Vorlesungsskript

Theoretische Informatik 2

Wintersemester 2009/10

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

22. Januar 2010

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einleitung

In der Vorlesung ThI 1 standen die mathematischen Grundlagen der Informatik im Vordergrund. Insbesondere lernten Sie, wie man folgerichtig argumentiert und wie man formale Beweise führt. Als universelle Sprache der Mathematik lernten Sie dabei die mathematische Logik kennen, insbesondere die Aussagenlogik und darauf aufbauend die Prädikatenlogik. In dieser Sprache lassen sich nicht nur algebraische und relationale Strukturen modellieren, sondern auch Rechenmaschinen wie zum Beispiel die Turingmaschine.

Ein weiteres wichtiges Thema der Vorlesung ThI1 war die Frage, welche Probleme algorithmisch lösbar sind.

Themen der Vorlesung ThI1

- Mathematische Grundlagen der Informatik, Beweise führen, Modellierung (Aussagenlogik, Prädikatenlogik)
- Welche Probleme sind lösbar? (Berechenbarkeitstheorie)

Dagegen stehen in dieser Vorlesung folgende Fragen im Mittelpunkt.

Themen der Vorlesung ThI2

- Welche Rechenmodelle sind für bestimmte Aufgaben adäquat? (Automatentheorie)
- Welcher Aufwand ist zur Lösung eines algorithmischen Problems nötig? (Komplexitätstheorie)

Schließlich wird es in der Vorlesung ThI 3 in erster Linie um folgende Frage gehen.

Thema der Vorlesung ThI3

- Wie lassen sich eine Reihe von praktisch relevanten Problemstellungen möglichst effizient lösen? (Algorithmik)

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 ThI3 wird das etwas flexibere Modell der Registermaschine (engl. random access machine; RAM) benutzt. Dieses Modell erlaubt den unmittelbaren Lese- und Schreibzugriff (random access) auf eine beliebige Speichereinheit (Register). Hier betrachten wir Einschränkungen des TM-Modells, die vielfältige praktische Anwendungen haben, wie z.B. endliche Automaten (DFA, NFA), Kellerautomaten (PDA, DPDA) etc.

Der Begriff *Algorithmus* geht auf den persischen Gelehrten Muhammed Al Chwarizmi (8./9. Jhd.) zurück. Der älteste bekannte nicht-triviale Algorithmus ist der nach *Euklid* benannte Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen (300 v. Chr.). Von einem Algorithmus wird erwartet, dass er jede *Problemeingabe* nach endlich vielen Rechenschritten löst (etwa durch Produktion einer Ausgabe). Ein Algorithmus ist ein „Verfahren“ zur Lösung eines Berechnungsproblems, das sich prinzipiell auf einer Turingmaschine implementieren lässt (Church-Turing-These).

Wir betrachten zunächst nur Entscheidungsprobleme, was der Berechnung von $\{0,1\}$-wertigen Funktionen entspricht. Problemeingaben können Zahlen, Formeln, Graphen etc. sein. Diese werden über einem *Eingabealphabet* Σ kodiert.

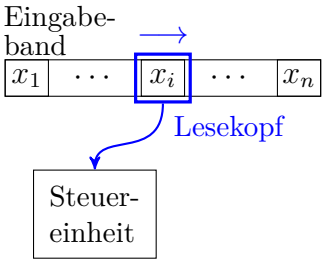
Definition 1. *Ein **Alphabet** ist eine geordnete endliche Menge $\Sigma = \{a_1, \ldots, a_m\}$, $m \geq 1$, von **Zeichen**. Eine Folge $x = x_1 \ldots x_n \in \Sigma^n$ heißt **Wort** (der **Länge** n). Die Menge aller Wörter über Σ ist*

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

*Das (einzige) Wort der Länge $n = 0$ ist das **leere Wort**, welches wir mit ε bezeichnen. Jede Teilmenge $L \subseteq \Sigma^*$ heißt **Sprache** über dem Alphabet Σ.*

2 Reguläre Sprachen

2.1 Endliche Automaten



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

Definition 2. *Ein **endlicher Automat** (kurz: DFA; deterministic finite automaton) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, q_0, E)$ beschrieben, wobei*

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

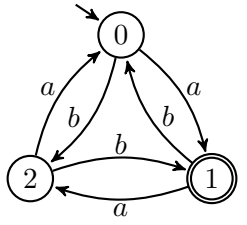
*Die von M **akzeptierte** oder **erkannte Sprache** ist*

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

Beispiel 3. *Betrachte den DFA $M = (Z, \Sigma, \delta, 0, E)$ mit $Z = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $E = \{1\}$ und der Überführungsfunktion*

δ	0	1	2
a	1	2	0
b	2	0	1

Graphische Darstellung:



Der Startzustand wird meist durch einen Pfeil und Endzustände werden durch einen doppelten Kreis gekennzeichnet. ◁

Bezeichne $\hat{\delta}(q, x)$ denjenigen Zustand, in dem sich M nach Lesen von x befindet, wenn M im Zustand q gestartet wird. Dann können wir die Funktion

$$\hat{\delta} : Z \times \Sigma^* \to Z$$

induktiv wie folgt definieren. Für $q \in Z$, $x \in \Sigma^*$ und $a \in \Sigma$ sei

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

Die von M erkannte Sprache lässt sich nun auch in der Form

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in E\}$$

schreiben.

Behauptung 1. *Der DFA M aus Beispiel 3 akzeptiert die Sprache*

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

wobei $\#_a(x)$ die Anzahl der Vorkommen des Buchstabens a in x bezeichnet und $j \equiv k \pmod m$ bedeutet, dass $j - k$ durch m teilbar ist. Für $j \equiv k \pmod m$ schreiben wir im Folgenden auch kurz $j \equiv_m k$.

Beweis. Da M nur den Endzustand 1 hat, ist $L(M) = \{x \in \Sigma^* \mid \hat{\delta}(0,x) = 1\}$. Daher reicht es, folgende Kongruenzgleichung zu zeigen:

$$\hat{\delta}(0,x) \equiv_3 \#_a(x) - \#_b(x).$$

Wir beweisen die Kongruenz induktiv über die Länge n von x.

Induktionsanfang ($n = 0$): klar, da $\hat{\delta}(0,\varepsilon) = \#_a(\varepsilon) = \#_b(\varepsilon) = 0$ ist.

Induktionsschritt ($n \rightsquigarrow n+1$): Sei $x = x_1 \cdots x_{n+1}$ gegeben und sei $i = \hat{\delta}(0, x_1 \cdots x_n)$. Nach IV ist

$$i \equiv_3 \#_a(x_1 \cdots x_n) - \#_b(x_1 \cdots x_n).$$

Wegen $\delta(i,a) \equiv_3 i+1$ und $\delta(i,b) \equiv_3 i-1$ folgt

$$\delta(i, x_{n+1}) \equiv_3 i + \#_a(x_{n+1}) - \#_b(x_{n+1}) = \#_a(x) - \#_b(x).$$

Folglich ist

$$\hat{\delta}(0,x) = \delta(\hat{\delta}(0, x_1 \cdots x_n), x_{n+1}) = \delta(i, x_{n+1}) \equiv_3 \#_a(x) - \#_b(x).$$

■

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

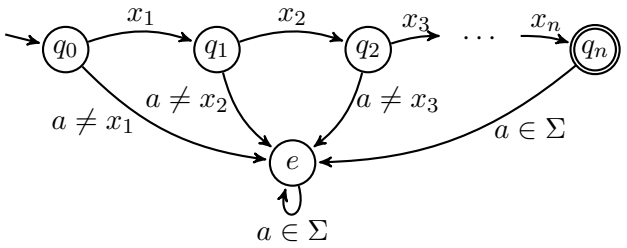
$$\mathsf{REG} = \{L(M) \mid M \text{ ist ein DFA}\}.$$

Um ein intuitives Verständnis für die Berechnungskraft von DFAs zu entwickeln, werden wir Antworten auf folgende Frage suchen.

Frage: Welche Sprachen gehören zu REG und welche nicht?

Dabei legen wir unseren Überlegungen ein beliebiges aber fest gewähltes Alphabet $\Sigma = \{a_1, \ldots, a_m\}$ zugrunde.

Beobachtung 4. *Alle Sprachen, die aus einem einzigen Wort $x = x_1 \cdots x_n \in \Sigma^*$ bestehen (diese Sprachen werden auch als* Singletonsprachen *bezeichnet), 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 $Z = \{q_0, \ldots, q_n, e\}$, $E = \{q_n\}$ und der Überführungsfunktion

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

beschreiben.

Als nächstes betrachten wir Abschlusseigenschaften der Sprachklasse REG.

Definition 5. *Ein* **(k-stelliger) Sprachoperator** *ist eine Abbildung op, die k Sprachen $L_1, \ldots, L_k$ auf eine Sprache $op(L_1, \ldots, L_k)$ abbildet.*

Beispiel 6. *Der 2-stellige Schnittoperator bildet zwei Sprachen L_1 und L_2 auf die Sprache $L_1 \cap L_2$ ab.* ◁

Definition 7. *Eine Sprachklasse $\mathcal{K}$ heißt unter op* **abgeschlossen**, *wenn gilt:*

$$L_1, \ldots, L_k \in \mathcal{K} \Rightarrow op(L_1, \ldots, L_k) \in \mathcal{K}.$$

Der **Abschluss** *von $\mathcal{K}$ unter op ist die kleinste Sprachklasse $\mathcal{K}'$, die $\mathcal{K}$ enthält und unter op abgeschlossen ist.*

Definition 8. *Für eine Sprachklasse $\mathcal{C}$ bezeichne co-$\mathcal{C}$ die Klasse $\{\bar{L} \mid L \in \mathcal{C}\}$ aller Komplemente von Sprachen in $\mathcal{C}$.*

Es ist leicht zu sehen, dass $\mathcal{C}$ genau dann unter Komplementbildung abgeschlossen ist, wenn *co-*$\mathcal{C} = \mathcal{C}$ ist.

Beobachtung 9. *Mit $L_1, L_2 \in$ REG sind auch die Sprachen $\overline{L_1} = \Sigma^* \setminus L_1$, $L_1 \cap L_2$ und $L_1 \cup L_2$ regulär. Sind nämlich $M_i = (Z_i, \Sigma, \delta_i, q_0, E_i)$, $i = 1, 2$, DFAs mit $L(M_i) = L_i$, so akzeptiert der DFA*

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

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

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

mit

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

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

Aus Beobachtung 9 folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da die in Beispiel 3 betrachtete Sprache weder endlich noch co-endlich ist, haben wir damit allerdings noch nicht alle regulären Sprachen erfasst.

Es stellt sich die Frage, ob REG neben den mengentheoretischen Operationen Schnitt, Vereinigung und Komplement unter weiteren Operationen wie etwa der **Produktbildung**

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

(auch **Verkettung** oder **Konkatenation** genannt) 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 definiert durch

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

Die **Plushülle** von L ist

$$L^+ = \bigcup_{n \geq 1} L^n = LL^*.$$

Ist $L_1 = \{x\}$ eine Singletonsprache, so schreiben wir für das Produkt $\{x\}L_2$ auch einfach xL_2.

Im übernächsten Abschnitt werden wir sehen, dass die Klasse REG als der Abschluss der endlichen Sprachen unter Vereinigung, Produktbildung und Sternhülle charakterisierbar ist.

