

Vorlesungsskript

Theoretische Informatik II

Wintersemester 2003/2004

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

21. Februar 2004

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Reguläre Sprachen | 1 |
| 1.1 | Endliche Automaten | 1 |
| 1.2 | Nichtdeterministische endliche Automaten | 5 |
| 1.3 | Reguläre Ausdrücke | 7 |
| 1.4 | Das Pumping-Lemma | 10 |
| 1.5 | Minimierung von DFAs | 11 |
| 1.6 | Grammatiken | 14 |
| 2 | Kontextfreie Sprachen | 19 |
| 2.1 | Äquivalenz von kontextfreien Grammatiken und Kellerautomaten | 19 |
| 2.2 | Chomsky-Normalform Grammatiken | 25 |
| 2.3 | Der CYK-Algorithmus | 28 |
| 2.4 | Das Pumping-Lemma | 30 |
| 2.5 | Deterministisch kontextfreie Sprachen | 34 |
| 3 | Kontextsensitive Sprachen | 39 |
| 3.1 | Kontextsensitive Grammatiken | 39 |
| 3.2 | Turingmaschinen | 40 |
| 3.3 | Linear beschränkte Automaten | 43 |
| 4 | Entscheidbare und rekursiv aufzählbare Sprachen | 46 |
| 4.1 | Kodierung (Gödelisierung) von TMs | 49 |
| 4.2 | Das Postsche Korrespondenzproblem (PCP) | 52 |
| 5 | Komplexitätsklassen und \mathcal{NP}-Vollständigkeit | 59 |
| 5.1 | Zeitkomplexität | 59 |
| 5.2 | \mathcal{NP} -Vollständigkeit und das P=NP Problem | 60 |

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.

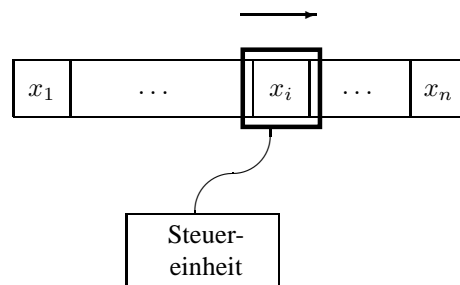
Definition 1 (Alphabet)

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

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

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 konstant 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**,
- $\delta : Z \times \Sigma \rightarrow Z$ die **Überföhrungsfunktion**,
- $q_0 \in Z$ der **Startzustand** und
- $E \subseteq Z$ die Menge der **Endzustände** ist.

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

$$L(M) = \{x_1 \cdots x_n \in \Sigma^* \mid \exists q_1, \dots, q_{n-1} \in Z \exists q_n \in E : \delta(q_i, x_{i+1}) = q_{i+1} \text{ für } i = 0, \dots, n-1\}.$$

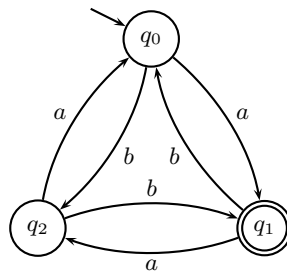
Beispiel 3 Betrachte den DFA $M = (Z, \Sigma, \delta, q_0, E)$ mit

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

und der Überföhrungsfunktion

| 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

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

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

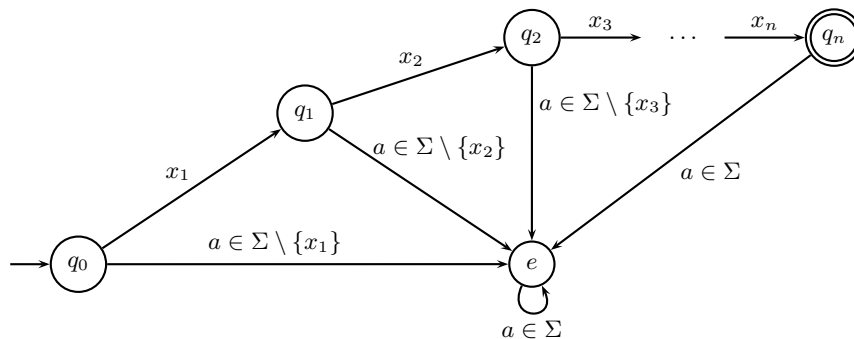
Eine von einem DFA akzeptierte Sprache wird als **regulär** bezeichnet. Die zugehörige Sprachklasse ist

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

Frage: Welche Sprachen gehören zu \mathcal{REG} und welche nicht?

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

Beobachtung 1: Alle Sprachen, die aus einem einzigen Wort $x = x_1 \dots x_n \in \Sigma^*$ 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

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

und der Überföhrungsfunktion

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

beschreiben.

Beobachtung 2: Mit $L_1, L_2 \in \mathcal{REG}$ 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

$$\overline{M} = (Z_1 \times Z_2, \Sigma, \delta, q_0, E_1 \times E_2)$$

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 Automat aus?)

Aus den beiden Beobachtungen folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da weder die in Beispiel 3 betrachtete Sprache noch ihr Komplement endlich sind, haben wir damit allerdings noch nicht alle regulären Sprachen erfasst. Es stellt sich daher die Frage, ob \mathcal{REG} auch unter anderen 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

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

definiert.

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. Später werden wir dann nachweisen, dass auch nichtdeterministische endliche Automaten nur reguläre Sprachen akzeptieren können.

1.2 Nichtdeterministische endliche Automaten

Definition 4 (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 Überföhrungsfunktion

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

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

$$L(M) = \{x_1 \cdots x_n \in \Sigma^* \mid \exists q_0 \in Q_0 \exists q_1, \dots, q_{n-1} \in Z \exists q_n \in E : \\ q_{i+1} \in \delta(q_i, x_{i+1}) \text{ f\u00fcr } i = 0, \dots, n-1\}$$

Ein NFA kann also nicht nur eine, sondern mehrere verschiedene Rechnungen ausföhren. Die Eingabe geh\u00f6rt 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 \u00dcberf\u00f6hrungsfunktion auf der gesamten Menge $Z \times \Sigma$ definiert ist, kann ein NFA „stecken bleiben“. Falls er n\u00e4mlich einen Zustand q erreicht, in dem das n\u00e4chste Eingabezeichen x_i wegen $\delta(q, x_i) = \emptyset$ nicht gelesen werden kann.

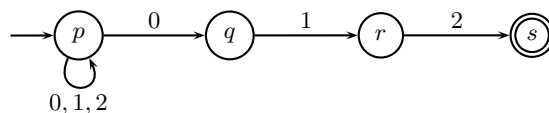
Beispiel 5 Betrachte den NFA $M = (Z, \Sigma, \delta, Q_0, E)$ mit

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

und der \u00dcberf\u00f6hrungsfunktion

| 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

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

Beobachtung 3: Sind $L_1, L_2 \in \mathcal{REG}$, so werden die Sprachen L_1L_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 6 $\mathcal{REG} = \{L(M) \mid M \text{ ist ein NFA}\}$.

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

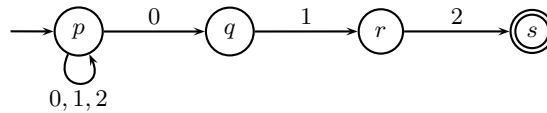
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \emptyset\}.$$

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

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

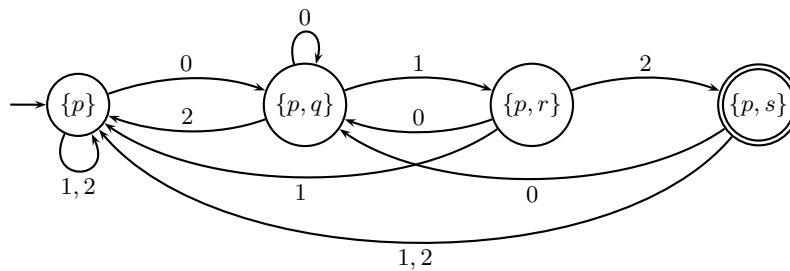
■

Beispiel 7 Für den NFA $M = (Z, \Sigma, \delta, Q_0, E)$ aus Beispiel 5



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 |



1.3 Reguläre Ausdrücke

Wir haben uns davon überzeugt, dass alle Sprachen, die aus einem einzelnen Wort bestehen, regulär sind, und dass sich daraus mittels der Operationen Durchschnitt, Vereinigung, Komplement, Produkt und Sternhülle weitere reguläre Sprachen bilden lassen. Wie wir gleich sehen werden, lassen sich auf diese Weise sogar alle regulären Sprachen erhalten. Abgesehen von der leeren Menge gilt dies auch dann noch, wenn auf die Durchschnitts- und Komplementbildung verzichtet wird.

Definition 8 (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 $L(\epsilon) = \{\epsilon\}$ 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 9 Über $\Sigma = \{0, 1\}$ sind ϵ , $(0|1)^*00$ und $(\epsilon 0|\emptyset 1)$ reguläre Ausdrücke, die die Sprachen $\{\epsilon\}$, $\{x00 \mid x \in \Sigma^*\}$ 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 10 $\mathcal{REG} = \{L(\gamma) \mid \gamma \text{ ist ein regulärer Ausdruck}\}$.

Beweis: Die Inklusion von rechts nach links folgt aus den Beobachtungen 1 bis 3, zusammen mit Satz 6. 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 $\hat{\delta}$:

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

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

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

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

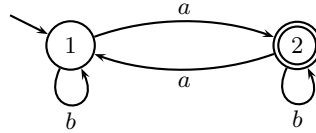
$$L_{p,q}^0 = \begin{cases} \{a \in \Sigma \mid \delta(p, a) = q\} \cup \{\epsilon\}, & p = q, \\ \{a \in \Sigma \mid \delta(p, a) = q\}, & \text{sonst} \end{cases}$$

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

$$\begin{aligned} L_{p,q}^{r+1} &= L_{p,q}^r \cup L_{p,r+1}^r (L_{r+1,r+1}^r)^* L_{r+1,q}^r \\ &= L(\gamma_{p,q}^r \mid \gamma_{p,r+1}^r (\gamma_{r+1,r+1}^r)^* \gamma_{r+1,q}^r) \end{aligned}$$

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

Beispiel 11 Die von dem DFA



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

| r | p, q | | | |
|-----|--------------|--|------|---|
| | 1, 1 | 1, 2 | 2, 1 | 2, 2 |
| 0 | ϵb | a | a | ϵb |
| 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 12 (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. $uv^i w \in L$ 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 $\hat{\delta}(q, v) = q$

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

ist. ■

Um also $L \notin \mathcal{REG}$ 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 12 aufgeführten Eigenschaften verletzt ist.

Beispiel 13 *Die Sprache*

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

ist nicht regulär, da $a^l b^l$ die Länge $2l \geq l$ hat und offensichtlich nicht in Teilwörter u, v, w mit $v \neq \varepsilon$ und $uv^2 w \in L$ zerlegbar ist.

