v) \in A\}$

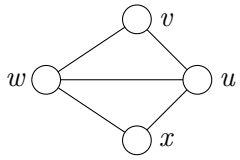
  Problem: Schleifen und Mehrfachknoten:



Definition (Nachbarschaft, Grad):
Sei $G = (V, E)$ ein ungerichteter Graph.

- ***Nachbarschaft:*** $\Gamma : V \to \mathfrak{P}(V),\ v \mapsto \{u \mid \{u, v\} \in E\}$
- ***Grad:*** $d : V \to \mathbb{N},\ v \mapsto |\Gamma(v)|$

Beispiel: $\Gamma(x) = \{u, w\},\ d(x) = 2$



Sei $D = (V, A)$ ein gerichteter Graph:

- ***Nachbarschaft:***
  - $\Gamma^+ : V \to \mathfrak{P}(V),\ v \mapsto \{w \mid (v, w) \in A\}$
  - $\Gamma^- : V \to \mathfrak{P}(V),\ v \mapsto \{u \mid (u, v) \in A\}$
  - $\Gamma : V \to \mathfrak{P}(V),\ v \mapsto \Gamma^+(v) \cup \Gamma^-(v)$
- ***Grad:***
  - $d^+ : V \to \mathbb{N},\ v \mapsto |\Gamma^+(v)|$
  - $d^- : V \to \mathbb{N},\ v \mapsto |\Gamma^-(v)|$
  - $d : V \to \mathbb{N},\ v \mapsto |\Gamma(v)|$

Proposition (Anzahl von Knoten):
Sei G ein ungerichteter und D ein gerichteter Graph.

(i) $G = (V, E) \Rightarrow \sum_{v \in V} d(v) = 2|E|$

(ii) $D = (V, A) \Rightarrow \sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = |A| \Rightarrow \sum_{v \in V} d(v) = 2|A|$

Beweis:
Informell:

1. Jede Kante hat genau zwei Endknoten, wird also zweimal in einem $d(v)$ gezählt, also zweimal in $\sum_{v \in V} d(v)$.

2. Übung

Definition (Teilgraph, induzierter Teilgraph):
Seien $G = (V, E)$ und $H(V', E')$ Graphen.

- *H ist **Teilgraph** von G, falls $V' \subseteq V$ und $E' \subseteq E \cap \binom{V'}{2}$*
- *$H = G[V']$ ist **induzierter Teilgraph** von G, falls $V' \subseteq V$ und $E' = E \cap \binom{V'}{2}$*

Definition (Weg, Pfad, Kreis):
Sei $G = (V, E)$ Graph mit $u, v \in V$.

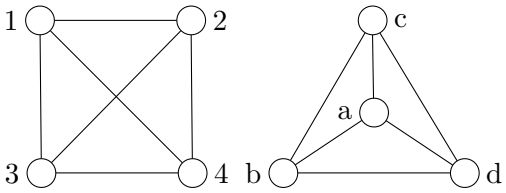
- *u-v-**Weg** in G: $(u, w_1, \ldots, w_k, v) : w_i \in V, \{u, w_1\}, \{w_k, v\}, \{w_i, w_{i+1}\} \in E$*
- *u-v-**Pfad** in G: u-v-Weg mit $|\{u, w_1, \ldots, w_k, v\}| = k + |\{u, v\}|$*
  (keine Zwischenstation)
- ***Kreis** in G: $(w_1, \ldots, w_k)$: w_1-w_k-Pfad mit $\{w_1, w_k\} \in E$*

Definition (Isomorphie):
*H und H' sind **isomorph** / $H \hat{=} H'$, genau dann wenn*

$$\exists \varphi : V' \mapsto V'' : \{u, v\} \in E' \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E''$$

Wenn $G = (V, E)$ ein Graph und H ein Teilgraph von G ist, sowie $H' \hat{=} H$, dann gilt: G enthält H'.

Beispiel 26 (isomorphe Graphen):



Die beiden Graphen sind isomorph mit der Abbildung
$\varphi(1) = a, \varphi(2) = c, \varphi(3) = b, \varphi(4) = d$.

Proposition:
Sei $G = (V, E)$ ein Graph und $\delta = \delta(G) := \min_{v \in V} d(V)$

1. G enthält P_δ

2. $\delta \geq 2 \Rightarrow G$ enthält $C_{k \geq \delta+1}$

Dabei wird $\delta \geq 2$ vorausgesetzt, da sonst isolierte Kanten existieren dürften.

Beweis: 1. Sei $P = (v_1, \ldots, v_l)$ längster Pfad in G
$\Rightarrow \Gamma(v_2) \subseteq \{v_1, \ldots, v_{l-1}\}$
$\Rightarrow l - 1 \geq d(v_l) \geq \delta$
$\Rightarrow G$ enthält P_δ

2. Sei i_0 erster Knoten in $\Gamma(v_l)$, d.h. $i_0 := \{i \mid v_i \in \Gamma(v_l)\}$
$\Rightarrow \Gamma(v_l) \subseteq \{v_{i_0}, \ldots, v_{l-1}\}$
$\Rightarrow l - i_0 \geq d(v_l) \geq \delta$
$\Rightarrow (v_{i_0}, v_{i_{0+1}}, \ldots, v_l, v_{i_0})$ ist Kreis G der Länge $l - i_0 + 1 \geq \delta + 1$
$\Rightarrow G$ enthält $C_{k \geq \delta+1}$

□

6.2 Bäume

Definition (zusammenhängender Graph):
*Ein Graph $G = (V, E)$ heißt **zusammenhängend/zsh** genau dann wenn*

$$\forall u, v \in V : \exists u - v\text{-Pfad in } G$$

Definition (Baum):
Ein Graph $T = (V, E)$ heißt Baum genau dann wenn

$$\forall u, v \in V : \exists! u - v\text{-Pfad in } T$$

Proposition (Eigenschaften von Bäumen):
T Baum $\Leftrightarrow$ T zsh und kreisfrei (d.h. T enthält keinen Kreis)

Beweis:
Richtung „⇒“:

siehe Übungsaufgabe 24

Richtung „⇐“:

$u, v \in V$, T zsh $\Rightarrow \exists$ $u - v$-Pfad in T

Annahme: $\exists$ zwei unterschiedliche $u - v$-Pfade: $(u, w_1, \ldots, w_k, v) = P^{(1)}$ und $(u, w'_1, \ldots, w'_l, v) = P^{(2)}$

Seien w_i, w_i' die ersten unterschiedlichen Knoten $i = \min\{j \mid w_j \neq w_j'\}$. Sei w_r der erste Knoten danach, der auf $P^{(2)}$ liegt: $r = \min\{s \geq 1 \mid w_s' \text{ auf } P^{(2)}\}$

$\Rightarrow T$ enthält Kreis $(w_{i-1}, w_i, \ldots, w_r = w_q', w_{q-1}, w_{q-2}, \ldots, w_{i-1}')$. Dies steht im Widerspruch zur Annahme, T sei kreisfrei.

□

Satz 6.1 (spannender Baum):
$G = (V, E)$ zusammenhängend $\Leftrightarrow$ G enthält Baum T auf allen Knoten (spannender Baum).

Beweis (von Satz 6.1):
Richtung „$\Leftarrow$“:

$\forall u, v \in V : \exists! u - v$-Pfad in $T \Rightarrow \exists u - v$-Pfade in G.

Richtung „$\Rightarrow$“:

Folgendes Verfahren SUCHE findet T in G.

Dabei sei o.B.d.A. der Graph $G = ([n], E)$ als Adjazenzliste gegeben. S kann ein Keller oder eine Schlange sein (vgl. Tiefen-/Breitensuche), M ist ein Feld mit Markierungen und E' eine Liste:

```
ALGORITHMUS SUCHE (G);
BEGIN
 S ← {1};
 M ← {1};
 E' ← ∅;
 WHILE S ≠ ∅
   WÄHLE i ∈ S;
   S ← S\{i};
   S ← S ∪ (Γ(i)\M);
   E' ← E' ∪ {{i, j} | j ∈ Γ(i)\M};
   M ← M ∪ Γ(i);
 ENDWHILE;
 AUSGABE T = (M, E');
END;
```

□

Behauptung:
T ist spannender Baum.

