Vorlesungsskript

Theoretische Informatik II

Wintersemester 2004/2005

Prof. Dr. Johannes Köbler
Humboldt-Universität zu Berlin
Lehrstuhl Komplexität und Kryptografie

4. März 2005

Inhaltsverzeichnis

1 Reguläre Sprachen

1.1 Endliche Automaten

Rechenmaschinen spielen in der Informatik eine zentrale Rolle. Hier beschäftigen wir uns mit mathematischen Modellen für Maschinentypen von unterschiedlicher Berechnungskraft. In der Vorlesung Theoretische Informatik 1 wurde die Turingmaschine als ein universales Berechnungsmodell eingeführt. In dieser Vorlesung werden wir eine Reihe von Einschränkungen dieses Maschinenmodells kennenlernen, die vielfältige praktische Anwendungen haben. Dabei betrachten wir zunächst nur Entscheidungsprobleme, was der Berechnung von $\{0,1\}$-wertigen Funktionen entspricht. Zur Beschreibung der Problemeingaben wird ein Eingabealphabet Σ verwendet.

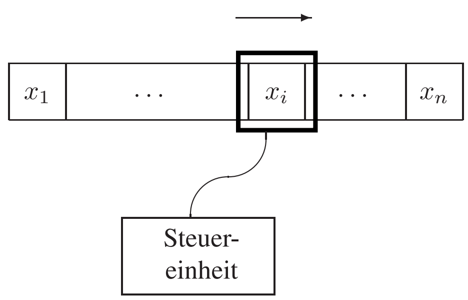
Definition 1 (Alphabet)

Ein **Alphabet** ist eine geordnete endliche Menge $\Sigma = \{a_1, \ldots, a_m\}$ von **Zeichen**. Eine Folge $x = x_1 \ldots x_n \in \Sigma^n$ heißt **Wort** (der **Länge** n). Die Menge aller Wörter über Σ ist

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n = \{x_1 \cdots x_n \mid n \geq 0 \text{ und } x_i \in \Sigma \text{ für } i = 1, \ldots, n\}.$$

Das (einzige) Wort der Länge $n = 0$ ist das **leere Wort**, welches wir mit ε bezeichnen.

Ein endlicher Automat ist eine auf ein Minimum „abgespeckte“ Turingmaschine, die nur kostant viel Speicherplatz zur Verfügung hat und bei Eingaben der Länge n nur n Rechenschritte ausführen darf. Um die gesamte Eingabe lesen zu können, muss der Automat also in jedem Schritt ein Zeichen der Eingabe verarbeiten.



Definition 2 (DFA)

Ein **endlicher Automat** (kurz: DFA; *deterministic finite automaton*) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, q_0, E)$ beschrieben, wobei

- Z eine endliche Menge von **Zuständen**,
- Σ das **Eingabealphabet**,
- $\delta : Z \times \Sigma \to Z$ die **Überführungsfunktion**,
- $q_0 \in Z$ der **Startzustand** und
- $E \subseteq Z$ die Menge der **Endzustände** ist.

Die von M **akzeptierte** oder **erkannte Sprache** ist

$$L(M) \;=\; \left\{ x_1 \cdots x_n \in \Sigma^* \;\middle|\; \begin{array}{c} \exists\, q_1, \ldots, q_{n-1} \in Z, q_n \in E: \\ \delta(q_i, x_{i+1}) = q_{i+1} \text{ für } i = 0, \ldots, n-1 \end{array} \right\}$$

Beispiel 3

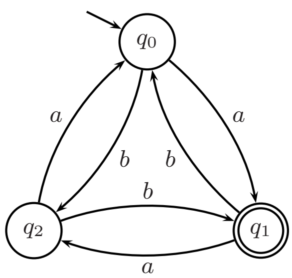
Betrachte den DFA $M = (Z, \Sigma, \delta, q_0, E)$ mit

$$\begin{aligned} Z &= \{q_0, q_1, q_2\}, \\ \Sigma &= \{a, b\}, \\ E &= \{q_1\} \end{aligned}$$

und der Überführungsfunktion

δ	q_0	q_1	q_2
a	q_1	q_2	q_0
b	q_2	q_0	q_1

Graphische Darstellung von M:



Hierbei wird der Startzustand durch einen Pfeil und Endzustände werden durch einen doppelten Kreis gekennzeichnet. Offensichtlich akzeptiert M die Sprache

$$L(M) \quad = \quad \{x \in \Sigma^* \mid \#_a(x) - \#_b(x) \equiv 1 \pmod 3\},$$

wobei $\#_a(x)$ die Anzahl der Vorkommen von a in x bezeichnet.

Eine von einem DFA akzeptierte Sprache wird als **regulär** bezeichnet. Die zugehörige Sprachklasse ist

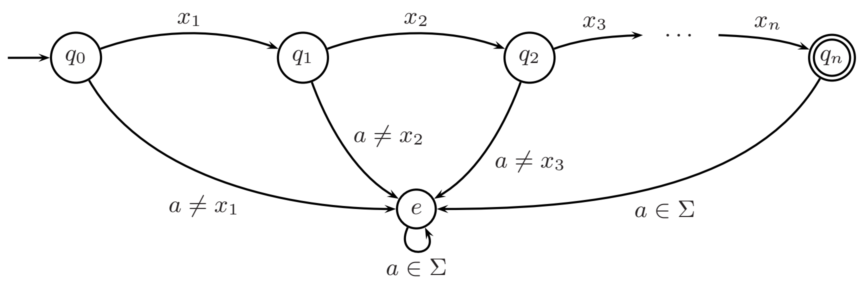
$$\mathrm{REG} = \{L(M) \mid M \text{ ist ein DFA}\}.$$

Frage: Welche Sprachen gehören zu REG und welche nicht?

Im folgenden sei $\Sigma = \{a_1, \ldots, a_m\}$ ein beliebiges aber fest gewähltes Alphabet.

Beobachtung 4

Alle Sprachen, die aus einem einzigen Wort $x = x_1 \cdots x_n \in \Sigma^*$ bestehen, sind regulär. Für folgenden DFA M gilt nämlich $L(M) = \{x\}$.



Formal lässt sich M also durch das Tupel $M = (Z, \Sigma, \delta, q_0, E)$ mit

$$\begin{aligned} Z &= \{q_0, \ldots, q_n, e\}, \\ E &= \{q_n\} \end{aligned}$$

und der Überführungsfunktion

$$\delta(q, a_j) = \begin{cases} q_{i+1}, & q = q_i \text{ für ein } i \text{ mit } 0 \leq i \leq n-1 \text{ und } a_j = x_{i+1} \\ e, & \text{sonst} \end{cases}$$

beschreiben.

Beobachtung 5

Mit $L_1, L_2 \in \mathrm{REG}$ sind auch die Sprachen $\overline{L_1} = \Sigma^* \setminus L_1$, $L_1 \cap L_2$ und $L_1 \cup L_2$ regulär. Sind nämlich $M_i = (Z_i, \Sigma, \delta_i, q_0, E_i)$, $i = 1, 2$, DFAs mit $L(M_i) = L_i$, so akzeptiert der DFA

$$\overline{M_1} = (Z_1, \Sigma, \delta_1, q_0, Z_1 \setminus E_1)$$

das Komplement $\overline{L_1}$ von L_1. Der Schnitt $L_1 \cap L_2$ von L_1 und L_2 wird dagegen von dem DFA

$$M' = (Z_1 \times Z_2, \Sigma, \delta', (q_0, q_0), E_1 \times E_2)$$

mit

$$\delta'((q, p), a) = (\delta_1(q, a), \delta_2(p, a))$$

akzeptiert. Wegen $L_1 \cup L_2 = \overline{(\overline{L_1} \cap \overline{L_2})}$ ist dann aber auch die Vereinigung von L_1 und L_2 regulär. (Wie sieht der zugehörige DFA aus?)

Aus den beiden Beobachtungen folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da die in Beispiel 3 betrachtete Sprache weder endlich noch co-endlich ist, haben wir damit allerdings noch nicht alle regulären Sprachen erfasst. Es stellt sich die Frage, ob REG neben den mengentheoretischen Operationen Schnitt, Vereinigung und Komplement unter weiteren Operationen wie etwa der **Produktbildung**

$$L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$$

oder der Bildung der **Sternhülle**

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

abgeschlossen ist. Die n-fache Potenz L^n von L ist dabei induktiv durch

$$L^0 = \{\varepsilon\},\ L^{n+1} = L^n L$$

definiert.

Im übernächsten Abschnitt werden wir sehen, dass die Klasse REG als der Abschluss der endlichen Sprachen unter Vereinigung, Produktbildung und Sternhülle charakterisierbar ist. Beim Versuch, einen endlichen Automaten für das Produkt L_1L_2 zweier regulärer Sprachen zu konstruieren, stößt man auf die Schwierigkeit, den richtigen Zeitpunkt für den Übergang von (der Simulation von) M_1 zu M_2 zu finden. Unter Verwendung eines nichtdeterministischen Automaten lässt sich dieses Problem jedoch leicht beheben, da dieser den richtigen Zeitpunkt „erraten“ kann. Im nächsten Abschnitt werden wir nachweisen, dass auch nichtdeterministische endliche Automaten nur reguläre Sprachen erkennen können.

1.2 Nichtdeterministische endliche Automaten

Definition 6 (NFA)

Ein **nichtdeterministischer endlicher Automat** (kurz: NFA; *nondeterministic finite automaton*) $M = (Z, \Sigma, \delta, Q_0, E)$ ist ähnlich aufgebaut wie ein DFA, nur dass er mehrere Startzustände (zusammengefasst in der Menge $Q_0 \subseteq Z$) haben kann und seine Überführungsfunktion

$$\delta : Z \times \Sigma \to \mathcal{P}(Z)$$

die Potenzmenge $\mathcal{P}(Z)$ von Z als Wertebereich hat. Die von M akzeptierte Sprache ist

$$L(M) = \left\{ x_1 \cdots x_n \in \Sigma^* \,\middle|\, \begin{array}{l} \exists\, q_0 \in Q_0, q_1, \ldots, q_{n-1} \in Z, q_n \in E: \\ q_{i+1} \in \delta(q_i, x_{i+1}) \text{ für } i = 0, \ldots, n-1 \end{array} \right\}$$

Ein NFA kann also nicht nur eine, sondern mehrere verschiedene Rechnungen ausführen. Die Eingabe gehört bereits dann zu $L(M)$, wenn bei einer dieser Rechnungen nach Lesen des gesamten Eingabewortes ein Endzustand erreicht wird. Im Gegensatz zu einem DFA, dessen Überführungsfunktion auf der gesamten Menge $Z \times \Sigma$ definiert ist, kann ein NFA „stecken bleiben". Das ist dann der Fall, wenn er in einen Zustand q gelangt, in dem das nächste Eingabezeichen x_i wegen $\delta(q, x_i) = \emptyset$ nicht gelesen werden kann.

Beispiel 7

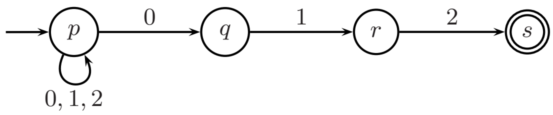
Betrachte den NFA $M = (Z, \Sigma, \delta, Q_0, E)$ mit

$$\begin{aligned} Z &= \{p, q, r, s\}, \\ \Sigma &= \{0, 1, 2\}, \\ Q_0 &= \{p\}, \\ E &= \{s\} \end{aligned}$$

und der Überführungsfunktion

δ	p	q	r	s
0	$\{p, q\}$	$\emptyset$	$\emptyset$	$\emptyset$
1	$\{p\}$	$\{r\}$	$\emptyset$	$\emptyset$
2	$\{p\}$	$\emptyset$	$\{s\}$	$\emptyset$

Graphische Darstellung von M:



Offensichtlich akzeptiert M die Sprache

$$L(M) = \{x012 \mid x \in \Sigma^*\}$$

aller Wörter, die mit dem Suffix 012 enden.

Beobachtung 8

Sind $L_1, L_2 \in$ REG, so werden die Sprachen $L_1 L_2$ und L_1^* von einem NFA erkannt. Sind nämlich $M_i = (Z_i, \Sigma, \delta_i, q_i, E_i)$, $i = 1, 2$, DFAs mit $L(M_i) = L_i$ (wobei wir $Z_1 \cap Z_2 = \emptyset$ annehmen können), so akzeptiert der NFA

$$M = (Z_1 \cup Z_2, \Sigma, \delta, \{q_1\}, E)$$

mit

$$\delta(q, a) = \begin{cases} \{\delta_1(q, a)\}, & q \in Z_1 \setminus E_1, \\ \{\delta_1(q, a), \delta_2(q_2, a)\}, & q \in E_1, \\ \{\delta_2(q, a)\}, & \text{sonst} \end{cases}$$

und

$$E = \begin{cases} E_1 \cup E_2, & q_2 \in E_2, \\ E_2, & \text{sonst} \end{cases}$$

die Sprache L_1L_2 und der NFA

$$M' = (Z_1 \cup \{q_{neu}\}, \Sigma, \delta', \{q_1, q_{neu}\}, E_1 \cup \{q_{neu}\})$$

mit

$$\delta'(q,a) = \begin{cases} \{\delta_1(q,a), \delta_1(q_1,a)\}, & q \in E_1, \\ \{\delta_1(q,a)\}, & \text{sonst} \end{cases}$$

die Sprache L_1^*.

Satz 9
REG $= \{L(M) \mid M$ ist ein NFA$\}$.

Beweis: Die Inklusion von links nach rechts ist klar, da jeder DFA auch als NFA aufgefasst werden kann. Für die Gegenrichtung konstruieren wir zu einem NFA $M = (Z, \Sigma, \delta, Q_0, E)$ einen DFA M' mit $L(M') = L(M)$. Und zwar ist

$$M' = (\mathcal{P}(Z), \Sigma, \delta', Q_0, E')$$

mit

$$\delta'(Q,a) = \bigcup_{q \in Q} \delta(q,a)$$

und

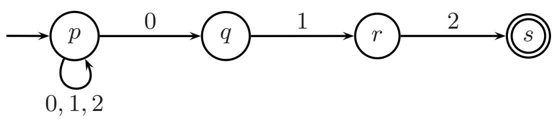
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \emptyset\}.$$

Dann gilt für alle Wörter $x = x_1 \cdots x_n \in \Sigma^*$:

$$\begin{aligned} x \in L(M) \quad &\Leftrightarrow \quad \exists\, q_0 \in Q_0\, \exists\, q_1, \ldots, q_{n-1} \in Z\, \exists\, q_n \in E : \\ &\qquad q_{i+1} \in \delta(q_i, x_{i+1}) \text{ für } i = 0, \ldots, n-1 \\ &\Leftrightarrow \quad \exists\, Q_1, \ldots, Q_n \subseteq Z : \\ &\qquad \delta'(Q_i, x_{i+1}) = Q_{i+1} \text{ und } Q_n \cap E \neq \emptyset \\ &\Leftrightarrow \quad x \in L(M'). \end{aligned}$$

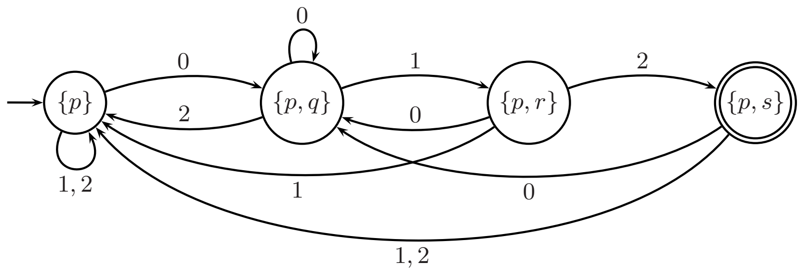
■

Beispiel 10
Für den NFA $M = (Z, \Sigma, \delta, Q_0, E)$ aus Beispiel 7



ergibt die Konstruktion des vorigen Satzes den folgenden DFA M' (nach Entfernung aller vom Startzustand $Q_0 = \{p\}$ aus nicht erreichbaren Zustände):

δ'	$Q_0 = \{p\}$	Q_1	Q_2	Q_3
0	$\{p,q\} = Q_1$	Q_1	Q_1	Q_1
1	Q_0	$\{p,r\} = Q_2$	Q_0	Q_0
2	Q_0	Q_0	$\{p,s\} = Q_3$	Q_0



Im obigen Beispiel wurden für die Konstruktion des DFA M' aus dem NFA M nur 4 der insgesamt $2^{\|Z\|} = 16$ Zustände benötigt, da die übrigen 12 Zustände in $\mathcal{P}(Z)$ nicht vom Startzustand $Q_0 = \{p\}$ aus erreichbar sind. Es gibt jedoch Beispiele, bei denen alle $2^{\|Z\|}$ Zustände in $\mathcal{P}(Z)$ für die Konstruktion von M' benötigt werden (siehe Übungen).

1.3 Reguläre Ausdrücke

Wir haben uns im letzten Abschnitt davon überzeugt, dass auch NFAs nur reguläre Sprachen erkennen können:

$$\text{REG} = \{L(M) \mid M \text{ ist ein DFA}\} = \{L(M) \mid M \text{ ist ein NFA}\}.$$

In diesem Abschnitt werden wir eine weitere Charakterisierung der regulären Sprachen kennen lernen:

> REG ist die Klasse aller Sprachen, die sich mittels der Operationen Vereinigung, Durchschnitt, Komplement, Produkt und Sternhülle aus der leeren Menge und den einelementigen Sprachen bilden lassen.

Tatsächlich kann hierbei sogar auf die Durchschnitts- und Komplementbildung verzichtet werden.

Definition 11 (regulärer Ausdruck)
Die Menge der **regulären Ausdrücke** γ (über einem Alphabet Σ) und die durch γ dargestellte Sprache $L(\gamma)$ sind induktiv wie folgt definiert:

- Die Symbole $\emptyset$ und ϵ sind reguläre Ausdrücke, welche die Sprachen $L(\emptyset) = \emptyset$ und $L(\epsilon) = \{\varepsilon\}$ beschreiben.
- Jedes Zeichen $a \in \Sigma$ ist ein regulärer Ausdruck und beschreibt die Sprache $L(a) = \{a\}$.
- Mit α und β sind auch $\alpha\beta$, $(\alpha|\beta)$, $(\alpha)^*$ reguläre Ausdrücke, die die Sprachen $L(\alpha\beta) = L(\alpha)L(\beta)$, $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$ und $L((\alpha)^*) = L(\alpha)^*$ beschreiben.

Beispiel 12
Über $\Sigma = \{0, 1\}$ sind ϵ, $(0|1)^*00$ und $(\epsilon 0|\emptyset 1)$ reguläre Ausdrücke, die die Sprachen $\{\varepsilon\}$, $\{x00 \mid x \in \Sigma^*\}$ und $\{0\}$ beschreiben.

Da die beiden regulären Ausdrücke $\gamma^*\gamma$ und $\gamma\gamma^*$ die Sprache $L(\gamma)^+$ beschreiben, soll γ^+ als Abkürzung für die regulären Ausdrücke $\gamma^*\gamma$ und $\gamma\gamma^*$ verwendet werden.

Satz 13
$\mathrm{REG} = \{L(\gamma) \mid \gamma$ ist ein regulärer Ausdruck$\}$.

Beweis: Die Inklusion von rechts nach links folgt aus den Beobachtungen 1 bis 3, zusammen mit Satz 9. Für die Gegenrichtung konstruieren wir zu einem DFA M einen regulären Ausdruck γ mit $L(\gamma) = L(M)$. Sei also $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA, wobei wir annehmen können, dass $Z = \{1, \dots, l\}$ und $q_0 = 1$ ist. Dann lässt sich $L(M)$ als Vereinigung

$$L(M) = \bigcup_{q \in E} L_{1,q}$$

von Sprachen der Form

$$L_{p,q} = \{x \in \Sigma^* \mid \hat{\delta}(p, x) = q\}$$

darstellen, wobei $\hat{\delta}(p, x)$ denjenigen Zustand bezeichnet, in dem sich M nach Lesen von x befindet, wenn M im Zustand p gestartet wird. Induktive Definition von $\hat{\delta}$:

$$\begin{aligned} \hat{\delta}(p, \varepsilon) &= p, \\ \hat{\delta}(p, xa) &= \delta(\hat{\delta}(p, x), a). \end{aligned}$$

Aufgrund obiger Darstellung von $L(M)$ reicht es zu zeigen, dass die Sprachen $L_{p,q}$ durch reguläre Ausdrücke beschreibbar sind. Hierzu betrachten wir die Sprachen

$$L^r_{p,q} = \{x \in \Sigma^* \mid \hat{\delta}(p, x) = q \text{ und für } i = 1, \dots, n-1 \text{ gilt } \hat{\delta}(p, x_1 \cdots x_i) \le r\}.$$

Wegen $L^l_{p,q} = L_{p,q}$ reicht es, reguläre Ausdrücke für die Sprachen $L^r_{p,q}$ anzugeben. Im Fall $r = 0$ enthält

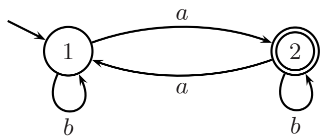
$$L^0_{p,q} = \begin{cases} \{a \in \Sigma \mid \delta(p, a) = q\} \cup \{\varepsilon\}, & p = q, \\ \{a \in \Sigma \mid \delta(p, a) = q\}, & \text{sonst} \end{cases}$$

nur Buchstaben (und eventuell das leere Wort) und ist somit leicht durch einen regulären Ausdruck beschreibbar. Nehmen wir nun an, dass die Sprachen $L^r_{p,q}$ durch reguläre Ausdrücke $\gamma^r_{p,q}$ beschreibbar sind, so folgt wegen

$$\begin{aligned} L^{r+1}_{p,q} &= L^r_{p,q} \cup L^r_{p,r+1}(L^r_{r+1,r+1})^* L^r_{r+1,q} \\ &= L(\gamma^r_{p,q} | \gamma^r_{p,r+1}(\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}) \end{aligned}$$

induktiv, dass auch die Sprachen $L^{r+1}_{p,q}$ durch reguläre Ausdrücke beschreibbar sind. ■

Beispiel 14
Die von dem DFA



erkannte Sprache ist $L^2_{1,2} = L(b^*a(b|ab^*a)^*)$. Die folgende Tabelle zeigt die regulären Ausdrücke $\gamma^r_{p,q}$, $p, q \in \{1, 2\}$ und $r \in \{0, 1, 2\}$, soweit sie zur Bildung von $\gamma^2_{1,2}$ benötigt werden.

r	p,q			
	$1,1$	$1,2$	$2,1$	$2,2$
0	$\epsilon\|b$	a	a	$\epsilon\|b$
1	-	$\underbrace{a\|(\epsilon\|b)(\epsilon\|b)^*a}_{b^*a}$	-	$\underbrace{(\epsilon\|b)\|a(\epsilon\|b)^*a}_{\epsilon\|b\|ab^*a}$
2	-	$\underbrace{b^*a\|b^*a(\epsilon\|b\|ab^*a)^*(\epsilon\|b\|ab^*a)}_{b^*a(b\|ab^*a)^*}$	-	-

1.4 Das Pumping-Lemma

Wie kann man von einer Sprache nachweisen, dass sie nicht regulär ist? Eine Möglichkeit besteht darin, die Kontraposition folgender Aussage anzuwenden.

Satz 15 (Pumping-Lemma für reguläre Sprachen)
Zu jeder regulären Sprache L gibt es eine Zahl l, so dass sich alle Wörter $x \in L$ mit $|x| \geq l$ in $x = uvw$ zerlegen lassen mit

1. $v \neq \varepsilon$,
2. $|uv| \leq l$ und
3. $uv^iw \in L$ für alle $i \geq 0$.

Das kleinste solche l wird auch **Pumping-Zahl** von L genannt.

Beweis: Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA mit $L(M) = L$ und sei l die Anzahl der Zustände von M. Setzen wir nun M auf eine Eingabe $x = x_1 \cdots x_n \in L$ der Länge $n \geq l$ an, so muss M nach spätestens l Schritten einen Zustand $q \in Z$ zum zweiten Mal annehmen:

$$\exists\, 0 \leq j < k \leq l : \hat{\delta}(q_0, x_1 \cdots x_j) = \hat{\delta}(q_0, x_1 \cdots x_k) = q.$$

Wählen wir nun $u = x_1 \cdots x_j$, $v = x_{j+1} \cdots x_k$ und $w = x_{k+1} \cdots x_n$, so ist $|v| = k - j \geq 1$ und $|uv| = k \leq l$. Ausserdem gilt $uv^i w \in L$ für $i \geq 0$, da wegen $\hat{\delta}(q, v) = q$

$$\hat{\delta}(q_0, uv^i w) = \hat{\delta}(\hat{\delta}(\underbrace{\hat{\delta}(q_0, u)}_{q}, v^i), w) = \hat{\delta}(\underbrace{\hat{\delta}(q, v^i)}_{q}, w) = \hat{\delta}(q_0, x) \in E$$

ist. ■

Um also $L \notin$ REG nachzuweisen, genügt es, für jede Zahl l ein Wort $x \in L$ der Länge $|x| \geq l$ anzugeben, so dass für jede Zerlegung von x in drei Teilwörter u, v, w mindestens eine der drei in Satz 15 aufgeführten Eigenschaften verletzt ist.

Beispiel 16

Jede endliche Sprache L lässt sich „pumpen". Wie man leicht sieht, genügt es im Fall $L \neq \emptyset$, die um Eins erhöhte Maximallänge

$$l = \max\{|x| + 1 \mid x \in L\}$$

aller in L enthaltenen Wörter als Pumping-Zahl zu wählen.

Beispiel 17

Die Sprache

$$L = \{a^j b^j \mid j \geq 0\}$$

ist nicht regulär, da sich für jedes $l \geq 0$ ein Wort $x = a^l b^l$ in L der Länge $|x| = 2l \geq l$ finden lässt, das offensichtlich nicht in Teilwörter u, v, w mit $v \neq \varepsilon$ und $uv^2 w \in L$ zerlegbar ist.

Beispiel 18

Die Sprache

$$L = \{a^{j^2} \mid j \geq 0\}$$

ist nicht regulär. Andernfalls müsste es nämlich eine Zahl l geben, so dass jede Quadratzahl $j^2 \geq l$ als Summe von natürlichen Zahlen $u + v + w$ darstellbar ist mit der Eigenschaft, dass $v \geq 1$ und $u + v \leq l$ ist und für jedes $i \geq 0$ auch $u + iv + w$ eine Quadratzahl ist. Insbesondere müsste also $u + 2v + w$ eine Quadratzahl sein, was wegen

$$j^2 = u + v + w < u + 2v + w < j^2 + l + 1 = (j + 1)^2 - (2j - l)$$

für alle $j \geq l/2$ ausgeschlossen ist.