Beispiel 14 *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 15 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: Mit dem Pumping-Lemma können nicht alle Sprachen $L \notin \mathcal{REG}$ 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 immer davon aus, dass M keine unerreichbaren Zustände enthält. Offensichtlich können zwei Zustände q und p zu einem Zustand verschmolzen werden, 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 16 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:

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

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} die Zustände $\tilde{q}_0, \tilde{q}_1, \dots, \tilde{q}_n$. 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 $\{L_x \mid x \in \Sigma^*\} = \{L_q \mid q \in Z\}$ und somit hängt

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

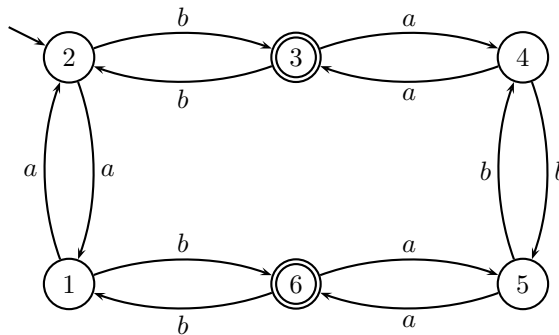
nur von L ab. ■

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

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

schrittweise um alle Paare $\{q, p\}$ erweitert, für die eines der Paare $\{\delta(q, a), \delta(p, a)\}$, $a \in \Sigma$, bereits zu V gehört. Wenn sich auf diese Weise keine weiteren Paare mehr zu V hinzufügen lassen, können alle Zustände q, p mit $\{q, p\} \notin V$ verschmolzen werden.

Beispiel 17 Betrachte den DFA



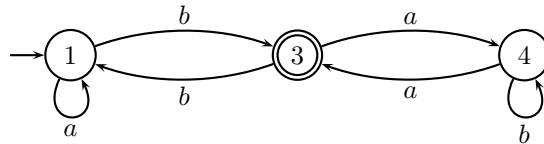
Zu Beginn enthält V die Paare $\{1, 3\}, \{2, 3\}, \{4, 3\}, \{5, 3\}, \{1, 6\}, \{2, 6\}, \{4, 6\}, \{5, 6\}$. Wegen

$$\frac{\{q, p\}}{\{\delta(q, a), \delta(p, a)\}} \left| \begin{array}{cccc} \{1, 4\} & \{1, 5\} & \{2, 4\} & \{2, 5\} \\ \{2, 3\} & \{2, 6\} & \{1, 3\} & \{1, 6\} \end{array} \right.$$

können die Paare $\{1, 4\}, \{1, 5\}, \{2, 4\}, \{2, 5\}$ zu V hinzugefügt werden. Da die verbliebenen Paare $\{1, 2\}, \{4, 5\}, \{3, 6\}$ wegen

$$\frac{\{q, p\}}{\{\delta(q, a), \delta(p, a)\} \quad \{\delta(q, b), \delta(p, b)\}} \left| \begin{array}{ccc} \{1, 2\} & \{4, 5\} & \{3, 6\} \\ \{1, 2\} & \{3, 6\} & \{4, 5\} \\ \{3, 6\} & \{4, 5\} & \{1, 2\} \end{array} \right.$$

nun nicht mehr zu V hinzugefügt werden können, können die entsprechenden Zustände 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

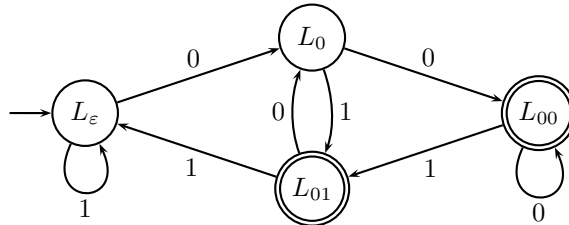
$$\begin{aligned} Z &= \{L_x \mid x \in \Sigma^*\}, \\ E &= \{L_x \mid x \in L\} \text{ und} \\ \delta(L_x, a) &= L_{xa}, \end{aligned}$$

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

Beispiel 18 Sei $L = \{x_1 \cdots x_n \mid x_i \in \{0, 1\} \text{ für } i = 1, \dots, n \text{ und } x_{n-1} = 0\}$. 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

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

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:

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

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

Beispiel 19 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 20 Sei Σ ein Alphabet. Dann lassen sich alle regulären Ausdrücke über Σ unter Gebrauch der folgenden Regeln aus dem Symbol R erzeugen:

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

Definition 21 (Grammatik)

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

- V eine endliche Menge von **Variablen** (oder **Nichtterminalsymbolen**),
- 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:

$$u \Rightarrow_G v, \text{ falls } x, y, y', z \in (V \cup \Sigma)^* \text{ existieren mit}$$

- $u = xyz$,
- $y \rightarrow y'$ und
- $v = xy'z$.

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 $u \Rightarrow_G^n v$. Die durch G **erzeugte Sprache** ist

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

Beispiel 22 (Fortsetzung) Eine Grammatik, die die Menge der regulären Ausdrücke über einem Alphabet Σ erzeugt, ist beispielsweise $G = (\{R\}, \Sigma \cup \{\emptyset, \epsilon, (,), *, |, \}, P, R)$, wobei P die oben angegebenen Regeln enthält.

Man unterscheidet vier verschiedene Typen von Grammatiken.

Definition 23 (Typ einer Grammatik)

Sei $G = (V, \Sigma, P, S)$ eine Grammatik.

1. G heißt **vom Typ 3** oder **regulär**, falls für alle Regeln $u \rightarrow v$ 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 $u \rightarrow v$ gilt: $u \in V$.

3. G heißt **vom Typ 1** oder **kontextsensitiv**, falls für alle Regeln $u \rightarrow v$ gilt: $|v| \geq |u|$ (mit Ausnahme der ε -Sonderregelung, siehe unten).
4. Jede Grammatik ist automatisch **vom Typ 0**.

ε -Sonderregelung: Bei einer kontextsensitiven Grammatik ist auch die Regel $S \rightarrow \varepsilon$ 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

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

(*context free languages*) und

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

(*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

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

Satz 24 $\mathcal{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}.$

Beweis: Sei $L \in \mathcal{REG}$ 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

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

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

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

Für die entgegengesetzte Inklusion sei nun $G = (V, \Sigma, P, A_0)$ eine reguläre Grammatik. Dann wird die Sprache $L = L(G)$ auch von der Grammatik $G' = (V', \Sigma, P', A_0)$ mit

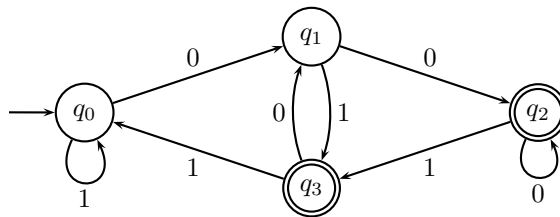
$$\begin{aligned} V' &= V \cup \{X_{neu}\}, \\ P' &= \{A \rightarrow aX_{neu} \mid A \rightarrow_G a\} \cup \{X_{neu} \rightarrow \varepsilon\} \cup P \setminus (V \times \Sigma) \end{aligned}$$

erzeugt. Da G' keine Produktionen der Form $A \rightarrow a$ mehr enthält, können wir die gerade beschriebene Konstruktion einer Grammatik aus einem DFA „umdrehen“, um ausgehend von G' einen NFA $M' = (Z', \Sigma, \delta', \{A_0\}, E')$ mit

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

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

Beispiel 25 Der DFA



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

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

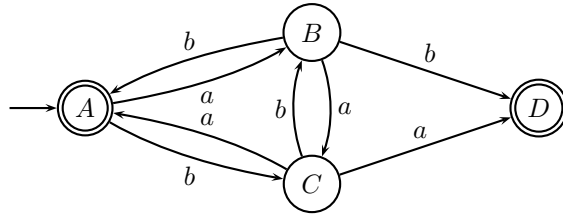
Umgekehrt führt die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ mit

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

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

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

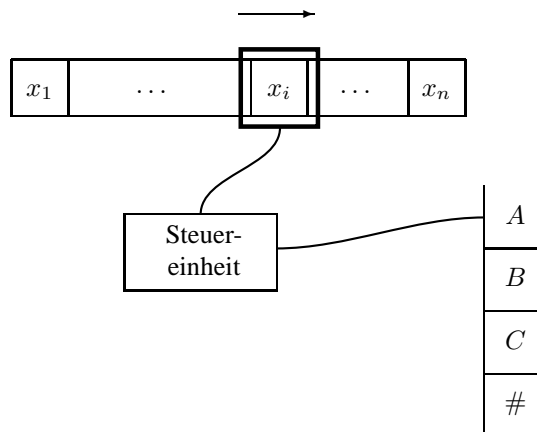
auf den NFA



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 26 (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**,
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Z \times \Gamma^*)$ die **Überföhrungsfunktion** ($\mathcal{P}_e(M) = \{A \subseteq M \mid A \text{ ist endlich}\}$) und
- $\# \in \Gamma$ das **Kelleranfangszeichen** ist.

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 und im Keller das Zeichen A durch die Zeichenfolge $B_1 \cdots B_k$ ersetzen. Hierfür schreiben wir auch kurz $quA \rightarrow pB_1 \cdots B_k$. 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). $K' = (p, x_j \cdots x_n, B_1 \cdots B_k A_2 \cdots A_l)$ heißt **Folgekonfiguration** von K (kurz: $K \vdash K'$), falls

- $qx_i A_1 \rightarrow pB_1 \cdots B_k$ und $j = i + 1$ oder
- $q\varepsilon A_1 \rightarrow pB_1 \cdots B_k$ und $j = i$ ist.

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

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

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

$$\begin{aligned} \delta : \quad q_0 \varepsilon \# &\rightarrow q_1 & (1) \\ q_0 a \# &\rightarrow q_0 A & (2) \\ q_0 a A &\rightarrow q_0 A A & (3) \\ q_0 b A &\rightarrow q_1 & (4) \\ q_1 b A &\rightarrow q_1 & (5) \end{aligned}$$

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:

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

mit $x_1 = \dots = x_n = a$ 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 $x_{n+2} = \dots = x_{2n} = b$ 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. Hierzu benötigen wir den Begriff der (Links-)ableitung.

Definition 28 (Linksableitung)

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik. Dann heißt

$$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_m$$

Ableitung von x (der Länge m), wenn $\alpha_0 = S$ und $\alpha_m = x$ ist. Wird in jedem Ableitungsschritt $\alpha_i \Rightarrow \alpha_{i+1}$ die am weitesten links stehende Variable in α_i ersetzt, so spricht man von einer **Linksableitung**.