1. $\exists 1-k$-Pfad in $G \Leftrightarrow k \in M$ am Schluss $\Leftrightarrow \exists 1-k$-Pfad in T.

2. T ist kreisfrei

Beweis: 1. Übung

2. Annahme: K enthält Kreis $(i_1, \ldots, i_k, i_1)$. O.B.d.A.: $i = i_1, i = i_{k-1}, i = i_k$ in dieser Reihenfolge.

   $\Rightarrow$ nach $i = i_1$ ist i_k markiert ($i_k \in M$)
   $\Rightarrow$ Kante $\{i_{k-1}, i_k\}$ kann nur in $i = i_{k-1}$ zu E' kommen.

   Dies ist ein Widerspruch, da i_k schon markiert ist.

□

6.3 Laufzeit und Datenstrukturen

➜ Literatur: [Hougardy]

Definition (Algorithmen, Laufzeit):
Algorithmen *bestehen aus:*

- *Eingabe/Ausgabe (durch Variablen)*
- *Variablenzuweisung (auch indirekt)*
- *arithmetische Operationen (in einem Schritt)*
- *Schleifen*

mit der gewohnten Semantik.

Die ***Laufzeit*** *ist die Anzahl der Schritte als Funktion der Eingabe bzw. Eingabegröße.*

Definition (indirekte Variablen, Listen, Schlangen, Keller):
***indirekte Variablen**: Variablen $R_1, R_2, \ldots$ mit Werten $r_1, r_2, \ldots$. Dabei ist beispielsweise die Zuweisung $R_{r_{17}} \leftarrow 5$ möglich. Dadurch können Datenstrukturen realisiert werden, die auf Zeigern beruhen:*

- ***Listen***
  die Operationen Elemente anhängen, Randelemente löschen und Listen verketten sind in $\mathcal{O}(1)$ möglich
- ***Schlangen/queues***
  die Operationen queue/dequeue sind in $\mathcal{O}(1)$ möglich
- ***Keller/stacks***
  die Operationen push/pop sind in $\mathcal{O}(1)$ möglich

Definition (Adjazenzmatrix):
*Sei $G = ([n], E)$ ein Graph. Er kann in der **Adjazenzmatrix***

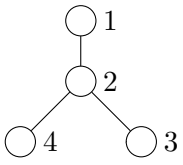
$$A = (a_{ij})_{i,j=1}^{n} \in \{0,1\}^{n \times n} \text{ mit } a_{ij} = \begin{cases} 1 & \{i,j\} \in E \\ 0 & \text{sonst} \end{cases}$$

gespeichert werden.

- *Speicherbedarf: $\mathcal{O}(n^2)$*
- *Knoten einfügen/entfernen: $\mathcal{O}(n)$ (genau $2n$)*
- *Kante einfügen/entfernen: $\mathcal{O}(1)$ (genau 2)*
- *Test: $\{i,j\} \in E$: $\mathcal{O}(1)$*

Beispiel 27 (Adjazenzmatrix):

Die Matrix $A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$ beschreibt den Graphen



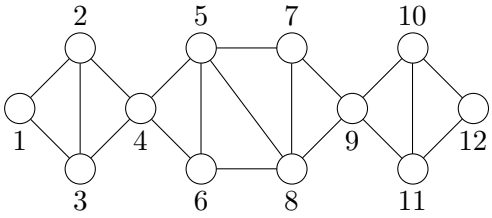
Definition (Adjazenzliste):
*Sei $G = ([n], E)$ ein Graph. Er kann in einer **Adjazenzliste** gespeichert werden. Sie ist eine Liste alle Knoten mit angehängten Nachbarlisten.*

- *Speicherbedarf: $\mathcal{O}(n+m)$ (genau $2m+n$)*
  - *$\mathcal{O}(n)$ für Knotenliste*
  - *$\mathcal{O}(\sum_{i=1}^{n} d(i))$ für Nachbarlisten (genau $2m$)*
- *Knoten einfügen: $\mathcal{O}(1)$*
- *Knoten entfernen: $\mathcal{O}(n)$*
- *Kante einfügen/entfernen (mit zusätzlicher Kantenliste): $\mathcal{O}(1)$*
- *Test: $\{i, j\} \in E$: $\mathcal{O}(n)$*

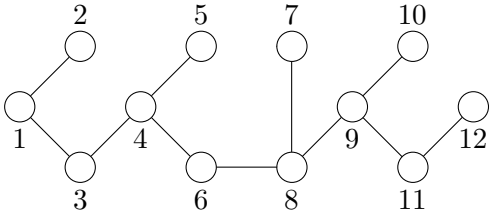
6.4 Tiefen- und Breitensuche

Definition (Tiefensuche, Breitensuche):
Betrachtung des Algorithmus SUCHE *aus Satz 6.1:*

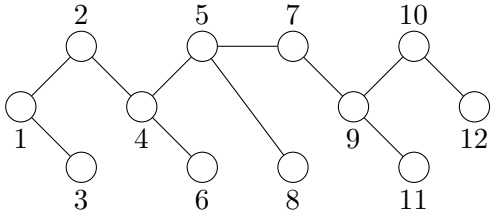
- ***Tiefensuche**: Algorithmus mit S als Keller*
- ***Breitensuche**: Algorithmus mit S als Schlange*



Beispiel 28 (Tiefensuche):



Beispiel 29 (Breitensuche):



Satz 6.2:
Tiefen- und Breitensuche haben die Laufzeit $\mathcal{O}(m)$.

Beweis (von Satz 6.2):
Betrachtung des Algorithmus SUCHE aus Satz 6.1:

- jeder Knoten wird nur einmal als i gewählt
- für Knoten i braucht Schleife $\mathcal{O}(d(i))$ Schritte

$\Rightarrow$ Laufzeit: $\sum_{i=1}^{n} \mathcal{O}(d(i)) = \mathcal{O}(m)$ □

Algorithmus TIEFENSUCHE**:**

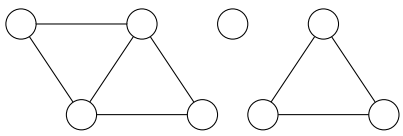
```
ALGORITHMUS Tiefensuche (G);
BEGIN
 S ← {1};
 M ← {1};
 E' ← ∅;
 WHILE S ≠ ∅
   i ← POP S;
   PUSH S, Γ(i)\M;
   E' ← E' ∪ {{i, j} | j ∈ Γ(i)\M};        (*)
   M ← M ∪ Γ(i);
 ENDWHILE;
 AUSGABE T = (M, E');
END;
```

Definition (Komponente):
Sei $G = (V, E)$ Graph mit $v \in V$.

$$W := \{w \in V \mid \exists v - w\text{-Pfad in } G\} \textbf{ bzw. } G[w]$$

*nennt man **Zusammenhangskomponente** (von v) oder **Komponente**.*

Beispiel 30 (Komponenten):



Dieser Graph hat 3 Komponenten.

Satz 6.3:
Die Komponenten eines Graphen können in $\mathcal{O}(n+m)$ bestimmt werden.

Beweis (von Satz 6.3):
Folgendes Verfahren leistet das Verlangte. Dabei sei $G = ([n], E)$ als Adjazenzliste gegeben. k ist der Komponentenzähler:

```
ALGORITHMUS KOMPONENTEN (G);
BEGIN
 k ← 1;
 GLOBAL M ← ∅;
 FOR v = 1...n DO
   IF v ∉ M:
     G_k ← TIEFENSUCHE'(G, v);
     k ← k + 1;
 AUSGABE: Liste G_1, ..., G_{k-1};
END;
```

Dabei ist TIEFENSUCHE' wie TIEFENSUCHE, jedoch ist die Zeile (*) nun wie folgt:

$$E' \leftarrow E' \cup \{\{i,j\} \mid j \in \Gamma(i)\}$$

- Laufzeit von TIEFENSUCHE' ist $\mathcal{O}(m(\text{Zusammenhangskomponente von } v))$
- Laufzeit durch Schleife: $\mathcal{O}(n)$
- Gesamtlaufzeit für TIEFENSUCHE' (alle $k-1$ Aufrufe):

$$\sum_{j=1}^{k-1} \mathcal{O}(m(G_j)) = \mathcal{O}(m(G))$$

□

Definition (Abstand zwischen Knoten):
*Der **Abstand zwischen** v **und** w / **dist(**v**,**w**)** ist die minimale Länge (Anzahl der Knoten) eines $v-w$-Pfades in G. Falls kein $v-w$-Pfad existiert, gilt: dist$(v,w)=\infty$.*

Satz 6.4:
Die Abstände $\text{dist}(v,w)$ für ein $v \in V$ und alle $w \in V$ können in $\mathcal{O}(m)$ bestimmt werden.