Beim Versuch, einen endlichen Automaten für das Produkt $L_1 L_2$ zweier regulärer Sprachen zu konstruieren, stößt man auf die Schwierigkeit, den richtigen Zeitpunkt für den Übergang von (der Simulation von) M_1 zu M_2 zu finden. Unter Verwendung eines nichtdeterministischen Automaten lässt sich dieses Problem jedoch leicht beheben, da dieser den richtigen Zeitpunkt „erraten“ kann.

Im nächsten Abschnitt werden wir nachweisen, dass auch nichtdeterministische endliche Automaten nur reguläre Sprachen erkennen können.

2.2 Nichtdeterministische endliche Automaten

Definition 10. *Ein* ***nichtdeterministischer endlicher Automat*** *(kurz: NFA; nondeterministic finite automaton) $N = (Z, \Sigma, \delta, Q_0, E)$ ist ähnlich aufgebaut wie ein DFA, nur dass er mehrere Startzustände (zusammengefasst in der Menge $Q_0 \subseteq Z$) haben kann und seine Überführungsfunktion die Form*

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

hat. Hierbei bezeichnet $\mathcal{P}(Z)$ die Potenzmenge *(also die Menge aller Teilmengen) von Z. Diese wird auch oft mit 2^Z bezeichnet. Die von N akzeptierte Sprache ist*

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

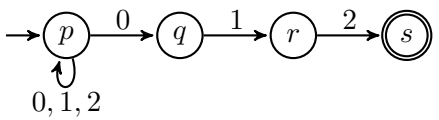
Ein NFA kann also nicht nur eine, sondern mehrere verschiedene Rechnungen ausführen. Die Eingabe gehört bereits dann zu $L(N)$, wenn bei einer dieser Rechnungen nach Lesen des gesamten Eingabewortes ein Endzustand erreicht wird.

Im Gegensatz zu einem DFA, dessen Überführungsfunktion auf der gesamten Menge $Z \times \Sigma$ definiert ist, kann ein NFA „stecken bleiben“. Das ist dann der Fall, wenn er in einen Zustand q gelangt, in dem das nächste Eingabezeichen x_i wegen $\delta(q, x_i) = \emptyset$ nicht gelesen werden kann.

Beispiel 11. *Betrachte den NFA $N = (Z, \Sigma, \delta, Q_0, E)$ mit Zustandsmenge $Z = \{p, q, r, s\}$, Eingabealphabet $\Sigma = \{0, 1, 2\}$, Start- und Endzustandsmenge $Q_0 = \{p\}$ und $E = \{s\}$ sowie der Überführungsfunktion*

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

Graphische Darstellung:



Offensichtlich akzeptiert N die Sprache $L(N) = \{x012 \mid x \in \Sigma^*\}$ aller Wörter, die mit dem Suffix 012 enden. ◁

Beobachtung 12. *Sind $N_i = (Z_i, \Sigma, \delta_i, Q_i, E_i)$ $(i = 1, 2)$ NFAs, so werden auch die Sprachen $L(N_1)L(N_2)$ und $L(N_1)^*$ von einem NFA erkannt. Wir können $Z_1 \cap Z_2 = \emptyset$ annehmen. Dann akzeptiert der*

NFA

$$N = (Z_1 \cup Z_2, \Sigma, \delta, Q_1, E)$$

mit

$$\delta(p, a) = \begin{cases} \delta_1(p, a), & p \in Z_1 \setminus E_1, \\ \delta_1(p, a) \cup \bigcup_{q \in Q_2} \delta_2(q, a), & p \in E_1, \\ \delta_2(p, a), & \text{sonst} \end{cases}$$

und

$$E = \begin{cases} E_1 \cup E_2, & Q_2 \cap E_2 \neq \emptyset \\ E_2, & \text{sonst} \end{cases}$$

die Sprache $L(N_1)L(N_2)$ und der NFA

$$N^* = (Z_1 \cup \{q_{neu}\}, \Sigma, \delta^*, Q_1 \cup \{q_{neu}\}, E_1 \cup \{q_{neu}\})$$

mit

$$\delta^*(p, a) = \begin{cases} \delta(p, a) \cup \bigcup_{q \in Q_1} \delta(q, a), & p \in E_1, \\ \delta(p, a), & \text{sonst} \end{cases}$$

die Sprache $L(N_1)^*$.

Satz 13 (Rabin und Scott)**.**
$\mathsf{REG} = \{L(N) \mid N \text{ ist ein NFA}\}$.

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 $N = (Z, \Sigma, \delta, Q_0, E)$ einen DFA $M = (\mathcal{P}(Z), \Sigma, \delta', Q_0, E')$ mit $L(M) = L(N)$. Wir definieren die Überführungsfunktion $\delta' : \mathcal{P}(Z) \times \Sigma \to \mathcal{P}(Z)$ von M mittels

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

Die Menge $\delta'(Q, a)$ enthält also alle Zustände, in die N gelangen kann, wenn N ausgehend von einem beliebigen Zustand $q \in Q$ das Zeichen

a liest. Intuitiv bedeutet dies, dass der DFA M den NFA N simuliert, indem M in seinem aktuellen Zustand Q die Information speichert, in welchen Zuständen sich N momentan befinden könnte. Für die Erweiterung $\hat{\delta}' : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ von δ' (siehe Seite 2) können wir nun folgende Behauptung zeigen:

> $\hat{\delta}'(Q_0, x)$ enthält alle Zustände, die N ausgehend von einem Startzustand nach Lesen der Eingabe x erreichen kann.

Wir beweisen die Behauptung induktiv über die Länge n von x.

Induktionsanfang ($n = 0$): klar, da $\hat{\delta}'(Q_0, \varepsilon) = Q_0$ ist.

Induktionsschritt ($n - 1 \rightsquigarrow n$): Sei $x = x_1 \cdots x_n$ gegeben. Nach Induktionsvoraussetzung enthält

$$Q_{n-1} = \hat{\delta}'(Q_0, x_1 \cdots x_{n-1})$$

alle Zustände, die $N(x)$ in genau $n-1$ Schritten erreichen kann. Wegen

$$\hat{\delta}'(Q_0, x) = \delta'(Q_{n-1}, x_n) = \bigcup_{q \in Q_{n-1}} \delta(q, x_n)$$

enthält dann aber $\hat{\delta}'(Q_0, x)$ alle Zustände, die $N(x)$ in genau n Schritten erreichen kann.

Deklarieren wir nun diejenigen Teilmengen $Q \subseteq Z$, die mindestens einen Endzustand von N enthalten, als Endzustände des **Potenzmengenautomaten** M, d.h.

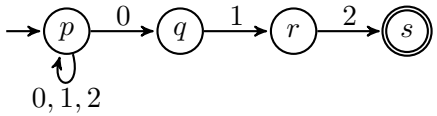
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \emptyset\},$$

so folgt für alle Wörter $x \in \Sigma^*$:

$$\begin{aligned} x \in L(N) &\Leftrightarrow N(x) \text{ kann in genau } |x| \text{ Schritten einen Endzustand erreichen} \\ &\Leftrightarrow \hat{\delta}'(Q_0, x) \cap E \neq \emptyset \\ &\Leftrightarrow \hat{\delta}'(Q_0, x) \in E' \\ &\Leftrightarrow x \in L(M). \end{aligned}$$

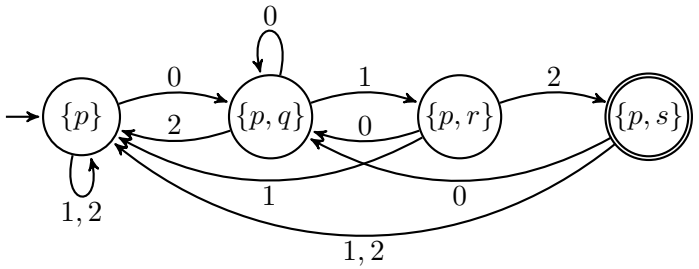
■

Beispiel 14. *Für den NFA $N = (Z, \Sigma, \delta, Q_0, E)$ aus Beispiel 11*



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

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



◁

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

Korollar 15. *Die Klasse* REG *der regulären Sprachen ist unter folgenden Operationen abgeschlossen:*

- *Komplement,*
- *Durchschnitt,*
- *Vereinigung,*
- *Produkt,*
- *Sternhülle.*

2.3 Reguläre Ausdrücke

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

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

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

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

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

Definition 16. *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* $\emptyset$*,* ϵ *und* a *(*$a \in \Sigma$*) sind reguläre Ausdrücke, die*

- *die leere Sprache* $L(\emptyset) = \emptyset$*,*
- *die Sprache* $L(\epsilon) = \{\varepsilon\}$ *und*
- *für jedes Zeichen* $a \in \Sigma$ *die Sprache* $L(a) = \{a\}$

beschreiben. Sind α *und* β *reguläre Ausdrücke, die die Sprachen* $L(\alpha)$ *und* $L(\beta)$ *beschreiben, so sind auch* $\alpha\beta$*,* $(\alpha|\beta)$ *und* $(\alpha)^*$ *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 17. *Die regulären Ausdrücke* ϵ^**,* $\emptyset^*$*,* $(0|1)^*00$ *und* $(\epsilon 0|\emptyset 1^*)$ *beschreiben folgende Sprachen:*

γ	ϵ^*	$\emptyset^*$	$(0\|1)^*00$	$(\epsilon 0\|\emptyset 1^*)$
$L(\gamma)$	$\{\varepsilon\}^* = \{\varepsilon\}$	$\emptyset^* = \{\varepsilon\}$	$\{x00 \mid x \in \{0,1\}^*\}$	$\{0\}$

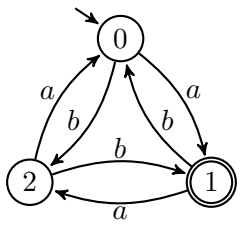
◁

Bemerkung 18.

- *Um Klammern zu sparen, definieren wir folgende* ***Präzedenzordnung****: Der Sternoperator* * *bindet stärker als der Produktoperator und dieser wiederum stärker als der Vereinigungsoperator. Für* $((a|b(c)^*)|d)$ *können wir also kurz* $a|bc^*|d$ *schreiben.*
- *Da der reguläre Ausdruck* $\gamma\gamma^*$ *die Sprache* $L(\gamma)^+$ *beschreibt, verwenden wir* γ^+ *als Abkürzung für den Ausdruck* $\gamma\gamma^*$*.*

Beispiel 19. *Betrachte nebenstehenden DFA* M*. Um für die von* M *erkannte Sprache*

$$L(M) = \{x \in \{a,b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}$$



einen regulären Ausdruck zu finden, betrachten wir zunächst die Sprache

$$L_0 = \{x \in \{a,b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 0\}.$$

L_0 enthält also alle Wörter x, die den DFA M ausgehend vom Zustand 0 in den Zustand 0 überführen. Jedes solche x setzt sich aus beliebig vielen Teilwörtern y zusammen, die M vom Zustand 0 in den Zustand 0 überführen, ohne zwischendurch den Zustand 0 anzunehmen. Jedes solche y beginnt entweder mit einem a (Übergang von 0 nach 1) oder mit einem b (Übergang von 0 nach 2). Im ersten Fall

folgt eine beliebige Anzahl von Teilwörtern ab (Wechsel zwischen 1 *und* 2*), an die sich entweder das Suffix aa (Rückkehr von* 1 *nach* 0 *über* 2*) oder das Suffix b (direkte Rückkehr von* 1 *nach* 0*) anschließt. Analog folgt im zweiten Fall eine beliebige Anzahl von Teilwörtern ba (Wechsel zwischen* 2 *und* 1*), an die sich entweder das Suffix a (direkte Rückkehr von* 2 *nach* 0*) oder das Suffix bb (Rückkehr von* 2 *nach* 0 *über* 1*) anschließt. Daher lässt sich* L_0 *durch den regulären Ausdruck*

$$\gamma_0 = (a(ab)^*(aa|b) \mid b(ba)^*(a|bb))^*$$

beschreiben. Eine ähnliche Überlegung zeigt, dass sich die Wörter, die M *ausgehend von* 0 *in den Zustand* 1 *überführen, ohne dass zwischendurch der Zustand* 0 *nochmals besucht wird, durch den regulären Ausdruck* $(a|bb)(ab)^*$ *beschrieben werden. Somit erhalten wir für* $L(M)$ *den regulären Ausdruck* $\gamma = \gamma_0(a|bb)(ab)^*$. ◁

Satz 20. $\mathsf{REG} = \{L(\gamma) \mid \gamma$ *ist ein regulärer Ausdruck*$\}$.