Beispiel 19

Die Sprache

$$L = \{a^p \mid p \text{ prim }\}$$

ist nicht regulär. Andernfalls ließe sich jede Primzahl p einer bestimmten Mindestgröße l als Summe von natürlichen Zahlen $u + v + w$ darstellen, so dass $v \geq 1$, $u + v \leq l$ und für alle $i \geq 0$ auch $u + iv + w$ prim ist. Insbesondere müsste also $u + (u + w)v + w$ eine Primzahl sein, was wegen

$$u + (u + w)v + w = (u + w)(v + 1) = \underbrace{(p - v)}_{\geq p - l} \underbrace{(v + 1)}_{\geq 2}$$

für alle Primzahlen $p \geq l + 2$ ausgeschlossen ist.

Bemerkung 20

Mit dem Pumping-Lemma können nicht alle Sprachen $L \notin \text{REG}$ als nicht regulär nachgewiesen werden, da seine Umkehrung falsch ist. Beispielsweise hat die Sprache

$$L = \{a^i b^j c^k \mid i = 0 \text{ oder } j = k\}$$

die Pumping-Zahl 1 (jedes Wort $x \in L$ mit Ausnahme von ε kann also „gepumpt" werden), obwohl L nicht regulär ist (siehe Übungen).

1.5 Minimierung von DFAs

Wie können wir feststellen, ob ein DFA $M = (Z, \Sigma, \delta, q_0, E)$ unnötige Zustände enthält? Zunächst einmal können alle Zustände entfernt werden, die nicht vom Startzustand aus erreichbar sind. Im folgenden gehen wir daher davon aus, dass M keine unerreichbaren Zustände enthält. Offensichtlich können zwei Zustände q und p zu einem Zustand verschmolzen werden (kurz: $q \sim p$), wenn M von q und von p ausgehend jeweils dieselben Wörter akzeptiert. Bezeichnen wir den DFA $(Z, \Sigma, \delta, q, E)$ mit M_q und $L(M_q)$ mit L_q, so sind q und p genau dann verschmelzbar, wenn $L_q = L_p$ ist. Für $q \in Z$ sei

$$\tilde{q} = \{p \in Z \mid L_q = L_p\}$$

die durch q repräsentierte Äquivalenzklasse und für eine Teilmenge $Q \subseteq Z$ sei $\tilde{Q} = \{\tilde{q} \mid q \in Q\}$.

Satz 21

Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA für $L = L(M)$. Dann ist auch $\tilde{M} = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E})$ mit

$$\delta'(\tilde{q}, a) = \widetilde{\delta(q, a)}$$

ein DFA für L, welcher zudem eine minimale Anzahl von Zuständen besitzt.

Beweis: Zuerst müssen wir zeigen, dass der Wert von $\delta'(\tilde{q}, a)$ nicht von der Wahl des Repräsentanten q abhängt. Hierzu zeigen wir, dass $\widetilde{\delta(q, a)} \neq \widetilde{\delta(p, a)}$ nur im Fall $\tilde{p} \neq \tilde{q}$ gelten kann:

$$\begin{aligned}
L_{\delta(q,a)} \neq L_{\delta(p,a)} \quad &\Leftrightarrow \quad \exists x \in \Sigma^* : x \in L_{\delta(q,a)} \Delta L_{\delta(p,a)} \\
&\Leftrightarrow \quad \exists x \in \Sigma^* : ax \in L_q \Delta L_p \\
&\Rightarrow \quad L_q \neq L_p.
\end{aligned}$$

Als nächstes zeigen wir, dass $L(\tilde{M}) = L(M)$ ist. Sei $x = x_1 \cdots x_n$ eine beliebige Eingabe und seien $q_i = \hat{\delta}(q_0, x_1 \cdots x_i)$ die von M beim Abarbeiten von x durchlaufenen Zustände. Dann durchläuft $\tilde{M}$ offensichtlich die Zustände $\tilde{q}_0, \tilde{q}_1, \cdots, \tilde{q}_n$. Da aber q_n genau dann zu E gehört, wenn $\tilde{q}_n \in \tilde{E}$ ist, folgt $L(\tilde{M}) = L(M) = L$.

Es bleibt zu zeigen, dass die Anzahl $k = \|\tilde{Z}\|$ der Zustände von $\tilde{M}$ minimal ist. Dies ist sicher dann der Fall, wenn bereits M minimal ist. Daher reicht es zu zeigen, dass k nur von L (und nicht von M) abhängt. Für $x \in \Sigma^*$ sei L_x die Sprache $\{y \in \Sigma^* \mid xy \in L\}$. Wegen $L_x = L_{\hat{\delta}(q_0,x)}$ folgt $\{L_x \mid x \in \Sigma^*\} \subseteq \{L_q \mid q \in Z\}$. Da nach Voraussetzung jeder Zustand $q \in Z$ von q_0 aus erreichbar ist, gilt auch die umgekehrte Inklusion, und somit hängt

$$k = \|\{\tilde{q} \mid q \in Z\}\| = \|\{L_q \mid q \in Z\}\| = \|\{L_x \mid x \in \Sigma^*\}\|$$

tatsächlich nur von L ab. ■

Bei der Konstruktion von $\tilde{M}$ aus M kann man so verfahren, dass man beginnend mit der Menge

$$V_0 = \{\{q, p\} \mid q \in E, p \notin E\}$$

die Menge V_{i+1} dadurch bildet, dass man zu V_i alle Paare $\{q, p\}$ hinzufügt, für die eines der Paare $\{\delta(q, a), \delta(p, a)\}$, $a \in \Sigma$, bereits zu V_i gehört:

$$V_{i+1} := V_i \cup \{\{q, p\} \mid \exists a \in \Sigma : \{\delta(q, a), \delta(p, a)\} \in V_i\}.$$

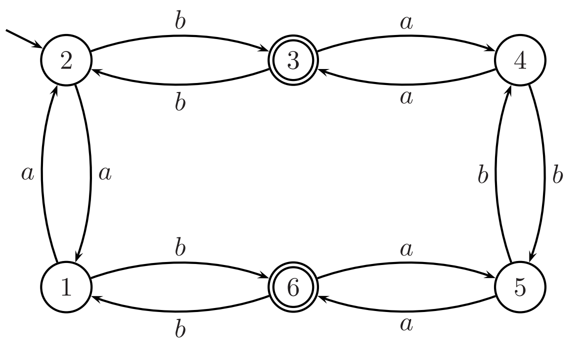
Sobald sich auf diese Weise keine weiteren Paare zu V_i mehr hinzufügen lassen (d.h. es gilt $V_{i+1} = V_i$), enthält V_i alle Paare von unverschmelzbaren Zuständen (siehe Übungen),

$$V_i = \{\{q, p\} \mid q \not\sim p\}.$$

Daher kann nun $\tilde{M}$ durch Verschmelzen aller Zustände q, p von M mit $\{q, p\} \notin V_i$ gebildet werden.

Beispiel 22

Betrachte den DFA



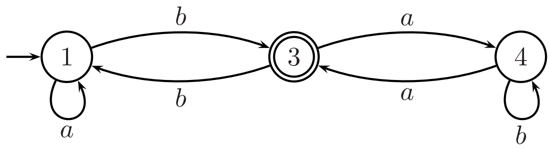
Dann enthält V_0 die Paare $\{1, 3\}$, $\{2, 3\}$, $\{4, 3\}$, $\{5, 3\}$, $\{1, 6\}$, $\{2, 6\}$, $\{4, 6\}$, $\{5, 6\}$. Wegen

$\{q, p\}$	$\{1, 4\}$	$\{1, 5\}$	$\{2, 4\}$	$\{2, 5\}$
$\{\delta(q, a), \delta(p, a)\}$	$\{2, 3\}$	$\{2, 6\}$	$\{1, 3\}$	$\{1, 6\}$

enthält V_1 zusätzlich die Paare $\{1,4\}$, $\{1,5\}$, $\{2,4\}$, $\{2,5\}$. Da jedoch die verbliebenen Paare $\{1,2\}$, $\{4,5\}$, $\{3,6\}$ wegen

$\{q,p\}$	$\{1,2\}$	$\{4,5\}$	$\{3,6\}$
$\{\delta(q,a),\delta(p,a)\}$	$\{1,2\}$	$\{3,6\}$	$\{4,5\}$
$\{\delta(q,b),\delta(p,b)\}$	$\{3,6\}$	$\{4,5\}$	$\{1,2\}$

nicht zu V_1 hinzugefügt werden können, ist $V_2 = V_1$ und die entsprechenden Zustände können verschmolzen werden:



Aus obigem Beweis ist ersichtlich, dass ein Minimalautomat auch direkt aus einer regulären Sprache L gewonnen werden kann: Da zwei Eingaben x und y den DFA $\tilde{M}$ genau dann in denselben Zustand überführen, wenn $L_x = L_y$ ist, können wir die Zustände von $\tilde{M}$ auch mit den Sprachen L_x benennen. Wir erhalten damit den DFA $M_L = (Z, \Sigma, \delta, L_\varepsilon, E)$ mit

$$\begin{aligned} Z &= \{L_x \mid x \in \Sigma^*\}, \\ E &= \{L_x \mid x \in L\} \text{ und} \\ \delta(L_x, a) &= L_{xa}, \end{aligned}$$

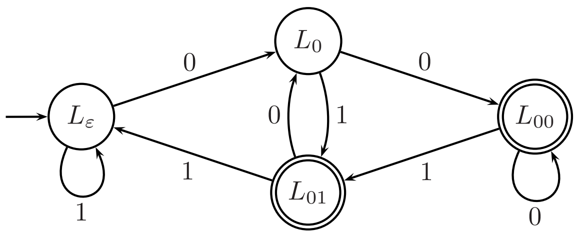
welcher isomorph zu $\tilde{M}$ ist (also bis auf die Benennung der Zustände mit diesem identisch ist).

Beispiel 23

Sei $L = \{x_1 \cdots x_n \mid x_i \in \{0,1\} \text{ für } i = 1, \ldots n \text{ und } x_{n-1} = 0\}$. Dann gilt

$$L_x = \begin{cases} L, & x \in \{\varepsilon, 1\} \text{ oder } x \text{ endet mit } 11, \\ L \cup \{0,1\}, & x = 0 \text{ oder } x \text{ endet mit } 10, \\ L \cup \{\varepsilon, 0, 1\}, & x \text{ endet mit } 00, \\ L \cup \{\varepsilon\}, & x \text{ endet mit } 01. \end{cases}$$

Somit erhalten wir den folgenden Minimalautomaten M_L:



Im Fall, dass M bereits ein Minimalautomat ist, sind alle Zustände von $\tilde{M}$ von der Form $\tilde{q} = \{q\}$, so dass M isomorph zu $\tilde{M}$ und damit auch isomorph zu M_L ist. Dies zeigt, dass alle Minimalautomaten für eine Sprache isomorph sind.

Häufig werden auch die Äquivalenzklassen $[x] = \{y \mid x\, R_L\, y\}$ der durch

$$x\, R_L\, y \Leftrightarrow L_x = L_y$$

definierten Relation R_L anstelle der Sprachen L_x zur Benennung der Zustände des Minimalautomaten verwendet. Dies führt auf den so genannten *Äquivalenzklassenautomaten* $(Z, \Sigma, \delta, [\varepsilon], E)$ mit

$$\begin{array}{rcl} Z & = & \{[x] \mid x \in \Sigma^*\}, \\ E & = & \{[x] \mid x \in L\} \text{ und} \\ \delta([x], a) & = & [xa]. \end{array}$$

Schließlich sei noch angemerkt, dass sich aus obigem Beweis eine weitere Charakterisierung der regulären Sprachen ergibt: Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn die Relation R_L endlich viele verschiedene Äquivalenzklassen $[x]$ hat (man sagt auch: R_L hat endlichen Index). Formal:

$$L \in \text{REG} \Leftrightarrow \|\{[x] \mid x \in \Sigma^*\}\| < \infty.$$

Dies ist offensichtlich genau dann der Fall, wenn die Sprachklasse $\{L_x \mid x \in \Sigma^*\}$ endlich ist.

Beispiel 24

Sei $L = \{a^i b^i \mid i \geq 0\}$. Wegen $b^i \in L_{a^i} \Delta L_{a^j}$ für $i \neq j$ hat R_L unendlichen Index, d.h. L ist nicht regulär. (Dies hatten wir bereits mit Hilfe des Pumping-Lemmas gezeigt.)

1.6 Grammatiken

Eine beliebte Methode, Sprachen zu beschreiben, sind Grammatiken. Implizit haben wir hiervon bei der Definition der regulären Ausdrücke Gebrauch gemacht.

Beispiel 25

Sei Σ ein Alphabet. Dann lassen sich alle regulären Ausdrücke über Σ unter Anwendung der Regeln

$$\begin{array}{rcl} R & \to & \emptyset, \epsilon \\ R & \to & a,\ a \in \Sigma \\ R & \to & RR,\ (R|R),\ (R)^* \end{array}$$

aus dem Symbol R erzeugen.

Definition 26 (Grammatik)
Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V eine endliche Menge von **Variablen** (auch **Nichtterminalsymbole** genannt),
- Σ das **Terminalalphabet**,
- $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ eine endliche Menge von **Regeln** (oder **Produktionen**) und
- $S \in V$ die **Startvariable** ist.

Für $(u, v) \in P$ schreiben wir auch kurz $u \rightarrow_G v$ bzw. $u \rightarrow v$, wenn die benutzte Grammatik aus dem Kontext ersichtlich ist. Durch G ist eine Relation $\Rightarrow_G$ auf $(V \cup \Sigma)^*$ wie folgt definiert:

$\alpha \Rightarrow_G \beta$, falls $l, r, u, v \in (V \cup \Sigma)^*$ existieren mit

- $\alpha = lur$,
- $u \rightarrow_G v$ und
- $\beta = lvr$.

Das n-fache Produkt von $\Rightarrow_G$ ist $\Rightarrow_G^n$, d.h. $u \Rightarrow_G^n v$ bedeutet, es gibt $u_0 = u$, $u_1, \dots, u_{n-1}, u_n = v$ mit $u_{i-1} \Rightarrow_G u_i$ für $i = 1, \dots, n$. Die reflexive, transitive Hülle von $\Rightarrow_G$ ist $\Rightarrow_G^*$, d.h. $u \Rightarrow_G^* v$ bedeutet, es gibt ein $n \geq 0$ mit $u \Rightarrow_G^n v$. Ein Wort $\alpha \in (V \cup \Sigma)^*$ heißt **Satzform**, wenn $S \Rightarrow_G^* \alpha$ gilt. Eine Folge

$$(l_0, u_0, r_0), \dots, (l_m, u_m, r_m) \quad \text{(kurz: } l_0\underline{u_0}r_0 \Rightarrow_G \cdots \Rightarrow_G l_m\underline{u_m}r_m\text{)}$$

heißt **Ableitung** von x (der Länge m), wenn $(l_0, u_0, r_0) = (\varepsilon, S, \varepsilon)$ und $l_m u_m r_m = x$ ist und $l_i u_i r_i \Rightarrow_G l_{i+1} u_{i+1} r_{i+1}$ für $i = 0, \dots, m-1$ mittels einer Regel der Form $u_i \rightarrow_G v$ gilt. Die durch G **erzeugte Sprache** ist

$$L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$$

Beispiel 25 (Fortsetzung)
Wir betrachten nochmals die Grammatik $G = (\{R\}, \Sigma \cup \{\emptyset, \epsilon, (,),^*, |\}, P, R)$, die die Menge der regulären Ausdrücke über dem Alphabet Σ erzeugt, wobei P die oben angegebenen Regeln enthält. Ist $\Sigma = \{0, 1\}$, so lässt sich der reguläre Ausdruck $(01)^*(\epsilon|\emptyset)$ beispielsweise wie folgt ableiten:

$$\begin{aligned} \underline{R} &\Rightarrow \underline{R}R \Rightarrow (\underline{R})^*R \Rightarrow (RR)^*R \Rightarrow (\underline{R}R)^*(R|R) \\ &\Rightarrow (0\underline{R})^*(R|R) \Rightarrow (01)^*(\underline{R}|R) \Rightarrow (01)^*(\epsilon|\underline{R}) \Rightarrow (01)^*(\epsilon|\emptyset) \end{aligned}$$

Man unterscheidet vier verschiedene Typen von Grammatiken.

Definition 27 (Typ einer Grammatik)
Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

1. G heißt **vom Typ 3** oder **regulär**, falls für alle Regeln $u \to v$ gilt: $u \in V$ und $v \in \Sigma V \cup \Sigma \cup \{\epsilon\}$.
2. G heißt **vom Typ 2** oder **kontextfrei**, falls für alle Regeln $u \to v$ gilt: $u \in V$.
3. G heißt **vom Typ 1** oder **kontextsensitiv**, falls für alle Regeln $u \to v$ gilt: $|v| \geq |u|$ (mit Ausnahme der ε-Sonderregel, siehe unten).
4. Jede Grammatik ist automatisch **vom Typ 0**.

ε-Sonderregel: Bei einer kontextsensitiven Grammatik ist auch die Regel $S \to \varepsilon$ zugelassen. Wird hiervon Gebrauch gemacht, so darf das Startsymbol S jedoch nicht auf der rechten Seite einer Regel vorkommen.

Die Sprechweisen „vom Typ i" bzw. „regulär", „kontextfrei" und „kontextsensitiv" werden auch auf die durch eine solche Grammatik erzeugte Sprache angewandt. (Der folgende Satz rechtfertigt dies für die regulären Sprachen, die wir bereits mit Hilfe von DFAs definiert haben.) Die zugehörigen neuen Sprachklassen sind

$$\text{CFL} = \{L(G) \mid G \text{ ist eine kontextfreie Grammatik}\},$$

(*context free languages*) und

$$\text{CSL} = \{L(G) \mid G \text{ ist eine kontextsensitive Grammatik}\}$$

(*context sensitive languages*). Da die Klasse der Typ 0 Sprachen mit der Klasse der rekursiv aufzählbaren (*recursively enumerable*) Sprachen übereinstimmt, bezeichnen wir diese Sprachklasse mit

$$\text{RE} = \{L(G) \mid G \text{ ist eine Grammatik}\}.$$

Wie wir noch sehen werden, bilden die Sprachklassen

$$\text{REG} \subset \text{CFL} \subset \text{CSL} \subset \text{RE}$$

eine Hierarchie (d.h. die Inklusionen sind echt), die so genannte **Chomsky-Hierarchie**.

Satz 28

$$\text{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}.$$

Beweis: Sei $L \in$ REG und sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA mit $L(M) = L$. Wir konstruieren eine reguläre Grammatik $G = (V, \Sigma, P, S)$ mit $L(G) = L$. Setzen wir

$$\begin{aligned} V &= Z, \\ S &= q_0 \text{ und} \\ P &= \{q \to ap \mid \delta(q,a) = p\} \cup \{q \to \varepsilon \mid q \in E\}, \end{aligned}$$

so gilt für alle Wörter $x = x_1 \cdots x_n \in \Sigma^*$:

$$\begin{aligned} x \in L(M) &\Leftrightarrow \exists\, q_1, \ldots, q_{n-1} \in Z\; \exists\, q_n \in E : \delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \ldots, n \\ &\Leftrightarrow \exists\, q_1, \ldots, q_n \in V : q_{i-1} \to x_i q_i \text{ für } i = 1, \ldots, n \text{ und } q_n \to \varepsilon \\ &\Leftrightarrow \exists\, q_1, \ldots, q_n \in V : q_0 \Rightarrow^i x_1 \cdots x_i q_i \text{ für } i = 1, \ldots, n \text{ und } q_n \to \varepsilon \\ &\Leftrightarrow x \in L(G) \end{aligned}$$

Für die entgegengesetzte Inklusion sei nun $G = (V, \Sigma, P, S)$ eine reguläre Grammatik. Dann wird die Sprache $L = L(G)$ auch von der Grammatik $G' = (V', \Sigma, P', S)$ mit

$$\begin{aligned} V' &= V \cup \{X_{neu}\}, \\ P' &= \{A \to aX_{neu} \mid A \to_G a\} \cup \{X_{neu} \to \varepsilon\} \cup P \setminus (V \times \Sigma) \end{aligned}$$

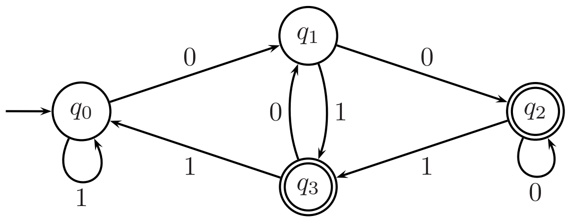
erzeugt. Da G' keine Produktionen der Form $A \to a$ mehr enthält, können wir die gerade beschriebene Konstruktion einer Grammatik aus einem DFA „umdrehen", um ausgehend von G' einen NFA $M' = (Z', \Sigma, \delta', \{S\}, E')$ mit

$$\begin{aligned} Z' &= V', \\ E' &= \{A \mid A \to_{G'} \varepsilon\} \text{ und} \\ \delta'(A, a) &= \{B \mid A \to_{G'} aB\} \end{aligned}$$

zu erhalten. Genau wie oben folgt nun $L(M') = L(G')$. ■

Beispiel 29

Der DFA



führt auf die Grammatik $(\{q_0, q_1, q_2, q_3\}, \{0, 1\}, P, q_0)$ mit

$$\begin{aligned} P: \quad & q_0 \to 1q_0, 0q_1, \\ & q_1 \to 0q_2, 1q_3, \\ & q_2 \to 0q_2, 1q_3, \varepsilon, \\ & q_3 \to 0q_1, 1q_0, \varepsilon. \end{aligned}$$

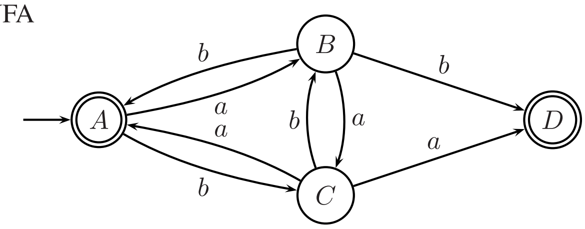
Umgekehrt führt die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ mit

$$\begin{aligned} P: \quad & A \to aB, bC, \varepsilon, \\ & B \to aC, bA, b, \\ & C \to aA, bB, a \end{aligned}$$

über die Grammatik $G' = (\{A, B, C, D\}, \{a, b\}, P', A)$ mit

$$\begin{aligned} P': \quad & A \to aB, bC, \varepsilon, \\ & B \to aC, bA, bD, \\ & C \to aA, bB, aD, \\ & D \to \varepsilon \end{aligned}$$

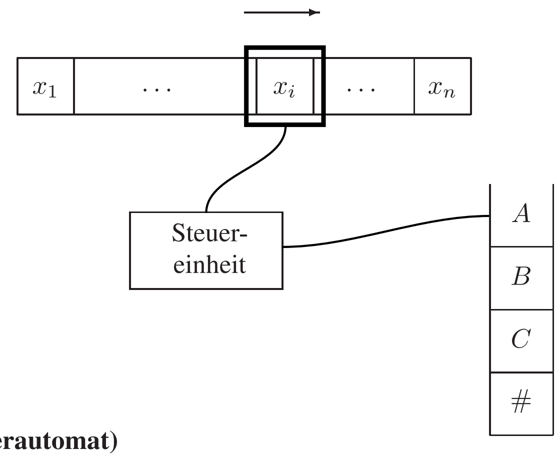
auf den NFA



2 Kontextfreie Sprachen

2.1 Äquivalenz von kontextfreien Grammatiken und Kellerautomaten

Dass ein DFA die Sprache $L = \{a^n b^n \mid n \geq 0\}$ nicht erkennen kann, liegt an der Beschränkung seines Speichervermögens durch eine Konstante (die zwar von L aber nicht von der Eingabe abhängen darf). Um L erkennen zu können, reicht bereits ein so genannter Kellerspeicher (engl. *stack, pushdown memory*) aus, der nur auf die höchste belegte Speicheradresse zugreifen darf.



Definition 30 (Kellerautomat)

Ein **Kellerautomat** (kurz: PDA; *pushdown automaton*) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ beschrieben, wobei

- Z, Σ und q_0 wie bei einem DFA,
- Γ das **Kelleralphabet**,
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ die **Überführungsfunktion** und
- $\# \in \Gamma$ das **Kelleranfangszeichen** ist.

Mit $\mathcal{P}_e(M)$ bezeichnen wir hierbei die Menge aller endlichen Teilmengen von M,

$$\mathcal{P}_e(M) = \{A \subseteq M \mid A \text{ ist endlich}\}.$$

Wenn q der momentane Zustand, A das oberste Kellerzeichen und $u \in \Sigma$ das nächste Eingabezeichen (bzw. $u = \varepsilon$) ist, so kann M im Fall $(p, B_1 \cdots B_k) \in \delta(q, u, A)$

- in den Zustand p wechseln,
- den Lesekopf auf dem Eingabeband um $|u|$ Positionen vorrücken und
- das Zeichen A im Keller durch die Zeichenfolge $B_1 \cdots B_k$ ersetzen.

Hierfür schreiben wir auch kurz $quA \to pB_1 \cdots B_k$. Da im Fall $u = \varepsilon$ kein Eingabezeichen gelesen wird, spricht man auch von einem spontanen Übergang (oder ε-Übergang).

Eine **Konfiguration** wird durch ein Tripel

$$K = (q, x_i \cdots x_n, A_1 \cdots A_l) \in Z \times \Sigma^* \times \Gamma^*$$

beschrieben und besagt, dass q der momentane Zustand, $x_i \cdots x_n$ der ungelesene Rest der Eingabe und $A_1 \cdots A_l$ der aktuelle Kellerinhalt ist (oberstes Kellerzeichen ist A_1). Im Fall $quA_1 \to pB_1 \cdots B_k$ mit $u \in \{\varepsilon, x_i\}$ heißt

$$K' = (p, x_j \cdots x_n, B_1 \cdots B_k A_2 \cdots A_l)$$

Folgekonfiguration von K (kurz: $K \vdash K'$), wenn $j = i + |u|$ ist.