Beweis (von 6.4):
Folgendes Verfahren leistet das Verlangte. Dabei sei S eine Schlange und a ein Feld:

```
ALGORITHMUS Abstände (G);
BEGIN
 M ← v;
 S ← v;
 a[v] ← 0;
 a[w] ← ∞ ∀w ≠ v;
 WHILE S ≠ ∅
   w ← DEQUEUE(S);
   FOR u ∈ Γ(w)\M
     QUEUE(S, u);
     M ← M ∪ {u};
     a[u] ← a[w]+1;     (**)
 ENDWHILE;
 AUSGABE: Feld a;
END;
```

Laufzeit: $\mathcal{O}(m)$ wie bei Breitensuche.

Noch zu zeigen: $a[w] = \text{dist}(v, w)\ \forall w \in V$

- x nach y in S aufgenommen: $a[x] \geq a[y]$
- x, y gleichzeitig in der Schlange $\Rightarrow |a[x] - a[y]| \leq 1$

  Induktion nach Anzahl der WHILE-Schleifendurchläufe$= k$

  - Induktionsanfang $k = 0$: OK.
  - Induktionsschritt $k \to k + 1$: w wird aus S entfernt, $\Gamma(w)$ zugefügt
    - $\forall x \in S\backslash(\Gamma(w)\backslash M) : |a[x] - a[w]| \leq 1$ nach Induktionsvoraussetzung
      $\forall u \in \Gamma(w)\backslash M : |a[x] - a[w]| = 1$ nach (**)
    - $\forall u \in \Gamma(w)\backslash M : a[u] = a[w] + 1 \geq a[w]$
- $a[w] \geq \text{dist}(v, w)$, denn das Verfahren findet einen $v - w$-Pfad der Länge $a[w]$
- $a[w] \leq \text{dist}(v, w)$:

  Induktion nach $\text{dist}(v, w)$:

  - Induktionsanfang: $\text{dist}(v, w) \Rightarrow v = w$: OK
  - Induktionsschritt $\text{dist}(v, w) \geq 1$:
    Sei z Nachbar von w auf kürzestem $v - w$-Pfad
    $\Rightarrow \underbrace{\text{dist}(v, z)}_{a[z] \text{ nach IV}} = \underbrace{\text{dist}(v, w) - 1}_{\leq a[w]-1}$
    Deshalb wird z vor w in S aufgenommen.

Fall 1: Als z aus S entfernt wird, ist w bereits in S
$\Rightarrow a[w] \leq a[z] + 1 = \text{dist}(v, z) + 1 = \text{dist}(v, w)$

Fall 2: sonst wird w zu diesem Zeitpunkt in S aufgenommen und $a[w] \leftarrow a[z] + 1$ zugewiesen.

□

7 Greedy-Algorithmen

➜ Literatur: [Hougardy]

Prinzip: Finden einer Lösung, die stückweise zusammengesetzt wird. Dabei sind die Stücke lokal optimal.

- Beispiele für Greedy-Algorithmen:
  - Prim
  - Dijkstra
  - Algorithmus aus Übungsaufgabe 51
- Gegenbeispiel: Übungsaufgabe 52

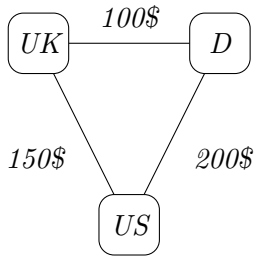
Frage: Wann ist dies auch global optimal? (greedy = gierig)

7.1 minimal spannende Bäume

Definition (gewichteter Graph):
(G, w) *ist ein* ***(kanten-)gewichteter Graph****:* $G = (V, E)$ *Graph und* $w : E \to \mathbb{R}_+$

Beispiele:

- *Rechenzentren und Kosten, je zwei davon zu verbinden*



- *Straßennetz und Entfernungen oder Fahrzeiten*

Definition (minimal spannender Baum/MST):
Sei $G = (V, E)$, $w : E \to \mathbb{R}_+$ ein kantengewichteter Graph.

*$T = (V, E')$ mit $E' \subseteq E$ ist ein **minimal spannender Baum/MST**, genau dann wenn*

- *T ein spannender Baum ist*
- $w(T) = w(E') = \sum_{e \in E} w(e) = \min\{w(T') \mid T' \text{ spannender Baum}\}$

Greedy-Algorithmus für MST (Prim):

Sei $G = ([n], E), w : E \to \mathbb{R}_+$.

```
ALGORITHMUS Prim (G, w);
BEGIN
 M ← {1};
 E' ← ∅;
 WHILE M ≠ [n]
    e = {u, v} ← argmin{w(e) | e ∈ E, u ∈ M, v ∈ V\M};          (*)
    E' ← E' ∪ {e};
    M ← M ∪ {v};
 ENDWHILE;
 AUSGABE T = (M, E');
END;
```

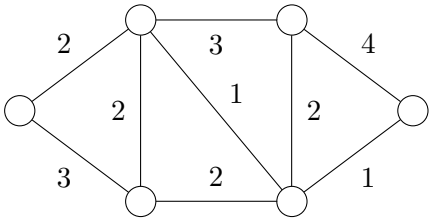
In Zeile (*) wird die Kante mit den geringsten Kosten gesucht, die aus der markierten Knotenmenge M herausführt.

Dabei sei argmin sei wie folgt definiert: Sei S eine Menge und $f : S \to \mathbb{R}_+$.

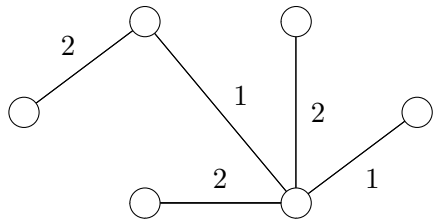
$$\text{argmin}\{f(x) \mid x \in S\} := \text{ein } x \in S : f(x) = \min\{f(x') \mid x' \in S\}$$

Beispiel 31 (minimal spannender Baum):
Der Algorithmus Prim arbeitet wie folgt:

Eingabe: gewichteter Graph



Ausgabe: minimal spannender Baum



Satz 7.1:
Die Ausgabe von PRIM ist ein MST.

Beweis (von Satz 7.1):
Sei T' ein MST, der möglichst viele Kanten mit T gemeinsam hat. Antithese $T' \neq T$.
$T' = \text{argmax}\{|E(T') \cap E(T)| \mid T' \text{ MST}\}$

- Sei $e = \{u, v\}$ die erste Kante, die zu T kommt und nicht in T' liegt.
- Sei P der $u - v$-Pfad in T' und $e' = \{u', v'\}$ die erste Kante auf P mit $u' \in M$, $v' \in V \backslash M$ (als e zu T kommt).
- Da PRIM e ausgewählt hat, muss gelten: $w(e) \leq w(e')$.
- Also ist $T'' = T' - e' + e$ ein MST mit einer mit T gemeinsamen Kante mehr als T'. Widerspruch!

□

7.2 kürzeste Wege

Definition (kürzester Weg):
Sei $G = (V, E)$ ein kantengerichteter Graph mit $w : E \to \mathbb{R}_+$ und $u, v \in V$.