Beweis. Die Inklusion von rechts nach links ist klar, da die Basisausdrücke $\emptyset$, ϵ und a, $a \in \Sigma^*$, nur reguläre Sprachen beschreiben und die Sprachklasse REG unter Produkt, Vereinigung und Sternhülle abgeschlossen ist (siehe Beobachtungen 9 und 12).

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 $Z = \{1, \ldots, m\}$ 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. Folglich reicht es zu zeigen, dass die Sprachen $L_{p,q}$ durch reguläre Ausdrücke beschreibbar sind. Hierzu betrachten wir die Sprachen

$$L^r_{p,q} = \left\{ x_1 \cdots x_n \in \Sigma^* \;\middle|\; \begin{array}{c} \hat{\delta}(p, x_1 \cdots x_n) = q \text{ und für} \\ i = 1, \ldots, n-1 \text{ gilt } \hat{\delta}(p, x_1 \cdots x_i) \leq r \end{array} \right\}.$$

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

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

nur Buchstaben (und eventuell das leere Wort) und ist somit leicht durch einen regulären Ausdruck $\gamma^0_{p,q}$ beschreibbar. Wegen

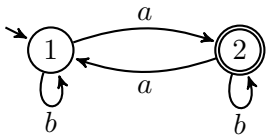
$$L^{r+1}_{p,q} = L^r_{p,q} \cup L^r_{p,r+1}(L^r_{r+1,r+1})^* L^r_{r+1,q}$$

lassen sich aus den regulären Ausdrücken $\gamma^r_{p,q}$ für die Sprachen $L^r_{p,q}$ leicht reguläre Ausdrücke für die Sprachen $L^{r+1}_{p,q}$ gewinnen:

$$\gamma^{r+1}_{p,q} = \gamma^r_{p,q} | \gamma^r_{p,r+1}(\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}.$$

■

Beispiel 21. *Betrachte den DFA*



Da M *insgesamt* $m = 2$ *Zustände und nur den Endzustand* 2 *besitzt, ist*

$$L(M) = \bigcup_{q \in E} L_{1,q} = L_{1,2} = L^2_{1,2} = L(\gamma^2_{1,2}).$$

Um $\gamma^2_{1,2}$ zu berechnen, benutzen wir die Rekursionsformel

$$\gamma^{r+1}_{p,q} = \gamma^r_{p,q} | \gamma^r_{p,r+1} (\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}$$

und erhalten

$$\begin{aligned}
\gamma^2_{1,2} &= \gamma^1_{1,2} | \gamma^1_{1,2} (\gamma^1_{2,2})^* \gamma^1_{2,2}, \\
\gamma^1_{1,2} &= \gamma^0_{1,2} | \gamma^0_{1,1} (\gamma^0_{1,1})^* \gamma^0_{1,2}, \\
\gamma^1_{2,2} &= \gamma^0_{2,2} | \gamma^0_{2,1} (\gamma^0_{1,1})^* \gamma^0_{1,2}.
\end{aligned}$$

Um den regulären Ausdruck $\gamma^2_{1,2}$ für $L(M)$ zu erhalten, genügt es also, die regulären Ausdrücke $\gamma^0_{1,1}$, $\gamma^0_{1,2}$, $\gamma^0_{2,1}$, $\gamma^0_{2,2}$, $\gamma^1_{1,2}$ und $\gamma^1_{2,2}$ zu berechnen:

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

◁

Korollar 22. *Sei L eine Sprache. Dann sind folgende Aussagen äquivalent:*

- *L ist regulär,*
- *es gibt einen DFA M mit $L = L(M)$,*
- *es gibt einen NFA N mit $L = L(N)$,*
- *es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$,*
- *L lässt sich mit den Operationen Vereinigung, Produkt und Sternhülle aus endlichen Sprachen gewinnen,*
- *L lässt sich mit den Operationen $\cap$, $\cup$, Komplement, Produkt und Sternhülle aus endlichen Sprachen gewinnen.*

Wir werden bald noch eine weitere Charakterisierung von REG kennenlernen, nämlich durch reguläre Grammatiken. Zuvor befassen wir uns jedoch mit dem Problem, DFAs zu minimieren. Dabei spielen Relationen (insbesondere Äquivalenzrelationen) eine wichtige Rolle.

2.4 Relationalstrukturen

Sei A eine nichtleere Menge, R_i eine k_i-stellige Relation auf A, d.h. $R_i \subseteq A^{k_i}$ für $i = 1, \ldots, n$. Dann heißt $(A; R_1, \ldots, R_n)$ ***Relationalstruktur***. Die Menge A heißt ***Grundmenge***, ***Trägermenge*** oder ***Individuenbereich*** der Relationalstruktur.

Wir werden hier hauptsächlich den Fall $n = 1$, $k_1 = 2$, also (A, R) mit $R \subseteq A \times A$ betrachten. Man nennt dann R eine **(binäre) Relation** auf A. Oft wird für $(a, b) \in R$ auch die **Infix-Schreibweise** aRb benutzt.

Beispiel 23.

- (F, M) *mit* $F = \{f \mid f \text{ ist Fluss in Europa}\}$ *und*
$$M = \{(f, g) \in F \times F \mid f \text{ mündet in } g\}.$$
- (U, B) *mit* $U = \{x \mid x \text{ ist Berliner}\}$ *und*
$$B = \{(x, y) \in U \times U \mid x \text{ ist Bruder von } y\}.$$
- $(\mathsf{P}(M), \subseteq)$, *wobei* $\mathsf{P}(M)$ *die Potenzmenge einer beliebigen Menge M und $\subseteq$ die Inklusionsbeziehung auf den Teilmengen von M ist.*
- (A, Id_A), *wobei* $Id_A = \{(x, x) \mid x \in A\}$ *die* ***Identität auf*** A *ist.*
- $(\mathbb{R}, \leq)$.

- $(\mathbb{Z}, |)$, *wobei* $|$ *die "teilt"-Relation bezeichnet.*
- $(\mathcal{F}ml, \Rightarrow)$ *mit* $\mathcal{F}ml = \{F \mid F$ *ist aussagenlogische Formel*$\}$ *und*

$$\Rightarrow = \{(F, G) \in \mathcal{F}ml \times \mathcal{F}ml \mid G \text{ ist Folgerung von } F\}. \quad \triangleleft$$

Da Relationen Mengen sind, sind auf ihnen die mengentheoretischen Operationen ***Durchschnitt***, ***Vereinigung***, ***Komplement*** und ***Differenz*** definiert. Seien R und S Relationen auf A, dann ist

$$\begin{aligned} R \cap S &= \{(x, y) \in A \times A \mid xRy \wedge xSy\}, \\ R \cup S &= \{(x, y) \in A \times A \mid xRy \vee xSy\}, \\ R - S &= \{(x, y) \in A \times A \mid xRy \wedge \neg xSy\}, \\ \overline{R} &= (A \times A) - R. \end{aligned}$$

Sei allgemeiner $\mathcal{M} \subseteq \mathcal{P}(A \times A)$ eine beliebige Menge von Relationen auf A. Dann sind der **Schnitt über** $\mathcal{M}$ und die **Vereinigung über** $\mathcal{M}$ folgende Relationen:

$$\begin{aligned} \bigcap \mathcal{M} &= \{(x, y) \mid \forall R \in \mathcal{M} : xRy\}, \\ \bigcup \mathcal{M} &= \{(x, y) \mid \exists R \in \mathcal{M} : xRy\}. \end{aligned}$$

Weiterhin ist die **Inklusionsrelation** $R \subseteq S$ auf Relationen von Bedeutung:

$$R \subseteq S \Leftrightarrow \forall x, y : xRy \rightarrow xSy.$$

Die ***transponierte (konverse) Relation*** zu R ist

$$R^T = \{(y, x) \mid xRy\}.$$

R^T wird oft auch mit R^{-1} bezeichnet. Zum Beispiel ist $(\mathbb{R}, \leq^T) = (\mathbb{R}, \geq)$.

Seien R und S Relationen auf A. Das ***Produkt*** oder die ***Komposition*** von R und S ist

$$R \circ S = \{(x, z) \in A \times A \mid \exists y \in A : xRy \wedge ySz\}.$$

Beispiel 24. *Ist B die Relation "ist Bruder von", V "ist Vater von", M "ist Mutter von" und $E = V \cup M$ "ist Elternteil von", so ist $B \circ E$ die Onkel-Relation.* $\triangleleft$

Übliche Bezeichnungen für das Relationenprodukt sind auch $R;S$ und $R \cdot S$ oder einfach RS. Das n-fache Relationenprodukt $R \circ \cdots \circ R$ von R wird mit R^n bezeichnet. Dabei ist $R^0 = Id$.

Vorsicht: Das n-fache Relationenprodukt R^n von R sollte nicht mit dem n-fachen kartesischen Produkt $R \times \cdots \times R$ der *Menge* R verwechselt werden. Wir vereinbaren, dass R^n das n-fache Relationenprodukt bezeichnen soll, falls R eine Relation ist.

Eigenschaften von Relationen

Sei R eine Relation auf A. Dann heißt R

reflexiv,	falls $\forall x \in A : xRx$	(also $Id_A \subseteq R$)
irreflexiv,	falls $\forall x \in A : \neg xRx$	(also $Id_A \subseteq \overline{R}$)
symmetrisch,	falls $\forall x, y \in A : xRy \Rightarrow yRx$	(also $R \subseteq R^T$)
asymmetrisch,	falls $\forall x, y \in A : xRy \Rightarrow \neg yRx$	(also $R \subseteq \overline{R^T}$)
antisymmetrisch,	falls $\forall x, y \in A : xRy \wedge yRx \Rightarrow x = y$	(also $R \cap R^T \subseteq Id$)
konnex,	falls $\forall x, y \in A : xRy \vee yRx$	(also $A \times A \subseteq R \cup R^T$)
semikonnex,	falls $\forall x, y \in A : x \neq y \Rightarrow xRy \vee yRx$	(also $\overline{Id} \subseteq R \cup R^T$)
transitiv,	falls $\forall x, y, z \in A : xRy \wedge yRz \Rightarrow xRz$	(also $R^2 \subseteq R$)

gilt.