Die reflexive, transitive Hülle von $\vdash$ bezeichnen wir wie üblich mit $\vdash^*$. Die von M **akzeptierte** oder **erkannte Sprache** ist dann

$$L(M) \quad = \quad \{x \in \Sigma^* \mid \exists\, p \in Z : (q_0, x, \#) \vdash^* (p, \varepsilon, \varepsilon)\}.$$

Ein Wort x wird also genau dann von M akzeptiert, wenn es eine Rechnung (Folge von Konfigurationen) von M bei Eingabe x gibt, bei der das gesamte Wort gelesen und der Keller geleert wird. (Man beachte, dass bei leerem Keller kein weiterer Übergang mehr möglich ist.)

Beispiel 31

Der PDA $M = (\{q_0, q_1\}, \{a, b\}, \{A, \#\}, \delta, q_0, \#)$ mit

$$\begin{array}{lllll} \delta: & q_0\varepsilon\# & \to & q_1 & (1) \\ & q_0 a\# & \to & q_0 A & (2) \\ & q_0 aA & \to & q_0 AA & (3) \\ & q_0 bA & \to & q_1 & (4) \\ & q_1 bA & \to & q_1 & (5) \end{array}$$

erkennt die Sprache $L(M) = \{a^n b^n \mid n \geq 0\}$. Das Wort $x = aabb$ wird beispielsweise durch folgende Rechnung akzeptiert:

$$(q_0, aabb, \#) \underset{(2)}{\vdash} (q_0, abb, A) \underset{(3)}{\vdash} (q_0, bb, AA) \underset{(4)}{\vdash} (q_1, b, A) \underset{(5)}{\vdash} (q_1, \varepsilon, \varepsilon).$$

Allgemein kann ein Wort $x = a^n b^n$ wie folgt akzeptiert werden:

$n = 0$: $(q_0, \varepsilon, \#) \underset{(1)}{\vdash} (q_1, \varepsilon, \varepsilon)$.

$n \geq 1$:

$$\begin{aligned}(q_0, a^n b^n, \#) &\underset{(2)}{\vdash} (q_0, a^{n-1} b^n, A) \underset{(3)}{\vdash^*} (q_0, b^n, A^n)\\ &\underset{(4)}{\vdash} (q_1, b^{n-1}, A^{n-1}) \underset{(5)}{\vdash^*} (q_1, \varepsilon, \varepsilon).\end{aligned}$$

Dies zeigt $\{a^n b^n \mid n \geq 0\} \subseteq L(M)$. Ist umgekehrt $x = x_1 \dots x_l \in L(M)$, so gibt es eine akzeptierende Rechnung

$$(q_0, x, \#) \vdash^* (p, \varepsilon, \varepsilon)$$

mit $p \in \{q_0, q_1\}$. Ausgehend von $(q_0, x, \#)$ sind nur die Anweisungen (1) und (2) anwendbar. Im ersten Fall muss $x = \varepsilon$ sein, da nach Ausführung von (1) der Keller leer ist.

Im zweiten Fall muss die Rechnung ebenfalls im Zustand $p = q_1$ enden, da der Keller nur bei Übergang in den Zustand q_1 geleert werden kann. Da kein Wechsel von q_1 nach q_0 möglich ist, findet genau ein Zustandswechsel (mittels (4)) statt. Bis dahin können nur a's gelesen werden und für jedes gelesene a wird ein A eingekellert. Ist n die Anzahl der gelesenen a's, so muss die Rechnung bis dahin also wie folgt aussehen:

$$\begin{aligned}(q_0, x_1 \dots x_l, \#) &\underset{(2)}{\vdash} (q_0, x_2 \dots x_l, A)\\ &\underset{(3)}{\vdash}^{n-1} (q_0, x_{n+1} \cdots x_l, A^n)\\ &\underset{(4)}{\vdash} (q_1, x_{n+2} \cdots x_l, A^{n-1})\end{aligned}$$

mit $x_1 = \cdots = x_n = a$ und $x_{n+1} = b$. Um den Keller leeren zu können, muss M jetzt noch genau $n - 1$ weitere b's lesen. Daraus ergibt sich, dass $l = 2n$ und $x_{n+2} = \cdots = x_{2n} = b$ ist, x also auch in diesem Fall die Form $a^n b^n$ hat.

Als nächstes wollen wir zeigen, dass eine Sprache genau dann kontextfrei ist, wenn sie von einem PDA erkannt wird.

Satz 32

CFL $= \{L(M) \mid M$ ist ein PDA$\}$.

Beweis: Wir zeigen zuerst, dass jede von einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ erzeugte Sprache L auch von einem PDA erkannt wird. Hierzu konstruieren wir den PDA $M = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$, wobei

$$\begin{aligned}\delta(q, \varepsilon, A) &= \{(q, \alpha) \mid A \rightarrow_G \alpha\} \text{ für jedes } A \in V \text{ und}\\ \delta(q, a, a) &= \{(q, \varepsilon)\} \text{ für jedes } a \in \Sigma \text{ ist.}\end{aligned}$$

Dann gilt

$$
\begin{aligned}
x \in L(G) \quad &\Leftrightarrow \quad \text{es gibt eine Linksableitung} \\
&\qquad S \Rightarrow \alpha_0 \Rightarrow^* x_1\alpha_1 \Rightarrow^* x_1x_2\alpha_2 \Rightarrow^* \cdots \Rightarrow^* x_1 \cdots x_{n-1}\alpha_{n-1} \Rightarrow^* x \\
&\Leftrightarrow \quad \text{es gibt eine Konfigurationenfolge} \\
&\qquad (q, x, S) \vdash (q, x, \alpha_0) \vdash^* (q, x_2 \cdots x_n, \alpha_1) \vdash^* (q, x_3 \cdots x_n, \alpha_2) \vdash^* \\
&\qquad \cdots \vdash^* (q, x_n, \alpha_n) \vdash^* (q, \varepsilon, \varepsilon) \\
&\Leftrightarrow \quad x \in L(M).
\end{aligned}
$$

Für die entgegengesetzte Inklusion sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ ein PDA und sei $L = L(M)$. Wir konstruieren eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit $L(G) = L$:

$$
\begin{aligned}
V &= \{S\} \cup \{X_{qAp} \mid q, p \in Z, A \in \Gamma\}, \\
P &= \{S \to X_{q_0\#p} \mid p \in Z\} \cup \{X_{p_0Ap_{k+1}} \to uX_{p_1B_1p_2}X_{p_2B_2p_3} \cdots X_{p_kB_kp_{k+1}} \mid \\
&\qquad (p_1, B_1 \cdots B_k) \in \delta(p_0, u, A) \text{ für ein } k \geq 0 \text{ und beliebige } p_2, \ldots, p_{k+1} \in Z\}.
\end{aligned}
$$

Die Gleichheit $L(G) = L(M)$ folgt nun unmittelbar aus der Äquivalenz

$$X_{qAp} \Rightarrow^m x \iff (q, x, A) \vdash^m (p, \varepsilon, \varepsilon),$$

deren Beweis wir gleich nachholen:

$$
\begin{aligned}
x \in L(M) \quad &\Leftrightarrow \quad (q_0, x, \#) \vdash^* (p, \varepsilon, \varepsilon) \text{ für ein } p \in Z \\
&\Leftrightarrow \quad S \Rightarrow X_{q_0\#p} \Rightarrow^* x \\
&\Leftrightarrow \quad x \in L(G).
\end{aligned}
$$

Beweis der Äquivalenz. Wir führen den Beweis durch Induktion über m:

$m = 0$: Da weder $X_{qAp} \Rightarrow^0 x$ (also $X_{qAp} = x$) noch $(q, x, A) \vdash^0 (p, \varepsilon, \varepsilon)$ (also $(q, x, A) = (p, \varepsilon, \varepsilon)$) gilt, sind beide Bedingungen falsch, also ist die Äquivalenz erfüllt.

$m \rightsquigarrow m + 1$: Seien $q, p \in Z$, $x \in \Sigma^*$ und $A \in \Gamma$ gegeben. Wir müssen zeigen, dass genau dann eine (Links-)Ableitung $X_{qAp} \Rightarrow^{m+1} x$ existiert, wenn es eine Rechnung $(q, x, A) \vdash^{m+1} (p, \varepsilon, \varepsilon)$ gibt.

Sei also $X_{qAp} = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_{m+1} = x$ eine Linksableitung von x aus X_{qAp} der Länge $m + 1$. Da α_1 aus X_{qAp} durch Anwendung einer Regel aus P entsteht, hat α_1 die Form $\alpha_1 = uX_{p_1B_1p_2} \cdots X_{p_kB_kp_{k+1}}$ für ein $k \geq 0$, wobei $u \in \Sigma \cup \{\varepsilon\}$, $(p_1, B_1 \cdots B_k) \in \delta(q, u, A)$, $p_{k+1} = p$ und $p_1, \ldots, p_k \in Z$ sind. Das Wort x muss sich daher in $k + 1$ Teilwörter $x = x_0x_1 \cdots x_k$ mit $x_0 = u$ zerlegen lassen, so dass jedes Teilwort x_i ($i = 1, \ldots, k$) in genau m_i Ableitungsschritten aus der Variablen $X_{p_iB_ip_{i+1}}$ ableitbar ist, wobei $m_1 + \cdots + m_k = m$ ist. Nach Induktionsvoraussetzung gibt es also Rechnungen

$$(p_i, x_i, B_i) \vdash^{m_i} (p_{i+1}, \varepsilon, \varepsilon), i = 1, \ldots, k$$

aus denen wir die gesuchte Rechnung der Länge $m+1$ wie folgt erhalten:

$$\begin{array}{lll} (q,x,A) & \vdash & (p_1, x_1\cdots x_k, B_1\cdots B_k) \\ & \vdash^{m_1} & (p_2, x_2\cdots x_k, B_2\cdots B_k) \\ & \vdots & \\ & \vdash^{m_{k-1}} & (p_k, x_k, B_k) \\ & \vdash^{m_k} & (p, \varepsilon, \varepsilon). \end{array}$$

Ist nun umgekehrt $(q,x,A) \vdash^{m+1} (p,\varepsilon,\varepsilon)$ eine Rechnung der Länge $m+1$, so sei $quA \to p_1B_1\cdots B_k$ die Anweisung, die im ersten Rechenschritt befolgt wird:

$$(q,x,A) \vdash (p_1, x', B_1\cdots B_k) \vdash^m (p,\varepsilon,\varepsilon),$$

wobei $x = ux'$ ist. Für $i = 2,\ldots,k$ sei p_i der Zustand, den M annimmt, wenn B_i oberstes Kellerzeichen wird und x_i sei das Teilwort von x', das M zwischen den Besuchen der Zustände p_i und p_{i+1} verarbeitet (wobei $p_{k+1} = p$ ist). Dann enthält P einerseits die Regel $X_{qAp_{k+1}} \to uX_{p_1B_1p_2}\cdots X_{p_kB_kp_{k+1}}$ und andererseits gilt

$$(p_i, x_i, B_i) \vdash^{m_i} (p_{i+1}, \varepsilon, \varepsilon), i = 1,\ldots,k$$

für Zahlen $m_i \le m$ mit $m_1 + \cdots + m_k = m$. Nach Induktionsvoraussetzung gibt es daher Ableitungen

$$X_{p_iB_ip_{i+1}} \Rightarrow^{m_i} x_i, i = 1,\ldots,k$$

die wir zu der gesuchten Ableitung von x zusammensetzen können:

$$\begin{array}{lll} X_{qAp_{k+1}} & \Rightarrow & uX_{p_1B_1p_2}\cdots X_{p_kB_kp_{k+1}} \\ & \Rightarrow^{m_1} & ux_1X_{p_2B_2p_3}\cdots X_{p_kB_kp_{k+1}} \\ & \vdots & \\ & \Rightarrow^{m_{k-1}} & ux_1\cdots x_{k-1}X_{p_kB_kp_{k+1}} \\ & \Rightarrow^{m_k} & ux_1\cdots x_k = x. \end{array}$$

■

Beispiel 33

Sei $G = (\{S\},\{a\},P,S)$ mit

$$\begin{array}{llc} P: & S \to SSS & (1) \\ & S \to a & (2) \end{array}$$

die Grammatik aus dem vorigen Beispiel. Der zugehörige Kellerautomat besitzt dann folgende Anweisungen:

$$\begin{array}{lcclc} \delta: & q\varepsilon S & \to & qSSS & (1') \\ & q\varepsilon S & \to & qa & (2') \\ & qaa & \to & q\varepsilon & (3) \end{array}$$

Der Linksableitung

$$S \underset{(1)}{\Rightarrow} SSS \underset{(2)}{\Rightarrow} aSS \underset{(2)}{\Rightarrow} aaS \underset{(2)}{\Rightarrow} aaa$$

entspricht dann die Rechnung

$$\begin{aligned}(q,aaa,S) &\underset{(1')}{\vdash} (q,aaa,SSS) \underset{(2')}{\vdash} (q,aaa,aSS) \underset{(3)}{\vdash} (q,aa,SS)\\ &\underset{(2')}{\vdash} (q,aa,aS) \underset{(3)}{\vdash} (q,a,S) \underset{(2')}{\vdash} (q,a,a) \underset{(3)}{\vdash} (q,\varepsilon,\varepsilon).\end{aligned}$$

Beispiel 34
Ist umgekehrt $M = (\{q_0, q_1\}, \{a, b\}, \{A, \#\}, \delta, q_0, \#)$ mit

$$\begin{array}{llcll} \delta: & q_0\varepsilon\# & \to & q_1 & (1)\\ & q_0a\# & \to & q_0A & (2)\\ & q_0aA & \to & q_0AA & (3)\\ & q_0bA & \to & q_1 & (4)\\ & q_1bA & \to & q_1 & (5)\end{array}$$

der PDA aus Beispiel 31, so erhalten wir daraus die Grammatik $G = (V, \Sigma, P, S)$ mit

$$V = \{S, X_{q_0\#q_0}, X_{q_0\#q_1}, X_{q_1\#q_0}, X_{q_1\#q_1}, X_{q_0Aq_0}, X_{q_0Aq_1}, X_{q_1Aq_0}, X_{q_1Aq_1}\}$$

und

$$\begin{array}{lll} P: & S \to X_{q_0\#q_0}, X_{q_0\#q_1} & (0,0')\\ & X_{q_0\#q_1} \to \varepsilon & (1')\\ & X_{q_0\#q_0} \to aX_{q_0Aq_0} & (2')\\ & X_{q_0\#q_1} \to aX_{q_0Aq_1} & (2'')\\ & X_{q_0Aq_0} \to aX_{q_0Aq_0}X_{q_0Aq_0}, aX_{q_0Aq_1}X_{q_1Aq_0} & (3',3'')\\ & X_{q_0Aq_1} \to aX_{q_0Aq_0}X_{q_0Aq_1}, aX_{q_0Aq_1}X_{q_1Aq_1} & (3''',3'''')\\ & X_{q_0Aq_1} \to b & (4')\\ & X_{q_1Aq_1} \to b & (5').\end{array}$$

Der Rechnung

$$(q_0, aabb, \#) \underset{(2)}{\vdash} (q_0, abb, A) \underset{(3)}{\vdash} (q_0, bb, AA) \underset{(4)}{\vdash} (q_1, b, A) \underset{(5)}{\vdash} (q_1, \varepsilon, \varepsilon).$$

entspricht dann die Linksableitung

$$S \underset{(0')}{\Rightarrow} X_{q_0\#q_1} \underset{(2'')}{\Rightarrow} aX_{q_0Aq_1} \underset{(3'''')}{\Rightarrow} aaX_{q_0Aq_1}X_{q_1Aq_1} \underset{(4')}{\Rightarrow} aabX_{q_1Aq_1} \underset{(5')}{\Rightarrow} aabb.$$

2.2 Chomsky-Normalform Grammatiken

Grammatiken in Chomsky-Normalform bilden einerseits die Basis für eine effiziente Lösung des Entscheidungsproblems für kontextfreie Sprachen. daneben ermöglichen sie auch den Beweis des Pumping-Lemmas (für kontextfreie Sprachen), mit dem sich viele Sprachen als nicht kontextfrei nachweisen lassen.

Definition 35 (Chomsky-Normalform)
Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ist in **Chomsky-Normalform** (kurz CNF), falls alle Regeln die Form $A \to BC$ oder $A \to a$ haben.

Zuerst zeigen wir, dass ε-Produktionen (Produktionen der Form $A \to \varepsilon$) überflüssig sind (sofern das leere Wort nicht zur Sprache gehört).

Satz 36
Sei $L \in \mathrm{CFL}$. Dann gibt es für die Sprache $L \setminus \{\varepsilon\}$ eine kontextfreie Grammatik ohne ε-Produktionen.

Beweis: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik für L.

1. Schritt: Bestimme die Menge V' aller Variablen $A \in V$ mit $A \Rightarrow^* \varepsilon$ durch folgenden Algorithmus:

Eingabe: $G = (V, \Sigma, P, S)$
  $V'' \leftarrow \{A \in V \mid A \to_G \varepsilon\}$
  $V' \leftarrow \emptyset$
  while $V' \neq V''$ **do**
    $V' \leftarrow V''$
    $V'' \leftarrow V' \cup \{A \in V \mid \exists B_1, \ldots, B_k \in V' : A \to_G B_1 \cdots B_k\}$
  end
Ausgabe: V'

2. Schritt: Entferne alle ε-Produktionen aus P und nehme für jede übrig gebliebene Produktion $A \to \alpha$ die Produktionen $A \to \alpha'$ hinzu, wobei $\alpha' \in (V \cup \Sigma)^+$ aus α durch Streichen von einer oder mehreren Variablen $A \in V'$ entsteht.

■

Beispiel 37
Betrachte die Grammatik $G = (\{S, A, B, C, D, E\}, \{a, b, c\}, P, S)$ mit

$$
\begin{aligned}
P: \quad & S \to aB, bA, C \\
& A \to aS, bAA \\
& B \to bS, aBB \\
& C \to \varepsilon, S, D, cC \\
& D \to E \\
& E \to abc.
\end{aligned}
$$

1. Schritt: Berechnung von V':

V'	V''
$\emptyset$	$\{C\}$
$\{C\}$	$\{C, S\}$
$\{C, S\}$	$\{C, S\}$

2. Schritt: Entferne die Regel $C \to \varepsilon$ und füge für $A \to aS$ die Regel $A \to a$, für $B \to bS$ die Regel $B \to b$ und für $C \to cC$ die Regel $C \to c$ zu P hinzu:

$$P: \begin{array}{l} S \to aB, bA, C \\ A \to a, aS, bAA \\ B \to b, bS, aBB \\ C \to c, S, D, cC \\ D \to E \\ E \to abc. \end{array}$$

Korollar 38

CFL $\subseteq$ CSL.

Beweis: Sei $L \in$ CFL. Nach vorigem Satz gibt es für die Sprache $L \setminus \{\varepsilon\}$ eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ohne ε-Produktionen. Da G dann auch kontextsensitiv ist, folgt im Fall $\varepsilon \notin L$ direkt $L \in$ CSL. Im Fall $\varepsilon \in L$ ist $G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \to S, \varepsilon\}, S')$ eine kontextsensitive Grammatik für L. ■

Der nächste Satz zeigt, dass auch auf Variablenumbenennungen (Produktionen der Form $A \to B$) verzichtet werden kann.

Satz 39

Für jede Sprache $L \in$ CFL gibt es eine kontextfreie Grammatik ohne Variablenumbenennungen.

Beweis: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik für L.

1. Schritt: Entfernung von Zyklen.
   Falls Variablen $A_1, \ldots, A_k \in V$ existieren mit $A_1 \to A_2 \to \cdots \to A_k \to A_1$, so entferne diese Regeln aus P und ersetze alle übrigen Vorkommen dieser Variablen durch A_1 (falls sich unter den A_i die Startvariable befindet, sei A_1 die neue Startvariable).
2. Schritt: Solange noch Regeln der Form $A \to B$ in P existieren, wähle eine solche Regel, für die keine Variable $D \in V$ existiert mit $B \to D$. Entferne $A \to B$ aus P und füge für jede in P vorkommende Regel $B \to \alpha$ die Regel $A \to \alpha$ zu P hinzu.

■

Beispiel 40 (Fortsetzung von Beispiel 37)

1. Schritt: Entfernung des Zyklus' $S \to C \to S$:
   Entferne die beiden Regeln $S \to C$ und $C \to S$ und ersetze alle Vorkommen von C durch S:

$$P: \begin{array}{l} S \to c, D, aB, bA, cS \\ A \to a, aS, bAA \\ B \to b, bS, aBB \\ D \to E \\ E \to abc. \end{array}$$

2. Schritt: Entferne die Regel $D \to E$ und füge für die Regel $E \to abc$ die Regel $D \to abc$ hinzu. Entferne dann auch die Regel $S \to D$ und füge wegen $D \to abc$ die neue Regel $S \to abc$ hinzu (da die Variablen D und E nun nicht mehr auf der rechten Seite einer Regel vorkommen, können wir die beiden Regeln $D \to abc$ und $E \to abc$ weglassen):

$$\begin{aligned} P: \quad & S \to c, abc, aB, bA, cS \\ & A \to a, aS, bAA \\ & B \to b, bS, aBB. \end{aligned}$$

Nun ist es nicht mehr schwierig, die Chomsky-Normalform herzustellen.

Satz 41

Sei $L \in$ CFL. Dann gibt es für die Sprache $L \setminus \{\varepsilon\}$ eine Grammatik in Chomsky-Normalform.

Beweis: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik für $L \setminus \{\varepsilon\}$. Aufgrund der beiden vorherigen Sätze können wir annehmen, dass in P keine ε-Produktionen und keine Variablenumbenennungen vorkommen. Aus G lässt sich nun leicht eine CNF-Grammatik erhalten.

1. Schritt: Füge für jedes $a \in \Sigma$ eine neue Variable X_a zu V und eine neue Regel $X_a \to a$ zu P hinzu. Ersetze alle Vorkommen von a, ausser wenn a alleine auf der rechten Seite einer Regel vorkommt, durch X_a.
2. Schritt: Ersetze jede Regel der Form $A \to B_1 \cdots B_k$ mit $k \geq 3$ durch die $k-1$ neuen Regeln $A \to B_1 C_1$, $C_1 \to B_2 C_2, \ldots, C_{k-3} \to B_{k-2} C_{k-2}$ und $C_{k-2} \to B_{k-1} B_k$, wobei $C_1, \ldots, C_{k-2}$ neue Variablen sind.

■

Beispiel 42 (Fortsetzung von Beispiel 40)

1. Schritt: Ersetze die Zeichen a, b und c durch X_a, X_b und X_c (ausser wenn sie alleine auf der rechten Seite einer Regel vorkommen) und ergänze die Regeln $X_a \to a$, $X_b \to b$ und $X_c \to c$.

$$\begin{aligned} P: \quad & S \to c, X_a X_b X_c, X_a B, X_b A, X_c S \\ & A \to a, X_a S, X_b AA \\ & B \to b, X_b S, X_a BB \\ & X_a \to a;\ X_b \to b;\ X_c \to c. \end{aligned}$$

2. Schritt: Ersetze die Regeln $S \to X_a X_b X_c$, $A \to X_b AA$ und $B \to X_a BB$ durch die Regeln $S \to X_a X'$, $X' \to X_b X_c$, $A \to X_b A'$, $A' \to AA$ und $B \to X_a B'$, $B' \to BB$.

$$\begin{aligned} P: \quad & S \to c, X_a X', X_a B, X_b A, X_c S \\ & A \to a, X_a S, X_b A' \\ & B \to b, X_b S, X_a B' \\ & X_a \to a;\ X_b \to b;\ X_c \to c \\ & X' \to X_b X_c;\ A' \to AA;\ B' \to BB. \end{aligned}$$

2.3 Syntaxbäume und Linksableitungen

Es ist leicht, eine kontextfreie Grammatik für die Sprache $L = \{a^n b^n \mid n \geq 0\}$ anzugeben: Es ist $L = L(G)$ für $G = (\{S\}, \{a, b\}, \{S \to aSb, \varepsilon\}, S)$. Das Wort $aabb$ lässt sich beispielsweise wie folgt ableiten:

$$\underline{S} \Rightarrow a\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb$$

Diese Grammatik ist sogar eindeutig, da es für jedes Wort $x \in L(G)$ genau eine Ableitung gibt. Zwar hat das Wort ab in der Grammatik $G' = (\{S\}, \{a, b, c\}, \{S \to aSbS, \varepsilon\}, S)$, welche die Sprache $L(G) = L^*$ erzeugt, zwei verschiedene Ableitungen

$$\underline{S} \Rightarrow aSb\underline{S} \Rightarrow a\underline{S}b \Rightarrow ab \text{ und } \underline{S} \Rightarrow a\underline{S}bS \Rightarrow ab\underline{S} \Rightarrow ab,$$

diese unterscheiden sich jedoch nur darin, in welcher Reihenfolge die Regeln zur Anwendung kommen. Im wesentlichen (d.h. bis auf die Regelanwendungsreihenfolge) hat auch in dieser Grammatik jedes Wort eine eindeutige Ableitung.

Definition 43 (Linksableitung)
Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Eine Ableitung

$$\underline{S} = l_0\underline{u_0}r_0 \Rightarrow_G \cdots \Rightarrow_G l_m\underline{u_m}r_m$$

heißt **Linksableitung**, wenn u_i für $i = 0, \ldots, m-1$ die am weitesten links stehende Variable in der Satzform $\alpha_i = l_i u_i r_i$ ist.

Es ist klar, dass jede Ableitung von x in eindeutiger Weise in eine Linksableitung umgewandelt werden kann, indem man die Reihenfolge der Anwendungen der Produktionen entsprechend verändert.

Beispiel 44
Sei $G = (\{S\}, \{a\}, \{S \to SSS, a\}, S)$. Dann ist

$$\underline{S} \Rightarrow S\underline{S}S \Rightarrow \underline{S}aS \Rightarrow aa\underline{S} \Rightarrow aaa$$

eine Ableitung des Wortes $x = aaa$. Die zugehörige Linksableitung ist

$$S \Rightarrow \underline{S}SS \Rightarrow a\underline{S}S \Rightarrow aa\underline{S} \Rightarrow aaa.$$

Definition 45 (eindeutige Grammatik)
Eine kontextfreie Grammatik G heißt **eindeutig**, falls es für jedes Wort $x \in L(G)$ genau eine Linksableitung gibt.

(Links)ableitungen lassen sich sehr gut durch Syntaxbäume veranschaulichen.