*Ein $u - v$-Pfad P ist **kürzester ($u - v$-)Weg** genau dann wenn*

$$d(u, v) := w(P) = \sum_{e \in E(P)} w(e) = \min\{w(P') \mid P' u - v\text{-Pfad}\}$$

Definition (kürzester-Wege-Baum):
*T ist ein **kürzester-Wege-Baum (zu** u**)**, wenn*

- *T ist spannender Baum*
- *der $u-v$-Pfad in T ist ein kürzester $u-v$-Weg ($\forall v \in V$)*

Greedy-Algorithmus für kürzeste-Wege-Baum (DIJKSTRA):

Sei $G = (V, E), w : E \to \mathbb{R}_+, u \in V$. Das Feld $d[x]$ speichert die Entfernungen von u zu den jeweiligen Knoten x. Das Feld $p[x]$ enthält den Vorgänger von x auf dem kürzesten Pfad von u. M sind die markierten Knoten und E' enthält die Kanten des entwickelnden Baumes.

```
ALGORITHMUS DIJKSTRA (G, w, u);
BEGIN
 EINGABE G, w, u;
 M ← ∅;
 d[u] ← 0;
 d[v] ← ∞   (∀v ∈ V\{u});
 p[v] ←⊥    (∀v ∈ V);
 WHILE M ≠ V
   v ← argmin{d[v'] | v' ∈ V\M};
   M ← M ∪ {v};
   FORALL v' ∈ Γ(v)\M
     IF d[v] + w({v, v'}) < d[v']
       d[v'] ← d[v] + w({v, v'});
       p[v'] ← v;
     ENDIF;
   ENDFOR;
 ENDWHILE;
 E' ← {{p[v], v} | v ∈ V\{u}};
 AUSGABE T = (M, E');
END;
```

Satz 7.2:
Die Ausgabe von DIJKSTRA ist ein kürzester-Wege-Baum für u.

Beweis (von Satz 7.2):
Es genügt zu zeigen, dass vor und nach jedem WHILE-Schleifendurchlauf gilt:

1. $d[v]$ ist die Länge eines kürzesten $u-v$-Weges in in $G[M \cup v]$.
   $p[v]$ ist der Vorgänger von v auf einem solchen.

2. $\forall x \in M\ \forall y \in V \backslash M : d[x] \leq d[y]$

Beweis durch Induktion über die Anzahl der Schleifendurchläufe:

Induktionsanfang ($= 1$):
Nach erstem Durchlauf:

- $d[u] = 0$, $d[v] = \begin{cases} w(\{u,v\}) & v \in \Gamma(u) \\ \infty & \text{sonst} \end{cases}$
- $p[u] = \perp$, $p[v] = \begin{cases} u & v \in \Gamma(u) \\ \perp & \text{sonst} \end{cases}$

Induktionsschritt ($+1$):
Sei v der Knoten, der im letzten Durchlauf zu M kommt; M, p, d der Zustand vor diesem.

Zu zeigen: 1. und 2. gelten auch nach diesem Durchlauf

Zu 2.:

- sei $x \in M, y \in V \backslash (M \cup \{v\})$.
- dann gilt: $\underbrace{d[x] \leq d[v]}_{\text{nach IV}} \leq d[y]$
- $d[x]$ und $d[v]$ ändern sich nicht;
- $d[y]$ *kann* auf $\underbrace{d[v] + w(\{v,y\})}_{\geq d[v]}$ geändert werden.
- also gilt nach dem Durchlauf $d[x] \leq d[v] \leq d[y] \Rightarrow$ **2.**

Zu 1.:

- sei $y \in V \backslash (M \cup \{v\})$

  zu zeigen: nach dem Durchlauf $d[y] \hat{=}$ Länge kürzester Weg in $G[M \cup \{v,y\}]$
- gibt es einen kürzesten $u-y$-Weg in $G[M \cup \{v,y\}]$, der v nicht benutzt, dann gilt 1. nach Induktionsvoraussetzung
- andernfalls benutzt jeder kürzeste $u-y$-Weg in $G[M \cup \{v,y\}]$ v, und zwar als Vorgänger von y

  (wäre $x \in M$ Vorgänger, dann $d[x] \leq d[v]$ wegen 2. und es gibt einen kürzesten Weg, der v nicht benutzt)

- dann gilt $d(u,y) = d(u,v) + w(\{v,y\}) \overset{\text{IV}}{=} d[v] + w(\{v,y\}) \geq d[y]$ (nach dem Schleifendurchlauf)
  $p[y] = v$

□

7.3 Laufzeit von Prim und Dijkstra

Beide beruhen auf SUCHE

Frage: Wie wird der nächste Knoten gewählt?

- PRIM: möglichst dicht an M
- DIJKSTRA: möglichst kurzer Weg zu u

In beiden Fällen: $V \backslash M$ geordnet durch Abstände: Zahlen = Schlüssel

Realisiere $V \backslash M$ als Liste

- Suche des kleinsten Schlüssels dauert $\mathcal{O}(|V \backslash M|)$: ein Durchlauf pro $v \in V$, jeder braucht $\mathcal{O}(d(v))$ Schritte
- Laufzeit: $\mathcal{O}(\sum_{i=1}^{n} i + \sum_{v \in V} d(v)) = \mathcal{O}(n^2 + m)$

Schneller mit Heaps

- speichern als Feld
- *Versickern*(i):
  - $a \leq b \leq c$ und $a \leq c \Rightarrow$ fertig
  - $a > b \geq c$: tausche a und c, danach *Versickern*$(2i+1)$

  Laufzeit: $\mathcal{O}(n \log n)$ (genau $\mathcal{O}(n)$)
- *build-heap* mit Laufzeit $\mathcal{O}(\log n)$
- *min-entf* = *Entferne*(1)
  Entferne(i) mit Laufzeit $\mathcal{O}(\log n)$
- *decrease-key*$(2i, b')$ mit Laufzeit $\mathcal{O}(\log n)$

$\Rightarrow$ Laufzeit von PRIM und DIJKSTRA: $\mathcal{O}((n+m) \log n)$

8 Dynamische Programmierung

➜ Literatur: [Cormen et al.]

Erinnerung: Divide & Conquer: Aufgabe (rekursiv) in Teilaufgaben zerlegen, diese rekursiv lösen und dann zusammensetzen (top-down).

- Beispiel: *Mergesort*
  Laufzeit (Anzahl Vergleiche): $T(n) = 2T(\frac{n}{2}) + n = \mathcal{O}(n \log n)$
- Anti-Beispiel: *Fibonacci-Zahlen*
  $F(0) := 0$, $F(1) := 1$, $F(n+2) = F(n+1) + F(n)$, $n \in \mathbb{N}$
  Laufzeit bei D&C (Anzahl Additionen): $T(n)$ um $F(n)$ zu berechnen:
  $T(n+2) = T(n+1) + T(n) + 1 > 2\,T(n) + 1 > 2^{\lfloor \frac{n}{2} \rfloor} \Rightarrow$ exponentiell!

  Besser: $F(2) = F(1) + F(0)$, $F(3) = F(2) + F(1)$, ..., $F(n+2) = F(n+1) + F(n)$
  $\Rightarrow n + 1$ Additionen

Fazit: Zwischenergebnisse merken (bottom-up)

8.1 Das Wortproblem für kontextfreie Sprachen

Definition (Wortproblem):
Gegeben: $L \subseteq \Sigma^*, x \in \Sigma^*$ *Frage:* $x \in L$*?*

Nach Satz 3.2 ist das Wortproblem für kontextfreie Sprachen, gegeben durch einen LBA entscheidbar.

Beweisprinzip:

- $D = (V, A)$ gerichteter Graph mit V Konfigurationen des LBA,
  $A \ni (K, K') :\Leftrightarrow K \vdash K'$.
- Anzahl der Konfigurationen $|V| = n \cdot |Z| \cdot |\Gamma|^n$ (für $x = x_1 \dots x_n$).

- Tiefensuche: Ist eine Endkonfiguration von K_x erreichbar?