Beispiel 25.

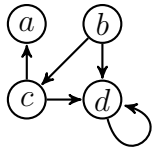
- *Die Relation "ist Schwester von" ist zwar in einer reinen Damengesellschaft symmetrisch, i.a. jedoch weder symmetrisch noch asymmetrisch noch antisymmetrisch.*

- $(\mathbb{R}, <)$ *ist irreflexiv, asymmetrisch, transitiv und semikonnex.*
- $(\mathbb{R}, \leq)$ *und* $(\mathsf{P}(M), \subseteq)$ *sind reflexiv, antisymmetrisch und transitiv.*
- $(\mathbb{R}, \leq)$ *ist auch konnex und* $(\mathsf{P}(M), \subseteq)$ *ist im Fall* $\|M\| \leq 1$ *zwar auch konnex, aber im Fall* $\|M\| \geq 2$ *weder semikonnex noch konnex.* ◁

Graphische Darstellung von Relationen

Eine Relation R auf einer endlichen Menge A kann durch einen ***gerichteten Graphen*** (oder ***Digraphen***) $G = (V, E)$ mit ***Knotenmenge*** $V = A$ und ***Kantenmenge*** $E = R$ veranschaulicht werden. Hierzu stellen wir jedes Element $x \in A$ als einen Knoten dar und verbinden jedes Knotenpaar $(x, y) \in R$ durch eine gerichtete Kante (Pfeil). Zwei durch eine Kante verbundene Knoten heißen ***benachbart*** oder ***adjazent***.

Beispiel 26. *Für die Relation* (A, R) *mit* $A = \{a, b, c, d\}$ *und* $R = \{(b, c), (b, d), (c, a), (c, d), (d, d)\}$ *erhalten wir folgende graphische Darstellung.*



◁

Der ***Ausgangsgrad*** eines Knotens $x \in V$ ist $\deg^+(x) = \|R(x)\|$, wobei $R(x) = \{y \in V \mid xRy\}$ der ***Nachbereich*** von x ist. Entsprechend ist $\deg^-(x) = \|\{y \in V \mid yRx\}\|$ der ***Eingangsgrad*** von x. Falls R symmetrisch ist, werden die Pfeilspitzen meist weggelassen. In diesem Fall ist $d(x) = \deg^-(x) = \deg^+(x)$ der ***Grad*** von x. Ist R zudem irreflexiv, so ist G ***schleifenfrei*** und wir erhalten einen ***(ungerichteten) Graphen***.

Darstellung durch eine Adjazenzmatrix

Eine Relation R auf einer endlichen (geordneten) Menge $A = \{a_1, \ldots, a_n\}$ lässt sich durch eine boolesche $n \times n$-Matrix $M_R = (m_{ij})$ mit

$$m_{ij} := \begin{cases} 1, & a_i R a_j, \\ 0, & \text{sonst} \end{cases}$$

darstellen. Beispielsweise hat die Relation

$$R = \{(b, c), (b, d), (c, a), (c, d), (d, d)\}$$

auf der Menge $A = \{a, b, c, d\}$ die Matrixdarstellung

$$M_R = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Darstellung durch eine Adjazenzliste

Eine weitere Möglichkeit besteht darin, eine endliche Relation R in Form einer Tabelle darzustellen, die jedem Element $x \in A$ seinen Nachbereich $R(x)$ in Form einer Liste zuordnet:

x	$R(x)$
a	-
b	c, d
c	a, d
d	d

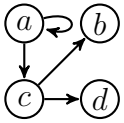
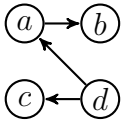
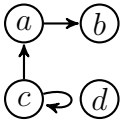
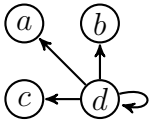
Sind $M_R = (r_{ij})$ und $M_S = (s_{ij})$ boolesche $n \times n$-Matrizen für R und S, so erhalten wir für $T = R \circ S$ die Matrix $M_T = (t_{ij})$ mit

$$t_{ij} = \bigvee_{k=1,\ldots,n} (r_{ik} \wedge s_{kj})$$

Der Nachbereich $T(x)$ von x bzgl. der Relation $T = R \circ S$ berechnet sich zu

$$T(x) = \bigcup\{S(y) \mid y \in R(x)\} = \bigcup_{y \in R(x)} S(y).$$

Beispiel 27. *Betrachte die Relationen* $R = \{(a,a),(a,c),(c,b),(c,d)\}$ *und* $S = \{(a,b),(d,a),(d,c)\}$ *auf der Menge* $A = \{a,b,c,d\}$.

Relation	R	S	$R \circ S$	$S \circ R$
Digraph				
Adjazenz-matrix	1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0	0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
Adjazenz-liste	$a : a, c$ b : - $c : b, d$ d : -	$a : b$ b : - c : - $d : a, c$	$a : b$ b : - $c : a, c$ d : -	a : - b : - c : - $d : a, b, c, d$

◁

Beobachtung: Das Beispiel zeigt, dass das Relationenprodukt nicht kommutativ ist, d.h. i.a. gilt nicht $R \circ S = S \circ R$.

Als nächstes zeigen wir, dass die Menge $\mathcal{R} = \mathcal{P}(A \times A)$ aller binären Relationen auf A mit dem Relationenprodukt $\circ$ als binärer Operation und der Relation Id_A als neutralem Element eine Halbgruppe (oder ***Monoid***) bildet.

Satz 28. *Seien Q, R, S Relationen auf A. Dann gilt*

(i) $(Q \circ R) \circ S = Q \circ (R \circ S)$, *d.h. $\circ$ ist assoziativ,*

(ii) $Id \circ R = R \circ Id = R$, *d.h. Id ist neutrales Element.*

Beweis.

(i) Es gilt:

$$\begin{aligned} x\ (Q \circ R) \circ S\ y &\Leftrightarrow \exists u \in A : x\ (Q \circ R)\ u\ \wedge\ u\ S\ y \\ &\Leftrightarrow \exists u \in A : (\exists v \in A : x\ Q\ v\ R\ u)\ \wedge\ u\ S\ y \\ &\Leftrightarrow \exists u, v \in A : x\ Q\ v\ R\ u\ S\ y \\ &\Leftrightarrow \exists v \in A : x\ Q\ v\ \wedge\ (\exists u \in A : v\ R\ u\ \wedge\ u\ S\ y) \\ &\Leftrightarrow \exists v \in A : x\ Q\ v\ (R \circ S)\ y \\ &\Leftrightarrow x\ Q \circ (R \circ S)\ y \end{aligned}$$

(ii) Wegen $x\ Id \circ R\ y \Leftrightarrow \exists z : x = z\ \wedge\ z\ R\ y \Leftrightarrow x\ R\ y$ folgt $Id \circ R = R$. Die Gleichheit $R \circ Id = R$ folgt analog.

■

Manchmal steht man vor der Aufgabe, eine gegebene Relation R durch eine möglichst kleine Modifikation in eine Relation R' mit vorgegebenen Eigenschaften zu überführen. Will man dabei alle in R enthaltenen Paare beibehalten, dann sollte R' aus R durch Hinzufügen möglichst weniger Paare hervorgehen.

Es lässt sich leicht nachprüfen, dass der Schnitt über eine Menge reflexiver (bzw. transitiver oder symmetrischer) Relationen wieder reflexiv (bzw. transitiv oder symmetrisch) ist. Folglich existiert zu jeder Relation R auf einer Menge A eine kleinste reflexive (bzw. transitive oder symmetrische) Relation R', die R enthält.

Definition 29. *Sei R eine Relation.*

- *Die* ***reflexive Hülle*** *von R ist*

$$h_{refl}(R) = \bigcap\{S \subseteq A \times A \mid S \text{ ist reflexiv und } R \subseteq S\}.$$

- *Die* ***symmetrische Hülle*** *von R ist*

$$h_{sym}(R) = \bigcap\{S \subseteq A \times A \mid S \text{ ist symmetrisch und } R \subseteq S\}.$$

- *Die* ***transitive Hülle*** *von R ist*

$$R^+ = \bigcap\{S \subseteq A \times A \mid S \text{ ist transitiv und } R \subseteq S\}.$$

- *Die **reflexiv-transitive Hülle** von R ist*

$$R^* = \bigcap\{S \subseteq A \times A \mid S \text{ ist reflexiv, transitiv und } R \subseteq S\}.$$

Satz 30. *Sei R eine Relation auf A.*

(i) $h_{refl}(R) = R \cup Id_A$,

(ii) $h_{sym}(R) = R \cup R^T$,

(iii) $R^+ = \bigcup_{n\geq 1} R^n$,

(iv) $R^* = \bigcup_{n\geq 0} R^n$.

Beweis. Siehe Übungen. ■

Anschaulich besagt der vorhergehende Satz, dass ein Paar (a, b) genau dann in der reflexiv-transitiven Hülle R^* von R ist, wenn es ein $n \geq 0$ gibt mit aR^nb, d.h. es gibt Elemente $x_0, \ldots, x_n \in A$ mit $x_0 = a$, $x_n = b$ und

$$x_0 R x_1 R x_2 \cdots x_{n-1} R x_n.$$

In der Graphentheorie nennt man $x_0, \ldots, x_n$ einen ***Weg*** der Länge n von a nach b.

2.4.1 Äquivalenz- und Ordnungsrelationen

Die nachfolgende Tabelle gibt einen Überblick über die wichtigsten Relationalstrukturen.

	refl.	sym.	trans.	antisym.	asym.	konnex	semikon.
Äquivalenzrelation	✓	✓	✓				
(Halb-)Ordnung	✓		✓	✓			
Striktordnung			✓		✓		
lineare Ordnung			✓	✓		✓	
lin. Striktord.			✓		✓		✓
Quasiordnung	✓		✓				

In der Tabelle sind nur die definierenden Eigenschaften durch ein "✓" gekennzeichnet. Das schließt nicht aus, dass gleichzeitig auch noch weitere Eigenschaften vorliegen können.

Wir betrachten zunächst ***Äquivalenzrelationen***, die durch die drei Eigenschaften reflexiv, symmetrisch und transitiv definiert sind.

Ist E eine Äquivalenzrelation, so nennt man den zu x gehörigen Nachbereich $E(x)$ die ***von x repräsentierte Äquivalenzklasse*** und bezeichnet sie mit $[x]_E$ oder einfach mit $[x]$. Die durch E auf A induzierte Partition $\{[x] \mid x \in A\}$ wird ***Quotienten- oder Faktormenge*** genannt und mit A/E bezeichnet. Die Anzahl der Äquivalenzklassen von E wird auch als der ***Index*** von E bezeichnet. Eine Menge $S \subseteq A$ heißt ***Repräsentantensystem***, falls sie genau ein Element aus jeder Äquivalenzklasse enthält.

Beispiel 31.