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 29 Sei $G = (\{S\}, \{a\}, \{S \rightarrow SSS, a\}, S)$. Dann ist

$$S \Rightarrow SSS \Rightarrow SaS \Rightarrow aaS \Rightarrow aaa$$

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

$$S \Rightarrow SSS \Rightarrow aSS \Rightarrow aaS \Rightarrow aaa.$$

Satz 30 $\mathcal{CFL} = \{L(M) \mid M \text{ ist ein PDA}\}$.

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

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

Dann gilt

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

F\"ur 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 $L(G) = L$:

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

Die Gleichheit $L(G) = L(M)$ folgt nun unmittelbar aus der \u00c4quivalenz

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

deren Beweis wir gleich nachholen:

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

Beweis der \u00c4quivalenz. Wir f\"uhren den Beweis durch Induktion \u00fcber 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 \u00c4quivalenz erf\u00fcllt.

$m \rightsquigarrow m + 1$: Seien $q, p \in Z$, $x \in \Sigma^*$ und $A \in \Gamma$ gegeben. Wir m\u00fcssen 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 L\u00e4nge $m + 1$. 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 $p_1, \dots, p_k \in Z$ 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 ($i = 1, \dots, k$) 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

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

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

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

Ist nun umgekehrt $(q, x, A) \vdash^{m+1} (p, \varepsilon, \varepsilon)$ eine Rechnung der Länge $m + 1$, so sei $q u A \rightarrow p_1 B_1 \cdots B_k$ 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 $i = 2, \dots, k$ 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 $X_{qAp_{k+1}} \rightarrow uX_{p_1B_1p_2} \cdots X_{p_kB_kp_{k+1}}$ und andererseits gilt

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

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

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

die wir zu der gesuchten Ableitung von x zusammensetzen können:

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

■

Beispiel 31 Sei $G = (\{S\}, \{a\}, P, S)$ mit

$$\begin{aligned} P : \quad S &\rightarrow SSS & (1) \\ S &\rightarrow a & (2) \end{aligned}$$

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

$$\begin{aligned} \delta : \quad q\varepsilon S &\rightarrow qSSS & (1') \\ q\varepsilon S &\rightarrow qa & (2') \\ qaa &\rightarrow q\varepsilon & (3) \end{aligned}$$

Der Linksableitung

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

entspricht dann die Rechnung

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

Beispiel 32 Ist umgekehrt $M = (\{q_0, q_1\}, \{a, b\}, \{A, \#\}, \delta, q_0, \#)$ mit

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

der PDA aus Beispiel 27, so erhalten wir daraus die Grammatik $G = (V, \Sigma, P, S)$ mit

$$\begin{aligned} V &= \{S, X_{q_0\#q_0}, X_{q_0\#q_1}, X_{q_1\#q_0}, X_{q_1\#q_1}, X_{q_0Aq_0}, X_{q_0Aq_1}, X_{q_1Aq_0}, X_{q_1Aq_1}\} \text{ und} \\ P : \quad S &\rightarrow X_{q_0\#q_0}, X_{q_0\#q_1} & (0, 0') \\ X_{q_0\#q_1} &\rightarrow \varepsilon & (1') \\ X_{q_0\#q_0} &\rightarrow aX_{q_0Aq_0} & (2') \\ X_{q_0\#q_1} &\rightarrow aX_{q_0Aq_1} & (2'') \\ X_{q_0Aq_0} &\rightarrow aX_{q_0Aq_0}X_{q_0Aq_0}, aX_{q_0Aq_1}X_{q_1Aq_0} & (3', 3'') \\ X_{q_0Aq_1} &\rightarrow aX_{q_0Aq_0}X_{q_0Aq_1}, aX_{q_0Aq_1}X_{q_1Aq_1} & (3''', 3''''') \\ X_{q_0Aq_1} &\rightarrow b & (4') \\ X_{q_1Aq_1} &\rightarrow b & (5'). \end{aligned}$$

Der Rechnung

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

entspricht dann die Linksableitung

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

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 33 (Chomsky-Normalform)

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

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

Satz 34 Sei $L \in \mathcal{CFL}$. 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:

```

1  Eingabe:  $G = (V, \Sigma, P, S)$ 
2   $V'' \leftarrow \{A \in V \mid A \rightarrow_G \varepsilon\}$ 
3   $V' \leftarrow \emptyset$ 
4  while  $V' \neq V''$  do
5     $V' \leftarrow V''$ 
6     $V'' \leftarrow V' \cup \{A \in V \mid \exists B_1, \dots, B_k \in V' : A \rightarrow_G B_1 \cdots B_k\}$ 
7  end
8  Ausgabe:  $V'$ 

```

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

■

Beispiel 35 Betrachte die Grammatik $G = (\{S, A, B, C, D, E\}, \{a, b, c\}, P, S)$ mit

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

1. Schritt: *Berechnung von V' :*

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

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

$P : S \rightarrow aB, bA, C$
 $A \rightarrow a, aS, bAA$
 $B \rightarrow b, bS, aBB$
 $C \rightarrow c, S, D, cC$
 $D \rightarrow E$
 $E \rightarrow abc.$

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

Satz 36 Für jede Sprache $L \in \mathcal{CFL}$ 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 $A_1, \dots, A_k \in V$ existieren mit $A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_k \rightarrow A_1$, 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 $A \rightarrow B$ in P existieren, wähle eine solche Regel, für die keine Variable $D \in V$ existiert mit $B \rightarrow D$. Entferne $A \rightarrow B$ aus P und füge für jede in P vorkommende Regel $B \rightarrow \alpha$ die Regel $A \rightarrow \alpha$ zu P hinzu.

■

Beispiel 37 (Fortsetzung)

1. Schritt: Entfernung des Zyklus' $S \rightarrow C \rightarrow S$:

Entferne die beiden Regeln $S \rightarrow C$ und $C \rightarrow S$ und ersetze alle Vorkommen von C durch S :

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

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

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

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

Satz 38 Sei $L \in \mathcal{CFL}$. 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 $X_a \rightarrow a$ 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 $A \rightarrow B_1 \cdots B_k$ mit $k \geq 3$ durch die $k - 1$ neuen Regeln $A \rightarrow B_1 C_1$, $C_1 \rightarrow B_2 C_2$, \dots , $C_{k-3} \rightarrow B_{k-2} C_{k-2}$ und $C_{k-2} \rightarrow B_{k-1} B_k$, wobei C_1, \dots, C_{k-2} neue Variablen sind.

■

Beispiel 39 (Fortsetzung)

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 $X_a \rightarrow a$, $X_b \rightarrow b$ und $X_c \rightarrow c$.

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

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

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

2.3 Der CYK-Algorithmus

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

Satz 40 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 $j = 1, \dots, n$ können die Mengen $V_{i,j}$, $i = 1, \dots, n - j + 1$, 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 $k \in \{1, \dots, j-1\}$ 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 $k \in \{1, \dots, j-1\}$ 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 41 Für die CNF-Grammatik aus dem vorigen Beispiel mit den Produktionen

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

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 | $\{S\}$ | | | | | |

2.4 Das Pumping-Lemma

Für unsere zweite Anwendung der Chomsky-Normalform — das Pumping-Lemma — benötigen wir noch den Begriff des Syntaxbaums.

Definition 42 (Syntaxbaum)

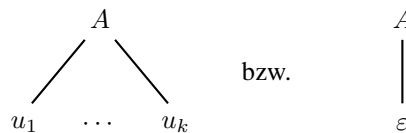
Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik und sei

$$S = \alpha_0 \Rightarrow \alpha_1 \cdots \Rightarrow \alpha_m = x_1 \cdots x_n = x$$

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

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

$T_i, i \geq 1$: Sei $A \rightarrow u_1 \cdots u_k$ mit $u_i \in \Sigma \cup V$ (bzw. $A \rightarrow \varepsilon$) die Regel, die im i -ten Ableitungsschritt $\alpha_{i-1} \Rightarrow \alpha_i$ angewandt wird. Dann erhalten wir den Baum T_i , indem wir im Baum T_{i-1} das zur Variablen A gehörige Blatt durch den Unterbaum

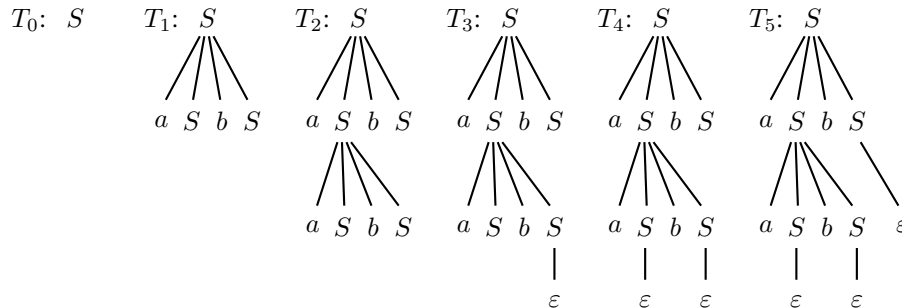


ersetzen.

Beispiel 43 Betrachte die Grammatik $G = (\{S\}, \{a, b, c\}, \{S \rightarrow aSbS, \varepsilon\}, S)$ und die Ableitung

$$\underbrace{S}_{\alpha_0} \Rightarrow \underbrace{aSbS}_{\alpha_1} \Rightarrow \underbrace{aaSbSbS}_{\alpha_2} \Rightarrow \underbrace{aaSbbS}_{\alpha_3} \Rightarrow \underbrace{aabbS}_{\alpha_4} \Rightarrow \underbrace{aabb}_{\alpha_5}$$

Die zugehörigen Syntaxbäume sind dann

**Bemerkungen:**

1. Verschiedene Ableitungen können durchaus auf denselben Syntaxbaum führen. Unterschiedliche Linksableitungen führen jedoch immer auf unterschiedliche Syntaxbäume. Deshalb entspricht jedem Syntaxbaum eindeutig eine Linksableitung und umgekehrt.
2. Ist G in Chomsky-Normalform, so hat jeder Syntaxbaum T eine spezielle Form: Jeder Knoten besitzt höchstens zwei Nachfolger, d.h. T ist ein Binärbaum.

Für den Beweis des Pumping-Lemmas benötigen wir noch die folgende Beziehung zwischen der Blätterzahl eines Binärbaumes und der Länge seiner Pfade.

Lemma 44 *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.

$k \rightsquigarrow k + 1$: 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 45 *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 46 (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 $L \in \mathcal{CFL}$. 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 $k = \lceil \log \|V\| \rceil$ 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 $A \rightarrow BC$ und genau n Regeln der Form $A \rightarrow a$ angewandt, die Länge der Ableitung ist also $m = 2n - 1$.

Dabei können wir annehmen, dass zuerst die Regeln der Form $A \rightarrow BC$ und erst danach die Regeln der Form $A \rightarrow a$ 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_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 $z = uvwx^iy$, 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 47 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 48 Die Klasse \mathcal{CFL} ist abgeschlossen unter

- Vereinigung
- Produkt
- Sternhülle,

aber nicht unter

- Durchschnitt und
- Komplement.