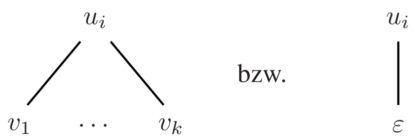
Definition 46 (Syntaxbaum)
Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik und sei

$$\underline{S} = l_0\underline{u_0}r_0 \Rightarrow_G \cdots \Rightarrow_G l_m\underline{u_m}r_m = x$$

eine Ableitung von $x \in \Sigma^*$. Dann ordnen wir dieser Ableitung den **Syntaxbaum** T_m zu, wobei die Syntaxbäume $T_0, \ldots, T_m$ induktiv wie folgt definiert sind:

T_0: Der Baum T_0 besteht aus einem einzigen Knoten, der mit S markiert ist.

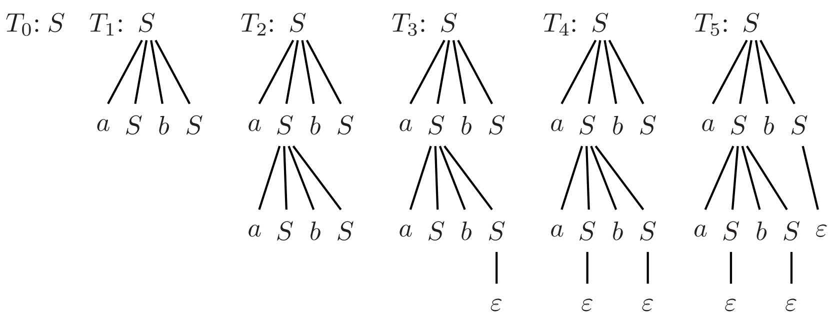
$T_{i+1}, i \geq 0$: Sei $u_i \to v_1 \cdots v_k$ mit $v_j \in \Sigma \cup V$ (bzw. $u_i \to \varepsilon$) die Regel, die im $i+1$-ten Ableitungsschritt $l_i\underline{u_i}r_i \Rightarrow l_{i+1}u_{i+1}r_{i+1}$ angewandt wird. Dann ensteht der Baum T_{i+1} aus T_i durch Ersetzen des zur Variablen u_i gehörigen Blattes durch den Unterbaum



Beispiel 47
Betrachte die Grammatik $G = (\{S\}, \{a, b, c\}, \{S \to aSbS, \varepsilon\}, S)$ und die Ableitung

$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSb\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$$

Die zugehörigen Syntaxbäume sind dann



Verschiedene Ableitungen können durchaus auf denselben Syntaxbaum führen, sofern sie sich lediglich in der Reihenfolge, in der die Regeln zur Anwendung kommen, unterscheiden. Unterschiedliche Linksableitungen führen dagegen immer auf unterschiedliche Syntaxbäume. Deshalb entspricht jedem Syntaxbaum eindeutig eine Linksableitung und umgekehrt.

2.4 Der CYK-Algorithmus

Als erste Anwendung der Chomsky-Normalform stellen wir einen effizienten Entscheidungsalgorithmus für kontextfreie Sprachen vor.

Satz 48
Jede kontextfreie Sprache L ist in Polynomialzeit entscheidbar.

Beweis: Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik in Chomsky-Normalform. Wir verwenden den nach seinen Autoren Cocke, Younger und Kasami benannten CYK-Algorithmus. Für jede Eingabe $x = x_1 \cdots x_n$ sei

$$V_{i,j} = \{A \in V \mid A \Rightarrow^* x_i \cdots x_{i+j-1}\}.$$

die Menge aller Variablen, aus denen das mit x_i beginnende Teilwort von x der Länge j ableitbar ist. Dann gilt

$$x \in L(G) \Leftrightarrow S \in V_{1,n}.$$

Für $j = 1, \ldots, n$ können die Mengen $V_{i,j}$, $i = 1, \ldots, n - j + 1$, wie folgt berechnet werden:

$$V_{i,j} = \begin{cases} \{A \in V \mid A \rightarrow_G x_i\}, & j = 1 \\ \bigcup_{1 \leq k \leq j-1} \{A \in V \mid \exists B \in V_{i,k} \exists C \in V_{i+k,j-k} : A \rightarrow_G BC\}, & j > 1. \end{cases}$$

Um $V_{i,j}$ zu bestimmen, testet man also für jede Regel der Form $A \rightarrow BC$, ob es ein $k \in \{1, \ldots, j-1\}$ gibt, so dass B zu $V_{i,k}$ und C zu $V_{i+k,j-k}$ gehört, und fügt im positiven Fall A zu $V_{i,j}$ hinzu.

Die polynomiale Zeitschranke ergibt sich aus der Tatsache, dass nur $n(n+1)/2$ Mengen $V_{i,j} \subseteq V$ zu bestimmen sind und hierfür jeweils nur eine konstante Anzahl von Tests für jede Zahl $k \in \{1, \ldots, j-1\}$ nötig sind, was auf eine Gesamtlaufzeit von $O(n^3)$ führt. ■

Da eine gegebene kontextfreie Grammatik in Polynomialzeit in CNF gebracht werden kann, folgt aus dem obigen Beweis, dass sogar das **Wortproblem für kontextfreie Grammatiken** in Polynomialzeit entscheidbar ist: Es gibt einen Polynomialzeitalgorithmus, der für eine gegebene kontextfreie Grammatik G und ein Wort x entscheidet, ob x zu $L(G)$ gehört.

Beispiel 49
Für die CNF-Grammatik aus dem vorigen Beispiel mit den Produktionen

$$\begin{array}{ll} P: & S \rightarrow c, X_aX', X_aB, X_bA, X_cS \\ & A \rightarrow a, X_aS, X_bA' \\ & B \rightarrow b, X_bS, X_aB' \\ & X_a \rightarrow a;\ X_b \rightarrow b;\ X_c \rightarrow c \\ & X' \rightarrow X_bX_c;\ A' \rightarrow AA;\ B' \rightarrow BB. \end{array}$$

und für das Wort $x = aaa$ berechnet der CYK-Algorithmus die folgenden Variablenmengen $V_{i,j}$:

j	x_i		
	$x_1 = a$	$x_2 = a$	$x_3 = a$
1	$\{A, X_a\}$	$\{A, X_a\}$	$\{A, X_a\}$
2	$\{A'\}$	$\{A'\}$	
3	$\emptyset$		

Wegen $S \notin V_{1,3}$ ist $x \notin L(G)$. Dagegen gehört das Wort $y = aababb$ zu $L(G)$:

j	y_i					
	$y_1 = a$	$y_2 = a$	$y_3 = b$	$y_4 = a$	$y_5 = b$	$y_6 = b$
1	$\{A, X_a\}$	$\{A, X_a\}$	$\{B, X_b\}$	$\{A, X_a\}$	$\{B, X_b\}$	$\{B, X_b\}$
2	$\{A'\}$	$\{S\}$	$\{S\}$	$\{S\}$	$\{B'\}$	
3	$\{A\}$	$\{A\}$	$\{B\}$	$\{B\}$		
4	$\{A'\}$	$\{S\}$	$\{B'\}$			
5	$\{A\}$	$\{B\}$				
6	$\{S\}$					

2.5 Das Pumping-Lemma

Als zweite Anwendung der Chomsky-Normalform beweisen wir nun das Pumping-Lemma für kontextfreie Sprachen. Eine Grammatik G in Chomsky-Normalform hat die spezielle Eigenschaft, dass jeder Knoten in einem Syntaxbaum T von G höchstens zwei Nachfolger besitzt (d.h. T ist ein Binärbaum). Für den Beweis des Pumping-Lemmas benötigen wir noch die folgende Abschätzung für die maximale Pfadlänge von T in Abhängigkeit von seiner Blätterzahl.

Lemma 50 ()
In jedem Binärbaum mit $\geq 2^{k-1} + 1$ Blättern existiert ein Pfad von der Wurzel zu einem Blatt mit einer Länge $\geq k$.

Beweis: Wir führen den Beweis durch Induktion über k.

$k = 0$: Ein Pfad der Länge 0 existiert in jedem Baum.

$k \leadsto k + 1$: Sei B ein Binärbaum mit $\geq 2^k + 1$ Blättern. Da an B's Wurzel maximal zwei Teilbäume hängen, muss einer dieser Teilbäume $\geq 2^{k-1} + 1$ Blätter besitzen. In diesem Teilbaum existiert nach IV ein Pfad der Länge $\geq k$. Nach Hinzufügen der Wurzel von B hat er also eine Länge von mindestens $k + 1$.

■

Korollar 51
Ein Binärbaum B, der nur Pfade der Länge $\leq k$ besitzt, hat $\leq 2^k$ Blätter.

Beweis: Die Annahme, dass B zwar nur Pfade der Länge $\leq k$, aber mindestens $2^k + 1$ Blätter besitzt, führt nach obigem Lemma auf einen Widerspruch. ■

Satz 52 (Pumping-Lemma für kontextfreie Sprachen)
Zu jeder kontextfreien Sprache L gibt es eine Zahl l, so dass sich alle Wörter $z \in L$ mit $|z| \geq l$ in $z = uvwxy$ zerlegen lassen mit

1. $vx \neq \varepsilon$,
2. $|vwx| \leq l$ und
3. $uv^iwx^iy \in L$ für alle $i \geq 0$.

Beweis: Sei $L \in$ CFL. Es genügt, die Behauptung für $L \setminus \{\varepsilon\}$ zu zeigen. Sei also $G = (V, \Sigma, P, S)$ eine CNF-Grammatik für $L \setminus \{\varepsilon\}$. Dann setzen wir $l = 2^k$, wobei $k = \|V\|$ ist. Ist nun $z = z_1 \cdots z_n \in L$ mit $n \geq l$, so existiert in G eine Ableitung

$$S = \alpha_0 \Rightarrow \alpha_1 \cdots \Rightarrow \alpha_m = z.$$

Da G in CNF ist, werden hierbei genau $n - 1$ Regeln der Form $A \to BC$ und genau n Regeln der Form $A \to a$ angewandt, die Länge der Ableitung ist also $m = 2n - 1$.

Dabei können wir annehmen, dass zuerst die Regeln der Form $A \to BC$ und erst danach die Regeln der Form $A \to a$ benutzt werden. Dann hat der zu α_{n-1} gehörige Syntaxbaum T_{n-1} bereits $n \geq l = 2^k$ Blätter, so dass in T_{n-1} ein Pfad der Länge $\geq k$ existieren muss. Wählen wir in T_{n-1} einen von der Wurzel ausgehenden Pfad π maximaler Länge, so sind (mindestens) zwei der letzten $k + 1$ Knoten von π mit derselben Variablen A markiert.

Bezeichnen wir die beiden Unterbäume von T_m (kein Druckfehler; wir betrachten nun den Gesamtbaum $T_{\mathbf{m}}$), deren Wurzeln mit diesen Vorkommen von A markiert sind, mit U und U' (wobei U' der kleinere von beiden sein soll), so erhalten wir die gesuchte Zerlegung von z wie folgt:

- w ist das Teilwort von z, das von dem Teilbaum U' erzeugt wird.
- vwx ist das Teilwort von $z = uvwxy$, das von dem Teilbaum U erzeugt wird.

Da U mindestens ein Blatt mehr als U' besitzt, ist w ein echtes Teilwort von vwx. Daher kann vx nicht das leere Wort sein, Bedingung (1) ist also erfüllt. Aufgrund der Maximalität von π kann der Baum $U^* = U \cap T_{n-1}$ keine Pfade der Länge $> k$ enthalten. Die Anzahl der Blätter von U^*, welche mit der Anzahl der Blätter von U und somit mit der Länge von vwx übereinstimmt, ist also $\leq 2^k$, womit auch Bedingung (2) gezeigt ist. Für den Nachweis von Bedingung (3) konstruieren wir induktiv eine Folge von Syntaxbäumen B^i für die Wörter uv^iwx^iy:

- B^0 entsteht aus T_m, indem wir U durch U' ersetzen.
- B^{i+1} entsteht aus B^i, indem wir U' durch U ersetzen.

■

Beispiel 53

Die Sprache $\{a^n b^n c^n \mid n \geq 0\}$ ist nicht kontextfrei. Für eine vorgegebene Zahl $l \geq 0$ hat nämlich $z = a^l b^l c^l$ die Länge $|z| = 3l \geq l$, dieses Wort lässt sich aber nicht pumpen: Für jede Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$ gehört $z' = uv^2wx^2y$ nicht zu L. Hierzu betrachten wir zwei Fälle.

1\. Fall: In vx kommt eines der drei Zeichen a, b, c nicht vor. Ist dies beispielsweise a, so gehört z' wegen

$$\#_a(z') = \#_a(z) = l = |z|/3 < |z'|/3,$$

nicht zu L.

2\. Fall: Dass in vx jedes der drei Zeichen vorkommt ist wegen Bedingung (2) ausgeschlossen.

Satz 54

Die Klasse CFL ist abgeschlossen unter

- Vereinigung
- Produkt
- Sternhülle,

aber nicht unter

- Durchschnitt und
- Komplement.

Beweis: Seien $L_1, L_2 \in$ CFL und seien $G_i = (V_i, \Sigma, P_i, S_i)$, $i = 1, 2$, zugehörige kontextfreie Grammatiken mit $V_1 \cap V_2 = \emptyset$. Dann gilt für die kontextfreien Grammatiken

$$\begin{aligned} G_3 &= (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S_2\}, S), \\ G_4 &= (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1S_2\}, S) \text{ und} \\ G_5 &= (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1S, \varepsilon\}, S), \end{aligned}$$

wobei S eine neue Variable ist:

$$\begin{aligned} L(G_3) &= L_1 \cup L_2, \\ L(G_4) &= L_1L_2 \text{ und} \\ L(G_5) &= L_1^*. \end{aligned}$$

Die beiden Sprachen $L_1 = \{a^i b^j c^j \mid i, j \geq 0\}$ und $L_2 = \{a^i b^i c^j \mid i, j \geq 0\}$ sind kontextfrei, nicht jedoch $L_1 \cap L_2$, also ist CFL nicht unter Durchschnitt abgeschlossen. Da CFL unter Vereinigung abgeschlossen ist, kann CFL wegen de Morgan dann auch nicht unter Komplementbildung abgeschlossen sein. ■

2.6 Deterministisch kontextfreie Sprachen

Für die Informatik von besonderem Interesse sind kontextfreie Sprachen, die von einem deterministischen Kellerautomaten erkannt werden können.

Definition 55 (deterministischer Kellerautomat)

Ein **deterministischer Kellerautomat** (kurz: DPDA; *deterministic pushdown automaton*) wird durch ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ beschrieben. Dabei sind $Z, \Sigma, \Gamma, \delta, q_0, \#$ dieselben Komponenten wie bei einem PDA,

- $E \subseteq Z$ ist die Menge der **Endzustände**

und δ muss zusätzlich die Bedingung

$$\|\delta(q, a, A)\| + \|\delta(q, \varepsilon, A)\| \leq 1 \text{ für alle } (q, a, A) \in Z \times \Sigma \times \Gamma$$

erfüllen. (Äquivalent hierzu ist, dass die Relation $\vdash$ **rechtseindeutig** ist: Aus $K \vdash K'$ und $K \vdash K''$ folgt $K' = K''$).

Die von M **akzeptierte** oder **erkannte Sprache** ist

$$L(M) \quad = \quad \{x \in \Sigma^* \mid \exists p \in E, \alpha \in \Gamma^* : (q_0, x, \#) \vdash^* (p, \varepsilon, \alpha)\}.$$

Weiter sei

$$\text{DCFL} = \{L(M) \mid M \text{ ist ein DPDA}\}$$

(*deterministic context free languages*).

Beispiel 56

Der DPDA $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#, \{q_2\})$ mit

$$\begin{array}{rlcl}
\delta: & q_0 a\# & \to & q_0 A\# \\
& q_0 b\# & \to & q_0 B\# \\
& q_0 aA & \to & q_0 AA \\
& q_0 bA & \to & q_0 BA \\
& q_0 aB & \to & q_0 AB \\
& q_0 bB & \to & q_0 BB \\
& q_0 cA & \to & q_1 A \\
& q_0 cB & \to & q_1 B \\
& q_1 aA & \to & q_1 \\
& q_1 bB & \to & q_1 \\
& q_1 \varepsilon\# & \to & q_2
\end{array}$$

oder in Tabellenform

δ	$q_0, \#$	q_0, A	q_0, B	$q_1, \#$	q_1, A	q_1, B	$q_2, \#$	q_2, A	q_2, B
ε	–	–	–	q_2	–	–	–	–	–
a	$q_0 A\#$	$q_0 AA$	$q_0 AB$	–	q_1	–	–	–	–
b	$q_0 B\#$	$q_0 BA$	$q_0 BB$	–	–	q_1	–	–	–
c	–	$q_1 A$	$q_1 B$	–	–	–	–	–	–

ist deterministisch, da jeder Tabelleneintrag höchstens eine Anweisung enthält und im Fall eines nicht-leeren ε-Eintrages der Rest der Spalte leer ist. Offensichtlich erkennt M die Sprache $L(M) = \{xcx^R \mid x \in \{a,b\}^+\}$.

Im Gegensatz zur Klasse CFL der kontextfreien Sprachen ist DCFL zwar nicht unter Vereinigung, dafür jedoch unter Komplementbildung abgeschlossen. Versuchen wir allerdings, einfach End- und Nichtendzustände zu vertauschen, um einen Automaten $\overline{M}$ für $\overline{L(M)}$ zu erhalten (diese Methode hatten wir ja bei DFAs angewandt), so ergeben sich folgende Schwierigkeiten:

1. Problem: Es kann sein, dass ein DPDA M eine Eingabe $x \notin L(M)$ nicht zu Ende liest. Dann wird x von $\overline{M}$ auch nicht akzeptiert.

2. Problem: Es kann sein, dass M nach dem Lesen einer Eingabe $x \in L(M)$ einen oder mehrere ε-Übergänge ausführt, wobei sowohl End- als auch Nichtendzustände angenommen werden. Dann wird x von $\overline{M}$ ebenfalls akzeptiert.

Der nächste Satz zeigt, wie sich Problem 1 beseitigen lässt.

Satz 57

Jede Sprache $L \in$ DCFL wird auch von einem DPDA M' erkannt, welcher alle Eingaben zu Ende liest.

Beweis: Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA mit $L(M) = L$. Es gibt drei Möglichkeiten, warum M eine Eingabe x nicht zu Ende lesen könnte:

- M gerät in eine Konfiguration mit leerem Keller.
- M gerät in eine Konfiguration $(q, x_i \cdots x_n, A\gamma)$, in der wegen $\delta(q, x_i, A) = \delta(q, \varepsilon, A) = \emptyset$ keine Anweisung ausführbar ist.
- M führt eine unendliche Folge von ε-Anweisungen aus.

Dem ersten Punkt begegnen wir, indem wir ein neues Zeichen $\square$ auf dem Boden des Kellers plazieren (siehe unten, Anweisung a). Die zweite und dritte Möglichkeit beseitigen wir durch Einführen eines Fehlerzustands q_- (siehe Anweisungen b, c und d), wobei wir im Falle einer unendlichen Folge von ε-Anweisungen, bei der auch Endzustände angenommen werden, einen Umweg über den neuen Endzustand q_+ vorsehen (siehe Anweisungen e und f). Konkret transformieren wir M in den DPDA

$$M' = (Z \cup \{q_0', q_+, q_-\}, \Sigma, \underbrace{\Gamma \cup \{\square\}}_{\Gamma'}, \delta', q_0', \#, E \cup \{q_+\}),$$

wobei δ' die folgenden Anweisungen enthält:

a) $q_0'\varepsilon\# \to q_0\#\square$,

b) $qaA \to q_-A$, für alle $(q, a, A) \in Z \times \Sigma \times \Gamma'$ mit $A = \square$ oder $\delta(q, a, A) = \delta(q, \varepsilon, A) = \emptyset$,

c) $q_-aA \to q_-A$, für alle $a \in \Sigma$ und $A \in \Gamma'$,

d) $q\varepsilon A \to q_-A$, für alle $q \in Z$ und $A \in \Gamma$, so dass ausgehend von der Konfiguration (q, ε, A) unendlich viele ε-Übergänge ausgeführt werden, ohne dass dabei ein Endzustand angenommen wird,

e) $q\varepsilon A \to q_+ A$, für alle $q \in Z$ und $A \in \Gamma$, so dass ausgehend von der Konfiguration (q, ε, A) unendlich viele ε-Übergänge ausgeführt werden, wobei mindestens einmal ein Endzustand angenommen wird,

f) $q_+\varepsilon A \to q_- A$, für alle $A \in \Gamma$,

g) alle Anweisungen aus δ, soweit sie nicht durch Anweisungen vom Typ d oder e überschrieben wurden.

Aufgrund dieser Konstruktion kann man sich nun leicht davon überzeugen, dass M' ebenfalls L erkennt und alle Eingaben zu Ende liest. ■

Beispiel 58

Wenden wir diese Konstruktion auf den DPDA

$$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#, \{q_2\})$$

mit der Überführungsfunktion

δ	$q_0, \#$	q_0, A	q_0, B	$q_1, \#$	q_1, A	q_1, B	$q_2, \#$	q_2, A	q_2, B
ε	–	–	–	q_2	–	–	$q_2\#$	–	–
a	$q_0A\#$	q_0AA	q_0AB	–	q_1	–	–	–	–
b	$q_0B\#$	q_0BA	q_0BB	–	–	q_1	–	–	–
c	–	q_1A	q_1B	–	–	–	–	–	–

an, so erhalten wir den DPDA

$$M' = (\{q_0, q_1, q_2, q_0', q_+, q_-\}, \{a, b, c\}, \{A, B, \#, \Box\}, \delta', q_0', \#, \{q_2, q_+\})$$

mit der Überführungsfunktion

δ'	$q_0', \#$	q_0', A	q_0', B	$q_0', \Box$	$q_0, \#$	q_0, A	q_0, B	$q_0, \Box$
ε	$q_0\#\Box$	–	–	–	–	–	–	–
a	–	–	–	–	$q_0A\#$	q_0AA	q_0AB	$q_-\Box$
b	–	–	–	–	$q_0B\#$	q_0BA	q_0BB	$q_-\Box$
c	–	–	–	–	$q_-\#$	q_1A	q_1B	$q_-\Box$
Typ	a	–	–	–	g, b	g	g	b

	$q_1, \#$	q_1, A	q_1, B	$q_1, \Box$	$q_2, \#$	q_2, A	q_2, B	$q_2, \Box$
ε	q_2	–	–	–	$q_+\#$	–	–	–
a	–	q_1	q_-B	$q_-\Box$	–	q_-A	q_-B	$q_-\Box$
b	–	q_-A	q_1	$q_-\Box$	–	q_-A	q_-B	$q_-\Box$
c	–	q_-A	q_-B	$q_-\Box$	–	q_-A	q_-B	$q_-\Box$
Typ	g	g, b	g, b	b	e	b	b	b

	$q_-, \#$	q_-, A	q_-, B	$q_-, \Box$	$q_+, \#$	q_+, A	q_+, B	$q_+, \Box$
ε	–	–	–	–	$q_-\#$	q_-A	q_-B	–
a	$q_-\#$	q_-A	q_-B	$q_-\Box$	–	–	–	–
b	$q_-\#$	q_-A	q_-B	$q_-\Box$	–	–	–	–
c	$q_-\#$	q_-A	q_-B	$q_-\Box$	–	–	–	–
Typ	c	c	c	c	f	f	f	–

Satz 59

Die Klasse DCFL ist unter Komplementbildung abgeschlossen.

Beweis: Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA mit $L(M) = L$, der alle Eingaben zu Ende liest. Betrachte den DPDA

$$M' = (Z \times \{1,2,3\}, \Sigma, \Gamma, \delta', q_0', \#, Z \times \{3\}),$$

wobei

$$q_0' \quad = \quad \begin{cases} (q_0, 1), & q_0 \notin E, \\ (q_0, 2), & \text{sonst,} \end{cases}$$

und δ' die folgenden Anweisungen enthält.

- Für alle $p, q \in Z$, $A \in \Gamma$ und $\gamma \in \Gamma^*$ mit $q\varepsilon A \to_M p\gamma$:

$$\begin{array}{lcll} (q,1)\varepsilon A & \to & (p,1)\gamma, & \text{falls } p \notin E, \\ (q,1)\varepsilon A & \to & (p,2)\gamma, & \text{falls } p \in E \text{ und} \\ (q,2)\varepsilon A & \to & (p,2)\gamma. & \end{array}$$

- Für alle $p, q \in Z$, $a \in \Sigma$, $A \in \Gamma$ und $\gamma \in \Gamma^*$ mit $qaA \to_M p\gamma$:

$$\begin{array}{lcll} (q,1)\varepsilon A & \to & (q,3)A, & \\ (q,2)aA & \to & (p,1)\gamma, & \text{falls } p \notin E, \\ (q,2)aA & \to & (p,2)\gamma, & \text{falls } p \in E, \\ (q,3)aA & \to & (p,1)\gamma, & \text{falls } p \notin E \text{ und} \\ (q,3)aA & \to & (p,2)\gamma, & \text{falls } p \in E. \end{array}$$

Der DPDA M' speichert also in seinem Zustand (q, i) die Information, ob seit dem Lesen des letzten Eingabesymbols nur Zustände in $Z - E$ (in diesem Fall ist $i = 1$) oder bereits ein Zustand in E (in diesem Fall ist $i = 2$) angenommen wurde. Nur im ersten Fall wird vor dem Lesen des nächsten Eingabesymbols ein Umweg über einen Endzustand der Form $(q, 3)$ gemacht. Auf Grund dieser Konstruktion ist nun klar, dass M' tatsächlich das Komplement von L erkennt. ∎

Beispiel 60

Führt ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ bei Eingabe $x = a$ beispielsweise die Rechnung

$$(q_0, a, \#) \vdash (p_0, \varepsilon, \gamma_0) \vdash (p_1, \varepsilon, \gamma_1)$$

aus, so würde diese im Fall $q_0, p_1 \in E$ und $p_0 \notin E$ von M' durch die Rechnung

$$((q_0, 2), a, \#) \vdash ((p_0, 1), \varepsilon, \gamma_0) \vdash ((p_1, 2), \varepsilon, \gamma_1)$$

simuliert. Dagegen würde M' im Fall $q_0 \in E$ und $p_0, p_1 \notin E$ die Rechnung

$$((q_0, 2), a, \#) \vdash ((p_0, 1), \varepsilon, \gamma_0) \vdash ((p_1, 1), \varepsilon, \gamma_1) \vdash ((p_1, 3), \varepsilon, \gamma_1)$$

ausführen.

Satz 61
Die Klasse DCFL ist nicht abgeschlossen unter Durchschnitt, Vereinigung, Produkt und Sternhülle.