- *Auf der Menge aller Geraden im $\mathbb{R}^2$ die Parallelität. Offenbar bilden alle Geraden mit derselben Richtung (oder Steigung) jeweils eine Äquivalenzklasse. Daher wird ein Repräsentantensystem beispielsweise durch die Menge aller Ursprungsgeraden gebildet.*
- *Auf der Menge aller Menschen "im gleichen Jahr geboren wie". Hier bildet jeder Jahrgang eine Äquivalenzklasse.*
- *Auf $\mathbb{Z}$ die Relation "gleicher Rest bei Division durch m". Die zugehörigen Äquivalenzklassen sind*

$$[r] = \{a \in \mathbb{Z} \mid a \bmod m = r\}.$$

  Ein Repräsentantensystem wird also durch die Reste $\{0, 1, \ldots, m-1\}$ gebildet.
- *Auf der Menge der aussagenlogischen Formeln die semantische Äquivalenz. Hier bilden beispielsweise alle Tautologien eine Äquivalenzklasse.* ◁

Definition 32. *Eine Familie $\{M_i \mid i \in I\}$ von nichtleeren Teilmengen $M_i \subseteq A$ heißt **Partition** der Menge A, falls gilt:*

*a) die Mengen M_i **überdecken** A, d.h. $A = \bigcup_{i \in I} M_i$ und*

*b) die Mengen M_i sind **paarweise disjunkt**, d.h. für je zwei verschiedene Mengen $M_i \neq M_j$ gilt $M_i \cap M_j = \emptyset$.*

Wie der nächste Satz zeigt, beschreiben Äquivalenzrelationen auf A und Partitionen von A denselben Sachverhalt.

Satz 33. *Sei E eine Relation auf A. Dann sind folgende Aussagen äquivalent.*

(i) E ist eine Äquivalenzrelation auf A.

(ii) Für alle $x, y \in A$ gilt

$$xEy \Leftrightarrow E(x) = E(y) \qquad (*)$$

(iii) E ist reflexiv und $\{E(x) \mid x \in A\}$ ist eine Partition von A.

Beweis.

$(i) \Rightarrow (ii)$ Sei E eine Äquivalenzrelation auf A. Da E transitiv ist, impliziert xEy die Inklusion $E(y) \subseteq E(x)$:

$$z \in E(y) \Rightarrow yEz \Rightarrow xEz \Rightarrow z \in E(x).$$

Da E symmetrisch ist, folgt aus xEy aber auch $E(x) \subseteq E(y)$.

Umgekehrt folgt aus $E(x) = E(y)$ wegen der Reflexivität von E, dass $x \in E(x) = E(y)$ enthalten ist, und somit xEy. Dies zeigt, dass E die Äquivalenz $(*)$ erfüllt.

$(ii) \Rightarrow (iii)$ Falls E die Bedingung $(*)$ erfüllt, so folgt sofort xEx (wegen $E(x) = E(x)$) und folglich überdecken die Nachbereiche $E(x)$ (wegen $x \in E(x)$) die Menge A.

Ist $E(x) \cap E(y) \neq \emptyset$ und z ein Element in $E(x) \cap E(y)$, so gilt xEz und yEz und daher folgt $E(x) = E(z) = E(y)$. Da also je zwei Nachbereiche $E(x)$ und $E(y)$ entweder gleich oder disjunkt sind, bildet $\{E(x) \mid x \in A\}$ sogar eine Partition von A.

$(iii) \Rightarrow (i)$ Wird schließlich A von den Mengen $E(x)$ partitioniert, wobei $x \in E(x)$ für alle $x \in A$ gilt, so folgt

$$xEy \Leftrightarrow y \in E(x) \cap E(y) \Leftrightarrow E(x) = E(y).$$

Daher übertragen sich die Eigenschaften Reflexivität, Symmetrie und Transitivität unmittelbar von der Gleichheitsrelation auf E.

■

Die kleinste Äquivalenzrelation auf A ist die **Identität** Id_A, die größte die **Allrelation** $A \times A$. Die Äquivalenzklassen der Identität enthalten jeweils nur ein Element, d.h. $A/Id_A = \{\{x\} \mid x \in A\}$, und die Allrelation erzeugt nur eine Äquivalenzklasse, nämlich $A/(A \times A) = \{A\}$.

Für zwei Äquivalenzrelationen $E \subseteq E'$ sind auch die Äquivalenzklassen $[x]_E$ von E in den Klassen $[x]_{E'}$ von E' enthalten. Folglich ist jede Äquivalenzklasse von E' die Vereinigung von (evtl. mehreren) Äquivalenzklassen von E_1. Im Fall $E \subseteq E'$ sagt man auch, E_1 bewirkt eine **feinere** Partitionierung als E_2. Demnach ist die Identität die **feinste** und die Allrelation die **gröbste** Äquivalenzrelation.

Da der Schnitt über eine Menge von Äquivalenzrelationen wieder eine Äquivalenzrelation ist, können wir für eine beliebige Relation R auf einer Menge A die kleinste R umfassende Äquivalenzrelation definieren:

$$h_{\text{äq}}(R) := \bigcap \{E \mid E \text{ ist eine Äquivalenzrelation auf } A \text{ mit } R \subseteq E\}$$

In der Sprache der Graphentheorie werden die durch die Äquivalenzklassen von $h_{\text{äq}}(R)$ induzierten Teilgraphen auch die **schwachen Zusammenhangskomponenten** des Digraphen (A, R) genannt (siehe Übungen). Als nächstes betrachten wir Ordnungen.

Definition 34. *(A, R) heißt **Ordnung** (auch **Halbordnung** oder **partielle Ordnung**), wenn R eine reflexive, antisymmetrische und transitive Relation auf A ist.*

Beispiel 35.

- *$(\mathbb{Z}, \leq)$ und $(\mathbb{N}, |)$ sind Ordnungen. Erstere ist linear, letztere nicht.*
- *Für jede Menge M ist die relationale Struktur $(\mathcal{P}(M); \subseteq)$ eine Ordnung. Diese ist nur im Fall $\|M\| \leq 1$ linear.*
- *Auf der Menge $\mathcal{A}(M)$ aller Äquivalenzrelationen auf M die Relation "feiner als". Dabei ist, wie wir gesehen haben, E_1 eine Verfeinerung von E_2, falls E_1 in E_2 enthalten ist. In diesem Fall bewirkt E_1 nämlich eine feinere Klasseneinteilung auf M als E_2, da jede Äquivalenzklasse von E_1 in einer Äquivalenzklasse von E_2 enthalten ist.*
- *Ist R eine Ordnung auf A und $B \subseteq A$, so heißt die Ordnung $R_B = R \cap (B \times B)$ die **Einschränkung** (oder **Restriktion**) von R auf B. Beispielsweise ist $(\mathcal{A}(M); \subseteq)$ die Einschränkung von $(\mathcal{P}(M \times M); \subseteq)$ auf $\mathcal{A}(M)$.* ◁

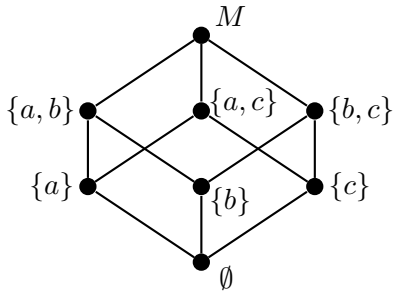
Ordnungen lassen sich sehr anschaulich durch Hasse-Diagramme darstellen. Sei $\leq$ eine Ordnung auf A und sei $<$ die Relation $\leq \cap \overline{Id}_A$. Um die Ordnung $\leq$ in einem ***Hasse-Diagramm*** darzustellen, wird nur der Graph der ***Nachbarrelation***

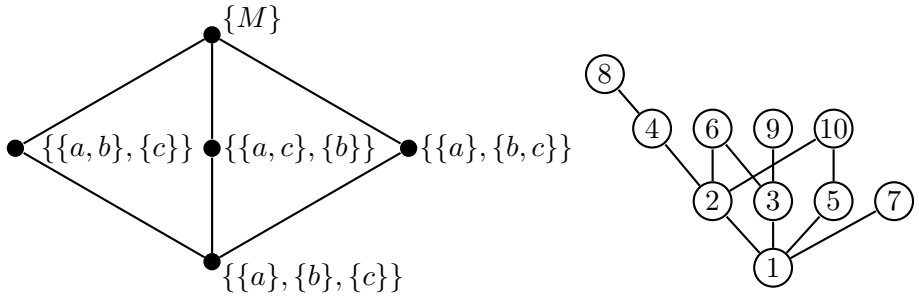
$$\lessdot \; = \; < \setminus <^2, \text{ d.h. } x \lessdot y \;\Leftrightarrow\; x < y \wedge \neg\exists z : x < z < y$$

gezeichnet. Für $x \lessdot y$ sagt man auch, y ist ***oberer Nachbar*** von x. Weiterhin wird im Fall $x \lessdot y$ der Knoten y oberhalb vom Knoten x gezeichnet, so dass auf Pfeilspitzen verzichtet werden kann.

Beispiel 36.

Die Inklusionsrelation auf der Potenzmenge $\mathcal{P}(M)$ von $M = \{a, b, c\}$ lässt sich durch nebenstehendes Hasse-Diagramm darstellen.





Das linke Hasse-Diagramm stellt die "feiner als" Relation auf der Menge aller Partitionen von $M = \{a, b, c\}$ dar. Rechts ist die Einschränkung der "teilt"-Relation auf die Zahlenmenge $\{1, 2, \ldots, 10\}$ abgebildet. ◁

Definition 37. *Sei $\leq$ eine Ordnung auf A und sei b ein Element in einer Teilmenge $B \subseteq A$.*

- *b heißt **kleinstes Element** oder **Minimum** von B (kurz $b = \min B$), falls gilt:*

$$\forall b' \in B : b \leq b'.$$

- *b heißt **größtes Element** oder **Maximum** von B (kurz $b = \max B$), falls gilt:*

$$\forall b' \in B : b' \leq b.$$

- *b heißt **minimal** in B, falls es in B kein kleineres Element gibt:*

$$\forall b' \in B : b' \leq b \Rightarrow b' = b.$$

- *b heißt **maximal** in B, falls es in B kein größeres Element gibt:*

$$\forall b' \in B : b \leq b' \Rightarrow b = b'.$$

Bemerkung 38. *Da Ordnungen antisymmetrisch sind, kann es in jeder Teilmenge B höchstens ein kleinstes und höchstens ein größtes Element geben. Die Anzahl der minimalen und maximalen Elemente in B kann dagegen beliebig groß sein.*

Definition 39. *Sei $\leq$ eine Ordnung auf A und sei $B \subseteq A$.*

- *Jedes Element $u \in A$ mit $u \leq b$ für alle $b \in B$ heißt* **untere** *und jedes $o \in A$ mit $b \leq o$ für alle $b \in B$ heißt* **obere Schranke** *von B.*
- *B heißt* **nach oben beschränkt***, wenn B eine obere Schranke hat, und* **nach unten beschränkt***, wenn B eine untere Schranke hat.*
- *B heißt* **beschränkt***, wenn B nach oben und nach unten beschränkt ist.*
- *Besitzt B eine größte untere Schranke i, d.h. besitzt die Menge U aller unteren Schranken von B ein größtes Element i, so heißt i das* **Infimum** *von B (kurz $i = \inf B$):*

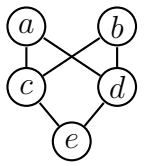
$$(\forall b \in B : b \geq i) \wedge [\forall u \in A : (\forall b \in B : b \geq u) \Rightarrow u \leq i].$$

- *Besitzt B eine kleinste obere Schranke s, d.h. besitzt die Menge O aller oberen Schranken von B ein kleinstes Element s, so heißt s das* **Supremum** *von B ($s = \sup B$):*

$$(\forall b \in B : b \leq s) \wedge [\forall o \in A : (\forall b \in B : b \leq o) \Rightarrow s \leq o]$$

Bemerkung 40. *B kann nicht mehr als ein Supremum und ein Infimum haben.*

Beispiel 41. *Betrachte nebenstehende Ordnung auf der Menge $A = \{a, b, c, d, e\}$. Die folgende Tabelle zeigt für verschiedene Teilmengen $B \subseteq A$ alle minimalen und maximalen Elemente in B Minimum und Maximum, alle unteren und oberen Schranken, sowie Infimum und Supremum von B (falls existent).*



B	*minimal*	*maximal*	*min*	*max*	*untere Schranken*	*obere Schranken*	*inf*	*sup*
$\{a, b\}$	a, b	a, b	-	-	c, d, e	-	-	-
$\{c, d\}$	c, d	c, d	-	-	e	a, b	e	-
$\{a, b, c\}$	c	a, b	c	-	c, e	-	c	-
$\{a, b, c, e\}$	e	a, b	e	-	e	-	e	-
$\{a, c, d, e\}$	e	a	e	a	e	a	e	a

◁

Bemerkung 42.