Beweis: Seien $L_1, L_2 \in \mathcal{CFL}$ 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

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

wobei S eine neue Variable ist:

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

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 \mathcal{CFL} nicht unter Durchschnitt abgeschlossen. Da \mathcal{CFL} unter Vereinigung abgeschlossen ist, kann \mathcal{CFL} wegen de Morgan dann auch nicht unter Komplementbildung abgeschlossen sein. ■

2.5 Deterministisch kontextfreie Sprachen

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

Definition 49 (deterministischer Kellerautomat)

Ein **deterministischer Kellerautomat** (kurz: DPDA; *deterministic push-down 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

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

Weiter sei

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

(*deterministic context free languages*).

Beispiel 50 Der DPDA $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#, \{q_2\})$ mit

$$\begin{array}{lll} \delta : & q_0 a \# \rightarrow q_0 A \# & q_0 c A \rightarrow q_1 A \\ & q_0 b \# \rightarrow q_0 B \# & q_0 c B \rightarrow q_1 B \\ & q_0 a A \rightarrow q_0 A A & q_1 a A \rightarrow q_1 \\ & q_0 b A \rightarrow q_0 B A & q_1 b B \rightarrow q_1 \\ & q_0 a B \rightarrow q_0 A B & q_1 \varepsilon \# \rightarrow q_2 \\ & q_0 b B \rightarrow q_0 B B & q_2 \varepsilon \# \rightarrow q_2 \# \# \end{array}$$

oder in Tabellenform

| δ | ε | a | b | c |
|-----------|---------------|------------|------------|---------|
| $q_0, \#$ | — | $q_0 A \#$ | $q_0 B \#$ | — |
| q_0, A | — | $q_0 A A$ | $q_0 B A$ | $q_1 A$ |
| q_0, B | — | $q_0 A B$ | $q_0 B B$ | $q_1 B$ |
| $q_1, \#$ | q_2 | — | — | — |
| q_1, A | — | q_1 | — | — |
| q_1, B | — | — | q_1 | — |
| $q_2, \#$ | $q_2 \# \#$ | — | — | — |
| q_2, A | — | — | — | — |
| q_2, B | — | — | — | — |

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

Im Gegensatz zur Klasse \mathcal{CFL} der kontextfreien Sprachen ist \mathcal{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 $x \in L(M)$ 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 51 Jede Sprache $L \in \mathcal{DCFL}$ 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 $L(M) = L$. Betrachte den DPDA

$$M' = (Z \cup \{q'_0, q_+, q_-\}, \Sigma, \underbrace{\Gamma \cup \{\square\}}_{\Gamma'}, \delta', q'_0, \#, E \cup \{q_+\}),$$

wobei δ' neben den Anweisungen aus δ (soweit sie nicht durch Anweisungen vom Typ 4 oder 5 überschrieben werden) die folgenden Anweisungen enthält:

- 1) $q'_0 \varepsilon \# \rightarrow q_0 \# \square$,
- 2) $q a A \rightarrow q_- A$, für alle $(q, a, A) \in Z \times \Sigma \times \Gamma'$ mit $A = \square$ oder $\delta(q, a, A) = \delta(q, \varepsilon, A) = \emptyset$,
- 3) $q_- a A \rightarrow q_- A$, für alle $a \in \Sigma$ und $A \in \Gamma'$,
- 4) $q \varepsilon A \rightarrow q_- A$, 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,
- 5) $q \varepsilon A \rightarrow q_+ A$, für alle $q \in Z$ und $A \in \Gamma$, so dass ausgehend von der Konfiguration (q, ε, A) unendlich viele ε -Übergänge ausgeführt werden und dabei mindestens einmal ein Endzustand angenommen wird,
- 6) $q_+ \varepsilon A \rightarrow q_- A$, für alle $A \in \Gamma$

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

Beispiel 52 Wenden wir diese Konstruktion auf den DPDA M aus unserem Beispiel an, so erhalten wir folgenden DPDA

$$M' = (\{q_0, q_1, q_2, q'_0, q_+, q_-\}, \{a, b, c\}, \{A, B, \#, \square\}, \delta', q'_0, \#, \{q_2, q_+\})$$

mit der Überföhrungsfunktion

| δ' | ε | a | b | c | Typ |
|-----------------|------------------|---------------|---------------|---------------|------|
| $q'_0, \#$ | $q_0 \# \square$ | – | – | – | 1 |
| q'_0, A | – | – | – | – | – |
| q'_0, B | – | – | – | – | – |
| q'_0, \square | – | – | – | – | – |
| $q_0, \#$ | – | $q_0 A \#$ | $q_0 B \#$ | $q_- \#$ | 0, 2 |
| q_0, A | – | $q_0 AA$ | $q_0 BA$ | $q_1 A$ | 0 |
| q_0, B | – | $q_0 AB$ | $q_0 BB$ | $q_1 B$ | 0 |
| q_0, \square | – | $q_- \square$ | $q_- \square$ | $q_- \square$ | 2 |
| $q_1, \#$ | q_2 | – | – | – | 0 |
| q_1, A | – | q_1 | $q_- A$ | $q_- A$ | 0, 2 |
| q_1, B | – | $q_- B$ | $q_- B$ | $q_- B$ | 2 |
| q_1, \square | – | $q_- \square$ | $q_- \square$ | $q_- \square$ | 2 |
| $q_2, \#$ | $q_+ \#$ | – | – | – | 5 |
| q_2, A | – | $q_- A$ | $q_- A$ | $q_- A$ | 2 |
| q_2, B | – | $q_- B$ | $q_- B$ | $q_- B$ | 2 |
| q_2, \square | – | $q_- \square$ | $q_- \square$ | $q_- \square$ | 2 |
| $q_-, \#$ | – | $q_- \#$ | $q_- \#$ | $q_- \#$ | 3 |
| q_-, A | – | $q_- A$ | $q_- A$ | $q_- A$ | 3 |
| q_-, B | – | $q_- B$ | $q_- B$ | $q_- B$ | 3 |
| q_-, \square | – | $q_- \square$ | $q_- \square$ | $q_- \square$ | 3 |
| $q_+, \#$ | $q_- \#$ | – | – | – | 6 |
| q_+, A | $q_- A$ | – | – | – | 6 |
| q_+, B | $q_- B$ | – | – | – | 6 |
| q_+, \square | – | – | – | – | – |

Die aus δ übernommenen Anweisungen sind hier mit Typ 0 bezeichnet.

Satz 53 Die Klasse $\mathcal{DCF}\mathcal{L}$ 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

$$q'_0 = \begin{cases} (q_0, 1), & q_0 \notin E, \\ (q_0, 2), & \text{sonst,} \end{cases}$$

und δ' die folgenden Anweisungen enthält.

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

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

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

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

Auf Grund dieser Konstruktion kann man sich nun davon überzeugen, dass M' tatsächlich das Komplement von L erkennt. ■

Satz 54 Die Klasse $\mathcal{DCF}\mathcal{L}$ ist nicht abgeschlossen unter

- Durchschnitt
- Vereinigung
- Produkt und
- Sternhülle.

Beweis: • Die im Beweis von Satz 48 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 $\mathcal{DCF}\mathcal{L}$ unter Komplementbildung abgeschlossen ist, kann $\mathcal{DCF}\mathcal{L}$ wegen de Morgan dann auch nicht unter Vereinigung abgeschlossen sein.

- Um zu zeigen, dass \mathcal{DCFL} nicht unter Produktbildung abgeschlossen ist, betrachten wir zunächst die beiden deterministisch kontextfreien Sprachen $L_3 = \{a^i b^j c^k \mid i, j, k \geq 1, i \neq j\}$ und $L_4 = \{a^i b^j c^k \mid i, j, k \geq 1, j \neq k\}$. Dann gilt zwar $L = L_0 L_3 \cup L_4 \in \mathcal{DCFL}$, wobei $L_0 = \{0\}$ ist. Da \mathcal{DCFL} unter Durchschnittsbildung mit regulären Sprachen abgeschlossen ist (siehe Übungen), gilt aber

$$L_3 \cup L_4 \notin \mathcal{DCFL}, \quad (2.1)$$

da sonst auch $(\overline{L_3 \cup L_4}) \cap L(a^* b^* c^*) = \{a^n b^n c^n \mid n \geq 0\}$ (deterministisch) kontextfrei sein müsste. Wir zeigen nun, dass das Produkt aus der regulären Sprache L_0^* und L nicht deterministisch kontextfrei ist.

Behauptung: $L_0^* L \notin \mathcal{DCFL}$.

Nehmen wir das Gegenteil an, so wäre auch die Sprache

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

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

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

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 \mathcal{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 \mathcal{REG} , \mathcal{DCFL} und \mathcal{CFL} zusammen.

| | Vereinigung | Durchschnitt | Komplement | Produkt | Sternhülle |
|------------------|-------------|--------------|------------|---------|------------|
| \mathcal{REG} | ja | ja | ja | ja | ja |
| \mathcal{DCFL} | nein | nein | ja | nein | nein |
| \mathcal{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 $\alpha \rightarrow \beta$ gilt: $|\beta| \geq |\alpha|$. Die einzige Ausnahme hiervon bildet die Regel $S \rightarrow \varepsilon$, die allerdings nur dann benutzt werden darf, wenn das Startsymbol S nicht auf der rechten Seite einer Regel vorkommt.

Beispiel 55 Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit

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

und den Regeln

$$\begin{aligned} P: \quad S &\rightarrow aSBC, aBC && (1), (2) \\ CB &\rightarrow BC && (3) \\ aB &\rightarrow ab && (4) \\ bB &\rightarrow bb && (5) \\ bC &\rightarrow bc && (6) \\ cC &\rightarrow cc && (7) \end{aligned}$$

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

$$\begin{aligned} S &\stackrel{(1)}{\Rightarrow} aSBC \stackrel{(2)}{\Rightarrow} aaBCBC \stackrel{(3)}{\Rightarrow} aaBBCC \\ &\stackrel{(4)}{\Rightarrow} aabBCC \stackrel{(5)}{\Rightarrow} aabbCC \stackrel{(6)}{\Rightarrow} aabbcC \stackrel{(7)}{\Rightarrow} aabbc \end{aligned}$$

Allgemein gilt für alle $n \geq 1$:

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

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 $w = a^n b^n c^n$ abgeleitet 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 \mathcal{CFL} der kontextfreien Sprachen echt in \mathcal{CSL} enthalten.

3.2 Turingmaschinen

Wir führen nun das Rechenmodell der Turingmaschine (TM) 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 56 (k-Band-Turingmaschine)

Eine **nichtdeterministische Turingmaschine** (kurz 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,

- Γ das Arbeitsalphabet (mit $\Sigma \subseteq \Gamma$ und $\square \in \Gamma \setminus \Sigma$),
- $\delta: Z \times \Gamma^k \rightarrow \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ für ein $k \geq 1$ die Überföhrungsfunktion,
- q_0 der Startzustand und
- $E \subseteq Z$ die Menge der Endzustände ist.