Beweis:

- Die im Beweis von Satz 54 betrachteten Sprachen $L_1 = \{a^i b^j c^j \mid i, j \geq 0\}$ und $L_2 = \{a^i b^i c^j \mid i, j \geq 0\}$ sind sogar deterministisch kontextfrei. Da $L_1 \cap L_2$ nicht kontextfrei ist, ist diese Sprache natürlich auch nicht deterministisch kontextfrei.
- Da DCFL unter Komplementbildung abgeschlossen ist, kann DCFL wegen de Morgan dann auch nicht unter Vereinigung abgeschlossen sein. So sind beispielsweise die beiden Sprachen

$$L_3 = \{a^i b^j c^k \mid i \neq j\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k\}$$

deterministisch kontextfrei, ihre Vereinigung aber nicht,

$$L_3 \cup L_4 = \{a^i b^j c^k \mid i \neq j \text{ oder } j \neq k\} \in \text{CFL} \setminus \text{DCFL} \tag{2.1}$$

Da nämlich DCFL unter Durchschnittsbildung mit regulären Sprachen abgeschlossen ist (siehe Übungen), wäre mit $L_3 \cup L_4$ auch die Sprache $(\overline{L_3 \cup L_4}) \cap L(a^*b^*c^*) = \{a^n b^n c^n \mid n \geq 0\}$ (deterministisch) kontextfrei.
- Um zu zeigen, dass DCFL nicht unter Produktbildung abgeschlossen ist, betrachten wir die beiden deterministisch kontextfreien Sprachen L_0^* und $L = L_0 L_3 \cup L_4$, wobei $L_0 = \{0\}$ ist.
Behauptung: $L_0^* L \notin$ DCFL.
Nehmen wir das Gegenteil an, so wäre auch die Sprache

$$L_0^* L \cap L_0 L(a^*b^*c^*) = \underbrace{L_0(L_3 \cup L_4)}_{L_5}$$

in DCFL, da DCFL unter Durchschnitt mit regulären Sprachen abgeschlossen ist. Sei also $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA mit $L(M) = L_5$. Dann wird die Sprache $L_3 \cup L_4$ von dem DPDA $M' = (Z \cup \{q'\}, \Sigma, \Gamma, \delta', q', \#, E)$ mit

$$\delta'(q, u, A) \quad = \quad \begin{cases} (p, \gamma), & (q, u, A) = (q', \varepsilon, \#), \\ \delta(q, u, A), & (q, u, A) \in Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma, \end{cases}$$

erkannt, wobei p der Zustand und γ der Kellerinhalt von M ist, nachdem M das Zeichen 0 gelesen hat. Dies steht jedoch im Widerspruch zu (2.1).
- Dass DCFL auch nicht unter Sternhüllenbildung abgeschlossen ist, zeigt man ganz ähnlich anhand der Sprache $L_0 \cup L$ (siehe Übungen).

■

Zum Schluss dieses Kapitels fassen wir nochmals die Abschlusseigenschaften der Sprachklassen REG, DCFL und CFL zusammen.

	Vereinigung	Durchschnitt	Komplement	Produkt	Sternhülle
REG	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	ja	nein	nein	ja	ja

3 Kontextsensitive Sprachen

In diesem Kapitel führen wir das Maschinenmodell des linear beschränkten Automaten (LBA) ein und zeigen, dass LBAs genau die kontextsensitiven Sprachen erkennen. Erst vor wenigen Jahren gelang der Nachweis, dass die Klasse CSL unter Komplementbildung abgeschlossen ist. Nach wie vor offen ist jedoch die Frage, ob die Klasse DCSL der deterministisch kontextsensitiven Sprachen eine echte Teilklasse von CSL ist oder nicht. (Eine Sprache ist in DCSL, wenn sie von einem deterministischen LBA erkannt wird.)

3.1 Kontextsensitive Grammatiken

Zur Erinnerung: Eine Grammatik $G = (V, \Sigma, P, S)$ heißt **kontextsensitiv**, falls für alle Regeln $\alpha \to \beta$ gilt: $|\beta| \geq |\alpha|$. Die einzige Ausnahme hiervon bildet die Regel $S \to \varepsilon$, die allerdings nur dann benutzt werden darf, wenn das Startsymbol S nicht auf der rechten Seite einer Regel vorkommt.

Beispiel 62

Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit

$$\begin{array}{rcl} V &=& \{S, B, C\}, \\ \Sigma &=& \{a, b, c\} \end{array}$$

und den Regeln

$$\begin{array}{lrll} P\colon & S \to aSBC, aBC & & (1,2) \\ & CB \to BC & & (3) \\ & aB \to ab & & (4) \\ & bB \to bb & & (5) \\ & bC \to bc & & (6) \\ & cC \to cc & & (7) \end{array}$$

In G läßt sich beispielsweise das Wort $w = aabbcc$ ableiten:

$$S \underset{(1)}{\Rightarrow} aSBC \underset{(2)}{\Rightarrow} aaBCBC \underset{(3)}{\Rightarrow} aaBBCC$$

$$\underset{(4)}{\Rightarrow} aabBCC \underset{(5)}{\Rightarrow} aabbCC \underset{(6)}{\Rightarrow} aabbcC \underset{(7)}{\Rightarrow} aabbcc$$

Allgemein gilt für alle $n \geq 1$:

$$\begin{aligned}
S \quad &\underset{(1,2)}{\Rightarrow^n} \quad \underbrace{a \cdots a}_{n\text{-mal}} \underbrace{BC \cdots BC}_{n\text{-mal}} \underset{(3)}{\Rightarrow^m} \quad \underbrace{a \cdots a}_{n\text{-mal}} \underbrace{B \cdots B}_{n\text{-mal}} \underbrace{C \cdots C}_{n\text{-mal}} \\
&\underset{(4,5)}{\Rightarrow^n} \quad \underbrace{a \cdots a}_{n\text{-mal}} \underbrace{b \cdots b}_{n\text{-mal}} \underbrace{C \cdots C}_{n\text{-mal}} \underset{(6,7)}{\Rightarrow^n} \quad \underbrace{a \cdots a}_{n\text{-mal}} \underbrace{b \cdots b}_{n\text{-mal}} \underbrace{c \cdots c}_{n\text{-mal}}
\end{aligned}$$

wobei Regel (3) genau $m = n(n-1)/2$-mal angewendet wird. Also gilt $a^n b^n c^n \in L(G)$ für alle $n \geq 1$.

Umgekehrt folgt durch Induktion über die Ableitungslänge, dass jede Satzform u mit $S \Rightarrow^* u$ die folgenden Bedingungen erfüllt:

- $\#_a(u) = \#_b(u) + \#_B(u) = \#_c(u) + \#_C(u)$,
- links von S und links von einem a kommen nur a's,
- links von einem b kommen nur a's oder b's.

Daraus ergibt sich, dass in G nur Wörter $w \in \Sigma^*$ der Form $w = a^n b^n c^n$ abgeleitetet werden können. Also ist

$$L(G) \ = \ \{a^n b^n c^n \mid n \geq 1\}.$$

Da wir bereits wissen, dass die Sprache $\{a^n b^n c^n \mid n \geq 1\}$ nicht kontextfrei ist, ist die Klasse CFL der kontextfreien Sprachen echt in CSL enthalten.

3.2 Turingmaschinen

Wir führen nun das Rechenmodell der Turingmaschine ein. Im Unterschied zum endlichen Automaten darf eine Turingmaschine ihre Eingabe überschreiben und somit das Eingabeband als Speicher benutzen. Von besonderem Interesse ist in diesem Kapitel der Fall, dass die TM den Bereich der Eingabe niemals verläßt. Eine solche TM wird als linear beschränkter Automat (LBA) bezeichnet. Zuvor betrachten wir jedoch das Modell einer Mehrband-Turingmaschine ohne Speicherplatzbeschränkung.

Definition 63 (k-Band-Turingmaschine)

Sei $k \geq 1$. Eine **nichtdeterministische k-Band-Turingmaschine** (kurz k-NTM oder einfach NTM) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ beschrieben, wobei

- Z eine endliche Menge von Zuständen,
- Σ das Eingabealphabet (wobei $\sqcup \notin \Sigma$),

- Γ das Arbeitsalphabet (wobei $\Sigma \cup \{\sqcup\} \subseteq \Gamma$),
- δ: $Z \times \Gamma^k \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ die Überführungsfunktion,
- q_0 der Startzustand und
- $E \subseteq Z$ die Menge der Endzustände ist.

Eine k-NTM M heißt **deterministisch** (kurz: M ist eine k-DTM oder einfach DTM), falls $\|\delta(q, a_1, \ldots a_k)\| \leq 1$ für alle $(q, a_1, \ldots a_k) \in Z \times \Gamma^k$ gilt.

Für $(q', a'_1, \ldots, a'_k, D_1, \ldots, D_k) \in \delta(q, a_1, \ldots a_k)$ schreiben wir auch

$$(q, a_1, \ldots, a_k) \to (q', a'_1, \ldots, a'_k, D_1, \ldots, D_k).$$

Eine solche Anweisung ist ausführbar, falls sich M im Zustand q befindet und sich die Leseköpfe auf mit $a_1, \ldots, a_k$ beschrifteten Feldern (a_i auf dem i-ten Band) befinden. Ihre Ausführung bewirkt, dass M in den Zustand q' übergeht, auf Band i das Symbol a_i durch a'_i ersetzt und den Kopf gemäß D_i bewegt (L: ein Feld nach links, R: ein Feld nach rechts, N: keine Bewegung).

Definition 64 (Konfiguration)
Eine **Konfiguration** ist ein $(3k+1)$-Tupel $K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \in Z \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$ und besagt, dass

- der momentane Zustand q ist und
- das i-te Band mit $\ldots \sqcup u_i a_i v_i \sqcup \ldots$ beschriftet ist, wobei sich der Kopf auf dem Zeichen a_i befindet.

Im Fall $k = 1$ schreiben wir für eine Konfiguration (q, u, a, v) auch kurz $uqav$. Eine Konfiguration $K' = (q', u'_1, a'_1, v'_1, \ldots, u'_k, a'_k, v'_k)$ heißt **Folgekonfiguration** von $K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ (kurz $K \vdash K'$), falls eine Anweisung

$$(q, a_1, \ldots, a_k) \to (q', b_1, \ldots, b_k, D_1, \ldots, D_k)$$

existiert, so dass für $i = 1, \ldots, k$ gilt:

1. Im Fall $D_i =$ N: $u'_i = u_i$, $a'_i = b_i$ und $v'_i = v_i$.
2. Im Fall $D_i =$ R: $u'_i = u_i b_i$ und $a'_i v'_i = \begin{cases} v_i, & v_i \neq \varepsilon, \\ \sqcup, & \text{sonst.} \end{cases}$
3. Im Fall $D_i =$ L: $v'_i = b_i v_i$, und $u'_i a'_i = \begin{cases} u_i, & u_i \neq \varepsilon, \\ \sqcup, & \text{sonst.} \end{cases}$

Man beachte, dass sich die Länge der Bandinschrift $u_i a_i v_i$ beim Übergang von K zu K' nicht verkleinern kann, d.h. $|u'_i a'_i v'_i| \geq |u_i a_i v_i|$.

Sei $x = x_1 \cdots x_n \in \Sigma^*$ eine Eingabe. Die zugehörige **Startkonfiguration** ist

$$K_x = \begin{cases} (q_0, \varepsilon, x_1, x_2 \cdots x_n, \varepsilon, \sqcup, \varepsilon, \ldots, \varepsilon, \sqcup, \varepsilon) & x \neq \varepsilon, \\ (q_0, \varepsilon, \sqcup, \varepsilon, \ldots, \varepsilon, \sqcup, \varepsilon) & x = \varepsilon. \end{cases}$$

Die von M **akzeptierte** oder **erkannte Sprache** ist

$$L(M) \quad = \quad \{x \in \Sigma^* \mid \exists K \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k : K_x \vdash^* K\}.$$

Ein Wort x wird also genau dann von M akzeptiert (kurz: $M(x)$ akzeptiert), wenn es eine Rechnung (Folge von Konfigurationen) von M bei Eingabe x gibt, bei der ein Endzustand erreicht wird.

Beispiel 65

Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit

$$\begin{aligned} Z &= \{q_0, \ldots q_4\}, \\ \Sigma &= \{a, b\}, \\ \Gamma &= \Sigma \cup \{A, B, \sqcup\}, \\ E &= \{q_4\} \end{aligned}$$

und den Anweisungen

δ:	$q_0 a \to q_1 AR$	(1)	Anfang der Schleife: Ersetze das erste a durch A
	$q_1 a \to q_1 aR$	(2)	Bewege den Kopf nach rechts bis zum ersten b und
	$q_1 B \to q_1 BR$	(3)	ersetze dies durch ein B (falls kein b mehr vorhan-
	$q_1 b \to q_2 BL$	(4)	den ist, dann halte ohne zu akzeptieren).
	$q_2 a \to q_2 aL$	(5)	Bewege den Kopf nach links bis ein A kommt, ge-
	$q_2 B \to q_2 BL$	(6)	he ein Feld nach rechts zurück und wiederhole die
	$q_2 A \to q_0 AR$	(7)	Schleife.
	$q_0 B \to q_3 BR$	(8)	Falls kein a am Anfang der Schleife, dann teste,
	$q_3 B \to q_3 BR$	(9)	ob noch ein b vorhanden ist. Falls nicht, akzeptiere,
	$q_3 \sqcup \to q_4 \sqcup N$	(10)	sonst halte ohne zu akzeptieren.

Wegen

$$\begin{array}{rcll}
q_0 aabb & \vdash & Aq_1 abb & (1) \\
& \vdash & Aaq_1 bb & (2) \\
& \vdash & Aq_2 aBb & (4) \\
& \vdash & q_2 AaBb & (5) \\
& \vdash & Aq_0 aBb & (7) \\
& \vdash & AAq_1 Bb & (1) \\
& \vdash & AABq_1 b & (3) \\
& \vdash & AAq_2 BB & (4) \\
& \vdash & Aq_2 ABB & (6) \\
& \vdash & AAq_0 BB & (7) \\
& \vdash & AABq_3 B & (8) \\
& \vdash & AABBq_3 \sqcup & (9) \\
& \vdash & AABBq_4 \sqcup & (10)
\end{array}$$

folgt beispielsweise $aabb \in L(M)$. Ähnlich läßt sich $a^n b^n \in L(M)$ für ein beliebiges $n \geq 1$ zeigen. Andererseits gehört das Wort abb wegen

$$q_0 abb \underset{(1)}{\vdash} Aq_1 bb \underset{(4)}{\vdash} q_2 ABb \underset{(7)}{\vdash} Aq_0 Bb \underset{(8)}{\vdash} ABq_3 b$$

nicht zu $L(M)$, da die Rechnung in der Konfiguration ABq_3b nicht fortgesetzt werden kann und kein Endzustand erreicht wurde. Tatsächlich ist $L(M) = \{a^n b^n \mid n \geq 1\}$ (ohne Beweis).

3.3 Linear beschränkte Automaten

Ein linear beschränkter Automat (LBA) ist eine 1-NTM, die nicht mehr Platz benötigt als ihr durch die Eingabe bereitgestellt wird. Dies scheint auf den ersten Blick der Begriffsbildung „linear beschränkt" zu widersprechen. Man kann jedoch zeigen, dass jede k-NTM, die bei Eingaben der Länge n nur linear viele (also $O(n)$) Bandfelder besucht, von einem LBA simuliert werden kann. Damit ein LBA das Ende der Eingabe erkennen kann (und nicht versehentlich darüber hinausliest), muss das letzte Zeichen der Eingabe markiert werden. (Das erste Zeichen der Eingabe braucht dagegen nicht markiert zu werden, da dies der LBA im ersten Rechenschritt selbst tun kann.)

Definition 66 (LBA)

Sei Σ ein Alphabet. Für $x = x_1 \cdots x_{n-1} x_n \in \Sigma^*$ bezeichne $\hat{x}$ das Wort

$$\hat{x} = \begin{cases} x, & x = \varepsilon, \\ x_1 \cdots x_{n-1} \hat{x}_n, & x \neq \varepsilon \end{cases}$$

über dem Alphabet $\Sigma' = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$. Eine 1-NTM $M = (Z, \Sigma', \Gamma, \delta, q_0, E)$ heißt **linear beschränkt** (kurz: M ist ein **LBA**), falls M für jedes Wort $x \in \Sigma^+$ ausgehend von der Startkonfiguration $K_{\hat{x}}$ nur Konfigurationen $K = uqav$ mit $|uv| \leq n$ erreichen kann:

$$\forall x \in \Sigma^+ : K_{\hat{x}} \vdash^* uqav \;\Rightarrow\; |uav| \leq |x|.$$

Die von einem LBA **akzeptierte** oder **erkannte Sprache** ist

$$L(M) \quad = \quad \{x \in \Sigma^* \mid M(\hat{x}) \text{ akzeptiert}\}.$$

Beispiel 67

Es ist nicht schwer, die 1-DTM M aus Beispiel 65 in einen LBA $M' = (Z, \Sigma', \Gamma', \delta', q_0, E)$ für $L = \{a^n b^n \mid n \geq 1\}$ umzuwandeln. Ersetze hierzu

- Σ durch $\Sigma' = \{a, b, \hat{a}, \hat{b}\}$,
- Γ durch $\Gamma' = \Sigma' \cup \{A, B, \hat{B}, \sqcup\}$ sowie

- die Anweisung $q_3\sqcup \to q_4\sqcup N$ (10) durch $q_3\hat{B} \to q_4\hat{B}N$ (10′) und
- füge die Anweisung $q_1\hat{b} \to q_2\hat{B}L$ (4a) hinzu.

Dann wird das Wort $aabb$ durch folgende Rechnung von M' bei Eingabe $aab\hat{b}$ akzeptiert:

$$q_0aab\hat{b} \ \vdash^* \ AABq_1\hat{b} \underset{(4a)}{\vdash} AAq_2B\hat{B} \ \vdash^* \ AABq_3\hat{B} \underset{(10')}{\vdash} AABq_4\hat{B}$$

Der LBA M' für die (kontextfreie) Sprache $\{a^nb^n \mid n \geq 1\}$ aus Beispiel 67 lässt sich leicht in einen LBA für die kontextsensitive Sprache $\{a^nb^nc^n \mid n \geq 1\}$ verwandeln (siehe Übungen). Als nächstes zeigen wir, dass LBAs genau die kontextsensitiven Sprachen erkennen.

Satz 68

CSL $= \{L(M) \mid M$ ist ein LBA$\}$.

Beweis: Wir zeigen zuerst die Inklusion von links nach rechts. Sei also $L \in$ CSL und sei $G = (V, \Sigma, P, S)$ eine kontextsensitive Grammatik mit $L = L(G)$. Dann wird L von folgendem LBA M akzeptiert, der bei Eingabe $\hat{x} = x_1 \cdots x_{n-1}\hat{x}_n$, $n \geq 1$, die folgenden Schritte ausführt (im Fall $\varepsilon \in L$ soll M auch ε akzeptieren).

1) Wähle nichtdeterministisch eine Regel $\alpha \to \beta$ aus $P \setminus \{S \to \varepsilon\}$.
2) Wähle nichtdeterministisch ein beliebiges Vorkommen von β auf dem Band. (Falls β auf dem Band nicht vorkommt, halte ohne zu akzeptieren.)
3) Verschiebe die Zeichen rechts von β um $|\beta| - |\alpha|$ Positionen nach links und ersetze die ersten $|\alpha|$ Zeichen von β durch α.
4) Enthält das erste Bandfeld das markierte Startsymbol $\hat{S}$, so halte in einem Endzustand. Ansonsten gehe zu Schritt 1).

Nun ist leicht zu sehen, dass M wegen $|\beta| \geq |\alpha|$ tatsächlich linear beschränkt ist und dass M genau diejenigen Wörter akzeptiert, für die es eine Ableitung in G gibt (welche M in umgekehrter Reihenfolge berechnet).

Für die Inklusion von rechts nach links sei $M = (Z, \Sigma', \Gamma, \delta, q_0, E)$ ein LBA und Σ ein Alphabet mit $\Sigma' = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$. Wir konstruieren eine kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit $L(G) = L(M)$. Für V wählen wir die Menge

$$V \ = \ \{S, A\} \cup ((\Gamma \cup (Z \times \Gamma)) \times \Sigma)$$

und P enthält folgende Regeln:

$S \to A(\hat{a}, a)$,	$a \in \Sigma$	(1)
$A \to A(a, a)$,	$a \in \Sigma$	(2)
$A \to ((q_0, a), a)$,	$a \in \Sigma$	(3)
$((q, c), a) \to ((q', c'), a)$,	$a \in \Sigma, (q', c', N) \in \delta(q, c)$	(4)
$((q, c), a)(d, b) \to (c', a)((q', d), b)$,	$a, b \in \Sigma, d \in \Gamma, (q', c', R) \in \delta(q, c)$	(5)
$(c, a)((q, d), b) \to ((q', c), a)(d', b)$,	$a, b \in \Sigma, c \in \Gamma, (q', d', L) \in \delta(q, d)$	(6)
$((q, c), a) \to a$,	$a \in \Sigma, c \in \Gamma, q \in E$	(7)
$(c, a) \to a$,	$a \in \Sigma, c \in \Gamma$	(8)

Im Fall $\varepsilon \in L(M)$ nehmen wir noch die Regel $S \to \varepsilon$ zu P hinzu. Durch Induktion über die Rechnungslänge m kann nun leicht für alle Eingaben $\hat{x} = x_1 \cdots x_{n-1}\hat{x}_n$ und alle $a_1, \ldots, a_n \in \Gamma$, $q \in Z$ die folgende Äquivalenz bewiesen werden:

$$
\begin{aligned}
&q_0 x_1 \cdots x_{n-1}\hat{x}_n \vdash^m a_1 \cdots a_{i-1} q a_i \cdots a_n \iff \\
&\qquad ((q_0, x_1), x_1) \cdots (\hat{x}_n, x_n) \underset{(4,5,6)}{\Rightarrow}{}^m (a_1, x_1) \cdots ((q, a_i), x_i) \cdots (a_n, x_n)
\end{aligned}
$$

Unter Benutzung dieser Äquivalenz folgt dann aus der Annahme $x \in L(M)$, dass

$$
\begin{aligned}
S &\underset{(1,2,3)}{\Rightarrow}{}^* ((q_0, x_1), x_1) \cdots (\hat{x}_n, x_n) \\
&\underset{(4,5,6)}{\Rightarrow}{}^* (a_1, x_1) \cdots ((q, a_i), x_i) \cdots (a_n, x_n),\ q \in E \\
&\underset{(7,8)}{\Rightarrow}{}^* x_1 \cdots x_n
\end{aligned}
$$

gilt, also $x \in L(G)$ ist. Ganz ähnlich folgt auch umgekehrt aus $S \Rightarrow^* x$, dass x von M akzeptiert wird. ■

Bemerkung 69

- Eine einfache Modifikation des Beweises zeigt, dass 1-NTMs genau die Sprachen vom Typ 0 akzeptieren, d.h.

$$\text{RE} = \{L(M) \mid M \text{ ist eine 1-NTM}\}.$$

- Bis heute ungelöst ist die Frage, ob es kontextsensitive Sprachen gibt, die nicht von einem deterministischen LBA akzeptiert werden. Anders ausgedrückt: Ist die Klasse

$$\text{DCSL} = \{L(M) \mid M \text{ ist ein deterministischer LBA}\}$$

  echt in der Klasse CSL enthalten?

Die folgenden Abschlusseigenschaften — abgesehen von der Komplementbildung bei den Klassen CSL (die Thema der Vorlesung Komplexitätstheorie ist) und RE (siehe nächstes Kapitel) — lassen sich nun leicht nachweisen (siehe Übungen).

	Vereinigung	Durchschnitt	Komplement	Produkt	Sternhülle
REG	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	ja	nein	nein	ja	ja
DCSL	ja	ja	ja	ja	ja
CSL	ja	ja	ja	ja	ja
RE	ja	ja	nein	ja	ja

4 Entscheidbare und rekursiv aufzählbare Sprachen

In diesem Kapitel beschäftigen wir uns mit der Klasse RE, die alle Typ-0 Sprachen enthält. Wir werden eine Reihe con Charakterisierungen für diese Klasse mittels Turingmaschinen beweisen, wodurch auch die Namensgebung (rekursiv aufzählbar) verständlich wird. Eine wichtige Teilklasse von RE bildet die Klasse REC der entscheidbaren (oder rekursiven) Sprachen, in der bereits alle kontextsensitiven Sprachen enthalten sind. Zunächst wenden wir uns jedoch der Berechnung von Funktionen zu.

Definition 70 (Berechenbarkeit von totalen Funktionen)

Eine k-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine Funktion $f : \Sigma^* \to \Gamma^*$, falls für alle $x \in \Sigma^*$ gilt:

> Es gibt genau eine Konfiguration $K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ in $E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$ mit $K_x \vdash^* K$ und für diese Konfiguration ist $u_k = f(x)$, $a_k = \sqcup$ und $v_k = \varepsilon$. Hierfür schreiben wir auch kurz $M(x) = f(x)$.

In diesem Fall heißt f **Turing-berechenbar** (oder einfach **berechenbar** oder **rekursiv**).

Um $f(x)$ zu berechnen, muss M bei Eingabe x im Verlauf der Rechnung also genau einmal in einen Endzustand gelangen und zu diesem Zeitpunkt muss das k-te Band mit $f(x)$ beschriftet sein. Obige Definition lässt sich auch auf partielle Funktionen übertragen, die nicht für alle Argumente definiert sein müssen.

Definition 71 (Berechenbarkeit von partiellen Funktionen)

Eine k-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine **partielle Funktion** $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$, falls für alle $x \in \Sigma^*$ gilt:

> Ist $f(x)$ undefiniert (in Zeichen: $f(x) = \uparrow$), so darf M bei Eingabe x keinen Endzustand erreichen (unabhängig davon, ob M hält oder nicht). Andernfalls muss (wie oben) $M(x) = f(x)$ gelten.

Mit FREC bezeichnen wir die Klasse aller partiellen rekursiven Funktionen und FREC_t bezeichnet die Teilklasse der totalen Funktionen in FREC.

Beispiel 72

Sei $\Sigma = \{0, 1\}$ und bezeichne x^+ den lexikografischen Nachfolger des Wortes x,

x	ε	0	1	00	01	10	11	000	$\ldots$
x^+	0	1	00	01	10	11	000	001	$\ldots$

Dann gehören die totalen Funktionen $f_i : \Sigma^* \to \Sigma^*$, $i = 1, 2, 3$ mit $f_1(x) = 0$, $f_2(x) = x$ und $f_3(x) = x^+$ zu FREC_t, während die partielle Funktion

$$f_4(x) = \begin{cases} \uparrow, & x = \varepsilon \\ y, & x = y^+ \end{cases}$$

zur Klasse FREC gehört.