  $\Rightarrow$ exponentielle Laufzeit in $|x|$

Satz 8.1:
Sei $L \in \mathcal{CFL}$ durch $G = (V, \Sigma, P, S)$ in CNF gegeben. Dann kann $x \in L$ in $\mathcal{O}(|P| \cdot |x|^3)$ entschieden werden.

Beweis (von Satz 8.1):
Beweisidee: Dynamische Programmierung: der folgende Algorithmus von Cocke, Younger und Kasami (CYK) leistet das Verlangte. Dabei sei $x = x_1 \dots x_n$:

```
ALGORITHMUS CYK (x);
BEGIN
 FOR j = 1...n
   FOR i = 1...n-j+1
     Berechne V_{i,j} = {A ∈ V | A ⇒*_G x_i ... x_{i+j-1}}
       aus V_{i,k} und V_{i+k,j-k} für alle k = 1...j-1;        (*)
   ENDFOR;
 ENDFOR;
 IF S ∈ V_{1,n}
   THEN Ausgabe 1;
   ELSE Ausgabe 0;
 ENDIF;
END;
```

$S \in V_{1,n}$ bedeutet: $S \Rightarrow_G^* x_1 \dots x_n$. Wenn (*) funktioniert, ist CYK korrekt und die gesamte Laufzeit von CYK ist n^2·Laufzeit für (*).

Beweis der Korrektheit von (*) durch Induktion über j:

Induktionsanfang ($j = 1$):
$V_{i,1} = \{A \in V \mid A \Rightarrow_G^* x_i\}$

- $A \Rightarrow_G^* x_i \Leftrightarrow A \to x_i \in P$ (G ist in CNF)
- Berechnung $V_{i,j}$ in $\mathcal{O}(|P|)$ (alle Produktionen müssen betrachtet werden)

Induktionsschritt ($j > 1$):
$V_{i,j} = \{A \in V \mid A \Rightarrow_G^* x_i \dots x_{i+j-1}\}$

- $A \Rightarrow^* x_i \dots x_{i+j-1} \Leftrightarrow A \Rightarrow_G BC \Rightarrow_G^* x_i \dots x_{i+j-1}$ (G ist in CNF)
  $\Leftrightarrow \exists k \in [j-1] : B \Rightarrow_G^* x_i \dots x_{i+k-1}, C \Rightarrow_G^* x_{i+k} \dots x_{i+j-1}, A \to BC \in P$
  $\Leftrightarrow \exists k : B \in V_{i,k}, C \in V_{i+k,j-k}, A \to BC \in P$

- Berechnung von (*): Gehe alle Produktionen $A \to BC$ durch,
  prüfe, ob $\exists k \in [j-1] : i : B \in V_{i,k}$ und $C \in V_{i+k,j-k}$
  ja: $V_{i,j} \leftarrow V_{i,j} \cup \{A\}$
- Laufzeit: $\mathcal{O}(|P| \cdot k)$

Da k höchstens n ist, gilt für die Gesamtlaufzeit von CYK: $\mathcal{O}(|P| \cdot n^3)$ □

Beschreibung von CYK:

Ausgangsfrage: $S \Rightarrow_G^* x$?

Teilfragen: $A \Rightarrow_G^* x_i \dots x_{i+j-1}$? ($A \in V, j = 1, \dots, n-j+1$ und $j = 1, \dots, n$)

Wir setzen

- $V_{i,1} = \{A \in V \mid A \to x_i \in P\}$ und
- $V_{i,j} = \bigcup_{k=1}^{j-1} \{A \in V \mid \exists B \in V_{i,k} \exists C \in V_{i+k,j-k} : A \to BC \in P\}$

Damit lassen sich also

- $V_{1,1}, V_{2,1}, \dots, V_{n-1,1}, V_{n,1}$
- $V_{1,2}, V_{2,2}, \dots, V_{n-1,2}$
  $\vdots$
- $V_{1,n}$

berechnen.

Es gilt: $x \in L \Leftrightarrow S \in V_{1,n}$.

Beispiel 32 (CYK):
Sei $G = (\{S, A, B, X_a, X_b\}, \{a, b\}, P, S)$ mit P:

$$S \to X_a A, BX_b, X_a X_b \quad A \to SX_b \quad B \to X_a S \quad X_a \to a \quad X_b \to b$$

gegeben. Gilt $aabb \in L(G)$?

	a	a	b	b
$j=1$	X_a	X_a	X_b	X_b
2	$\emptyset$	S	$\emptyset$	
3	B	A		
4	S			

Da $S \in V_{1,4}$ gilt: $aabb \in L(G)$.

Bemerkung:
Speichere $V_{i,j}$ als Feld mit Eintrag pro $A \in V$:

$$V_{i,j}[A] = \begin{cases} A \to x_i & \text{falls } j = 1, A \to x_i \in P \\ (A \to BC, k) & \text{falls } j > 1, B \in V_{i,k}, C \in V_{i+k,j-k}, A \to BC \in P \\ \bot & \text{sonst} \end{cases}$$

```
FOR k = 1 ... j − 1
 FORALL A → BC ∈ P
   IF V_{i,k}[B] ≠⊥ ∧V_{i+k,j−k}[C] ≠⊥
     V_{i,j}[A] ← (A → BC, k);
   ENDIF;
 ENDFOR;
ENDFOR;
```

Laufzeit: $\mathcal{O}(j \cdot |P|) = \mathcal{O}(|x|^3 \cdot |P|)$, weil $\mathcal{O}(|x|^2)$ $V_{i,j}$ berechnet werden müssen.

Falls $A \in V_{i,j}$, also $V_{i,j}[A] \neq \bot$ (z.B. $V_{i,j}[A] = (A \to BC, k)$) kann (rekursiv) gefunden werden:

1. $B \Rightarrow_G^* x_i \dots x_{i+k-1}$
2. $C \Rightarrow_G^* x_{i+k} \dots x_{i+j-1}$

Daraus wird $A \Rightarrow_G AB \Rightarrow_G^* x_i \dots x_{i+k-1}C \Rightarrow_G^* x_i \dots x_{i+j-1}$ (Backtracking)

Bemerkung:
Der CYK-Algorithmus hat bei dynamischer Programmierung polynominelle Laufzeit, während er bei Rekursion exponentielle Laufzeit hat, da hier keine Zwischenergebnisse gemerkt werden.

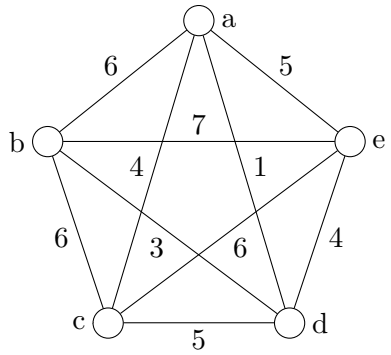
8.2 Das Travelling-Salesman-Problem

Definition (Travelling-Salesman-Problem):
*Das **Travelling-Salesman-Problem / TSP** ist wie folgt definiert:*

- *gegeben:* $n \geq 3, w : \binom{[n]}{2} \to \mathbb{R}_+$
- *gesucht: kürzeste Tour im* K_n *mit Kantengewichten* w,
  d.h. Kreis $C = ([n], \{\{i_1, i_2\}, \{i_2, i_3\}, \ldots, \{i_n, i_1\}\})$
  mit $w(C) := \sum_{e \in E(C)} w(e)$ *minimal.*

Dies wird als $TSP(n, w)$ *bezeichnet.*

Beispiel 33 (Travelling-Salesman-Problem):



Es gibt $\frac{4!}{2}$ mögliche Lösungen für dieses TSP.

Bemerkung:
Für das TSP ist kein Polynominalzeit-Algorithmus bekannt. Gibt es ein gutes Exponentialzeitverfahren?