- *Auch in linearen Ordnungen muss nicht jede beschränkte Teilmenge ein Supremum oder Infimum besitzen.*
- *So hat in der linear geordneten Menge $(\mathbb{Q}, \leq)$ die Teilmenge*

$$B \;=\; \{x \in \mathbb{Q} \mid x^2 \leq 2\} \;=\; \{x \in \mathbb{Q} \mid x^2 < 2\}$$

  weder ein Supremum noch ein Infimum.
- *Dagegen hat in einer linearen Ordnung jede* endliche *Teilmenge ein kleinstes und ein größtes Element und somit erst recht ein Supremum und ein Infimum.*

2.4.2 Abbildungen

Definition 43. *Sei R eine binäre Relation auf einer Menge M.*

- *R heißt* ***rechtseindeutig****, falls gilt:*

$$\forall x, y, z \in M : xRy \wedge xRz \Rightarrow y = z.$$

- *R heißt* ***linkseindeutig****, falls gilt:*

$$\forall x, y, z \in M : xRz \wedge yRz \Rightarrow x = y.$$

- *Der **Nachbereich** $N(R)$ und der **Vorbereich** $V(R)$ von R sind*
$$N(R) = \bigcup_{x \in M} R(x) \quad \text{und} \quad V(R) = \bigcup_{x \in M} R^T(x).$$
- *Eine rechtseindeutige Relation R mit $V(R) = A$ und $N(R) \subseteq B$ heißt **Abbildung** oder **Funktion von A nach B** (kurz $R : A \to B$).*

Bemerkung 44.

- *Wie üblich werden wir Abbildungen meist mit kleinen Buchstaben $f, g, h, \ldots$ bezeichnen und für $(x, y) \in f$ nicht xfy sondern $f(x) = y$ oder $f : x \mapsto y$ schreiben.*
- *Ist $f : A \to B$ eine Abbildung, so wird der Vorbereich $V(f) = A$ der **Definitionsbereich** und die Menge B der **Wertebereich** oder **Wertevorrat** von f genannt.*
- *Der Nachbereich $N(f)$ wird als **Bild** von f bezeichnet.*

Definition 45.

- *Im Fall $N(f) = B$ heißt f **surjektiv**.*
- *Ist f linkseindeutig, so heißt f **injektiv**. In diesem Fall impliziert $f(x) = f(y)$ die Gleichheit $x = y$.*
- *Eine injektive und surjektive Abbildung heißt **bijektiv**.*
- *Für eine injektive Abbildung $f : A \to B$ ist auch f^T eine Abbildung, die mit f^{-1} bezeichnet und die **inverse Abbildung** zu f genannt wird.*

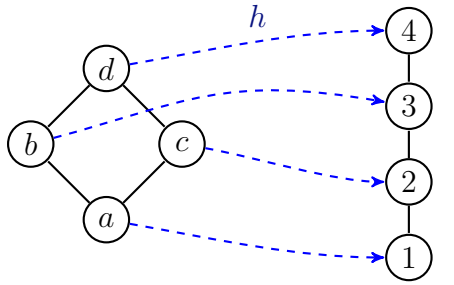
Man beachte, dass der Definitionsbereich $V(f^{-1}) = N(f)$ von f^{-1} nur dann gleich B ist, wenn f auch surjektiv, also eine Bijektion ist.

2.4.3 Homo- und Isomorphismen

Definition 46. *Seien (A_1, R_1) und (A_2, R_2) Relationalstrukturen.*

- *Eine Abbildung $h : A_1 \to A_2$ heißt **Homomorphismus**, falls für alle $a, b \in A_1$ gilt:*
$$aR_1b \Rightarrow h(a)R_2h(b).$$
- *Sind (A_1, R_1) und (A_2, R_2) Ordnungen, so spricht man von **Ordnungshomomorphismen** oder einfach von **monotonen** Abbildungen.*
- *Injektive Ordnungshomomorphismen werden auch **streng monotone** Abbildungen genannt.*

Beispiel 47. *Folgende Abbildung $h : A_1 \to A_2$ ist ein bijektiver Ordnungshomomorphismus.*



Obwohl h ein bijektiver Homomorphismus ist, ist die Umkehrung h^{-1} kein Homomorphismus, da h^{-1} nicht monoton ist. Es gilt nämlich
$$2 \sqsubseteq 3, \text{ aber } h^{-1}(2) = b \not\leq c = h^{-1}(3).$$
◁

Definition 48. *Ein bijektiver Homomorphismus $h : A_1 \to A_2$, bei dem auch h^{-1} ein Homomorphismus ist, d.h. es gilt*
$$\forall a, b \in A_1 : aR_1b \Leftrightarrow h(a)R_2h(b).$$
*heißt **Isomorphismus**. In diesem Fall heißen die Strukturen (A_1, R_1) und (A_2, R_2) **isomorph** (kurz: $(A_1, R_1) \cong (A_2, R_2)$).*

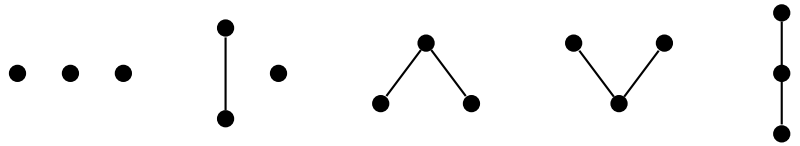
Beispiel 49.

- *Die Abbildung* $h : \mathbb{R} \to \mathbb{R}^+$ *mit*

$$h : x \mapsto e^x$$

ist ein Ordnungsisomorphismus zwischen $(\mathbb{R}, \leq)$ *und* $(\mathbb{R}^+, \leq)$.

- *Es existieren genau 5 nichtisomorphe Ordnungen mit 3 Elementen:*



Anders ausgedrückt: Die Klasse aller dreielementigen Ordnungen zerfällt unter der Äquivalenzrelation $\cong$ *in fünf Äquivalenzklassen, die durch obige fünf Hasse-Diagramme repräsentiert werden.*

- *Für* $n \in \mathbb{N}$ *sei*

$$T_n \;=\; \{k \in \mathbb{N} \mid k \text{ teilt } n\}$$

die Menge aller Teiler von n *und*

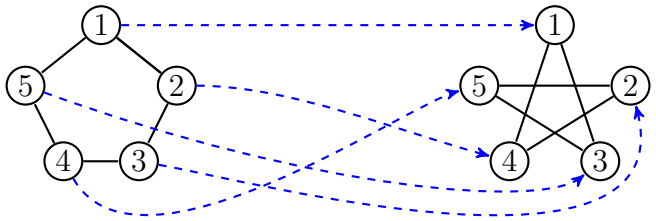
$$P_n \;=\; \{k \in \mathbb{N} \mid k \text{ ist Primteiler von } n\}$$

die Menge aller Primteiler von n. *Dann ist die Abbildung*

$$h : k \mapsto P_k$$

ein (surjektiver) Ordnungshomomorphismus von $(T_n, |)$ *auf* $(\mathcal{P}(P_n), \subseteq)$. h *ist sogar ein Isomorphismus, falls* n quadratfrei *ist (d.h. es gibt kein* $k \geq 2$, *so dass* k^2 *die Zahl* n *teilt).*

- *Die beiden folgenden Graphen* G *und* G' *sind isomorph. Zwei Isomorphismen sind beispielsweise* h_1 *und* h_2.



$G = (V, E)$ $\qquad$ $G' = (V, E')$

v	1 2 3 4 5
$h_1(v)$	1 3 5 2 4
$h_2(v)$	1 4 2 5 3

- *Während auf der Knotenmenge* $V = [3]$ *insgesamt* $2^3 = 8$ *verschiedene Graphen existieren, gibt es auf dieser Menge nur* 4 *verschiedene nichtisomorphe Graphen:*

◁

Bemerkung 50. *Auf der Knotenmenge* $V = \{1, \ldots, n\}$ *existieren genau* $2^{\binom{n}{2}}$ *verschiedene Graphen. Sei* $a(n)$ *die Anzahl aller nichtisomorphen Graphen auf* V. *Da jede Isomorphieklasse mindestens einen und höchstens* $n!$ *verschiedene Graphen enthält, ist* $2^{\binom{n}{2}}/n! \leq a(n) \leq 2^{\binom{n}{2}}$. *Tatsächlich ist* $a(n)$ ***asymptotisch gleich*** $u(n) = 2^{\binom{n}{2}}/n!$ *(in Zeichen:* $a(n) \sim u(n)$*), d.h.*

$$\lim_{n\to\infty} a(n)/u(n) = 1.$$

Also gibt es auf $V = \{1, \ldots, n\}$ *nicht wesentlich mehr als* $u(n)$ *nichtisomorphe Graphen.*

2.5 Minimierung von DFAs

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

Fassen wir alle zu einem Zustand z äquivalenten Zustände in dem neuen Zustand

$$[z]_\sim = \{z' \in Z \mid L_{z'} = L_z\}$$

zusammen (wofür wir auch kurz $[z]$ oder $\tilde{z}$ schreiben) und ersetzen wir Z und E durch $\tilde{Z} = \{\tilde{z} \mid z \in Z\}$ und $\tilde{E} = \{\tilde{z} \mid z \in E\}$, so erhalten wir den DFA $\tilde{M} = (\tilde{Z}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{E})$ mit

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

Hierbei bezeichnet $\tilde{Q}$ für eine Teilmenge $Q \subseteq Z$ die Menge $\{\tilde{q} \mid q \in Q\}$ aller Äquivalenzklassen $\tilde{q}$, die mindestens ein Element $q \in Q$ enthalten. Der nächste Satz zeigt, dass $\tilde{M}$ tatsächlich der gesuchte Minimalautomat ist.

Satz 51. *Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA, der nur Zustände enthält, die vom Startzustand q_0 aus erreichbar sind. Dann ist $\tilde{M} = (\tilde{Z}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{E})$ mit*

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

ein DFA für $L(M)$ mit einer minimalen Anzahl von Zuständen.