Definition 73 ((semi-)entscheidbare und rekursiv aufzählbare Sprachen)

- Eine Sprache $L \subseteq \Sigma^*$ heißt **entscheidbar** (oder **rekursiv**, kurz: $L \in$ REC), falls ihre **charakteristische Funktion** $\chi_L : \Sigma^* \to \{0, 1\}$, definiert durch

$$\chi_L(x) \quad = \quad \begin{cases} 1, & x \in L, \\ 0, & x \notin L, \end{cases}$$

  berechenbar ist.

- Eine Sprache $L \subseteq \Sigma^*$ heißt **semi-entscheidbar**, falls die partielle Funktion

$$\hat{\chi}_L(x) \quad = \quad \begin{cases} 1, & x \in L, \\ \uparrow, & x \notin L \end{cases}$$

  berechenbar ist.

- Eine Sprache $L \subseteq \Sigma^*$ heißt **rekursiv aufzählbar** (kurz: $L \in$ RE), falls L leer ist oder eine Turing-berechenbare Funktion $f : \Gamma^* \to \Sigma^*$ mit

$$L = \underbrace{\{f(x) \mid x \in \Gamma^*\}}_{=:W(f)}$$

  existiert. Die Menge $W(f)$ nennen wir die **Wertemenge** von f.

Der nächste Satz gibt eine Reihe von nützlichen Charakterisierungen der Typ-0 Sprachen (z.B. dass sie mit den rekursiv aufzählbaren Sprachen übereinstimmen).

Satz 74

Folgende Eigenschaften sind äquivalent:

1. A ist semi-entscheidbar,
2. A wird von einer k-NTM akzeptiert,
3. A wird von einer 1-NTM akzeptiert,
4. A ist vom Typ 0,
5. A ist rekursiv aufzählbar,
6. A wird von einer k-DTM akzeptiert,
7. A wird von einer 1-DTM akzeptiert.

Beweis: $1) \Rightarrow 2)$: Klar, da eine k-NTM, die $\hat{\chi}_A$ berechnet, A akzeptiert.

$2) \Rightarrow 3)$: Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM, die die Sprache A akzeptiert. Wir konstruieren eine 1-NTM $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$, die ebenfalls A akzeptiert. M' simuliert M, indem sie jede Konfiguration K von M der Form

$$\begin{array}{c|c|c|c|c|c}
\hline
\cdots & a & b & c & d & \cdots \\
\hline
 & & \vdots & & \uparrow & \\
\hline
\cdots & e & f & g & h & \cdots \\
\hline
 & & \uparrow & & &
\end{array}$$

durch eine Konfiguration K' der Form

$$\begin{array}{c|c|c|c|c|c}
\hline
\cdots & \begin{pmatrix} a \\ \vdots \\ e \end{pmatrix} & \begin{pmatrix} b \\ \vdots \\ \hat{f} \end{pmatrix} & \begin{pmatrix} c \\ \vdots \\ g \end{pmatrix} & \begin{pmatrix} \hat{d} \\ \vdots \\ h \end{pmatrix} & \cdots \\
\hline
\end{array}$$

nachbildet. Das heißt, M' arbeitet mit dem Alphabet

$$\Gamma' = \Gamma \cup (\Gamma \cup \{\hat{a} \mid a \in \Gamma\})^k$$

und erzeugt bei Eingabe $x = x_1 \cdots x_n \in \Sigma^*$ zuerst die der Startkonfiguration

$$K_x = (q_0, \varepsilon, x, \varepsilon, \sqcup, \ldots, \varepsilon, \sqcup)$$

von M bei Eingabe x entsprechende Konfiguration

$$K'_x = q'_0 \begin{pmatrix} \hat{x}_1 \\ \hat{\sqcup} \\ \vdots \\ \hat{\sqcup} \end{pmatrix} \begin{pmatrix} x_2 \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix} \cdots \begin{pmatrix} x_n \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix}.$$

Dann simuliert M' jeweils einen Schritt von M durch folgende Sequenz von Rechenschritten.

> Zuerst geht M' solange nach rechts, bis sie alle mit ^ markierten Zeichen (z.B. $\hat{a}_1, \ldots, \hat{a}_k$) gefunden hat. Diese Zeichen speichert M' in ihrem Zustand. Anschließend geht M' wieder nach links und realisiert dabei die durch $\delta(q, a_1, \ldots, a_k)$ vorgegebene Anweisung von M (den momentanen Zustand q von M speichert M' ebenfalls in ihrem Zustand).

Sobald M in einen Endzustand übergeht, wechselt M' ebenfalls in einen Endzustand und hält. Nun ist leicht zu sehen, dass $L(M') = L(M)$ ist.

$3) \Rightarrow 4)$: Konstruktion einer Grammatik wie beim Beweis von $\{L(M) \mid M \text{ ist ein LBA }\} \subseteq \text{CSL}$.

$4) \Rightarrow 5)$: Sei $G = (V, \Sigma, P, S)$ eine Grammatik und sei $A = L(G) \neq \emptyset$. Dann lassen sich die Wörter von A durch folgende Funktion $f : \Gamma^* \to \Gamma^*$ aufzählen, wobei

$\Gamma = V \cup \Sigma \cup \{\#\}$ und $\# \notin V \cup \Sigma$ ist:

$$f(x) = \begin{cases} \alpha_m, & x \text{ hat die Form } \alpha_0\#\alpha_1\#\ldots\#\alpha_m \text{ und } S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \\ & \cdots \Rightarrow \alpha_m \text{ ist eine Ableitung von } \alpha_m \in \Sigma^* \text{ in } G \\ x_0, & \text{sonst.} \end{cases}$$

Es ist klar, dass f von einer k-DTM berechnet werden kann, also ist A rekursiv aufzählbar.

$5) \Rightarrow 6)$: Sei $f : \Gamma^* \to \Sigma^*$ eine Funktion mit $A = W(f)$ und sei M eine k-DTM, die f berechnet. Betrachte folgende $(k+1)$-DTM M':

> Bei Eingabe x erzeugt M' auf dem 2. Band alle Wörter y in Γ^* (etwa in lexikographischer Reihenfolge) und berechnet durch Simulation von M jeweils den Wert $f(y)$. Sobald $f(y) = x$ ist, akzeptiert M' ihre Eingabe.

$6) \Rightarrow 7)$: Eine k-DTM kann genau wie im Beweis der Implikation $2) \Rightarrow 3)$ durch eine 1-DTM simuliert werden.

$7) \Rightarrow 1)$: Klar, da aus einer 1-DTM, die A akzeptiert, leicht eine 2-DTM gewonnen werden kann, die $\hat{\chi}_A$ berechnet.

■

Satz 75

A ist genau dann entscheidbar, wenn A und $\bar{A}$ semi-entscheidbar sind.

Beweis: Falls A entscheidbar ist, ist auch $\bar{A}$ entscheidbar, d.h. A und $\bar{A}$ sind dann auch semi-entscheidbar. Für die Rückrichtung seien $f_1, f_2 : \Gamma^* \to \Sigma^*$ Turing-berechenbare Funktionen mit $W(f_1) = A$ und $W(f_2) = \bar{A}$. Wir betrachten folgende k-DTM M:

> Bei Eingabe x bestimmt M der Reihe nach für jedes $y \in \Gamma^*$ die beiden Werte $f_1(y)$ und $f_2(y)$. Sobald ein y auf den Wert $f_1(y) = x$ führt, gibt M den Wert $M(x) = 1$ aus und falls $f_2(y) = x$ ist, gibt M den Wert $M(x) = 0$ aus.

Da jede Eingabe x entweder in $W(f_1) = A$ oder in $W(f_2) = \bar{A}$ enthalten ist, berechnet M die Funktion χ_A. ■

Bezeichnen wir für eine Sprachklasse $\mathcal{C}$ die Klasse $\{\bar{A} \mid A \in \mathcal{C}\}$ der Komplementärsprachen mit co-$\mathcal{C}$, so gilt also REC = RE $\cap$ co-RE.

4.1 Kodierung (Gödelisierung) von Turingmaschinen

Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine 1-NTM mit Eingabealphabet $\Sigma = \{0, 1, \#\}$ und einem beliebigen Arbeitsalphabet $\Gamma = \{a_0 = 0, a_1 = 1, a_2 = \#, a_3 = \sqcup, a_4, \ldots, a_l\}$

sowie Zustandsmenge $Z = \{q_0, \ldots, q_m\}$ (o.B.d.A. sei $E = \{q_m\}$). Dann können wir jede Anweisung der Form

$$q_i a_j \to q_{i'} a_{j'} D$$

durch das Wort $\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#b_D\#$ kodieren, wobei $bin(n)$ die Binärdarstellung von n und

$$b_D = \begin{cases} 0, & D = N. \\ 1, & D = L, \\ 11, & D = R \end{cases}$$

ist. Eine Binärkodierung $w_M \in \{0,1\}^*$ von M erhalten wir, indem wir alle Anweisungen von M in kodierter Form hintereinander schreiben, wobei wir das Alphabet $\{0, 1, \#\}$ binär kodieren (z.B. $0 \mapsto 00, 1 \mapsto 11, \# \mapsto 01$).

Entsprechend können wir auch k-NTMs, $k \geq 1$, sowie Konfigurationen und Rechnungen (Konfigurationsfolgen) von k-NTMs kodieren.

Jedem Binärstring $w \in \{0,1\}^*$ können wir umgekehrt eine k-NTM M_w wie folgt zuordnen:

$$M_w = \begin{cases} M, & \text{falls eine } k\text{-NTM } M \text{ mit } w_M = w \text{ existiert,} \\ M_0, & \text{sonst.} \end{cases}$$

Dabei ist M_0 eine beliebig, aber fest gewählte Turingmaschine mit Eingabealphabet $\Sigma = \{0, 1, \#\}$.

Definition 76 (Halteproblem)

Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM und sei $x \in \Sigma^*$ eine Eingabe. Dann ist

$$time_M(x) = \max\{t \geq 0 \mid \exists K : K_x \vdash^t K\}$$

die **Rechenzeit** von M bei Eingabe x, wobei $\max \mathbb{N} = \infty$ ist.

M **hält** bei Eingabe x (kurz: $M(x)$ hält), falls $time_M(x) < \infty$ ist. Das **Halteproblem** ist die Sprache

$$H = \{w\#x \mid w, x \in \{0,1\}^* \text{ und } M_w \text{ ist eine 1-DTM, die bei Eingabe } x \text{ hält}\}$$

und das **spezielle Halteproblem** ist

$$K = \{w \in \{0,1\}^* \mid M_w \text{ ist eine 1-DTM, die bei Eingabe } w \text{ hält}\}.$$

Satz 77

$K \in \text{RE} \setminus \text{REC}$.

Beweis: Sei w_0 die Kodierung einer 1-DTM, die bei jeder Eingabe hält und betrachte die Funktion

$$f(x) = \begin{cases} w, & x \text{ ist Kodierung einer haltenden Berechnung einer 1-DTM } M_w \text{ bei Eingabe } w, \\ w_0, & \text{sonst.} \end{cases}$$

Da f Turing-berechenbar und $W(f) = K$ ist, folgt $K \in$ RE.

Die Unentscheidbarkeit von K zeigen wir indirekt. Unter der Annahme, dass K entscheidbar ist, gibt es eine 1-DTM M, die bei Eingabe x genau dann hält, wenn $x \notin K$ ist. Für die Kodierung w von M folgt dann aber

$$\begin{aligned} w \in K \quad &\Rightarrow \quad M_w(w) \text{ hält} \\ &\Rightarrow \quad w \notin K, \text{ da } M_w = M \text{ ist,} \\ w \notin K \quad &\Rightarrow \quad M_w(w) \text{ hält nicht} \\ &\Rightarrow \quad w \in K, \text{ da } M_w = M \text{ ist.} \end{aligned}$$

Sowohl die Annahme $w \in K$ als auch die konträre Annahme $w \notin K$ führen also auf einen Widerspruch. ∎

Korollar 78

$K \notin co$-RE.

Beweis: Wegen $K \in$ RE folgt aus der Annahme, dass $K \in co$-RE ist, mit Satz 75, dass K entscheidbar ist. Dies steht jedoch im Widerspruch zu Satz 77. ∎

Definition 79 (reduzierbar)

Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. Dann heißt A auf B **reduzierbar** (kurz: $A \leq B$), falls eine Turing-berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ existiert mit

$$x \in A \Leftrightarrow f(x) \in B$$

für alle $x \in \Sigma^*$.

Sei $\mathcal{C}$ eine Sprachklasse. Eine Sprache $A \in \mathcal{C}$ heißt **$\mathcal{C}$-vollständig**, falls jede Sprache $L \in \mathcal{C}$ auf A reduzierbar ist.

Beispiel 80

$K \leq H$ mittels $f(w) = w\#w$ für alle $w \in \{0,1\}^*$.

Satz 81

Die Klassen RE und REC sind unter $\leq$ abgeschlossen. Dabei heißt eine Sprachklasse $\mathcal{C}$ **unter $\leq$ abgeschlossen**, wenn für alle Sprachen A, B gilt:

$$A \leq B \wedge B \in \mathcal{C} \Rightarrow A \in \mathcal{C}.$$

Beweis: Gelte $A \leq B$ mittels f und sei M eine 1-DTM, die χ_B (bzw. $\hat{\chi}_B$) berechnet. Betrachte folgende 1-DTM M':

> M' berechnet bei Eingabe x zuerst den Wert $f(x)$ und simuliert dann M bei Eingabe $f(x)$.

Wegen

$$x \in A \Leftrightarrow f(x) \in B$$

folgt $\chi_A(x) = \chi_B(f(x)) = M(f(x)) = M'(x)$ (bzw. $\hat{\chi_A}(x) = \hat{\chi_B}(f(x)) = M(f(x)) = M'(x)$), d.h. M' berechnet χ_A (bzw. $\hat{\chi}_A$). ∎

Korollar 82

$H \notin$ REC.

Beweis: Aus der Annahme, dass H entscheidbar ist, folgt wegen $K \leq H$, dass auch K entscheidbar ist, was im Widerspruch zu Satz 77 steht. ■

Definition 83 (Halteproblem bei leerem Band)

Das **Halteproblem bei leerem Band** ist die Sprache

$$H_0 = \{w \in \{0,1\}^* \mid M_w \text{ ist eine 1-DTM, die bei Eingabe } \varepsilon \text{ hält}\}.$$

Satz 84

$H_0 \notin$ REC.

Beweis: Wir zeigen $H \leq H_0$. Dadurch überträgt sich die Unentscheidbarkeit von H auf H_0, da andernfalls mit H_0 auch H entscheidbar wäre, was nach Korollar 82 nicht der Fall ist. Sei $w_0 \notin H_0$ und betrachte die Funktion

$$f(y) = \begin{cases} w', & y \text{ hat die Form } y = w\#x \text{ und } M_w \text{ ist eine 1-DTM; dabei ist } w' \text{ die Kodierung einer 1-DTM, die bei Eingabe } \varepsilon \text{ zuerst das Wort } x \text{ auf das Eingabeband schreibt und dann die 1-DTM } M_w \text{ bei Eingabe } x \text{ simuliert,} \\ w_0, & \text{sonst.} \end{cases}$$

Dann ist f eine Turing-berechenbare Funktion, die H auf H_0 reduziert. ■

Der folgende Satz von Rice besagt, dass man einer k-DTM nicht ansehen kann, ob die von ihr berechnete Funktion eine gewisse Eigenschaft hat oder nicht. Einzige Ausnahme: Keine oder jede berechenbare Funktion hat die fragliche Eigenschaft.

Satz 85 (Satz von Rice)

Sei $\Sigma = \{0, 1, \#\}$ und FREC_Σ bezeichne die Klasse aller partiellen Turing-berechenbaren Funktionen $f : \Sigma^* \to \Sigma^* \cup \{\uparrow\}$. Dann ist für jede Klasse $\mathcal{F}$ von partiellen Funktionen mit $\emptyset \subsetneq \mathcal{F} \subsetneq \mathrm{FREC}_\Sigma$ die Sprache

$$K(\mathcal{F}) = \{w \mid \ w \text{ kodiert eine } k\text{-DTM } M_w, \text{ die eine Funktion in } \mathcal{F} \text{ berechnet}\}$$

unentscheidbar, d.h. $K(\mathcal{F}) \notin$ REC.

Beweis: Wir reduzieren H_0 (oder $\overline{H_0}$) auf $K(\mathcal{F})$. Für eine gegebene 1-DTM M_w müssen wir also eine k-DTM $M_{w'}$ mit

$$\begin{aligned} w \in H_0 \quad &\Rightarrow \quad M_{w'} \text{ berechnet eine Funktion in } \mathcal{F}, \\ w \notin H_0 \quad &\Rightarrow \quad M_{w'} \text{ berechnet eine Funktion in } \overline{\mathcal{F}} \end{aligned}$$

konstruieren. Die Idee besteht nun darin, $M_{w'}$ bei Eingabe x zunächst einmal die 1-DTM M_w bei Eingabe ε simulieren zu lassen. Falls $w \notin H_0$ ist, berechnet $M_{w'}$ also die überall undefinierte Funktion u mit $u(x) =\uparrow$ für alle $x \in \Sigma^*$. Damit die Reduktion gelingt, muss $M_{w'}$ im Fall $w \in H_0$ eine Funktion $v \in \mathcal{F}$ berechnen mit

$$v \in \mathcal{F} \Leftrightarrow u \notin \mathcal{F}.$$

Wegen $\emptyset \subsetneq \mathcal{F} \subsetneq \text{FREC}_\Sigma$ muss eine solche Funktion $v \in \text{FREC}_\Sigma$ existieren.

Sei also M eine k-DTM, die v berechnet und betrachte die Reduktionsfunktion $f : \{0,1\}^* \to \{0,1\}^*$ mit

$h(w) \quad = \quad w'$, wobei w' die Kodierung einer k-DTM ist, die bei Eingabe x zunächst einmal die 1-DTM M_w bei Eingabe ε simuliert und im Fall, dass M_w hält, mit der Simulation von M bei Eingabe x fortfährt.

Dann ist h eine totale berechenbare Funktion und es gilt

$$\begin{aligned} w \in H_0 &\quad\rightarrow\quad M_{w'} \text{ berechnet } v, \\ w \notin H_0 &\quad\rightarrow\quad M_{w'} \text{ berechnet } u. \end{aligned}$$

Im Fall $v \in \mathcal{F}$ folgt daher

$$\begin{aligned} w \in H_0 &\quad\rightarrow\quad w' \in K(\mathcal{F}), \\ w \notin H_0 &\quad\rightarrow\quad w' \notin K(\mathcal{F}), \end{aligned}$$

das heißt, h reduziert H_0 auf $K(\mathcal{F})$. Im Fall $v \notin \mathcal{F}$ gilt dagegen

$$\begin{aligned} w \in H_0 &\quad\rightarrow\quad w' \notin K(\mathcal{F}), \\ w \notin H_0 &\quad\rightarrow\quad w' \in K(\mathcal{F}), \end{aligned}$$

das heißt, h reduziert $\overline{H_0}$ auf $K(\mathcal{F})$. Da h Turing-berechenbar ist und sowohl H_0 als auch $\overline{H_0}$ unentscheidbar sind, folgt $K(\mathcal{F}) \notin \text{REC}$. ■

Beispiel 86

Die Sprache

$$L = \{w \in \{0,1\}^* \mid M_w(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

ist unentscheidbar. Dies folgt aus dem Satz von Rice, da $L = K(\mathcal{F})$ für die durch $\mathcal{F} = \{f \in \text{FREC}_\Sigma \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$ beschriebene Eigenschaft ist, welche zwar mindestens auf eine, aber nicht auf alle Funktionen in FREC_Σ zutrifft (beispielsweise ist die Funktion

$$a(x) = \begin{cases} 0^{n+1}, & x = 0^n \\ \uparrow, & \text{sonst} \end{cases}$$

in $\mathcal{F}$ enthalten, aber nicht die konstante Funktion $f(x) = 0$).

4.2 Das Postsche Korrespondenzproblem (PCP)

Sei Σ ein beliebiges Alphabet mit $\# \notin \Sigma$. Dann enthält PCP_Σ alle positiven (mit ja zu beantwortenden) Eingaben für das folgende Entscheidungsproblem:

Gegeben: k Wortpaare $(x_1, y_1), \ldots, (x_k, y_k)$, kodiert durch $x_1\#y_1\#\cdots\#x_k\#y_k$ (die wir oft auch in Form einer Matrix $\binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ schreiben).

Gefragt: Gibt es eine PCP-Lösung für die Probleminstanz $\binom{x_1 \cdots x_k}{y_1 \ldots y_k}$? Dabei ist eine PCP-Lösung eine Folge $\alpha = (i_1, \ldots, i_n)$, $n \geq 1$, von Indizes $i_j \in \{1, \ldots, k\}$ mit

$$x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}.$$

Im Fall $i_1 = 1$ heißt α auch MPCP-Lösung (Lösung für das modifizierte PCP). Entsprechend enthält MPCP alle Problemeingaben, für die eine MPCP-Lösung existiert.

Beispiel 87

Die Instanz $I = \binom{1 \ \ 10 \ 011}{101 \ 00 \ \ 11}$ besitzt wegen

$$x_1x_3x_2x_3 = 101110011 = y_1y_3y_2y_3$$

die PCP-Lösung $\alpha = (1, 3, 2, 3)$, die auch eine MPCP-Lösung ist.

Lemma 88 ()

Für jedes Alphabet Σ gilt $\text{PCP}_\Sigma \leq \text{PCP}_{\{0,1\}}$ und $\text{MPCP}_\Sigma \leq \text{MPCP}_{\{0,1\}}$.

Beweis: Sei $\Sigma = \{a_1, \ldots, a_m\}$. Für ein Zeichen $a_i \in \Sigma$ sei $\hat{a}_i = 01^i$ und für ein Wort $w = w_1 \cdots w_n \in \Sigma^*$ mit $w_i \in \Sigma$ sei $\hat{w} = \hat{w}_1 \cdots \hat{w}_n$. Dann folgt $\text{PCP}_\Sigma \leq \text{PCP}_{\{0,1\}}$ und $\text{MPCP}_\Sigma \leq \text{MPCP}_{\{0,1\}}$ mittels der Reduktionsfunktion

$$f\begin{pmatrix} x_1 \cdots x_k \\ y_1 \ldots y_k \end{pmatrix} = \begin{pmatrix} \hat{x}_1 \cdots \hat{x}_k \\ \hat{y}_1 \ldots \hat{y}_k \end{pmatrix}.$$

■

Im Folgenden schreiben wir für $\text{PCP}_{\{0,1\}}$ ($\text{MPCP}_{\{0,1\}}$) kurz PCP (bzw. MPCP).

Satz 89

MPCP $\leq$ PCP.

Beweis: Sei $\Sigma' = \{0, 1, \$, \S\}$. Es genügt, $\text{MPCP} \leq \text{PCP}_{\Sigma'}$ zu zeigen. Betrachte die Reduktionsfunktion f, die eine MPCP-Instanz $I = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ auf die $\text{PCP}_{\Sigma'}$-Instanz

$$f(I) = \begin{pmatrix} \overrightarrow{x_1} & \overrightarrow{x_2} & \cdots & \overrightarrow{x_k} & \S\overrightarrow{x_1} & \S \\ \overleftarrow{y_1} & \overleftarrow{y_2} & & \overleftarrow{y_k} & \S\bar{y_1} & \$\S \end{pmatrix}$$

abbildet, wobei für $w = a_1 \cdots a_n$, $a_i \in \Sigma$,

$$\begin{aligned} \bar{w} &= a_1\$ \cdots \$a_n, \\ \overrightarrow{w} &= a_1\$ \cdots \$a_n\$, \\ \overleftarrow{w} &= \$a_1\$ \cdots \$a_n, \end{aligned}$$

ist. Da sich jede MPCP-Lösung $\alpha = (1, i_2, \ldots, i_n)$ für I in eine PCP-Lösung $\alpha' = (k+1, i_2, \ldots, i_n, k+2)$ für $f(I)$ verwandeln lässt, folgt

$$I \in \text{MPCP} \Rightarrow f(I) \in \text{PCP}_{\Sigma'}.$$

Für die umgekehrte Implikation sei $\alpha = (i_1, \ldots, i_n)$ eine (o.B.d.A. kürzeste) PCP-Lösung für $f(I)$. Dann muss $i_1 = k + 1$ sein, da $(\S\,\overrightarrow{x_1}, \S\,\overline{y_1})$ das einzige Paar in $f(I)$ ist, bei dem beide Komponenten mit demselben Buchstaben anfangen. Zudem muss $i_n = k+2$ sein, da $(\S, \$\S)$ das einzige Paar in $f(I)$ ist, bei dem beide Komponenten mit demselben Buchstaben aufhören. Aufgrund der minimalen Länge von α folgt $i_j \leq k$ für die übrigen Indizes i_j in α. Dann ist aber

$$\alpha' = (1, i_2, \ldots, i_{n-1})$$

eine MPCP-Lösung für I. ■

Satz 90

PCP ist RE-vollständig und damit unentscheidbar.

Beweis: Sei $A \in$ RE und sei $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$ eine 1-NTM mit $L(M) = A$. Sei $\Sigma' = \Gamma \cup Z \cup \{\$\}$. Wegen $\text{MPCP}_{\Sigma'} \leq \text{PCP}$ reicht es zu zeigen, dass zu jeder Eingabe $w \in \Sigma^*$ eine Instanz $f(w) = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ mit

$$w \in L(M) \Leftrightarrow f(w) \in \text{MPCP}_{\Sigma'}$$

berechnet werden kann.

Idee: Eine Indexfolge $\alpha = (i_1, \ldots, i_n)$ soll genau dann eine Lösung von $f(w)$ sein, wenn das zugehörige Lösungswort

$$x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$$

eine akzeptierende Rechnung von M bei Eingabe w kodiert.