Eine NTM M heißt **deterministische Turingmaschine** (kurz DTM), falls $\|\delta(q, a_1, \dots, a_k)\| \leq 1$ für alle $(q, a_1, \dots, a_k) \in Z \times \Gamma^k$ gilt.

Das Arbeitsalphabet Γ muss also zumindest alle Zeichen aus Σ und zusätzlich das **Blank** \square enthalten. Die Anzahl der Bänder einer TM M wird durch den Parameter k festgelegt. Wir bezeichnen dann M auch als k -Band-Turingmaschine (kurz k -TM bzw. k -NTM oder k -DTM).

Für

$$(q', a'_1, \dots, a'_k, D_1, \dots, D_k) \in \delta(q, a_1, \dots, a_k)$$

schreiben wir auch $(q, a_1, \dots, a_k) \rightarrow (q', a'_1, \dots, a'_k, D_1, \dots, D_k)$. Eine solche Anweisung ist ausführbar, falls sich M im Zustand q befindet und sich die Leseköpfe auf mit a_1, \dots, 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 57 (Konfiguration)

Eine **Konfiguration** ist ein $(2k + 1)$ -Tupel $K = (q, u_1, v_1, \dots, u_k, v_k) \in Z \times (\Gamma^* \times \Gamma^+)^k$ und besagt, dass

- der momentane Zustand q ist und
- das i -te Band mit $\dots \square u_i v_i \square \dots$ beschriftet ist, wobei sich der Kopf auf dem ersten Zeichen von v_i befindet.

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

$$(q, a_1, \dots, a_k) \rightarrow (q', a'_1, \dots, a'_k, D_1, \dots, D_k)$$

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

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

Sei $x \in \Sigma^*$ eine Eingabe für M . Die zugehörige **Startkonfiguration** ist

$$K_x = \begin{cases} (q_0, \varepsilon, x, \varepsilon, \square, \dots, \varepsilon, \square) & x \neq \varepsilon, \\ (q_0, \varepsilon, \square, \varepsilon, \square, \dots, \varepsilon, \square) & x = \varepsilon. \end{cases}$$

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

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

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 ein Endzustand erreicht wird.

Beispiel 58 Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit

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

und den Anweisungen

- δ :
- $q_0a \rightarrow q_1AR$ (1) *Anfang der Schleife: Ersetze das erste a durch A*
 - $q_1a \rightarrow q_1aR$ (2) *Bewege den Kopf nach rechts bis zum ersten b und*
 - $q_1B \rightarrow q_1BR$ (3) *ersetze dies durch ein B (falls kein b mehr vorhan-*
 - $q_1b \rightarrow q_2BL$ (4) *den ist, dann halte ohne zu akzeptieren).*
 - $q_2a \rightarrow q_2aL$ (5) *Bewege den Kopf nach links bis ein A kommt, ge-*
 - $q_2B \rightarrow q_2BL$ (6) *he ein Feld nach rechts zurück und wiederhole die*
 - $q_2A \rightarrow q_0AR$ (7) *Schleife.*
 - $q_0B \rightarrow q_3BR$ (8) *Falls kein a am Anfang der Schleife, dann teste, ob*
 - $q_3B \rightarrow q_3BR$ (9) *noch ein b vorhanden ist. Falls nicht, akzeptiere,*
 - $q_3\square \rightarrow q_4\square N$ (10) *sonst halte ohne zu akzeptieren.*

Wegen

$$\begin{aligned} q_0aabb &\vdash Aq_1abb & (1) \\ &\vdash Aaq_1bb & (2) \\ &\vdash Aq_2aBb & (4) \\ &\vdash q_2AaBb & (5) \\ &\vdash Aq_0aBb & (7) \\ &\vdash AAq_1Bb & (1) \\ &\vdash AABq_1b & (3) \\ &\vdash AAq_2BB & (4) \\ &\vdash Aq_2ABB & (6) \\ &\vdash AAq_0BB & (7) \\ &\vdash AABq_3B & (8) \\ &\vdash AABBq_3\square & (9) \\ &\vdash AABBq_4\square & (10) \end{aligned}$$

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

$$q_0abb \vdash_{(1)} Aq_1bb \vdash_{(4)} q_2ABb \vdash_{(7)} Aq_0Bb \vdash_{(8)} ABq_3b$$

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. Wir werden aber später sehen, dass jede 1-NTM, die linear viel Platz braucht, 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 59 (LBA)

Sei Σ ein Alphabet. Eine 1-NTM $M = (Z, \Sigma', \Gamma, \delta, q_0, E)$ mit Eingabealphabet $\Sigma' = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$ heißt **linear beschränkt**, falls sie bei jeder Eingabe der Form $x_1 \cdots x_{n-1} \hat{x}_n$, $n \geq 1$, ausgehend von der Startkonfiguration $K_{x_1 \cdots x_{n-1} \hat{x}_n}$ nur Konfigurationen $K = uqv$ mit $|uv| \leq n$ erreichen kann:

$$K_{x_1 \cdots x_{n-1} \hat{x}_n} \vdash^* uqv \quad \Rightarrow \quad |uv| \leq n.$$

Eine linear beschränkte 1-NTM wird als **LBA** bezeichnet. Die von einem LBA **akzeptierte** oder **erkannte Sprache** ist

$$L(M) = \{x_1 \cdots x_n \in \Sigma^* \mid \exists K \in E \times \Gamma^* \times \Gamma^+ : K_{x_1 \cdots x_{n-1} \hat{x}_n} \vdash^* K\}.$$

Beispiel 60 Es ist nicht schwer, die 1-DTM M aus Beispiel 58 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 $\Gamma' = \Sigma' \cup \{A, B, \hat{B}, \square\}$ sowie
- die Anweisung $q_3 \square \rightarrow q_4 \square N$ (10) durch $q_3 \hat{B} \rightarrow q_4 \hat{B} N$ (10') und
- füge die Anweisung $q_1 \hat{b} \rightarrow q_2 \hat{B} L$ (4a) hinzu.

Dann wird das Wort $aabb$ durch folgende Rechnung von M' bei Eingabe $aabb$ akzeptiert:

$$q_0 a a b \hat{b} \vdash^* A A B q_1 \hat{b} \underset{(4a)}{\vdash} A A q_2 B \hat{B} \vdash^* A A B q_3 \hat{B} \underset{(10')}{\vdash} A A B q_4 \hat{B}$$

Als nächstes zeigen wir, dass LBAs genau die kontextsensitiven Sprachen erkennen.

Satz 61 $CS\mathcal{L} = \{L(M) \mid M \text{ ist ein LBA}\}$.

Beweis: Wir zeigen zuerst die Inklusion von links nach rechts. Sei also $L \in CS\mathcal{L}$ 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 $x = x_1 \cdots x_{n-1} \hat{x}_n$, $n \geq 1$, die folgenden Schritte ausführt (im Fall $\varepsilon \in L$ soll M auch ε akzeptieren).

- 1) Wähle nichtdeterministisch eine Regel $\alpha \rightarrow \beta$ aus $P \setminus \{S \rightarrow \varepsilon\}$.
- 2) Wähle nichtdeterministisch ein beliebiges Vorkommen von β auf dem Band.
(Falls β auf dem Band nicht vorkommt, halte ohne zu akzeptieren.)
- 3) Ersetze dieses Vorkommen von β durch α (linksbündig). Falls das letzte Zeichen von β markiert war, markiere auch das letzte Zeichen von α . Ist dies nicht der Fall und ist α kürzer als β , so verschiebe alle Zeichen rechts von β um $|\beta| - |\alpha|$ Positionen nach links.
- 4) Enthält das Band nur noch das (markierte) Startsymbol, so halte in einem Endzustand. Ansonsten gehe zurück 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 für P die Vereinigung folgender Regelmengen

$$\{S \rightarrow A(\hat{a}, a) \mid a \in \Sigma\} \quad (1)$$

$$\{A \rightarrow A(a, a) \mid a \in \Sigma\} \quad (2)$$

$$\{A \rightarrow ((q_0, a), a) \mid a \in \Sigma\} \quad (3)$$

$$\{((q, c), a) \rightarrow ((q', c'), a) \mid a \in \Sigma, (q', c', N) \in \delta(q, c)\} \quad (4)$$

$$\{((q, c), a)(d, b) \rightarrow (c', a)((q', d), b) \mid a, b \in \Sigma, (q', c', R) \in \delta(q, c)\} \quad (5)$$

$$\{(c, a)((q, d), b) \rightarrow ((q', c), a)(d', b) \mid a, b \in \Sigma, (q', d', L) \in \delta(q, d)\} \quad (6)$$

$$\{((q, c), a) \rightarrow a \mid a \in \Sigma, c \in \Gamma, q \in E\} \quad (7)$$

$$\{(c, a) \rightarrow a \mid a \in \Sigma, c \in \Gamma\} \quad (8)$$

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

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

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

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

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

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

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

Eine bis heute ungelöste Frage ist, ob es kontextsensitive Sprachen gibt, die nicht von einem deterministischen LBA akzeptiert werden: Ist die Klasse

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

eine echte Teilklasse von CSL ?

Die folgenden Abschlusseigenschaften (abgesehen von der Komplementbildung bei den Klassen CSL und \mathcal{RE}) 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 |
| \mathcal{RE} | ja | ja | nein | ja | ja |

4 Entscheidbare und rekursiv aufzählbare Sprachen

Definition 62 (berechenbar)

Sei $f : \Sigma^* \rightarrow \Gamma^*$ eine Funktion. Eine k -DTM M **berechnet** f , falls für alle $x \in \Sigma^*$ und $y \in \Gamma^*$ gilt:

$$f(x) = y \Leftrightarrow K_x \vdash^* K \text{ für eine Konfiguration der Form} \\ K = (q_e, u_1, v_1, \dots, u_{k-1}, v_{k-1}, y, \square) \text{ mit } q_e \in E$$

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 mindestens einmal in einen Endzustand gelangen. Zudem muss das k -te Band immer wenn M in einen Endzustand gelangt mit $f(x)$ beschriftet sein.

Diese Definition lässt sich auch auf **partielle Funktionen** $f : \Sigma^* \rightarrow \Gamma^* \cup \{\uparrow\}$ übertragen: 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).

Definition 63 (entscheidbar)

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

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

berechenbar ist.

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

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

berechenbar ist.

- c) Eine Sprache $L \subseteq \Sigma^*$ heißt **rekursiv aufzählbar**, falls $L = \emptyset$ ist oder eine Turing-berechenbare Funktion $f : \Gamma^* \rightarrow \Sigma^*$ existiert mit

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

$W(f)$ heißt der **Werte-** oder **Bildbereich** von f .