Naiver Ansatz: „Brute Force“ mit „vollständiger Enumeration“ (d.h. Errechnen aller Möglichkeiten und Ausgabe der besten Lösung). Sei dazu f eine Liste für den jeweils aktuell errechneten Kreis, mf eine Liste des bisher besten Kreises, m das Gewicht dieses Kreises (intialisiert mit der oberen Schranke des $TSP(n, w)$, nämlich der Summe aller Gewichte), sowie

$$C_f := ([n], \{\{f[1], f[2]\}, \ldots, \{f[n-1], f[n]\}, \{f[n], f[1]\}\})$$

```
ALGORITHMUS TSPBF (n, w);
BEGIN
 M ← geordnete Liste: 1, ..., n;
 f ← leere Liste;
 mf ← leere Liste;
 m ← Σ_{1≤i<j≤n} w({i, j});
 BRANCH(M, f);
 AUSGABE C_mf;
END;
```

Dazu ist BRANCH wie folgt definiert:

```
PROCEDURE BRANCH (M, f);
BEGIN
 IF L = ∅
   IF w(C_f) < m
     m ← w(C_f);
     mf ← f;
   ENDIF;
 ELSE
     FORALL i ∈ L
       L ← L\{i};
       f ← f ∪ {i};
       BRANCH(L, f);
       f ← f\{i};
       L ← L ∪ {i};                                    (*)
     ENDFOR;
 ENDIF;
END;
```

An der Stelle (*) wird i an der alten Position in die Liste eingefügt.

Die Laufzeit von TSPBF ist $\Theta(n!) = \Theta((\frac{n}{2})^{n+\frac{1}{2}}) = 2^{\Theta(n \log n)}$

Dabei gilt: $f(n) = \Theta(g(n)) \ := \ f(n) = \mathcal{O}(g(n)) \wedge g(n) = \mathcal{O}(f(n))$ Die Θ-Schreibweise ist genauer als die $\mathcal{O}$-Schreibweise, da letztere nur eine Obergrenze der Laufzeit beschreibt.

Durch dynamische Programmierung kann eine Komplexität von $\mathcal{O}(n^2 2^n)$ erreicht werden.

- Problem bei TSP: Wie sehen Teilprobleme aus?

- Überlegung: Pfade sind einfacher → betrachte Pfade, die vorgegeben Knotenmenge besuchen

Lemma 8.2:
Sei $l \in [n]\backslash\{1\}$ und $1 \in S \subseteq [n]\backslash\{l\}$. Setzen $f(S,l) =$ Länge eines kürzesten $1-l$-Pfades, der alle Knoten in S besucht.

(i) $|S| = 1 \Rightarrow f(S,l) = w(\{1,l\})$

(ii) $|S| > 1 \Rightarrow f(S,l) = \min\{f(S',l') + w(\{l',l\}) \mid S' \dot\cup \{l'\} = S, l \neq 1\} =: M(S,l)$

(iii) $TSP(n,w) = \min\{f(S,l) + w(\{l,1\}) \mid [n] = S \dot\cup \{l\}, l \neq 1\}$

Beweis (von Lemma 8.2):
Die Punkte (i) und (iii) sollten klar sein.

Beweis von (ii) durch Induktion nach $|S|$:

Induktionsanfang ($|S| = 1$):
siehe (i)

Induktionsschritt:
Zu zeigen: $f(S,l) = M(S,l)$

- Richtung $\geq$:

  Sei P kürzester $1-l$-Pfad der alle in S besucht, l' der Vorgänger von l auf P', dem in P enthaltenden $1-l'$-Pfad.

  $w(P) = w(P') + w(\{l,l'\}) \overset{\text{IV}}{\geq} f(S\backslash\{l\},l') + w(\{l,l'\}) \geq M(S,l)$

- Richtung $\leq$:

  Das Minimum $M(S,l)$ werde für S' und l' mit $S = S' \dot\cup \{l'\}$ angenommen.

  Sei P' ein kürzester $1-l'$-Pfad, der alle in S' besucht.
  $\overset{\text{IV}}{\Rightarrow} w(P') = f(S',l')$

  P sei der Pfad, der durch Anhängen von l an P' entsteht
  $\Rightarrow w(P) = f(S',l') + w(\{l,l'\}) = M(S,l)$

  P besucht alle in S
  $\Rightarrow f(S,l) \leq w(P) = M(S,l)$

□

Satz 8.3:
Das $TSP(n,w)$ kann in $\mathcal{O}(n^2 2^n)$ berechnet werden.

Beweis (von Satz 8.3):
Der folgende Algorithmus TSPDP benutzt dynamische Programmierung und leistet das Verlangte:

```
ALGORITHMUS TSPDP;
BEGIN
 FOR l = 2...n
   f[{1}, l] ← w({1, l});
 ENDFOR;
 FOR s = 2...n − 1                                              (*)
   FORALL S ∈ ([n] s) mit 1 ∈ S
     FORALL l ∉ S
       f[S, l] ← min{f[S', l'] + w({l', l}) | S' ∪̇ {l'} = S, l ≠ 1};
     ENDFOR;
   ENDFOR;
 ENDFOR;
 Ausgabe: min{f[S, l] + w({l, 1}) | S ∪̇ {l} = [n], l ≠ 1};
END;
```

Dass TSPDP gerade TSP berechnet, folgt gerade aus Lemma 8.2.

Für die Laufzeit ist die Schleife (*) entscheidend:

- die innere FOR-Schleife hat $n - s \leq n$ Iterationen
- das Minimum wird über $s \leq n$ Terme gebildet, kann also in $\mathcal{O}(n)$ bestimmt werden
  $\Rightarrow$ Laufzeit der inneren FOR-Schleife ist $\mathcal{O}(n^2)$
- die innere Schleife wird für jedes $S \subseteq [n]$ höchstens einmal durchlaufen, also höchstens 2^n mal

$\Rightarrow$ Laufzeit von (*): $\mathcal{O}(n^2 2^n)$ □

Bemerkung:
Der Algorithmus TSPBF braucht nur $\mathcal{O}(n)$ Speicher.

Speicherbedarf bei dynamische Programmierung (TSPDP):

- Annahme: n gerade, betrachte Fall $s = \frac{n}{2}$
- es müssen $\binom{n}{\frac{n}{2}}$ Spalten gemerkt werden
- $\binom{n}{\frac{n}{2}} = \frac{n(n-1)\dots(\frac{n}{2}+1)}{\frac{n}{2}(\frac{n}{2}-1)\dots 1} \geq 2^{\frac{n}{2}} = \sqrt{2}^n$, weil $\frac{n-k}{\frac{n}{2}-k} \geq \frac{n-2k}{\frac{n}{2}-k} = 2$

$\Rightarrow$ exponentieller Speicherbedarf

Die optimale Tour kann mit Backtracking gefunden werden.

8.3 Knapsack-Problem

Definition (Knapsack-Problem, KP):
*Das **Knapsack-Problem / KP** ist wie folgt definiert:*

- *gegeben:* $a_1, \ldots, a_n$, $g_1, \ldots, g_n$, G
- *gesucht:* $S \subseteq [n]$ *mit* $g(S) = \sum_{i \in S} g_i \leq G$ *und* $a(S) = \sum_{i \in S} a_i$ *maximal.*

Dies wird als $KP(a_1, \ldots, a_n, g_1, \ldots, g_n, G)$ *bezeichnet.*

Beispiel 34 (Knapsack-Problem):
Das gegebene KP hat die Lösung $S = \{a_2\}$

i	1	2	3	4	5
g_i	25	49	12	12	13
a_i	49	80	12	12	20

Idee: dynamische Programmierung

Entscheidung, ob $n \in S$ oder nicht, abhängig davon, welcher Nutzen in $[n-1]$ möglich ist bei Rucksackgröße $G' = 0 \ldots G$

Lemma 8.4:

$$\begin{aligned} KP(a_1, \ldots, a_n, g_1, \ldots, g_n, G') &= \max\{KP(a_1, \ldots, a_{n-1}, g_1, \ldots, g_{n-1}, G'); \\ &\quad \underbrace{KP(a_1, \ldots, a_{n-1}, g_1, \ldots, g_{n-1}, G' - g_n)}_{=:0, \text{ falls } G - g_n < 0} + a_n\} \end{aligned}$$