Wir bilden $f(w)$ aus folgenden Wortpaaren:

1. $(\S, \S\$z_0x)$, „Startregel“
2. (a, a) für alle $a \in \Gamma \cup \{\$\}$, „Kopierregeln“
3. Für alle $a, a', b \in \Gamma$, $z, z' \in Z$ die folgenden Wortpaare,

$(za, z'a')$,	falls $\delta(z, a) = (z', a', N)$,
$(za, a'z')$,	falls $\delta(z, a) = (z', a', R)$,
$(bza, z'ba')$,	falls $\delta(z, a) = (z', a', L)$,

   sowie „Überführungsregeln“

$(\$za, \$z' \sqcup a')$,	falls $\delta(z, a) = (z', a', L)$,
$(z\$, z'a'\$)$,	falls $\delta(z, \sqcup) = (z', a', N)$,
$(z\$, a'z'\$)$,	falls $\delta(z, \sqcup) = (z', a', R)$,
$(bz\$, z'ba'\$)$,	falls $\delta(z, \sqcup) = (z', a', L)$,

4. (az_e, z_e) und (z_ea, z_e) für alle $z_e \in E$ und $a \in \Gamma$, „Löschregeln“
5. $(z_e\$\S, \S)$ für alle $z_e \in E$. „Abschlussregeln“

Nun lässt sich leicht aus einer akzeptierenden Rechnung

$$K_0 = z_0 w \vdash K_1 \vdash \cdots \vdash K_t = u z_e v$$

mit $z_e \in E$ und $u, v \in \Gamma^*$ eine MPCP-Lösung mit einem Lösungswort der Form

$$\S\$K_0\$K_1\$ \cdots \$K_t\$K_{t+1}\$ \cdots \$K_{t+|K_t|-1}\$\S$$

angeben, wobei K_{t+i} aus K_t durch Löschen von i Zeichen in der Nachbarschaft von z_e entsteht.

Umgekehrt lässt sich aus jeder MPCP-Lösung auch eine akzeptierende Rechnung von M bei Eingabe w gewinnen, womit $A \leq \text{MPCP}_{\Sigma'}$ gezeigt ist. ∎

Satz 91

Das Schnittproblem

$$\{w_1\#w_2 \mid w_1, w_2 \text{ kodieren DPDAs } M_1, M_2 \text{ mit } L(M_1) \cap L(M_2) = \emptyset\}$$

für DCFL ist unentscheidbar.

Beweis: Betrachte die Turing-berechenbare Funktion

$$f\begin{pmatrix} x_1 \cdots x_k \\ y_1 \ldots y_k \end{pmatrix} = (M_1, M_2),$$

wobei die DPDAs M_j, $j = 1, 2$, die von den Grammatiken $G_j = (\{S_j\}, \{0, 1\}, P_j, S_j)$ mit den Regeln

$$\begin{aligned} P_1 : S_1 \to \quad & 0^i 1 S_1 x_i, \\ & 0^i 11 y_i, \qquad i = 1, \ldots, k \\ P_2 : S_2 \to \quad & 0^i 1 S_2 y_i, \\ & 0^i 11 y_i, \qquad i = 1, \ldots, k \end{aligned}$$

erzeugten Sprachen $L_j = L(G_j)$ entscheiden. Dann gilt

$$L(M_1) = \{0^{i_1} 1 0^{i_2} \cdots 1 0^{i_n} 11 x_{i_n} \cdots x_{i_1} \mid n \geq 1 \text{ und } 1 \leq i_1, \ldots, i_n \leq k\}$$

und

$$L(M_2) = \{0^{i_1} 1 0^{i_2} \cdots 1 0^{i_n} 11 y_{i_n} \cdots y_{i_1} \mid n \geq 1 \text{ und } 1 \leq i_1, \ldots, i_n \leq k\}.$$

Somit ist

$$L(M_1) \cap L(M_2) = \{0^{i_1} 1 \cdots 1 0^{i_n} 11 z \mid n \geq 1 \text{ und } x_{i_n} \cdots x_{i_1} = z = y_{i_n} \cdots y_{i_1}\},$$

d.h. die Instanz $I = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ hat genau dann eine PCP-Lösung $\alpha = (i_n \cdots i_1)$, wenn $0^{i_1} 1 \cdots 1 0^{i_n} 11 x_{i_n} \cdots x_{i_1} \in L(M_1) \cap L(M_2)$, also $L(M_1) \cap L(M_2) \neq \emptyset$ ist. Dies zeigt, dass f eine Reduktion von PCP auf das Schnittproblem für DCFL vermittelt. ∎

Korollar 92

Das Inklusionsproblem für DCFL ist unentscheidbar.

Beweis: In obigem Beweis besitzt I genau dann eine PCP-Lösung, wenn $L_1 \cap L_2 \neq \emptyset$, d.h. $L_1 \subseteq L_2$ ist. ■

Korollar 93

Für kontextfreie Grammatiken sind folgende Fragestellungen unentscheidbar:

1. Ist G mehrdeutig?
2. Ist $L(G_1) = L(G_2)$? (Äquivalenzproblem)
3. Ist $L(G) = \Sigma^*$? (Ausschöpfungsproblem)

Beweis:

1. Betrachte die Turing-berechenbare Funktion
$$f\begin{pmatrix} x_1 \cdots x_k \\ y_1 \ldots y_k \end{pmatrix} = G,$$
wobei $G = (\{S, S_1, S_2\}, \{0, 1\}, P_1 \cup P_2 \cup \{S \to S_1, S_2\}, S)$ und P_j die Regelmengen aus vorigem Beweis sind. Da alle von S_1 oder S_2 ausgehende Ableitungen eindeutig sind, ist G genau dann mehrdeutig, wenn es ein Wort $w \in L(G)$ gibt mit
$$S \;\Rightarrow\; S_1 \;\Rightarrow^*\; w \text{ und } S \;\Rightarrow\; S_2 \;\Rightarrow^*\; w.$$
Wie wir im vorigen Beweis gesehen haben, ist dies genau dann der Fall, wenn I eine PCP-Lösung hat.
2. Seien $L_j = L(M_j)$, $j = 1, 2$, die beiden Sprachen aus obigem Beweis. Dann können bei Eingabe von $I = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ DPDAs M_1', M_2' mit $L(M_j') = \overline{L_j}$ und kontextfreie Grammatiken G_1', G_2', G mit $L(G_j') = \overline{L_j}$ und $L(G') = L_1 \cup \overline{L_2}$ berechnet werden. Wegen
$$\begin{aligned} I \notin \text{PCP} \quad &\Leftrightarrow \quad L_1 \cap L_2 = \emptyset \\ &\Leftrightarrow \quad L_1 \subseteq \overline{L_2} \\ &\Leftrightarrow \quad L_1 \cup \overline{L_2} = \overline{L_2} \\ &\Leftrightarrow \quad L(G') = L(G_2') \end{aligned}$$
vermittelt die Funktion $g(I) = (G', G_2')$ eine Reduktion des Komplements von PCP auf das Äquivalenzproblem von kontextfreien Grammatiken.
3. Aus den beiden Grammatiken G_1' und G_2' kann leicht eine kontextfreie Grammatik G'' mit $L(G'') = L(G_1') \cup L(G_2') = \overline{L_1} \cup \overline{L_2}$ gebildet werden. D.h. die Funktion $h(I) = G''$ ist Turing-berechenbar und wegen
$$\begin{aligned} I \notin \text{PCP} \quad &\Leftrightarrow \quad L_1 \cap L_2 = \emptyset \\ &\Leftrightarrow \quad \overline{L_1} \cup \overline{L_2} = \Sigma^* \\ &\Leftrightarrow \quad L(G'') = \Sigma^* \end{aligned}$$
reduziert sie das Komplement von PCP auf das Ausschöpfungsproblem für CFL.

■

Dagegen ist es nicht schwer, für eine kontextfreie Grammatik G zu entscheiden, ob ein Wort in G ableitbar ist.

Satz 94

Das Leerheitsproblem für CFL ist entscheidbar.

Beweis: Betrachte folgenden Algorithmus:

Eingabe: eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$
  $U' \leftarrow \emptyset$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow \{A \mid$ es gibt eine Regel $A \rightarrow \alpha \in P$ mit $\alpha \in (\Sigma \cup U)^*\}$
  until $(U = U')$
  if $S \in U$ **then**
    output „$L(G) \neq \emptyset$“
  else
    output „$L(G) = \emptyset$“
  end

Es ist klar, dass dieser Algorithmus korrekt arbeitet und auch von einer 1-DTM ausgeführt werden kann. ■

Satz 95

Das Leerheitsproblem für CSL ist unentscheidbar.

Beweis: Wir reduzieren das Schnittproblem für CFL auf das (Nicht-)Leerheitsproblem für CSL. Betrachte hierzu die Funktion

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

wobei G eine kontextsensitive Grammatik mit $L(G) = L(G_1) \cap L(G_2)$ ist. Es bleibt nur zu zeigen, dass f berechenbar ist:

> Bei Eingabe (G_1, G_2) bestimme zunächst LBAs M_1, M_2 mit $L(M_i) = L(G_i)$, $i = 1, 2$. Konstruiere daraus einen LBA M für die Sprache $L = L(M_1) \cap L(M_2)$, der erst M_1 und dann M_2 bei Eingabe x simuliert, ohne bei der Simulation von M_1 die Eingabe zu zerstören. Transformiere M in eine kontextsensitive Grammatik G mit $L(G) = L(M)$.

■

Satz 96

Das Wortproblem für CSL ist entscheidbar (Beweis siehe Übungen).

Zum Schluss dieses Kapitels fassen wir nochmals zusammen, welche der hier betrachteten Eigenschaften für die einzelnen Sprachtypen entscheidbar sind und welche nicht.

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

*Die Entscheidbarkeit des Äquivalenzproblems für die Klasse DCFL konnte erst vor wenigen Jahren (1997) nachgewiesen werden.

5 Komplexitätsklassen und NP-Vollständigkeit

5.1 Zeitkomplexität

Der Zeitverbrauch $time_M(x)$ einer Turingmaschine M bei Eingabe x ist die Anzahl der Schritte, die diese Maschine ausgehend von der Startkonfiguration K_x ausführt (bzw. ∞, falls sie nicht stoppt).

Definition 97 (Rechenzeit)

Sei M eine Turingmaschine und sei $x \in \Sigma^*$ eine Eingabe. Dann ist

$$time_M(x) = \max\{t \geq 0 \mid \exists K : K_x \vdash^t K\}$$

die **Rechenzeit** von M bei Eingabe x, wobei $\max \mathbb{N} := \infty$.

Sei $t : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M **$t(n)$-zeitbeschränkt**, falls für alle $x \in \Sigma^*$ gilt:

$$\mathrm{time}_M(x) \leq t(|x|).$$

Alle Sprachen, die in (nicht-) deterministischer Zeit $t(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\text{DTIME}(t(n)) := \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\}$$

und

$$\text{NTIME}(t(n)) := \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}$$

zusammen. Ferner sei

$$\text{FTIME}(t(n)) := \{f \mid f \text{ wird von einer } t(n)\text{-zeitbeschränkten DTM berechnet}\}.$$

Die wichtigsten Zeitkomplexitätsklassen sind

$$\begin{aligned}
\text{LINTIME} &= \bigcup_{c\geq 1} \text{DTIME}(cn), \\
\text{P} &= \bigcup_{c\geq 1} \text{DTIME}(n^c), \\
\text{E} &= \bigcup_{c\geq 1} \text{DTIME}(2^{cn}), \\
\text{EXP} &= \bigcup_{c\geq 1} \text{DTIME}(2^{n^c}).
\end{aligned}$$

Die Klassen NP, NE, NEXP und FP, FE, FEXP sind analog definiert.

5.2 Platzkomplexität

Als nächstes definieren wir den Platzverbrauch einer Turingmaschine M als die Summe aller während einer Rechnung besuchten Bandfelder. Um auch sublineare Platzkomplexitätsklassen sinnvoll definieren zu können, zählen wir jedoch die Bandfelder, auf denen die Eingabe steht, nicht mit. Damit M diesen Platz nicht auch zum Rechnen benutzt, darf M die Inschrift des ersten Bandes nicht verändern und nur bis zum Ende der Eingabe lesen.

Definition 98 (Offline-Turingmaschine)
Eine Turingmaschine M heißt **Offline-Turingmaschine,** falls für jede Anweisung $(q', a'_1, \ldots, a'_k, D_1, \ldots, D_k) \in \delta(q, a_1, \ldots a_k)$

- $a'_1 = a_1$ und
- $a_1 = \sqcup \Rightarrow D_1 = L$

gilt.

Definition 99 (Platzverbrauch)
Sei M eine Offline-Turingmaschine und sei $x \in \Sigma^*$ eine Eingabe. Dann ist

$$\begin{aligned}\mathrm{space}_M(x) = \max\{s \geq 1 \mid \exists K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \\ \text{mit } s = \sum_{i=2}^{k} |u_i a_i v_i| \text{ und } K_x \vdash^t K\}\end{aligned}$$

der **Platzverbrauch** von M bei Eingabe x. Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $s(n)$**-platzbeschränkt**, falls für alle $x \in \Sigma^*$ gilt:

$$\mathrm{space}_M(x) \leq s(|x|).$$

Alle Sprachen, die in (nicht-) deterministischem Platz $s(n)$ entscheidbar sind, fassen wir in den Komplexitätsklassen

$$\mathrm{DSPACE}(s(n)) := \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-DTM}\}$$

und

$$\mathrm{NSPACE}(s(n)) := \{L(M) \mid M \text{ ist eine } s(n)\text{-platzbeschränkte Offline-NTM}\}$$

zusammen. Die wichtigsten Platzkomplexitätsklassen sind

$$\begin{aligned}
\mathrm{L} &= \mathrm{DSPACE}(\log n)\\
\mathrm{LINSPACE} &= \mathrm{DSPACE}(n) \;=\; \bigcup_{c>0} \mathrm{DSPACE}(cn)\\
\mathrm{PSPACE} &= \bigcup_{c>0} \mathrm{DSPACE}(n^c)\\
\mathrm{ESPACE} &= \bigcup_{c>0} \mathrm{DSPACE}(2^{cn})\\
\mathrm{EXPSPACE} &= \bigcup_{c>0} \mathrm{DSPACE}(2^{n^c})
\end{aligned}$$

Die Klassen NL, NLINSPACE und NPSPACE sind analog definiert, wobei letztere wegen NSPACE$(s(n)) \subseteq$ DSPACE$(s^2(n))$ mit PSPACE zusammenfällt (dies wird in der Vorlesung Komplexitätstheorie gezeigt).

Die Klassen der Chomsky-Hierarchie lassen sich gemäß ihrer Zeit- und Platzkomplexität wie folgt einordnen.

$$\begin{aligned}
\mathrm{REG} \subseteq \mathrm{DCFL} &\subseteq \mathrm{LINTIME} \subseteq \mathrm{P}\\
\mathrm{CFL} &\subseteq \mathrm{DTIME}(n^3) \subseteq \mathrm{P}\\
\mathrm{DCSL} &= \mathrm{LINSPACE} \subseteq \mathrm{CSL} = \mathrm{NLINSPACE} \subseteq \mathrm{E}\\
\mathrm{REC} &= \bigcup_f \mathrm{DSPACE}(f(n)) = \bigcup_f \mathrm{NSPACE}(f(n))\\
&= \bigcup_f \mathrm{DTIME}(f(n)) = \bigcup_f \mathrm{NTIME}(f(n)),
\end{aligned}$$

wobei f alle berechenbaren Funktionen $f : \mathbb{N} \to \mathbb{N}$ durchläuft.

5.3 NP-Vollständigkeit und das P=NP Problem

Wie wir im letzten Kapitel gesehen haben, sind NTMs nicht mächtiger als DTMs, da erstere von letzteren simuliert werden können (siehe Satz 74). Die Frage, wieviel Zeit eine NTM gegenüber einer DTM einsparen kann, ist eines der wichtigsten offenen Probleme der Informatik. So enthält die Klasse NP viele für die Praxis überaus wichtige Probleme, von denen nicht bekannt ist, ob sie auch zu P gehören. Da jedoch nur Probleme in P als effizient lösbar gelten können, ist das so genannte **P $=$ NP-Problem**, also die Frage, ob die Inklusion von P in NP echt ist, nicht nur von theoretischem Interesse, sondern hat auch eine große praktische Bedeutung.

Wie bereits erwähnt, ist diese Frage im Kontext von Platzklassen weitgehend geklärt, da eine $s(n)$-platzbeschränkte NTM durch eine $s^2(n)$-platzbeschränkte DTM simuliert werden kann.

Definition 100 (Polynomialzeitreduktion)

Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. Dann heißt A auf B **in Polynomialzeit reduzierbar** (kurz: $A \leq^p B$), falls eine Funktion $f : \Sigma^* \to \Gamma^*$ in FP existiert mit

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B$$

Lemma 101 (D)

e Reduktionsrelation $\leq^p$ ist reflexiv und transitiv (siehe Übungen).

Satz 102

Die Klassen P und NP sind unter $\leq^p$ abgeschlossen.

Beweis: Sei $B \in$ P (bzw. $B \in$ NP) und gelte $A \leq^p B$ mittels einer Funktion $f \in$ FP. Sei M eine (nicht)deterministische Turingmaschine mit $L(M) = B$, sei T eine DTM, die f berechnet, und seien p, q polynomiale Zeitschranken für M und T. Betrachte folgende Turingmaschine M', die sich bei Eingabe x wie M bei Eingabe $f(x)$ verhält.

Eingabe: x
  $y \leftarrow f(x)$
  simuliere M bei Eingabe y

Dann gilt

$$\begin{aligned} x \in L(M') &\Leftrightarrow f(x) \in L(M) \\ &\Leftrightarrow f(x) \in B \\ &\Leftrightarrow x \in A, \end{aligned}$$

d.h. $L(M') = A$. Wegen

$$\begin{aligned} \mathit{time}_{M'}(x) &\leq \mathit{time}_T(x) + \mathit{time}_M(f(x)) \\ &\leq q(|x|) + p(q(|x|)), \end{aligned}$$

ist M' polynomial zeitbeschränkt, weshalb A in P (bzw. NP) ist. ■

Definition 103 (NP-vollständig)

Eine Sprache A heißt **NP-schwer** (oder **NP-hart**), falls gilt:

$$\forall L \in \text{NP} : L \leq^p A.$$

Ist darüber hinaus $A \in$ NP, so heißt A **NP-vollständig** (kurz $A \in$ NPC; NP-complete).

Satz 104

1. $A \leq^p B$ und A ist NP-schwer $\Rightarrow$ B ist NP-schwer.
2. $A \leq^p B$, $A \in$ NPC und $B \in$ NP $\Rightarrow$ $B \in$ NPC.

3. NPC $\cap$ P $\neq \emptyset \Rightarrow$ P $=$ NP.

Beweis: 1. Wir müssen für eine beliebige Sprache $L \in$ NP zeigen, dass $L \leq^p B$ gilt. Da A nach Voraussetzung NP-schwer ist, folgt zunächst $L \leq^p A$. Da zudem $A \leq^p B$ gilt und die Relation $\leq^p$ transitiv ist, folgt $L \leq^p B$.

2. Mit 1. folgt, dass B NP-schwer ist. Da zudem $B \in$ NP ist, folgt $B \in$ NPC.
3. Falls eine Sprache A in NPC $\cap$ P existiert, so lässt sich jede NP-Sprache L auf $A \in$ P reduzieren. Da P unter $\leq^p$ abgeschlossen ist, folgt $L \in$ P.

■

Satz 105
Die Sprache

$$A = \{M\#x\#0^m \mid M \text{ ist eine NTM, die } x \text{ in } \leq m \text{ Schritten akzeptiert}\}$$

ist NP-vollständig.

Beweis: Zunächst ist klar, dass $A \in$ NP mittels folgender NTM ist:

> Bei Eingabe $M\#x\#0^m$ simuliere M bei Eingabe x für höchstens m Schritte. Falls M in dieser Zeit akzeptiert, akzeptiere ebenfalls. Andernfalls verwerfe.

Zum Nachweis der NP-Härte von A sei eine beliebige NP-Sprache L gegeben. Dann existiert eine NTM M mit $L(M) = L$ und ein Polynom p, das die Rechenzeit von M begrenzt. Also lässt sich L mittels der FP-Funktion

$$f : x \mapsto M\#x\#0^{p(|x|)}$$

auf A reduzieren. ■

Definition 106 (Boolesche Formeln)
Die Menge der **Booleschen** (oder **aussagenlogischen**) **Formeln** über den Variablen $x_1, \ldots, x_n$ ist induktiv wie folgt definiert:

- Jede Variable x_i ist eine Boolesche Formel.
- Mit G und H sind auch die Negation $\neg G$ von G, die Konjunktion $(G \wedge H)$ von G und H, die Disjunktion $(G \vee H)$ sowie die Implikation $(G \rightarrow H)$ Boolesche Formeln.

Eine **Belegung** $a = a_1 \ldots a_n \in \{0,1\}^n$ ordnet jeder Variablen x_i den Wert $a_i \in \{0,1\}$ zu. Der Wert $F(a)$ von F unter a ist induktiv über den Aufbau von F definiert:

F	$F(a)$
x_i	a_i
$\neg G$	$1 - G(a)$
$G \wedge H$	$G(a)H(a)$
$G \vee H$	$G(a) + H(a) - G(a)H(a)$
$G \rightarrow H$	$1 - G(a)(1 - H(a))$

Durch F wird also eine n-stellige Boolesche Funktion berechnet, die wir ebenfalls mit F bezeichnen. F heißt **erfüllbar**, falls es eine Belegung a mit $F(a) = 1$ gibt.

Beispiel 107
Die Formel $F = (x_1 \vee x_2) \to (x_2 \wedge x_3)$ ist erfüllbar und berechnet folgende Boolesche Funktion:

$a_1a_2a_3$	$a_1 \vee a_2$	$a_2 \wedge a_3$	$(a_1 \vee a_2) \to (a_2 \wedge a_3)$
000	0	0	1
001	0	0	1
010	1	0	0
011	1	1	1
100	1	0	0
101	1	0	0
110	1	0	0
111	1	1	1

Beispiel 108
Die Formel

$$G(x_1, \ldots, x_n) = (x_1 \vee \cdots \vee x_n) \wedge \bigwedge_{1 \leq i < j \leq n} \neg(x_i \wedge x_j)$$

liefert unter einer Belegung $a = a_1 \cdots a_n$ genau dann den Wert $G(a_1 \cdots a_n) = 1$, wenn $\sum_{i=1}^{n} a_i = 1$ ist (d.h. wenn eine einzige Variable mit 1 belegt ist). Diese Formel werden wir im Beweis des nächsten Satzes benötigen.

Bei vielen praktischen Anwendungen ist es erforderlich, eine erfüllende Belegung für eine vorliegende Boolesche Formel zu finden (sofern es eine gibt). Die Bestimmung der Komplexität des folgenden Entscheidungsproblems hat also große praktische Bedeutung.

Erfüllbarkeitsproblem für Boolesche Formeln (SAT; *satisfiability*):

Gegeben: Eine Boolesche Formel F in den Variablen $x_1, \ldots, x_n$.

Gefragt: Ist F erfüllbar?

Satz 109
SAT ist NP-vollständig.

Beweis: Es ist leicht zu sehen, dass SAT $\in$ NP ist: Eine NTM kann bei Eingabe einer Booleschen Formel F zunächst eine Belegung a nichtdeterministisch raten und dann in Polynomialzeit überprüfen, ob $F(a) = 1$ ist.

Um zu zeigen, dass SAT NP-schwer ist, sei L eine beliebige NP-Sprache und sei $M = (Z, \Sigma, \Gamma, \delta, q_0)$ eine k-NTM mit $L(M) = L$, deren Rechenzeit durch ein Polynom p beschränkt ist (wir können o.B.d.A. $k = 1$ annehmen, da bei der Simulation einer k-NTM durch eine 1-NTM die Rechenzeit höchstens quadratisch ansteigt; siehe Satz 74).

Unsere Aufgabe besteht nun darin, zu einer gegebenen Eingabe $w = w_1 \cdots w_n$ eine Formel F_w zu finden, die genau dann erfüllbar ist, wenn $w \in L$ ist (natürlich müssen wir auch nachweisen, dass F_w in Polynomialzeit aus w berechenbar ist).

O.B.d.A. sei $Z = \{q_0, \ldots, q_k\}$, $E = \{q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$. Dann bilden wir F_w über den Variablen

$$x_{t,q}, y_{t,i} \text{ und } z_{t,i,a} \text{ für } 0 \leq t \leq p(n), -p(n) \leq i \leq p(n), q \in \{q_0, \ldots, q_k\} \text{ und } a \in \Gamma,$$

die für die folgenden Aussagen über eine Rechnung von M stehen sollen:

$x_{t,q}$:	zum Zeitpunkt t befindet sich M im Zustand q.
$y_{t,i}$:	zur Zeit t befindet sich der Kopf auf dem Bandfeld mit der Nummer i.
$z_{t,i,a}$:	zur Zeit t befindet sich das Zeichen a auf dem Feld mit der Nummer i.

Unser Ziel ist es, die Formel F_w so zu konstruieren, dass F_w unter einer Belegung genau dann wahr wird, wenn diese Belegung eine akzeptierende Rechnung von M bei Eingabe w beschreibt. Konkret sei $F_w = R \wedge S \wedge T_1 \wedge T_2 \wedge T_3 \wedge E$, wobei R (steht für Randbedingungen) sicherstellt, dass wir jeder erfüllenden Belegung eindeutig eine Folge von Konfigurationen $K_0, \ldots, K_{p(n)}$ zuordnen können:

$$R = \bigwedge_{0 \leq t \leq p(n)} R_t,$$

wobei

$$R_t = G(x_{t,q_0}, \ldots, x_{t,q_k}) \wedge G(y_{t,-p(n)}, \ldots, y_{t,p(n)}) \wedge \bigwedge_{-p(n) \leq i \leq p(n)} G(z_{t,i,a_1}, \ldots, z_{t,i,a_l})$$

ist. Die Teilformel R sorgt also dafür, dass zu jedem Zeitpunkt t

- genau ein Zustand $q \in \{q_0, \ldots, q_k\}$ eingenommen wird,
- genau ein Bandfeld $i \in \{-p(n), \ldots, p(n)\}$ besucht wird und
- auf jedem Feld i genau ein Zeichen $a \in \Gamma$ steht.