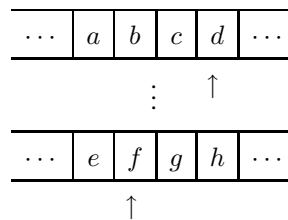
Die Klasse der entscheidbaren Sprachen bezeichnen wir mit \mathcal{REC} und die Klasse der semi-entscheidbaren Sprachen mit \mathcal{RE} (*recursively enumerable*; vergleiche Satz 64).

Satz 64 *Folgende Eigenschaften sind äquivalent:*

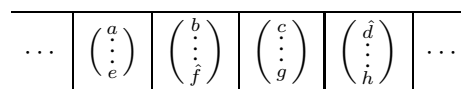
1. A ist semi-entscheidbar,
2. A wird von einer k -NTM akzeptiert,
3. A wird von einer 1-NTM akzeptiert,
4. A ist vom Typ 0,
5. A ist rekursiv aufzählbar,
6. A wird von einer k -DTM akzeptiert,
7. A wird von einer 1-DTM akzeptiert.

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



durch eine Konfiguration K' der Form



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

$$K_x = (q_0, \varepsilon, x, \varepsilon, \square, \dots, \varepsilon, \square)$$

von M bei Eingabe x entsprechende Konfiguration

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

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

Zuerst geht M' solange nach rechts, bis sie alle mit $\hat{}$ markierten Zeichen (z.B. $\hat{a}_1, \dots, \hat{a}_k$) gefunden hat. Diese Zeichen speichert M' in ihrem Zustand. Anschließend geht M' wieder nach links und realisiert dabei die durch $\delta(q, a_1, \dots, a_k)$ 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 $f : \Gamma^* \rightarrow \Gamma^*$ aufzählen, wobei $\Gamma = V \cup \Sigma \cup \{\#\}$ und $\# \notin V \cup \Sigma$ ist:

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

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

5) \Rightarrow 6): Sei $f : \Sigma^* \rightarrow \Sigma^*$ 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 Σ^* (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 k -DTM gewonnen werden kann, die χ_A berechnet. ■

Satz 65 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 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 zu jedem Wort $y \in \Sigma^*$ 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 1 aus und falls $f_2(y) = x$ ist, gibt M den Wert 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{K} die Klasse $\{\bar{A} \mid A \in \mathcal{K}\}$ der Komplementärsprachen mit $co\text{-}\mathcal{K}$, so gilt also $\mathcal{RE} \cap co\text{-}\mathcal{RE} = \mathcal{REC}$.

4.1 Kodierung (Gödelisierung) von TMs

Sei M eine 1-TM mit Arbeitsalphabet $\Gamma = \{a_0 = \square, a_1, \dots, a_l\}$ und Zustandsmenge $Z = \{z_0, \dots, z_m\}$ (o.B.d.A. sei $E = \{z_m\}$). Dann können wir jede Anweisung der Form

$$z_i, a_j \rightarrow z_{i'} a_{j'} D$$

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 Kodierung $w_M \in \{0, 1\}^*$ 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 -TMs, Konfigurationen und Rechnungen (Konfigurationsfolgen) kodieren.

Jedem Binärstring w können wir umgekehrt eine TM M_w wie folgt zuordnen:

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

Dabei ist M_0 eine beliebig, aber fest gewählte TM.

Definition 66 (Halteproblem)

Das **Halteproblem** ist die Sprache

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

Das **spezielle Halteproblem** ist

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

Satz 67 $K \in \mathcal{RE} \setminus \mathcal{REC}$.

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

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

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

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

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

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

Korollar 68 $K \notin \text{co-}\mathcal{RE}$.

Beweis: Wegen $K \in \mathcal{RE}$ folgt aus der Annahme, dass $K \in \text{co-}\mathcal{RE}$ ist, mit Satz 65, dass K entscheidbar ist. Dies steht jedoch im Widerspruch zu Satz 67. ■

Definition 69 (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 $f : \Sigma^* \rightarrow \Gamma^*$ existiert mit

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

für alle $x \in \Sigma^*$.

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

Beispiel 70 $K \leq H$ mittels $f(w) = w\#w$ für alle $w \in \{0,1\}^*$.

Satz 71 Die Klassen \mathcal{RE} und \mathcal{REC} sind unter \leq abgeschlossen. Dabei heißt eine Sprachklasse \mathcal{K} **unter \leq abgeschlossen**, wenn für alle Sprachen A, B gilt:

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

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 72 $H \notin \mathcal{REC}$.

Beweis: Aus der Annahme, dass H entscheidbar ist, folgt wegen $K \leq H$, dass auch K entscheidbar ist, was im Widerspruch zu Satz 67 steht. ■

Definition 73 (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 74 $H_0 \notin \mathcal{REC}$.

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 72 nicht der Fall ist. Sei $w_0 \notin H_0$ und betrachte die Funktion

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

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

Der folgende Satz von Rice besagt, dass man einer 1-DTM nicht ansehen kann, ob die von ihr berechnete Funktion eine gewisse Eigenschaft hat oder nicht.

Satz 75 (Satz von Rice) Sei \mathcal{F} die Klasse aller partiellen Turing-berechenbaren Funktionen $g : \{0, 1\}^* \rightarrow \{0, 1\}^*$ und sei $\mathcal{S} \subseteq \mathcal{F}$ mit $\mathcal{S} \notin \{\emptyset, \mathcal{F}\}$. Dann ist die Sprache

$$K(\mathcal{S}) = \{w \mid M_w \text{ ist eine DTM, die eine Funktion in } \mathcal{S} \text{ berechnet}\}$$

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

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

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

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 $x \in \{0, 1\}^*$. Damit die Reduktion gelingt, muss $M_{w'}$ im Fall $w \in H_0$ eine Funktion $v \in \mathcal{F}$ berechnen mit

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

Wegen $\mathcal{S} \notin \{\emptyset, \mathcal{F}\}$ muss eine solche Funktion $v \in \mathcal{F}$ existieren.

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

$$f(w) = w', \text{ wobei } w' \text{ die Kodierung einer DTM ist, die bei Eingabe } x \text{ zun\u00e4chst einmal die 1-DTM } M_w \text{ bei Eingabe } \varepsilon \text{ simuliert und im Fall, dass } M_w \text{ h\u00e4lt, mit der Simulation von } M \text{ bei Eingabe } x \text{ fortf\u00e4hrt.}$$

Dann gilt

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

Im Fall $v \in \mathcal{S}$ gilt daher

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

das hei\u00dft, f reduziert H_0 auf $K(\mathcal{S})$. Im Fall $v \notin \mathcal{S}$ gilt dagegen

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

das hei\u00dft, f reduziert $\overline{H_0}$ auf $K(\mathcal{S})$. Da f Turing-berechenbar ist und sowohl H_0 als auch $\overline{H_0}$ unentscheidbar sind, folgt $K(\mathcal{S}) \notin \mathcal{RE}$. ■

Beispiel 76 *Einer Turingmaschine kann man nicht ansehen, ob sie die Identit\u00e4t auf $\{0, 1\}^*$ (also die Funktion $id(x) = x$, $x \in \{0, 1\}^*$) berechnet oder nicht. Genauer: die Sprache*

$$L = \{w \in \{0, 1\}^* \mid M_w \text{ berechnet die Identit\u00e4t auf } \{0, 1\}^*\}$$

ist unentscheidbar, da die durch $\mathcal{S} = \{id\}$ beschriebene Eigenschaft nicht trivial (also $\mathcal{S} \notin \{\emptyset, \mathcal{F}\}$) ist.

4.2 Das Postsche Korrespondenzproblem (PCP)

Sei Σ ein beliebiges Alphabet und sei $\# \notin \Sigma$. Dann enth\u00e4lt PCP (genauer: PCP_Σ) alle positiven (mit ja zu beantwortenden) Eingaben f\u00fcr das folgende Entscheidungsproblem:

Gegeben: k Wortpaare $(x_1, y_1), \dots, (x_k, y_k)$, kodiert durch $x_1\#y_1\#\dots\#x_k\#y_k$ (die wir oft auch in Form einer Matrix $\begin{pmatrix} x_1 & \dots & x_k \\ y_1 & \dots & y_k \end{pmatrix}$ schreiben).

Gefragt: Gibt es eine PCP-Lösung für die Problem Instanz $\begin{pmatrix} x_1 \cdots x_k \\ y_1 \cdots y_k \end{pmatrix}$? Dabei ist eine PCP-Lösung eine Folge $\alpha = (i_1, \dots, i_n)$, $n \geq 1$, von Indizes $i_j \in \{1, \dots, k\}$ 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 77 Die Instanz $I = \begin{pmatrix} 1 & 10 & 011 \\ 101 & 00 & 11 \end{pmatrix}$ besitzt wegen

$$x_1 x_3 x_2 x_3 = 101110011 = y_1 y_3 y_2 y_3$$

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

Lemma 78 Für jedes Alphabet Σ gilt $\mathcal{PCP}_\Sigma \leq \mathcal{PCP}_{\{0,1\}}$ und $\mathcal{MPCP}_\Sigma \leq \mathcal{MPCP}_{\{0,1\}}$.

Beweis: Sei $\Sigma = \{a_1, \dots, a_m\}$. 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 $\mathcal{PCP}_\Sigma \leq \mathcal{PCP}_{\{0,1\}}$ und $\mathcal{MPCP}_\Sigma \leq \mathcal{MPCP}_{\{0,1\}}$ mittels der Reduktionsfunktion

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

■

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

Satz 79 $\mathcal{MPCP} \leq \mathcal{PCP}$.

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

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

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

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

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

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

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

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

eine MPCP-Lösung für I . ■

Satz 80 *PCP ist RE-vollständig und damit unentscheidbar.*

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

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

berechnet werden kann.

Idee: Eine Indexfolge $\alpha = (i_1, \dots, i_n)$ 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. $(\$, \$z_0x\$)$, „Startregel“
2. (a, a) für alle $a \in \Gamma \cup \{\S\}$, „Kopierregeln“
3. Für alle $a, a', b \in \Gamma, z, z' \in Z$ die folgenden Wortpaare,

$$\begin{array}{ll} (za, z'a'), & \text{falls } \delta(z, a) = (z', a', N), \\ (za, a'z'), & \text{falls } \delta(z, a) = (z', a', R), \\ (bza, z'ba'), & \text{falls } \delta(z, a) = (z', a', L), \end{array}$$

sowie

„Überführungsregeln“

$$\begin{array}{ll} (\$za, \$z'\square a'), & \text{falls } \delta(z, a) = (z', a', L), \\ (z\S, z'a'\$), & \text{falls } \delta(z, \square) = (z', a', N), \\ (z\S, a'z'\$), & \text{falls } \delta(z, \square) = (z', a', R), \\ (bz\S, z'ba'\$), & \text{falls } \delta(z, \square) = (z', a', L), \end{array}$$

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

Nun lässt sich leicht aus einer akzeptierenden Rechnung

$$K_0 = z_0 w \vdash K_1 \vdash \cdots \vdash K_t = uz_e v$$

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

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

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 $A \leq \mathcal{MPCP}_{\Sigma'}$ gezeigt ist. ■

Satz 81 Für zwei gegebene kontextfreie Grammatiken G_1, G_2 sind folgende Fragestellungen unentscheidbar:

1. Ist $L(G_1) \cap L(G_2) \neq \emptyset$? (Schnittproblem)
2. Ist $L(G_1) = L(G_2)$? (Äquivalenzproblem)
3. Ist $L(G_1) = \Sigma^*$?
4. Ist G_1 mehrdeutig?