Beweis:
Zunächst werden die Teile des Lemmas benannt:

$$\begin{aligned} \underbrace{KP(a_1, \ldots, a_n, g_1, \ldots, g_n, G')}_{KP} &= \max\{\overbrace{KP(a_1, \ldots, a_{n-1}, g_1, \ldots, g_{n-1}, G')}^{KP'}; \\ &\quad \underbrace{KP(a_1, \ldots, a_{n-1}, g_1, \ldots, g_{n-1}, G' - g_n) + a_n}_{KP''}\} \end{aligned}$$

Richtung $\leq$:

Sei $S \subseteq [n]$, sodass $a(S) = KP,\ g(S) \leq G$.
Fallunterscheidung:

- $n \in S$: $S' := S \backslash \{n\}$
  $\Rightarrow a(S') = a(S) - a_n,\ g(S') = g(S) - g_n \leq G - g_n$
  $\Rightarrow KP'' \geq a(S') + a_n = a(S)$
  $\Rightarrow \max\{KP', KP''\} \geq KP$
- $n \notin S$: $KP' \geq a(S) = KP$
  $\Rightarrow \max\{KP', KP''\} \geq KP$

Richtung $\geq$:

Sei $S' \subseteq [n-1]$ mit $a(S') = KP'$ und $g(S) \leq G$
$\Rightarrow KP \geq a(S') = KP'$

Sei $S'' \subseteq [n-1]$ mit $a(S'') = KP'' - a_n$ und $g(S'') \leq G - g_n$
$\Rightarrow S'' \cup \{n\}$ erfüllt $a(S'' \cup \{n\}) = KP''$ mit $g(S'' \cup \{n\}) \leq G$
$\Rightarrow KP \geq KP''$

□

Satz:
$KP(a_1, \ldots, a_n, g_1, \ldots, g_n, G)$ kann in $\mathcal{O}(n - G)$ berechnet werden.

A Einführung in die Graphen-Theorie

A.1 Definitionen

Definition (Graph):
Ein ***Graph*** G *ist ein 2-Tupel* $G = (V, E)$.

- V *ist eine endliche Menge von* ***Knoten*** *(****vertices****).*
- $E \subseteq \binom{V}{2}$ *(die zweielementigen Teilmengen von* V*) ist die Menge der* ***Kanten*** *(****edges****).*

Definition (Ordnung, gerichtete Graphen, einfache Graphen):
$|V|$ *heißt* ***Ordnung von*** G *und wird oft mit* n *beschrieben.* $|E| := m$

G *heißt* ***gerichtet****, falls* $E \subseteq V \times V$ *(d.h. die Kanten nicht Mengen, sondern geordneten Paare).*

Einfache Graphen *enthalten keine Loops (Kanten der Form* (v, v)*) oder Multikanten (mehrere Kanten zwischen zwei Knoten).*

Definition (adjazent, inzident, Nachbarschaft, Grad):
Zwei Knoten heißen ***adjazent****, falls* $(u, v) \in E$*. (Dies gilt für gerichtete und ungerichtete Graphen.)*

Ein Knoten $v \in V$ *und eine Kante* $e \in E$ *sind* ***inzident****, falls* $v \in e$.

Die ***Nachbarschaft eines Knotens*** $v \in V$ *wird beschrieben durch* $\Gamma(v) := \{u \mid (v, u) \in E\}$.

Der ***Grad eines Knotens*** $v \in V$ *ist die Anzahl seiner Nachbarn:* $d(v) := |\Gamma(v)|$

Definition (Weg):
Ein ***Weg*** *W im Graph $G = (V, E)$ ist ein t-Tupel $(w_1, \ldots, w_t)$, so dass $\forall w_i, w_{i+1} \in W$ gilt: $(w_i, w_{i+1}) \in E$.*

Ein Weg heißt ***Pfad****, falls jedes w_i nur einmal in W vorkommt (schleifenfreier Weg).*

Definition (Baum):
Ein Graph ist ein ***Baum****, falls je zwei Knoten aus G durch genau ein Pfad verbunden sind.*

A.2 Speicherung von Graphen

1. Adjazenzliste

   Notiere zu jedem Knoten seine Nachbarn in einer Liste.

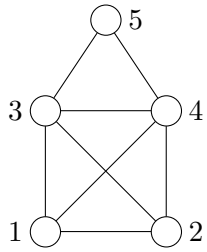
   - Platzbedarf: $\mathcal{O}(m)$
   - Zeitbedarf für die Suche $e \in E$?: $\mathcal{O}(n)$

2. Adjazenzmatrix

   - Platzbedarf: $\mathcal{O}(n^2)$
   - Zeitbedarf für die Suche $e \in E$?: $\mathcal{O}(1)$

Beispiel 35 (Graph):
$G_1 = (V, E)$ mit

- $V = \{1, 2, 3, 4, 5\}$
- $E = \{\{1,2\}, \{1,3\}, \{1,4\}, \{2,4\}, \{2,3\}, \{3,4\}, \{3,5\}, \{4,5\}\}$



Der Graph G_1 hat folgende Adjazenzmatrix:

	1	2	3	4	5
1	0	1	1	1	0
2		0	1	1	0
3			0	1	1
4				0	1
5					0

Die Zellen unter der Hauptdiagonale müssen nicht ausgefüllt werden, da G_1 ungerichtet ist und somit keine neuen Informationen gespeichert werden würden.

Es gilt: G_1 hat die Ordnung $n = 5$, $|E| = 8$, $d(1) = |\Gamma(1)| = |\{2, 3, 4\}| = 3$.

Satz:
Jeder Algorithmus der im „intuitiven“ Modell Zeitaufwand $O\left(t(n)\right)$ hat, kann durch eine 2-TM mit Zeitaufwand $O\left(t(n)^3\right)$ simuliert werden.

Beweis:
Siehe [Papadimitriou, Chapter 1.2].

Unter dem „intuitiven“ Modell versteht man eine beliebige Programmiersprache, die mittels Schleifen (z.B. WHILE oder FOR), Verzweigungen (z.B. IF), Grundrechenarten (Addition und Subtraktion), Speicherzugriffen (mit konstanter Zugriffszeit) und Rekursion ein Problem lösen kann. Die Bezeichnung „intuitiv“ bezieht sich auf die Church'sche These, die sich auf „intuitiver Berechenbarkeit“ bezieht.

A.3 Tiefensuche

Gegeben sei folgender Algorithmus für die Tiefensuche in Graphen:

```
PROCEDURE Initialisierung (G, w);
BEGIN
 FOR v ∈ V DO num[v] := 0;
 FOR i ∈ {1, ..., n} DO knot[i] := u;
 FOR v ∈ V DO bekannt[v] := FALSE;
 FOR v ∈ V DO Vor[v] := v;
 t := 0;
 bekannt[u] := TRUE;
 Tiefensuche(G, u);
END;
```

```
ALGORITHMUS TIEFENSUCHE(G, v)
BEGIN
 t := t + 1;
 num[v] := t;
 knot[t] := v;
 FOR w ∈ Γ(v) DO IF NOT bekannt[w] THEN BEGIN
   bekannt[w] := TRUE;
   Vor[w] := v;
   TIEFENSUCHE(G, w);
 END;
END;
```

Satz:
Die Laufzeit des Algorithmus ist $\mathcal{O}(m+n)$.

Beweis:
Initialisierung ist $\mathcal{O}(n)$ wegen der FOR-Schleifen. In der Prozedur Tiefensuche wird die FOR-Schleife für jeden Knoten genau einmal aufgerufen.

$$\begin{aligned} t &= \mathcal{O}\left(\sum\nolimits_{v\in V} d(v)\right) + \mathcal{O}(n) \\ &= \mathcal{O}(2m) + \mathcal{O}(n) \\ &= \mathcal{O}(m) + \mathcal{O}(n) \\ &= \mathcal{O}(m+n) \end{aligned}$$

Da jeder Kante bei der Nachbarsuche zweimal gezählt wird, gilt $\sum_{v\in V} d(v) = 2m$.

Für unzusammenhängende Graphen kann die Initialisierung geändert werden, sodass die Tiefensuche solange aufgerufen wird, bis alle Knoten besucht wurden.