Beweis. Wir zeigen zuerst, dass $\tilde{\delta}$ wohldefiniert ist, also der Wert von $\tilde{\delta}(\tilde{q}, a)$ nicht von der Wahl des Repräsentanten q abhängt. Hierzu zeigen wir, dass im Fall $p \sim q$ auch $\delta(q, a)$ und $\delta(p, a)$ äquivalent sind:

$$\begin{aligned} L_q = L_p &\Rightarrow \forall x \in \Sigma^* : x \in L_q \leftrightarrow x \in L_p \\ &\Rightarrow \forall x \in \Sigma^* : ax \in L_q \leftrightarrow ax \in L_p \\ &\Rightarrow \forall x \in \Sigma^* : x \in L_{\delta(q,a)} \leftrightarrow x \in L_{\delta(p,a)} \\ &\Rightarrow L_{\delta(q,a)} = L_{\delta(p,a)}. \end{aligned}$$

Als nächstes zeigen wir, dass $L(\tilde{M}) = L(M)$ ist. Sei $x = x_1 \cdots x_n$ eine Eingabe und seien

$$q_i = \hat{\delta}(q_0, x_1 \cdots x_i),\ i = 0, \ldots, n$$

die von M beim Abarbeiten von x durchlaufenen Zustände. Wegen

$$\tilde{\delta}(\tilde{q}_{i-1}, x_i) = \widetilde{\delta(q_{i-1}, x_i)} = \tilde{q}_i$$

durchläuft $\tilde{M}$ dann die Zustände

$$\tilde{q}_0, \tilde{q}_1, \cdots, \tilde{q}_n.$$

Da aber q_n genau dann zu E gehört, wenn $\tilde{q}_n \in \tilde{E}$ ist, folgt $L(\tilde{M}) = L(M)$ (man beachte, dass $\tilde{q}_n$ entweder nur Endzustände oder nur Nicht-Endzustände enthält, vgl. Beobachtung 52).

Es bleibt zu zeigen, dass $\tilde{M}$ eine minimale Anzahl $\|\tilde{Z}\|$ von Zuständen hat. Dies ist sicher dann der Fall, wenn bereits M minimal ist. Es reicht also zu zeigen, dass die Anzahl $k = \|\tilde{Z}\| = \|\{L_q \mid q \in Z\}\|$ der Zustände von $\tilde{M}$ nicht von M, sondern nur von $L = L(M)$ abhängt. Für $x \in \Sigma^*$ sei

$$L_x = \{y \in \Sigma^* \mid xy \in L\}.$$

Dann gilt $\{L_x \mid x \in \Sigma^*\} \subseteq \{L_q \mid q \in Z\}$, da $L_x = L_{\hat{\delta}(q_0,x)}$ ist. Die umgekehrte Inklusion gilt ebenfalls, da nach Voraussetzung jeder Zustand $q \in Z$ über ein $x \in \Sigma^*$ erreichbar ist. Also hängt $k = \|\{L_q \mid q \in Z\}\| = \|\{L_x \mid x \in \Sigma^*\}\|$ nur von L ab. ∎

Für die algorithmische Konstruktion von $\tilde{M}$ aus M ist es notwendig herauszufinden, ob zwei Zustände p und q von M äquivalent sind oder nicht.

Bezeichne $A\Delta B = (A \setminus B) \cup (B \setminus A)$ die *symmetrische Differenz* von zwei Mengen A und B. Dann ist die Inäquivalenz $p \not\sim q$ zweier Zustände p und q gleichbedeutend mit $L_p \Delta L_q \neq \emptyset$. Wir nennen ein Wort $x \in L_p \Delta L_q$ einen *Unterscheider* zwischen p und q.

Beobachtung 52.

- *Endzustände $p \in E$ sind nicht mit Zuständen $q \in Z \setminus E$ äquivalent (da sie durch ε unterschieden werden).*
- *Wenn $\delta(p,a)$ und $\delta(q,a)$ inäquivalent sind, dann auch p und q (da jeder Unterscheider x von $\delta(p,a)$ und $\delta(q,a)$ einen Unterscheider ax von p und q liefert).*

Wenn also D nur Paare von inäquivalenten Zuständen enthält, dann trifft dies auch auf die Menge

$$D' = \{\{p,q\} \mid \exists a \in \Sigma : \{\delta(p,a), \delta(q,a)\} \in D\}$$

zu. Wir können somit ausgehend von der Menge

$$D_0 = \{\{p,q\} \mid p \in E, q \notin E\}$$

eine Folge von Mengen

$$D_0 \subseteq D_1 \subseteq \cdots \subseteq \{\{z,z'\} \subseteq Z \mid z \neq z'\}$$

mittels der Vorschrift

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

berechnen, indem wir zu D_i alle Paare $\{p,q\}$ hinzufügen, für die eines der Paare $\{\delta(p,a), \delta(q,a)\}$, $a \in \Sigma$, bereits zu D_i gehört. Da Z endlich ist, muss es ein j mit $D_{j+1} = D_j$ geben. In diesem Fall gilt (siehe Übungen):

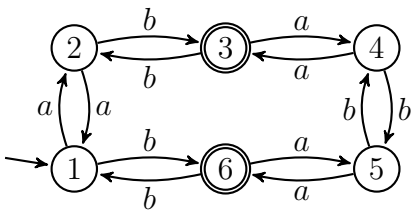
$$p \not\sim q \Leftrightarrow \{p,q\} \in D_j.$$

Folglich kann $\tilde{M}$ durch Verschmelzen aller Zustände p, q mit $\{p,q\} \notin D_j$ gebildet werden. Der folgende Algorithmus berechnet für einen beliebigen DFA M den zugehörigen Minimal-DFA $\tilde{M}$.

Algorithmus `min-DFA`(M)

```
1  Input: DFA M = (Z, Σ, δ, q_0, E)
2    entferne alle nicht erreichbaren Zustaende
3    D' := {{z, z'} | z ∈ E, z' ∉ E}
4    repeat
5      D := D'
6      D' := D ∪ {{p, q} | ∃a ∈ Σ : {δ(p, a), δ(q, a)} ∈ D}
7    until D' = D
8  Output: M̃ = (Z̃, Σ, δ̃, q̃_0, Ẽ), wobei für jeden Zustand
             z ∈ Z gilt: z̃ = {z} ∪ {z' ∈ Z | {z, z'} ∉ D}
```

Beispiel 53. *Betrachte den DFA M*



Dann enthält D_0 die Paare

$$\{1,3\}, \{1,6\}, \{2,3\}, \{2,6\}, \{3,4\}, \{3,5\}, \{4,6\}, \{5,6\}.$$

Die Paare in D_0 sind in der folgenden Matrix durch den Unterscheider

ε markiert.

2					
3	ε	ε			
4	a	a	ε		
5	a	a	ε		
6	ε	ε		ε	ε
	1	2	3	4	5

Wegen

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

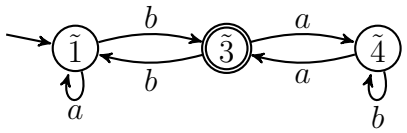
enthält D_1 zusätzlich die Paare $\{1,4\}$, $\{1,5\}$, $\{2,4\}$, $\{2,5\}$ (in obiger Matrix durch den Unterscheider a markiert). Da die verbliebenen Paare $\{1,2\}$, $\{3,6\}$, $\{4,5\}$ wegen

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

nicht zu D_1 hinzugefügt werden können, ist $D_2 = D_1$. Aus den unmarkierten Paaren $\{1,2\}$, $\{3,6\}$ und $\{4,5\}$ erhalten wir die Äquivalenzklassen

$$\tilde{1} = \{1,2\},\ \ \tilde{3} = \{3,6\}\ \ \textit{und}\ \ \tilde{4} = \{4,5\},$$

die auf folgenden Minimal-DFA $\tilde{M}$ führen:



◁

Es ist auch möglich, einen Minimalautomaten M_L direkt aus einer regulären Sprache L zu gewinnen (also ohne einen DFA M für L zu kennen). Da wegen

$$\widetilde{\hat{\delta}(q_0,x)} = \widetilde{\hat{\delta}(q_0,y)} \Leftrightarrow \hat{\delta}(q_0,x) \sim \hat{\delta}(q_0,y) \Leftrightarrow L_{\hat{\delta}(q_0,x)} = L_{\hat{\delta}(q_0,y)} \Leftrightarrow L_x = L_y$$

zwei Eingaben x und y den DFA $\tilde{M}$ genau dann in denselben Zustand $\widetilde{\hat{\delta}(q_0,x)} = \widetilde{\hat{\delta}(q_0,y)}$ überführen, wenn $L_x = L_y$ ist, können wir den von $\tilde{M}$ bei Eingabe x erreichten Zustand auch mit der Sprache L_x bezeichnen. Dies führt auf den zu $\tilde{M}$ isomorphen (also bis auf die Benennung der Zustände mit $\tilde{M}$ identischen) DFA $M_L = (Z_L, \Sigma, \delta_L, L_\varepsilon, E_L)$ mit

$$\begin{aligned} Z_L &= \{L_x \mid x \in \Sigma^*\}, \\ E_L &= \{L_x \mid x \in L\} \text{ und} \\ \delta_L(L_x,a) &= L_{xa}. \end{aligned}$$

Notwendig und hinreichend für die Existenz von M_L ist, dass die Menge $\{L_x \mid x \in \Sigma^*\}$ nur endlich viele verschiedene Sprachen enthält. L ist also genau dann regulär, wenn die durch

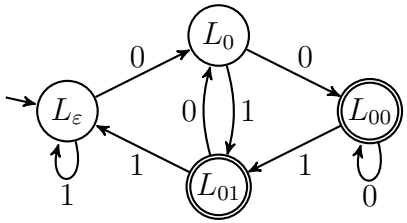
$$x\, R_L\, y \Leftrightarrow L_x = L_y$$

auf Σ^* definierte Äquivalenzrelation R_L endlichen Index hat.

Beispiel 54. *Für $L = \{x_1 \cdots x_n \in \{0,1\}^* \mid n \geq 2$ und $x_{n-1} = 0\}$ ist*

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

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 L isomorph sind.

Satz 55 (Myhill und Nerode)**.**
Für eine Sprache L bezeichne $index(R_L) = \|\{[x]_{R_L} \mid x \in \Sigma^*\}\|$ den Index der Äquivalenzrelation R_L.

1. $\mathsf{REG} = \{L \mid index(R_L) < \infty\}$.
2. *Für jede reguläre Sprache L gibt es bis auf Isomorphie genau einen Minimal-DFA. Dieser hat $index(R_L)$ Zustände.*

Beispiel 56. *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.* ◁

Die Zustände von M_L können anstelle von L_x auch mit den Äquivalenzklassen $[x]_{R_L}$ (bzw. mit geeigneten Repräsentanten) benannt werden. Der resultierende Minimal-DFA $M_{R_L} = (Z, \Sigma, \delta, [\varepsilon], E)$ mit

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

wird auch als **Äquivalenzklassenautomat** bezeichnet.