Beweis: 1. Betrachte die Turing-berechenbare Funktion

$$f \left(\begin{array}{c} x_1 \cdots x_k \\ y_1 \cdots y_k \end{array} \right) = (G_1, G_2),$$

wobei G_1 die Regeln

$$S \rightarrow 0^i 1 S x_i, 0^i 1 1 x_i, i = 1, \dots, k$$

und G_2 die Regeln

$$S \rightarrow 0^i 1 S y_i, 0^i 1 1 y_i, i = 1, \dots, k$$

enthält. Dann gilt

$$L(G_1) = \{0^{i_1} 1 0^{i_2} \cdots 0^{i_n} 1 1 x_{i_n} \cdots x_{i_1} \mid n \geq 1 \text{ und } 1 \leq i_1, \dots, i_n \leq k\}$$

und

$$L(G_2) = \{0^{i_1} 1 0^{i_2} \cdots 0^{i_n} 1 1 y_{i_n} \cdots y_{i_1} \mid n \geq 1 \text{ und } 1 \leq i_1, \dots, i_n \leq k\}.$$

Somit ist

$$L(G_1) \cap L(G_2) = \{0^{i_1} 1 \cdots 0^{i_n} 1 1 z \mid n \geq 1 \text{ und } x_{i_n} \cdots x_{i_1} = z = y_{i_n} \cdots y_{i_1}\},$$

d.h. die Instanz $I = \begin{pmatrix} x_1 \cdots x_k \\ y_1 \cdots y_k \end{pmatrix}$ hat genau dann eine PCP-Lösung $\alpha = (i_n \cdots i_1)$, wenn $0^{i_1} 1 \cdots 0^{i_n} 1 1 x_{i_n} \cdots x_{i_1} \in L(G_1) \cap L(G_2)$, also $L(G_1) \cap L(G_2) \neq \emptyset$ ist. Dies zeigt, dass f eine Reduktion von PCP auf das Schnittproblem für kontextfreie Grammatiken vermittelt.

2. Sei $L_i = L(G_i)$, $i = 1, 2$. Dann ist leicht zu sehen, dass $L_1, L_2 \in \mathcal{DCFL}$ sind. Zudem können bei Eingabe von $I = \begin{pmatrix} x_1 \cdots x_k \\ y_1 \cdots y_k \end{pmatrix}$ DPDAs M_1, M_2 mit $L(M_i) = \overline{L_i}$ und kontextfreie Grammatiken G'_1, G'_2, G mit $L(G'_i) = \overline{L_i}$ und $L(G) = L_1 \cup \overline{L_2}$ berechnet werden. Wegen

$$\begin{aligned} I \notin \mathcal{PCP} &\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset \\ &\Leftrightarrow L_1 \cap L_2 = \emptyset \\ &\Leftrightarrow L_1 \subseteq \overline{L_2} \\ &\Leftrightarrow L_1 \cup \overline{L_2} = \overline{L_2} \\ &\Leftrightarrow L(G) = L(G'_2) \end{aligned}$$

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

$$\begin{aligned} I \notin \mathcal{PCP} &\Leftrightarrow L_1 \cap L_2 = \emptyset \\ &\Leftrightarrow \overline{L_1} \cup \overline{L_2} = \Sigma^* \\ &\Leftrightarrow L(G') = \Sigma^* \end{aligned}$$

reduziert sie das Komplement von PCP auf die Sprache $\{G \mid G \text{ ist eine kontextfreie Grammatik mit } L(G) = \Sigma^*\}$.

4. Betrachte die Turing-berechenbare Funktion

$$f \begin{pmatrix} x_1 \cdots x_k \\ y_1 \cdots y_k \end{pmatrix} = G'',$$

wobei G'' die Regeln

$$\begin{aligned} S &\rightarrow A, B \\ A &\rightarrow 0^i 1 A x_i, 0^i 1 x_i, i = 1, \dots, k \\ B &\rightarrow 0^i 1 B y_i, 0^i 1 y_i, i = 1, \dots, k \end{aligned}$$

enthält. Da von A oder B ausgehende Ableitungen eindeutig sind, ist G'' genau dann mehrdeutig, wenn es ein Wort $w \in L(G'')$ gibt mit

$$S \Rightarrow A \Rightarrow^* w \text{ und } S \Rightarrow B \Rightarrow^* w.$$

Wie wir bereits in 1. gesehen haben, ist dies genau dann der Fall, wenn I eine PCP-Lösung hat. ■

Da die in obigem Beweis benutzten Sprachen $L_1 = L(G_1)$ und $L_2 = L(G_2)$ deterministisch kontextfrei sind, können wir auch die beiden folgenden Unentscheidbarkeitsresultate festhalten.

Korollar 82 *Das Schnitt- und das Inklusionsproblem für $\mathcal{DCF}\mathcal{L}$ sind unentscheidbar.*

Beweis: Wie wir im Beweis von Satz 81 gesehen haben, können zu jeder PCP-Instanz I deterministische Kellerautomaten für $\mathcal{DCF}\mathcal{L}$ -Sprachen L_1, L_2 berechnet werden, so dass I genau dann eine PCP-Lösung besitzt, wenn $L_1 \cap L_2 \neq \emptyset$ (bzw. $L_1 \subseteq L_2$) ist. ■

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

Satz 83 *Das Leerheitsproblem für $\mathcal{CF}\mathcal{L}$ ist entscheidbar.*

Beweis: Betrachte folgenden Algorithmus:

```

1  Eingabe: eine kontextfreie Grammatik  $G = (V, \Sigma, P, S)$ 
2   $U' \leftarrow \emptyset$ 
3  repeat
4   $U \leftarrow U'$ 
5   $U' \leftarrow \{A \mid \text{es gibt eine Regel } A \rightarrow \alpha \in P \text{ mit } \alpha \in (\Sigma \cup U)^*\}$ 
6  until ( $U = U'$ )
7  if  $S \in U$  then
8  output „ $L(G) \neq \emptyset$ “
9  else
10 output „ $L(G) = \emptyset$ “
11 end

```

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

Satz 84 *Das Leerheitsproblem für $\mathcal{CS}\mathcal{L}$ ist unentscheidbar.*

Beweis: Wir reduzieren das Schnittproblem für $\mathcal{CF}\mathcal{L}$ auf das (Nicht-)Leerheitsproblem für $\mathcal{CS}\mathcal{L}$. 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 85 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 \mathcal{NP} -Vollständigkeit

5.1 Zeitkomplexität

Der Zeitverbrauch $time_M(x)$ einer TM 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 86 (Rechenzeit)

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

$$time_M(x) \leq \max(t(|x|), n + 1).$$

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

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

und

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

zusammen. Ferner sei

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

Die wichtigsten Zeitkomplexitätsklassen sind

$$\begin{aligned} \mathcal{P} &= \bigcup_{c>0} DTIME(n^c + c), \\ \mathcal{E} &= \bigcup_{c>0} DTIME(2^{cn}), \\ \mathcal{EXPTIME} &= \bigcup_{c>0} DTIME(2^{n^c+c}). \end{aligned}$$

Die Klassen \mathcal{NP} , \mathcal{NE} , $\mathcal{NEXPTIME}$ und \mathcal{FP} , \mathcal{FE} , $\mathcal{FEXPTIME}$ sind analog definiert.

5.2 \mathcal{NP} -Vollständigkeit und das P=NP Problem

Wie wir in der Berechenbarkeit gesehen haben, sind NTMs nicht mächtiger als DTMs, da erstere von letzteren simuliert werden können (siehe Satz 64). Die Frage, wieviel Zeit eine NTM gegenüber einer DTM einsparen kann, ist eines der wichtigsten offenen Probleme der Informatik.

Es ist unmittelbar klar, dass $\mathcal{P} \subseteq \mathcal{NP}$ gilt. Die Klasse \mathcal{NP} enthält viele für die Praxis überaus wichtige Probleme. Da andererseits Probleme in \mathcal{P} als effizient lösbar angesehen werden können, hat das so genannte $\mathcal{P} = \mathcal{NP}$ -**Problem**, also die Frage, ob \mathcal{NP} gleich \mathcal{P} ist, eine große praktische Bedeutung.

Definition 87 (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 $f : \Sigma^* \rightarrow \Gamma^*$ in \mathcal{FP} existiert mit

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

für alle $x \in \Sigma^*$.

Lemma 88 Die Reduktionsrelation \leq^p ist reflexiv und transitiv (Beweis siehe Übungen).

Satz 89 Die Klassen \mathcal{P} und \mathcal{NP} sind unter \leq^p abgeschlossen.

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

- 1 **Eingabe:** x
- 2 $y \leftarrow f(x)$
- 3 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

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

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

Definition 90 (\mathcal{NP} -vollständig)

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

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

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

Satz 91 1. $A \leq^p B$ und A ist \mathcal{NP} -schwer $\Rightarrow B$ ist \mathcal{NP} -schwer.

2. $A \leq^p B$, $A \in \mathcal{NPC}$ und $B \in \mathcal{NP} \Rightarrow B \in \mathcal{NPC}$.

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

Beweis: 1. Wir müssen für eine beliebige Sprache $L \in \mathcal{NP}$ zeigen, dass $L \leq^p B$ gilt. Da A nach Voraussetzung \mathcal{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 \mathcal{NP} -schwer ist. Da zudem $B \in \mathcal{NP}$ ist, folgt $B \in \mathcal{NPC}$.

3. Falls eine Sprache A in $\mathcal{NPC} \cap \mathcal{P}$ existiert, so lässt sich jede \mathcal{NP} -Sprache L auf $A \in \mathcal{P}$ reduzieren. Da \mathcal{P} unter \leq^p abgeschlossen ist, folgt $L \in \mathcal{P}$. ■

Satz 92 Die Sprache

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

ist \mathcal{NP} -vollständig.

Beweis: Zunächst ist klar, dass $A \in \mathcal{NP}$ 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 verwirfe.