Glossar

$\vdash$	Schritt einer Turingmaschine von einer Konfiguration in die nächste. *Beispiel:* $K_x \vdash K_1 \vdash \ldots \vdash K_t$
$\leq$	Reduzierbar auf. *Beispiel:* $A \leq B$
$(\cdot)$	Abkürzungszeichen bei Funktionen: 1. $M_w(\cdot)$ ist die Funktion, die von der Turingmaschine M_w berechnet wird 2. $\uparrow(\cdot) = \uparrow$ ist äquivalent zur Schreibweise $\uparrow(x) = \uparrow \ \forall x$
$\overrightarrow{w}$	Zeichenkettenvariation, die nach jedem Zeichen ein \$ schreibt. Sei $w = w_1 \ldots w_n$. Dann ist $\overrightarrow{w} = w_1\$w_2\$\ldots w_n\$$.
$\overleftarrow{w}$	Zeichenkettenvariation, die vor jedes Zeichen ein \$ schreibt. Sei $w = w_1 \ldots w_n$. Dann ist $\overleftarrow{w} = \$w_1\$w_2\$\ldots\$w_n$
$\overleftrightarrow{w}$	Zeichenkettenvariation, die vor und nach jedem Zeichen ein \$ schreibt. Sei $w = w_1 \ldots w_n$. Dann ist $\overleftrightarrow{w} = \$w_1\$w_2\$\ldots\$w_n\$$
$\uparrow$	Nicht definiert. *Beispiel:* Für $f(x) = \frac{1}{x}$ gilt: $f(0) = \uparrow$.
$\nearrow$	Maschine stoppt nicht.
$\Rightarrow_G$	Ableitungsschritt über Grammatik G *Beispiel:* $S \Rightarrow_G \varepsilon$
$\leftarrow$	Variablenzuweisung *Beispiel:* $A \leftarrow \{a\}$ bedeutet: $A := \{a\}$
$\square$	Blank. Leere Zelle auf dem Band einer Turingmaschine.
$\#$	Raute/Hash. In der Kodierung einer TM Trennzeichen von Binärcodes. *Beispiel:* $y = K_x\#\#K_1\#\#\ldots\#\#K_t$ Außerdem Kelleranfangszeichen bei Kellerautomaten ($\# \in \Gamma$).
$\dot{\cup}$	Disjunkte Vereinigung. *Beispiel:* $A \dot{\cup} B \mathrel{\hat{=}} A \cup B \wedge A \cap B = \emptyset$

Δ	Symmetrische Differenz. $A \Delta B = A \backslash B \cup B \backslash A$
$\binom{V}{2}$	Menge der zweielementigen Teilmengen der Menge V: *Beispiel:* Sei $V = \{a, b, c, d\}$. Dann ist $\binom{V}{2} = \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}\}$
$\chi_L(x)$	Charakteristische Funktion, die entscheidet, ob $x \in L$.
$\hat{\chi}_L(x)$	Semi-charakteristische Funktion, die semi-entscheidet, ob $x \in L$.
C_n	Kreis.
$\mathcal{CFL}$	Context-Free Languages. Kontextfreie Sprachen. $\mathcal{CFL} = \{L \mid L \text{ vom Typ 2}\}$
$\mathcal{CSL}$	Context-Sensitive Languages. Kontextsensitive Sprachen. $\mathcal{CSL} = \{L \mid L \text{ vom Typ 1}\}$
co-$\mathfrak{K}$	Menge der Komplemente. co-$\mathfrak{K} = \{\bar{A} \mid A \in \mathfrak{K}\}$
DFA	Deterministic finite automaton. Deterministischer endlicher Automat.
DTM	Deterministische Turingmaschine.
E_n	Leerer Graph.
ε	Das leere Wort. Es gilt: $\|\varepsilon\| = 0$
$\Gamma(v)$	Nachbarschaft des Knotens v: $\Gamma(v) := \{u \mid (v, u) \in E\}$
H	Halteproblem. $H = \{w\#x \mid M_w \text{ ist 1-DTM, } M_w(x) \neq\uparrow; \quad w, x \in \{0,1\}^*\}$
K	spezielles Halteproblem. $K = \{w \in \{0,1\}^* \mid M_w \text{ 1-DTM}, M_w \neq\uparrow\}$
K_n	Vollständiger Graph/Clique.
$K(\mathcal{F})$	Die Menge der Turingmaschinen M_w, die die Funktionen aus $\mathcal{F}$ berechnen. $K(\mathcal{F}) = \{w \mid M_w(\cdot) \in \mathcal{F}\} \notin \mathcal{REC}$
k-DTM	Deterministische Turingmaschine mit k Bändern
kf	Kontextfrei.
$L(M)$	Die Sprache, die von der Turingmaschine M akzeptiert wird. *Beispiel:* $L(M) = \{x \mid M \text{ akzeptiert die Eingabe } x\}$
LBA	Linear beschränkter Automat.

M_w	Turingmaschine mit Gödelzahl w.
$M_w(\cdot)$	Die Funktion, die die Turingmaschine M_w berechnet.
MPKP	Modifiziertes Post'sches Korrespondenzproblem. Erste Ziffer der Lösung muss 1 sein ($i = 1$).
MST	Minimal spannender Baum.
NFA	Non-deterministic finite automaton. Nicht-deterministischer endlicher Automat.
$\mathcal{O}(f)$	Komplexitätsklasse. $\mathcal{O}(f) := \{g : \mathbb{N} \to \mathbb{R} \mid \exists c \in \mathbb{R}^{>0}, \exists n_0 \in \mathbb{N} : g(n) \leq c \cdot f(n), \forall n \geq n_0\}$
$\mathfrak{P}_e$	Endliche Potenzmenge. $\mathfrak{P}_e(A) = \{B \subseteq A \mid B \text{ endlich}\}$
P_n	Pfad.
PKP	Post'sches Korrespondenzproblem.
Q_n	Hyperwürfel.
$\mathcal{RE}$	Recursively enumerable. Menge der rekursive aufzählbaren Sprachen. $\mathcal{RE} = \{A \mid A \text{ ist rekursiv aufzählbar}\}$
$\mathcal{REC}$	Recursive. Menge der entscheidbaren Sprachen. $\mathcal{REC} = \{A \mid A \text{ ist entscheidbar}\}$
$\mathcal{REG}$	Reguläre Sprachen. $\mathcal{REG} = \{L \mid L \text{ vom Typ 3}\}$
RegExp_Σ	Reguläre Ausdrücke über Σ. $\text{RegExp}_\Sigma = (\{R\}, \Sigma \cup \{(,), \mid, *, \emptyset, \varepsilon\}, P, R)$
TM	Turingmaschine.
TSP	Travelling Salesman Problem
UTM	Universelle Turingmaschine.
zsh	Zusammenhängend.

Beispielverzeichnis

Literaturverzeichnis

[Cormen et al.] **Introduction to Algorithms**
Cormen, Leiserson, Rivest
McGraw-Hill, 1990

[Emden-Weinert] **Einführung in Graphen und Algorithmen**
Thomas Emden-Weinert, Stefan Hougardy, Bernd Kreuter, Hans Jürgen Prömel, Angelika Steger
Humboldt-Universität zu Berlin, 1996

[Hopcroft] **Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie**
John E. Hopcroft, Jeffrey D. Ullman
Addison-Wesley, 1988

[Hougardy] **Graphen und Algorithmen I**
Stefan Hougardy, Hans-Jürgen Prömel
Humboldt-Universität zu Berlin, 2002

[Köbler] **Theoretische Informatik II**
Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin, 2001

[Papadimitriou] **Computational Complexity**
Christos H. Papadimitriou
Addison-Wesley, 1994

[Schöning] **Theoretische Informatik kurz gefasst**
Prof. Dr. Uwe Schöning
BI Wissenschaftsverlag Mannheim, 1992

[Starke] **Logische Grundlagen der Informatik**
Prof. Dr. Peter H. Starke
Humboldt-Universität zu Berlin, 2000

[Wegener] **Theoretische Informatik – eine algorithmische Einführung**
Prof. Dr. Ingo Wegener
B.G. Teubner Stuttgart · Leipzig, 1999

Index