Die übrigen Teilformeln sind

$$
\begin{aligned}
S &= x_{0,q_0} \wedge y_{0,1} \wedge z_{0,-p(n),\sqcup} \wedge \cdots \wedge z_{0,-1,\sqcup} \wedge z_{0,0,w_1} \wedge \cdots \wedge z_{0,n-1,w_n} \wedge \\
&\quad z_{0,n,\sqcup} \wedge \cdots \wedge z_{0,p(n),\sqcup}, \\
T_1 &= \bigwedge_{\substack{0 \le t < p(n) \\ -p(n) \le i \le p(n) \\ a \in \Gamma}} [(\neg y_{t,i} \wedge z_{t,i,a}) \rightarrow z_{t+1,i,a}], \\
T_2 &= \bigwedge_{\substack{0 \le t < p(n) \\ -p(n) \le i \le p(n) \\ 0 \le j \le k-1, a \in \Gamma}} [(x_{t,q_j} \wedge y_{t,i} \wedge z_{t,i,a}) \rightarrow \bigvee_{(q',a',D) \in \delta(q_j,a)} (x_{t+1,q'} \wedge y_{t+1,i+D} \wedge z_{t+1,i,a'})], \\
T_3 &= \bigwedge_{\substack{0 \le t < p(n) \\ -p(n) \le i \le p(n) \\ a \in \Gamma}} [(x_{t,q_k} \wedge y_{t,i} \wedge z_{t,i,a}) \rightarrow (x_{t+1,q_k} \wedge y_{t+1,i} \wedge z_{t+1,i,a})] \text{ und} \\
E &= x_{p(n),q_k},
\end{aligned}
$$

wobei

$$
i + D = \begin{cases} i, & D = N \\ i+1, & D = R \\ i-1, & D = L \end{cases}
$$

ist. Die Formel S (wie Startbedingung) stellt also sicher, dass zum Zeitpunkt 0 tatsächlich die Startkonfiguration

$\sqcup$	$\cdots$	$\sqcup$	w_1	$\cdots$	w_n	$\sqcup$	$\cdots$	$\sqcup$
			$\uparrow$					
			q_0					

vorliegt. Mit der Formel T_1 wird gewährleistet, dass der Inhalt von nicht besuchten Feldern beim Übergang zwischen den Konfigurationen K_t und K_{t+1} unverändert bleibt. Dagegen achtet die Formel T_2 auf die Einhaltung der durch δ vorgegebenen Anweisungen, solange nicht der Zustand q_k angenommen wird. Schließlich sorgt T_3 dafür, dass sich nach Erreichen des Zustands q_k nichts mehr ändert und E überprüft, ob zum Zeitpunkt $p(n)$ der Zustand q_k vorliegt.

Da die Länge der Formel $f(w) = F_w$ polynomiell in n ist und ihr Aufbau einem einfachen Bildungsgesetz folgt, ist die Reduktionsfunktion f in FP berechenbar. Es bleibt zu zeigen, dass alle Eingaben w die Äquivalenz

$$
w \in L(M) \Leftrightarrow F_w \in \textsc{Sat}
$$

erfüllen.

„$\Rightarrow$“: Ist $w \in L(M)$, so gibt es eine akzeptierende Rechnung von M bei Eingabe w der Länge $\leq p(n)$. Belegen wir nun die Variablen $x_{t,q}$, $y_{t,i}$ und $z_{t,i,a}$ gemäß dieser Rechnung, so erhalten wir offensichtlich eine erfüllende Belegung für F_w.

„$\Leftarrow$“: Ist a eine erfüllende Belegung für F_w, so können wir daraus in eindeutiger Weise (wegen $R(a) = 1$) eine Folge von Konfigurationen $K_0, \ldots, K_{p(n)}$ gewinnen. Wegen $S(a) = 1$ ist K_0 die Startkonfiguration von M bei Eingabe w. Wegen $T_1(a) = T_2(a) = T_3(a) = 1$ muss für $i = 0, \ldots, p(n) - 1$ entweder $K_i \vdash K_{i+1}$ oder $K_i = K_{i+1}$ gelten. Schließlich muss $K_{p(n)}$ wegen $E(a) = 1$ eine akzeptierende Endkonfiguration und damit $w \in L(M)$ sein.

■

Gelingt es also, einen Polynomialzeit-Algorithmus für SAT zu finden, so lässt sich daraus leicht ein effizienter Algorithmus für jedes Problem in NP ableiten.

Korollar 110

SAT $\in$ P $\Leftrightarrow$ P $=$ NP.

Definition 111 (Boolesche Schaltkreise)

Ein **Boolescher Schaltkreis** c mit n Eingängen ist eine Folge $(g_1, \ldots, g_m)$ von **Gattern** $g_l \in \{0, 1, x_1, \ldots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$ mit $1 \leq j, k < l$. Die am Gatter g_l berechnete n-stellige Boolesche Funktion ist induktiv wie folgt definiert:

g_l	$g_l(a)$
0	0
1	1
x_i	a_i
$(\neg, j)$	$1 - g_j(a)$
$(\wedge, j, k)$	$g_j(a) g_k(a)$
$(\vee, j, k)$	$g_j(a) + g_k(a) - g_j(a) g_k(a)$

c berechnet die Boolesche Funktion $c(a) = g_m(a)$. c heißt **erfüllbar**, wenn es eine Eingabe $a \in \{0,1\}^n$ mit $c(a) = 1$ gibt.

Bemerkung: Die Anzahl der Eingänge eines Gatters g wird als **Fanin** von g bezeichnet, die Anzahl der Ausgänge (also die Anzahl der Gatter, die g als Eingabe benutzen) als **Fanout**. Boolesche Formeln entsprechen also den Booleschen Schaltkreisen mit (maximalem) Fanout 1 und umgekehrt.

Erfüllbarkeitsproblem für Boolesche Schaltkreise (CIRSAT):

Gegeben: Ein Boolescher Schaltkreis c mit n Eingängen.

Gefragt: Ist c erfüllbar?

Aufgrund obiger Bemerkung ist klar, dass SAT in Polynomialzeit auf CIRSAT reduzierbar ist. Da CIRSAT offensichtlich in NP enthalten ist, haben wir ein weiteres NP-vollständiges Problem gefunden.

Satz 112

CIRSAT ist NP-vollständig.

Da SAT NP-vollständig ist, ist CIRSAT in Polynomialzeit auf SAT reduzierbar. Tatsächlich können wir CIRSAT sogar auf ein Teilproblem von SAT reduzieren.

Definition 113 (konjunktive Normalform)

Eine Boolesche Formel F über den Variablen $x_1, \ldots, x_n$ ist in **konjunktiver Normalform** (kurz KNF), falls F eine Konjunktion

$$F = \bigwedge_{j=1}^{m} \underbrace{\bigvee_{l=1}^{k_i} L_{jl}}_{C_j}$$

von Disjunktionen C_j von **Literalen** $L_{jl} \in \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ ist. Hierbei verwenden wir $\bar{x}$ als abkürzende Schreibweise für $\neg x$. Gilt $k_i \leq k$ für $j = 1, \ldots, m$, so heißt F in k**-KNF**.

Eine Disjunktion $C = \bigvee_{l=1}^{k} L_l$ von Literalen wird auch als **Klausel** bezeichnet und oft auch in Mengennotation $C = \{L_1, \ldots, L_k\}$ der in ihr vorkommenden Literale dargestellt. Eine KNF-Formel $F = \bigwedge_{j=1}^{m} C_j$ kann dann als die Menge $\{C_1, \ldots, C_m\}$ ihrer Klauseln dargestellt werden.

Beispiel 114

Die Formel $F = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$ ist in 3-KNF und lässt sich in Mengennotation durch $F = \{\{x_1, \bar{x}_2\}, \{\bar{x}_1, x_3\}, \{x_2, \bar{x}_3, x_4\}\}$ beschreiben.

Erfüllbarkeitsproblem für k-KNF Formeln (k-SAT):

Gegeben: Eine Boolesche Formel in k-KNF.

Gefragt: Ist F erfüllbar?

Proposition 115

1-SAT und 2-SAT liegen in P.

Satz 116

3-SAT ist NP-vollständig.

Beweis: Wir zeigen CIRSAT $\leq^p$ 3-SAT. Hierzu transformieren wir einen Schaltkreis $c = (g_1, \ldots, g_m)$ mit n Eingängen in eine 3-KNF Formel $F_c = z_m \wedge \bigwedge_{l=1}^{m} G_l$ über den

Variablen $x_1, \ldots, x_n, z_1, \ldots, z_m$, wobei die Formeln G_l wie folgt definiert sind:

g_l	G_l
0	$\bar{z}_l$
1	z_l
x_i	$\underbrace{(\bar{z}_l \vee x_i)}_{z_l \to x_i} \wedge \underbrace{(\bar{x}_i \vee z_l)}_{x_i \to z_l}$
$(\neg, j)$	$\underbrace{(z_l \vee z_j)}_{\bar{z}_j \to z_l} \wedge \underbrace{(\bar{z}_l \vee \bar{z}_j)}_{z_l \to \bar{z}_j}$
$(\wedge, j, k)$	$\underbrace{(\bar{z}_l \vee z_j)}_{z_l \to z_j} \wedge \underbrace{(\bar{z}_l \vee z_k)}_{z_l \to z_k} \wedge \underbrace{(z_l \vee \bar{z}_j \vee \bar{z}_k)}_{(z_j \wedge z_k) \to z_l}$
$(\vee, j, k)$	$\underbrace{(z_l \vee \bar{z}_j)}_{z_j \to z_l} \wedge \underbrace{(z_l \vee \bar{z}_k)}_{z_k \to z_l} \wedge \underbrace{(\bar{z}_l \vee z_j \vee z_k)}_{z_l \to (z_j \vee z_k)}$

Offensichtlich ist die Reduktionsfunktion $f : c \mapsto F_c$ in FP berechenbar. Es bleibt zu zeigen, dass für jeden Schaltkreis c die Äquivalenz

$$c \in \text{CIRSAT} \Leftrightarrow F_c \in \text{3-SAT}$$

gilt.

„$\Rightarrow$“: Sei $a_1 \cdots a_n \in \{0,1\}^n$ eine Eingabe mit $c(a_1 \cdots a_n) = 1$. Dann liefert

$$b_l = g_l(a_1 \cdots a_n), l = 1, \ldots, m$$

eine erfüllende Belegung $a_1 \cdots a_n b_1 \cdots b_m$ für $F_c(x_1, \ldots, x_n, z_1, \ldots, z_m)$.

„$\Leftarrow$“: Ist umgekehrt $a_1 \cdots a_n b_1 \cdots b_m$ eine erfüllende Belegung für F_c, so muss $b_m = 1$ sein, da z_m eine Klausel in F_c ist. Aufgrund der Konstruktion von F_c folgt andererseits durch Induktion über $l = 1, \ldots, m$

$$g_l(a_1 \cdots a_n) = b_l.$$

Insbesondere folgt $c(a_1 \cdots a_n) = g_m(a_1 \cdots a_n) = b_m = 1$, also ist $c \in \text{CIRSAT}$.

■

Wenn eine KNF-Formel F nicht erfüllbar ist, kann es nützlich sein, herauszufinden, wieviele Klauseln in F maximal erfüllbar sind. Es ist klar, dass sich die Lösung dieses Optimierungsproblems leicht auf die Lösung des folgenden Entscheidungsproblems zurückführen lässt.

Definition 117 (MAX-k-SAT)

Gegeben: eine Formel F in k-KNF und eine Zahl l.

Gefragt: existiert eine Belegung a, die mindestens l Klauseln in F erfüllt?

Proposition 118

MAX-1-SAT liegt in P.

Satz 119

$\textsc{Max-2-Sat} \in$ NPC.

Beweis: Die Zugehörigkeit von $\textsc{Max-2-Sat}$ zu NP ist klar. Dass $\textsc{Max-2-Sat}$ NP-hart ist, zeigen wir durch eine Reduktion von 3-$\textsc{Sat}$ auf $\textsc{Max-2-Sat}$. Hierzu betrachten wir die 2-KNF-Formel $G(x, y, z, w)$ die aus den 10 Klauseln

$$\{x\}, \{y\}, \{z\}, \{w\}, \{\overline{x}, \overline{y}\}, \{\overline{y}, \overline{z}\}, \{\overline{x}, \overline{z}\}, \{x, \overline{w}\}, \{y, \overline{w}\}, \{z, \overline{w}\}$$

besteht. Die folgenden drei Eigenschaften von G lassen sich leicht verifizieren.

- Jede Belegung a erfüllt maximal 7 Klauseln von G.
- Jede Belegung $a = a_1a_2a_3$, die die Disjunktion $x \vee y \vee z$ erfüllt (d.h. $a \neq 000$), kann zu einer Belegung $a' = a_1a_2a_3a_4$ erweitert werden, die 7 Klauseln von G erfüllt.
- Die Belegung $a = 000$ kann nicht zu einer Belegung a' erweitert werden, die 7 Klauseln von G erfüllt.

Sei F eine 3-KNF-Formel mit m Klauseln und seien $C_i = \{l_{i1}, l_{i2}, l_{i3}\}$, $i = 1, \ldots, k$, die Klauseln in F, die genau 3 Literale enthalten, und seien C_i, $i = k+1, \ldots, m$ die übrigen Klauseln. Wir transformieren F auf das Paar $g(F) = (F', m + 6k)$, wobei die 2-KNF Formel F' aus F durch Anwendung folgender Regel entsteht:

> Ersetze jede Klausel $C_i = \{l_{i1}, l_{i2}, l_{i3}\}$ in F, die genau 3 Literale enthält, durch die 10 Klauseln der Formel $G_i = G(l_{i1}, l_{i2}, l_{i3}, w_i)$.

Dann ist leicht zu sehen, dass g in FL berechenbar ist. Unter Benutzung der 3 Eigenschaften der Formel G lässt sich nun leicht die Korrektheit der Reduktion, also die Äquivalenz

$$F \in 3\text{-}\textsc{Sat} \Leftrightarrow (F', m + 6k) \in \textsc{Max-2-Sat}$$

zeigen. Die Vorwärtsrichtung ergibt sich unmittelbar aus der Tatsache, dass jede erfüllende Belegung a für F zu einer Belegung a' erweiterbar ist, die $7k + m - k = m + 6k$ Klauseln von F' erfüllt.

Für die Rückwärtsrichtung sei a eine Belegung für F', die mindestens $m + 6k$ Klauseln erfüllt. Da a maximal 7 Klauseln in jeder 10er-Gruppe G_i, $i = 1, \ldots, k$ erfüllen kann, muss a genau 7 Klauseln in jeder Gruppe und alle Klauseln C_i für $i = k+1, \ldots, m$ erfüllen. Daher erfüllt a alle Klauseln C_i von F. ■

Als nächstes betrachten wir das Wortproblem für NFAs und für reguläre Ausdrücke und zeigen, dass beide effizient lösbar sind. Dagegen wird sich das Äquivalenzproblem für (sternfreie) reguläre Ausdrücke als co-NP-hart herausstellen.

Satz 120

Das Wortproblem für reguläre Sprachen, beschrieben durch NFAs,

$$\text{WP}_{\text{NFA}} = \{M\#x \mid M \text{ ist ein NFA und } x \in L(M)\}$$

ist in P entscheidbar.

Beweis: Betrachte folgenden Algorithmus:

Eingabe: $M\#x$ mit $x = x_1 \dots x_n$ und $M = (Z, \Sigma, \delta, Q_0, E)$
  $Q \leftarrow Q_0$
  for $i \leftarrow 1$ **to** n **do**
    $Q \leftarrow \bigcup_{q \in Q} \delta(q, x_i)$
  end
  if $Q \cap E \neq \emptyset$ **then**
    accept
  else
    reject
  end

Es ist klar, dass dieser Algorithmus korrekt arbeitet und auch in eine polynomiell zeitbeschränkte DTM übersetzt werden kann, die die Sprache WP_{NFA} entscheidet. ■

Korollar 121
Das Wortproblem für reguläre Ausdrücke,

$$\text{WP}_{\text{RA}} = \{\alpha\#x \mid \alpha \text{ ist ein regulärer Ausdruck und } x \in L(\alpha)\}$$

ist in P entscheidbar.

Beweis: Da ein regulärer Ausdruck α in Polynomialzeit in einen äquivalenten NFA N_α transformiert werden kann, gilt $\text{WP}_{\text{RA}} \leq^p \text{WP}_{\text{NFA}}$ mittels $f(\alpha\#x) = (N_\alpha\#x)$. Da nach vorigem Satz $L \in$ P ist, und da P unter $\leq^p$ abgeschlossen ist, folgt $\text{WP}_{\text{RA}} \in$ P. ■

Mit einem ganz ähnlichen Beweis zeigt man, dass auch das Leerheitsproblem und das Schnittproblem für NFAs in Polynomialzeit lösbar sind (siehe Übungen).

Korollar 122
Das Leerheitsproblem für NFAs,

$$\text{LP}_{\text{NFA}} = \{M \mid M \text{ ist ein NFA und } L(M) = \emptyset\}$$

und das Schnittproblem für NFAs,

$$\text{SP}_{\text{NFA}} = \{M_1\#M_2 \mid M_1 \text{ und } M_2 \text{ sind NFAs mit } L(M_1) \cap L(M_2) = \emptyset\}$$

sind in P lösbar.

Definition 123 (sternfrei)
Ein regulärer Ausdruck $\alpha \in (\Sigma \cup \{\emptyset, \epsilon, (,), |\})^*$, in dem kein Stern vorkommt, heißt **sternfrei**.

Satz 124
Sei $\Sigma = \{a_1, \dots, a_m\}$ ein Alphabet mit $m \geq 2$. Dann ist die Sprache $L = \{\alpha\#\mathit{bin}(n) \mid \alpha \text{ ist ein sternfreier regulärer Ausdruck über } \Sigma \text{ mit } L(\alpha) \neq \Sigma^n\}$ NP-vollständig.

Beweis: Wir zeigen zuerst $L \in \mathrm{NP}$. Durch Induktion über die Länge von α überzeugt man sich leicht davon, dass für einen sternfreien regulären Ausdruck α immer

$$L(\alpha) \subseteq \Sigma^{\leq|\alpha|}$$

gilt, wobei $\Sigma^{\leq m} := \bigcup_{n=0}^{m} \Sigma^n$ ist. Daher akzeptiert folgender NP-Algorithmus die Sprache L:

Eingabe: $\alpha\#\mathit{bin}(n)$
  if $n > |\alpha|$ **then accept**
  rate $x \in \Sigma^n$ und $y \in \Sigma^{\leq|\alpha|} - \Sigma^n$
  if $x \notin L(\alpha) \vee y \in L(\alpha)$ **then accept else reject end**

Als nächstes zeigen wir, dass 3-SAT auf L reduzierbar ist. Sei

$$F = \bigwedge_{j=1}^{m} C_j$$

mit $C_j \subseteq \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\}$ und $\|C_j\| \leq 3$. Betrachte die Reduktionsfunktion

$$f(F) = \underbrace{\alpha_1|\cdots|\alpha_m}_{\alpha_F}$$

mit $\alpha_j = \beta_{1j} \cdots \beta_{nj}$, wobei

$$\beta_{ij} = \begin{cases} 0, & x_i \text{ kommt in } C_j \text{ vor} \\ 1, & \bar{x}_i \text{ kommt in } C_j \text{ vor} \\ (0|1), & \text{sonst} \end{cases}$$

ist. Dann gilt $L(\alpha_j) = \{a \in \{0,1\}^n \mid C_j(a) = 0\}$ und daher folgt

$$\begin{aligned} F \notin \text{3-SAT} &\Leftrightarrow \forall a \in \{0,1\}^n : F(a) = 0 \\ &\Leftrightarrow \forall a \in \{0,1\}^n \exists j : C_j(a) = 0 \\ &\Leftrightarrow \forall a \in \{0,1\}^n \exists j : a \in L(\alpha_j) \\ &\Leftrightarrow L(\alpha_F) = \{0,1\}^n. \end{aligned}$$

■

Korollar 125
Das Äquivalenzproblem für sternfreie reguläre Ausdrücke,

$$\text{ÄP}_{\text{SFRA}} = \{\alpha\#\beta \mid \alpha, \beta \text{ sind sternfreie reguläre Ausdrücke mit } L(\alpha) = L(\beta)\}$$

ist co-NP-vollständig.

Beweis: Die Funktion

$$f(\alpha\#\mathit{bin}(n)) = \begin{cases} \underbrace{(0|1)\cdots(0|1)}_{n\text{-mal}}\#\alpha, & n \leq \alpha \\ 0\#1, & \text{sonst} \end{cases}$$

vermittelt eine Polynomialzeit-Reduktion von L auf $\overline{\text{ÄP}_{\text{SFRA}}}$. Dies zeigt, dass ÄP_{SFRA} co-NP-hart ist. Es ist auch leicht zu sehen, dass ÄP_{SFRA} in co-NP entscheidbar ist. ■

Es ist nicht schwer, mit derselben Beweistechnik auch das Äquivalenzproblem und das Inklusionsproblem für NFAs als co-NP-hart nachzuweisen (siehe Übungen). Tatsächlich sind diese Probleme sogar PSPACE-vollständig.

Korollar 126
Das Äquivalenzproblem für NFAs,

$$\text{ÄP}_{\text{NFA}} = \{M_1\#M_2 \mid M_1 \text{ und } M_2 \text{ sind NFAs mit } L(M_1) = L(M_2)\}$$

und das Inklusionsproblem für NFAs,

$$\text{IP}_{\text{NFA}} = \{M_1\#M_2 \mid M_1 \text{ und } M_2 \text{ sind NFAs mit } L(M_1) \subseteq L(M_2)\}$$

sind co-NP-hart.

Zum Abschluss betrachten wir das Problem, einen Rucksack der Größe v optimal mit einer Auswahl von k Gegenständen der Größe $u_1, \dots, u_k$ zu bepacken. Dieses Optimierungsproblem lässt sich leicht auf folgendes Entscheidungsproblem reduzieren.

Rucksack-Problem (Subset Sum):

Gegeben: Ein Vektor $(u_1, \dots, u_k, v, w) \in \mathbb{N}^{k+2}$ von natürlichen Zahlen.

Gefragt: Gibt es eine Auswahl $I \subseteq \{1, \dots, k\}$ mit $w \leq \sum_{i \in I} u_i \leq v$?

Satz 127
Rucksack ist NP-vollständig.

Beweis: Wir zeigen 3-SAT $\leq^p$ RUCKSACK. Sei

$$F = \bigwedge_{j=1}^{m} \underbrace{\bigvee_{l=1}^{k_j} L_{jl}}_{C_j}$$

mit $k_j \leq 3$ und $L_{jl} \in \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$. Betrachte die Reduktionsfunktion

$$f(F) = (u_1, \dots, u_n, \bar{u}_1, \dots, \bar{u}_n, v_1, \dots, v_m, w_1, \dots, w_m, v, v),$$

wobei u_i und $\bar{u}_i$ die Zahlen

$$u_i = b_{i1} \cdots b_{im} 0^{i-1} 1 0^{n-i-1} \text{ und } \bar{u}_i = \bar{b}_{i1} \cdots \bar{b}_{im} 0^{i-1} 1 0^{n-i-1} \text{ (zur Basis 10)}$$

mit

$$b_{ij} = \begin{cases} 1, & x_i \text{ kommt in } C_j \text{ vor} \\ 0, & \text{sonst} \end{cases} \quad \text{und} \quad \bar{b}_{ij} = \begin{cases} 1, & \bar{x}_i \text{ kommt in } C_j \text{ vor} \\ 0, & \text{sonst} \end{cases}$$

und $v_j = w_j = 0^{j-1} 1 0^{m-j-1}$ ist. Die Größe v des Rucksacks setzen wir auf

$$v = \underbrace{3 \cdots 3}_{m\text{-mal}} \underbrace{1 \cdots 1}_{n\text{-mal}}.$$

Für die Formel $F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$ erhalten wir beispielsweise den Vektor $(u_1, u_2, u_3, u_4, \bar{u}_1, \bar{u}_2, \bar{u}_3, \bar{u}_4, v_1, v_2, v_3, w_1, w_2, w_3, v)$ mit

$$\begin{array}{lll@{\qquad}lll} u_1 & = & 010\,1000 & \bar{u}_1 & = & 100\,1000 \\ u_2 & = & 011\,0100 & \bar{u}_2 & = & 001\,0100 \\ u_3 & = & 000\,0010 & \bar{u}_3 & = & 100\,0010 \\ u_4 & = & 110\,0001 & \bar{u}_4 & = & 001\,0001 \end{array}$$

und

$$\begin{array}{lll} v_1 = w_1 & = & 100\,0000 \\ v_2 = w_2 & = & 010\,0000 \\ v_3 = w_3 & = & 001\,0000 \end{array}$$

sowie $v = 333\,1111$. Da f in Polynomialzeit berechenbar ist, müssen wir nur noch die Äquivalenz

$$F \in \text{3-SAT} \;\Leftrightarrow\; f(F) \in \text{RUCKSACK}$$

zeigen, d.h. F ist genau dann erfüllbar, wenn es eine Auswahl von Gegenständen gibt, mit denen sich der Rucksack randvoll bepacken lässt.

„$\Rightarrow$“: Sei $a = a_1 \cdots a_n$ eine erfüllende Belegung für F. Dann ist

$$\sum_{a_i=1} u_i + \sum_{a_i=0} \bar{u}_i$$

eine Zahl der Form $b_1 \cdots b_m\, 1 \cdots 1$ mit $1 \leq b_j \leq 3$, da die Belegung a in jeder Klausel mindestens ein und höchstens drei Literale wahr macht. Durch Addition von

$$\sum_{b_j \leq 2} v_j + \sum_{b_j = 1} w_j$$

erhalten wir v.

„$\Leftarrow$“: Sei umgekehrt eine Lösung

$$P, N \subseteq \{1, \ldots, n\} \qquad I, J \subseteq \{1, \ldots, m\}$$

für die Rucksack-Instanz $f(F)$ mit

$$\sum_{i \in P} u_i + \sum_{i \in N} \bar{u}_i + \sum_{j \in I} v_j + \sum_{j \in J} w_j = v$$

gegeben. Dann gilt $P = \{1, \ldots, n\} - N$, da keine Überträge auftreten können. Ausserdem muss die Zahl $\sum_{i \in P} u_i + \sum_{i \in N} \bar{u}_i$ die Form $b_1 \cdots b_m\, 1 \cdots 1$ mit $b_j \geq 1$ haben, da die Restsumme $\sum_{j \in I} v_j + \sum_{j \in J} w_j$ die Form $c_1 \cdots c_m\, 0 \cdots 0$ mit $c_j \leq 2$ hat. Folglich enthält jede Klausel C_j mindestens ein Literal $L \in \{x_i \mid i \in P\} \cup \{\bar{x}_i \mid i \in N\}$, so dass die Belegung $a = a_1 \cdots a_n$ mit

$$a_i = \begin{cases} 1, & i \in P \\ 0, & i \in N \end{cases}$$

die Formel F erfüllt.

■