Zum Nachweis der \mathcal{NP} -Härte von A sei eine beliebige \mathcal{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 \mathcal{FP} -Funktion

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

auf A reduzieren. ■

Definition 93 (Boolesche Formeln)

Die Menge der **Booleschen** (oder **aussagenlogischen**) **Formeln** über den Variablen x_1, \dots, 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** $a = a_1 \dots a_n \in \{0, 1\}^n$ 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 94 Die Formel $F = (x_1 \vee x_2) \rightarrow (x_2 \wedge x_3)$ ist erfüllbar und berechnet folgende Boolesche Funktion:

| $a_1 a_2 a_3$ | $a_1 \vee a_2$ | $a_2 \wedge a_3$ | $(a_1 \vee a_2) \rightarrow (a_2 \wedge a_3)$ |
|---------------|----------------|------------------|---|
| 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 95 Die Formel

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

liefert unter einer Belegung $a = a_1 \dots a_n$ genau dann den Wert $G(a_1 \dots a_n) = 1$, 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, \dots, x_n .

Gefragt: Ist F erfüllbar?

Satz 96 SAT ist \mathcal{NP} -vollständig.

Beweis: Es ist leicht zu sehen, dass $\text{SAT} \in \mathcal{NP}$ 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 \mathcal{NP} -schwer ist, sei L eine beliebige \mathcal{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 64).

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 $Z = \{q_0, \dots, q_k\}$, $E = \{q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$. Dann bilden wir F_w über den Variablen

$x_{t,q}$, $y_{t,i}$ und $z_{t,i,a}$ für $0 \leq t \leq p(n)$, $-p(n) \leq i \leq p(n)$, $q \in \{q_0, \dots, q_k\}$ 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 $F_w = R \wedge S \wedge_1 \wedge_2 \wedge_3 \wedge E$, wobei R (steht für Randbedingungen) sicherstellt, dass wir jeder Belegung eindeutig eine Folge von Konfigurationen $K_0, \dots, K_{p(n)}$ zuordnen können:

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

wobei

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

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

- genau ein Zustand $q \in \{q_0, \dots, q_k\}$ eingenommen wird,
- genau ein Bandfeld $i \in \{-p(n), \dots, p(n)\}$ besucht wird und

- auf jedem Feld i genau ein Zeichen $a \in \Gamma$ steht.

Die übrigen Teilformeln sind

$$S = x_{0,q_0} \wedge y_{0,1} \wedge z_{0,-p(n),\square} \wedge \cdots \wedge z_{0,-1,\square} \wedge z_{0,0,w_1} \wedge \cdots \wedge z_{0,n-1,w_n} \wedge z_{0,n,\square} \wedge \cdots \wedge z_{0,p(n),\square},$$

$$1 = \bigwedge_{\substack{0 \leq t < p(n) \\ -p(n) \leq i \leq p(n) \\ a \in \Gamma}} [(\neg y_{t,i} \wedge z_{t,i,a}) \rightarrow z_{t+1,i,a}],$$

$$2 = \bigwedge_{\substack{0 \leq t < p(n) \\ -p(n) \leq i \leq p(n) \\ 0 \leq j \leq k-1, a \in \Gamma}} [(x_{t,q_j} \wedge y_{t,i} \wedge z_{t,i,a}) \rightarrow \bigvee_{(q',a',D) \in \delta(q_j,a)} (x_{t+1,q'} \wedge y_{t+1,i+D} \wedge z_{t+1,i,a})],$$

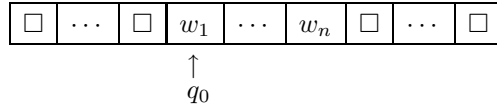
$$3 = \bigwedge_{\substack{0 \leq t < p(n) \\ -p(n) \leq i \leq p(n) \\ a \in \Gamma}} [(x_{t,q_k} \wedge y_{t,i} \wedge z_{t,i,a}) \rightarrow (x_{t+1,q_k} \wedge y_{t+1,i} \wedge z_{t+1,i,a})] \text{ und}$$

$$E = x_{p(n),q_k},$$

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



vorliegt. Mit der Formel 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 2 auf die Einhaltung der durch δ vorgegebenen Anweisungen, solange nicht der Zustand q_k angenommen wird. Schließlich sorgt 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 \mathcal{FP} berechenbar. Es bleibt zu zeigen, dass alle Eingaben w die Äquivalenz

$$w \in L(M) \Leftrightarrow F_w \in \text{SAT}$$

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 $K_0, \dots, K_{p(n)}$ gewinnen. Wegen $S(a) = 1$ ist K_0 die Startkonfiguration von M bei Eingabe w . Wegen $1(a) = 2(a) = 3(a) = 1$ muss für $i = 0, \dots, p(n) - 1$ 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 jemandem, einen Polynomialzeit-Algorithmus für SAT zu finden, so lässt sich daraus leicht ein effizienter Algorithmus für jedes Problem in NP ableiten.

Korollar 97 $\text{SAT} \in \mathcal{P} \Leftrightarrow \mathcal{P} = \mathcal{NP}$.

Definition 98 (Boolesche Schaltkreise)

Ein **Boolescher Schaltkreis** c mit n Eingängen ist eine Folge (g_1, \dots, g_m) von **Gattern** $g_l \in \{0, 1, x_1, \dots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$ 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 \mathcal{NP} enthalten ist, haben wir ein weiteres \mathcal{NP} -vollständiges Problem gefunden.

Satz 99 CIRSAT ist \mathcal{NP} -vollständig.

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

Definition 100 (konjunktive Normalform)

Eine Boolesche Formel F über den Variablen x_1, \dots, x_n ist in **konjunktiver Normalform** (kurz KNF), falls F eine Konjunktion $F = \bigwedge_{j=1}^m C_j$ von Disjunktionen (auch **Klauseln** genannt) $C_j = \bigvee_{l=1}^{k_j} L_{jl}$ von **Literalen** $L_{jl} \in \{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$ ist (hierbei verwenden wir \bar{x} als abkürzende Schreibweise für $\neg x$). Gilt $k_j \leq k$ für $j = 1, \dots, m$, so heißt F in **k -KNF**.

Beispiel 101 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.

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

Gegeben: Eine Boolesche Formel in k -KNF.

Gefragt: Ist F erfüllbar?

Satz 102 3-SAT ist \mathcal{NP} -vollständig.

Beweis: Wir zeigen CIRSAT \leq^P 3-SAT. Hierzu transformieren wir einen Schaltkreis $c = (g_1, \dots, g_m)$ mit n Eingängen in eine 3-KNF Formel $F_c = z_m \wedge \bigwedge_{l=1}^m G_l$ über den Variablen $x_1, \dots, x_n, z_1, \dots, z_m$, wobei die Formeln G_l wie folgt definiert sind:

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

Offensichtlich ist die Reduktionsfunktion $f : c \mapsto F_c$ in \mathcal{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 erhalten wir daraus eine erfüllende Belegung $a_1 \cdots a_n b_1 \cdots b_m$ für $F_c(x_1, \dots, x_n, z_1, \dots, z_m)$, indem wir

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

setzen.

„ \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 $l = 1, \dots, m$

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

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

Als nächstes betrachten wir das Problem, einen Rucksack optimal zu bepacken. D.h. wir haben k Gegenstände der Größe u_1, \dots, u_k und einen Rucksack der Größe v und möchten eine Auswahl von Gegenständen treffen, so dass der Rucksack randvoll wird.

Rucksack-Problem (Subset Sum):

Gegeben: Ein Vektor $(u_1, \dots, u_k, v) \in \mathbb{N}^{k+1}$ von natürlichen Zahlen.

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

Satz 103 *Rucksack ist \mathcal{NP} -vollständig.*

Beweis: Wir zeigen $\text{3-SAT} \leq^P \text{RUCKSACK}$. 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

$$f(F) = (u_1, \dots, u_n, \bar{u}_1, \dots, \bar{u}_n, v_1, \dots, v_m, w_1, \dots, w_m, v),$$

wobei u_i und \bar{u}_i die Zahlen

$$u_i = b_{i1} \cdots b_{im} 0^{i-1} 10^{n-i-1} \text{ und } \bar{u}_i = \bar{b}_{i1} \cdots \bar{b}_{im} 0^{i-1} 10^{n-i-1} \text{ (zur Basis 10)}$$

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 $v_j = w_j = 0^{j-1} 10^{m-j-1}$ 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

$$\begin{array}{ll} u_1 = 010\ 1000 & \bar{u}_1 = 100\ 1000 \\ u_2 = 011\ 0100 & \bar{u}_2 = 001\ 0100 \\ u_3 = 000\ 0010 & \bar{u}_3 = 100\ 0010 \\ u_4 = 110\ 0001 & \bar{u}_4 = 001\ 0001 \end{array}$$

und

$$\begin{array}{ll} v_1 = w_1 = 100\ 0000 \\ v_2 = w_2 = 010\ 0000 \\ v_3 = w_3 = 001\ 0000 \end{array}$$

sowie $v = 333\ 1111$. Da f in Polynomialzeit berechenbar ist, müssen wir nur noch die Äquivalenz

$$F \in 3\text{-SAT} \Leftrightarrow f(F) \in \text{RUCKSACK}$$

zeigen.

„ \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

$$P, N \subseteq \{1, \dots, n\} \quad I, J \subseteq \{1, \dots, m\}$$

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 $P = \{1, \dots, n\} - N$, 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. ■

Satz 104 Die Sprache $L = \{M\#x \mid M \text{ ist ein NFA und } x \in L(M)\}$ ist in \mathcal{P} .

Beweis: Betrachte folgenden Algorithmus:

```

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

```

Es ist klar, dass dieser Algorithmus korrekt arbeitet und auch in eine polynomiell zeitbeschränkte DTM übersetzt werden kann. ■

Korollar 105 Die Sprache $L' = \{\alpha\#x \mid \alpha \text{ ist ein regulärer Ausdruck und } x \in L(\alpha)\}$ ist in \mathcal{P} .

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

Definition 106 (sternfrei)

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

Satz 107 Sei $\Sigma = \{a_1, \dots, a_m\}$ ein Alphabet mit $m \geq 2$. Dann ist die Sprache $L_0 = \{\alpha\#\text{bin}(n) \mid \alpha \text{ ist ein sternfreier regulärer Ausdruck über } \Sigma \text{ mit } L(\alpha) \neq \Sigma^n\}$ \mathcal{NP} -vollständig.

Beweis: Wir zeigen zuerst $L_0 \in \mathcal{NP}$. 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 \mathcal{NP} -Algorithmus die Sprache L_0 :

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

Als nächstes zeigen wir, dass 3-SAT auf L_0 reduzierbar ist. Sei

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

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

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

mit $\alpha_j = \beta_{1j} \cdots \beta_{n_j}$, 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 108 Die Sprache $L_1 = \{\alpha \# \beta \mid \alpha \text{ und } \beta \text{ sind sternfreie reguläre Ausdrücke und } L(\alpha) \neq L(\beta)\}$ ist \mathcal{NP} -vollständig.

Beweis: Die Funktion

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

vermittelt eine Polynomialzeit-Reduktion von L_0 auf L_1 . ■