Die Konstruktion von M_{R_L} ist meist einfacher als die von M_L, da die Bestimmung der Sprachen L_x entfällt. Um die Überführungsfunktion

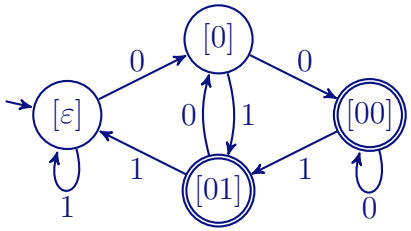
von M_{R_L} aufzustellen, reicht es, ausgehend von $r_1 = \varepsilon$ eine Folge $r_1, \ldots, r_k$ von paarweise bzgl. R_L inäquivalenten Wörtern zu bestimmen, so dass zu jedem Wort $r_i a$, $a \in \Sigma$, ein r_j mit $r_i a R_L r_j$ existiert. In diesem Fall ist $\delta([r_i], a) = [r_i a] = [r_j]$.

Beispiel 57. *Für die Sprache $L = \{x_1 \cdots x_n \in \{0,1\}^* \mid x_{n-1} = 0\}$ lässt sich M_{R_L} wie folgt konstruieren:*

1. *Wir beginnen mit $r_1 = \varepsilon$.*
2. *Da $r_1 0 = 0 \notin [\varepsilon]$ ist, wählen wir $r_2 = 0$ und setzen $\delta([\varepsilon], 0) = [0]$.*
3. *Da $r_1 1 = 1 \in [\varepsilon]$ ist, setzen wir $\delta([\varepsilon], 1) = [\varepsilon]$.*
4. *Da $r_2 0 = 00 \notin [\varepsilon] \cup [0]$ ist, ist $r_3 = 00$ und wir setzen $\delta([0], 0) = [00]$.*
5. *Da $r_2 1 = 01 \notin [\varepsilon] \cup [0] \cup [00]$ ist, wählen wir $r_4 = 01$ und setzen $\delta([0], 1) = [01]$.*
6. *Da die Wörter $r_3 0 = 000 \in [00]$, $r_3 1 = 001 \in [01]$, $r_4 0 = 010 \in [0]$ und $r_4 1 = 011 \in [\varepsilon]$ sind, setzen wir $\delta([00], 0) = [00]$, $\delta([00], 1) = [01]$, $\delta([01], 0) = [0]$ und $\delta([01], 1) = [\varepsilon]$.*

Wir erhalten also folgenden Minimal-DFA M_{R_L}:

r	ε	0	00	01
$[r0]$	$[0]$	$[00]$	$[00]$	$[0]$
$[r1]$	$[\varepsilon]$	$[01]$	$[01]$	$[\varepsilon]$



◁

Wir fassen nochmals die wichtigsten Ergebnisse zusammen.

Korollar 58. *Sei L eine Sprache. Dann sind folgende Aussagen äquivalent:*

- *L ist regulär,*

- *es gibt einen DFA M mit $L = L(M)$,*
- *es gibt einen NFA N mit $L = L(N)$,*
- *es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$,*
- *die Äquivalenzrelation R_L hat endlichen Index.*

Wir werden im nächsten Abschnitt noch eine weitere Charakterisierung von REG kennenlernen, nämlich durch reguläre Grammatiken.

2.6 Grammatiken

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

Beispiel 59. *Die Sprache RA aller regulären Ausdrücke über einem Alphabet $\Sigma = \{a_1, \ldots, a_k\}$ lässt sich aus dem Symbol R durch wiederholte Anwendung folgender Regeln erzeugen:*

$$\begin{array}{ll} R \to \emptyset, & R \to RR, \\ R \to \epsilon, & R \to (R|R), \\ R \to a_i,\ i = 1, \ldots, k, & R \to (R)^*. \end{array}$$

◁

Definition 60. *Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei*

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

Für $(u, v) \in P$ schreiben wir auch kurz $u \to_G v$ bzw. $u \to v$, wenn die benutzte Grammatik aus dem Kontext ersichtlich ist.

Definition 61. *Seien $\alpha, \beta \in (V \cup \Sigma)^*$.*

*a) Wir sagen, β **ist aus** α **in einem Schritt ableitbar** (kurz: $\alpha \Rightarrow_G \beta$), falls eine Regel $u \to_G v$ und Wörter $l, r \in (V \cup \Sigma)^*$ existieren mit*

$$\alpha = lur \text{ und } \beta = lvr.$$

Hierfür schreiben wir auch $l\underline{u}r \Rightarrow_G lvr$. (Man beachte, dass durch Unterstreichen von u in α sowohl die benutzte Regel als auch die Stelle in α, an der u durch v ersetzt wird, eindeutig erkennbar sind.)

*b) Eine Folge $\sigma = (l_0, u_0, r_0), \ldots, (l_m, u_m, r_m)$ von Tripeln (l_i, u_i, r_i) heißt **Ableitung von** β **aus** α, falls gilt:*

- *$l_0 u_0 r_0 = \alpha$, $l_m u_m r_m = \beta$ und*
- *$l_i \underline{u_i} r_i \Rightarrow l_{i+1} u_{i+1} r_{i+1}$ für $i = 0, \ldots, m-1$.*

*Die **Länge** von σ ist m und wir notieren σ auch in der Form*

$$l_0 \underline{u_0} r_0 \Rightarrow l_1 \underline{u_1} r_1 \Rightarrow \cdots \Rightarrow l_{m-1} \underline{u_{m-1}} r_{m-1} \Rightarrow l_m u_m r_m.$$

*c) Die durch G **erzeugte Sprache** ist*

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

*d) Ein Wort $\alpha \in (V \cup \Sigma)^*$ mit $S \Rightarrow_G^* \alpha$ heißt **Satzform** von G.*

Zur Erinnerung: Die Relation $\Rightarrow^*$ bezeichnet die reflexive, transitive Hülle der Relation $\Rightarrow$, d.h. $\alpha \Rightarrow^* \beta$ bedeutet, dass es ein $n \geq 0$ gibt mit $\alpha \Rightarrow^n \beta$. Hierzu sagen wir auch, β **ist aus** α **(in** n **Schritten) ableitbar**. Die Relation $\Rightarrow^n$ bezeichnet das n-fache Produkt der Relation $\Rightarrow$, d.h. es gilt $\alpha \Rightarrow^n \beta$, falls Wörter $\alpha_0, \ldots, \alpha_n$ existieren mit

- $\alpha_0 = \alpha$, $\alpha_n = \beta$ und
- $\alpha_i \Rightarrow \alpha_{i+1}$ für $i = 0, \ldots, n-1$.

Beispiel 62. *Wir betrachten nochmals die Grammatik* $G = (\{R\}, \Sigma \cup \{\emptyset, \epsilon, (,), ^*, |\}, P, R)$*, die die Menge der regulären Ausdrücke über dem Alphabet* Σ *erzeugt, wobei* P *die oben angegebenen Regeln enthält. Ist* $\Sigma = \{0, 1\}$*, so lässt sich der reguläre Ausdruck* $(01)^*(\epsilon|\emptyset)$ *beispielsweise wie folgt ableiten:*

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

◁

Man unterscheidet vier verschiedene Typen von Grammatiken.

Definition 63. *Sei* $G = (V, \Sigma, P, S)$ *eine Grammatik.*

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

ε-Sonderregel: In einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ kann auch die Regel $S \to \varepsilon$ benutzt werden. Aber nur, wenn das Startsymbol S nicht auf der rechten Seite einer Regel in P vorkommt.

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

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

(*context free languages*) und

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

(*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

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

Die Sprachklassen

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

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

Als nächstes zeigen wir, dass sich mit regulären Grammatiken gerade die regulären Sprachen erzeugen lassen. Hierbei erweist sich folgende Beobachtung als nützlich.

Lemma 64. *Zu jeder regulären Grammatik* $G = (V, \Sigma, P, S)$ *gibt es eine äquivalente reguläre Grammatik* G'*, die keine Produktionen der Form* $A \to a$ *hat.*

Beweis. Betrachte die Grammatik $G' = (V', \Sigma, P', S)$ mit

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

Es ist leicht zu sehen, dass G' die gleiche Sprache wie G erzeugt. ■

Satz 65. $\mathsf{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}$.

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

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

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

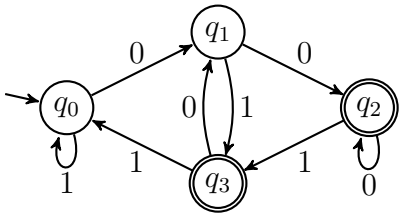
$$\begin{aligned} x \in L(M) &\Leftrightarrow \exists\, q_1, \ldots, q_{n-1} \in Z\, \exists\, q_n \in E : \\ &\quad \delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \ldots, n \\ &\Leftrightarrow \exists\, q_1, \ldots, q_n \in V : \\ &\quad q_{i-1} \to_G x_i q_i \text{ für } i = 1, \ldots, n \text{ und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

Für die entgegengesetzte Inklusion sei nun $G = (V, \Sigma, P, S)$ eine reguläre Grammatik, die keine Produktionen der Form $A \to a$ enthält. Dann können wir die gerade beschriebene Konstruktion einer Grammatik aus einem DFA „umdrehen“, um ausgehend von G einen NFA $M = (Z, \Sigma, \delta, \{S\}, E)$ mit

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

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

Beispiel 66. *Der DFA*



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

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

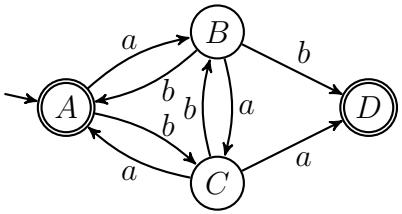
Umgekehrt führt die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ *mit*

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

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

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

auf den NFA



◁

2.7 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 67 (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* $i \geq 0$.

Falls eine Zahl l mit diesen Eigenschaften existiert, wird das kleinste solche l die ***Pumping-Zahl*** *von L genannt.*

Beweis. Sei $G = (V, \Sigma, P, S)$ eine reguläre Grammatik für L, die keine Regeln der Form $A \to a$ enthält, und sei

$$\underline{A_0} \Rightarrow x_1\underline{A_1} \Rightarrow x_1x_2\underline{A_2} \Rightarrow \cdots \Rightarrow x_1x_2\cdots x_n\underline{A_n} \Rightarrow x_1x_2\cdots x_n$$

eine beliebige Ableitung von $x = x_1 \cdots x_n \in L$ aus $A_0 = S$. Setzen wir $l = \|V\|$, so muss im Fall $|x| = n \geq l$ unter $A_0, \ldots, A_l$ eine Variable A mehrfach vorkommen, d.h. es ex. $0 \leq j < k \leq l$ mit $A_j = A_k = A$. Somit können wir die Ableitung von x wie folgt zerlegen:

$$A_0 \Rightarrow^j x_1 \cdots x_j A_j = uA \Rightarrow^{k-j} ux_{j+1} \cdots x_k A_k = uvA \Rightarrow^{n+1-k} uvw,$$

wobei $u = x_1 \cdots x_j$, $v = x_{j+1} \cdots x_k$ und $w = x_{k+1} \cdots x_n$ ist. Dann gilt $|v| = k - j \geq 1$ (d.h. $v \neq \varepsilon$), $k = |uv| \leq l$ und für $i \geq 0$ zeigt die Ableitung

$$A_0 \Rightarrow^j uA \Rightarrow^{(k-j)i} uv^iA \Rightarrow^{n+1-k} uv^iw,$$

dass $uv^iw \in L$ ist.

Das Pumping-Lemma lässt sich alternativ unter Benutzung eines DFA $M = (Z, \Sigma, \delta, q_0, E)$ für L beweisen. Ist l die Anzahl der Zustände von M und setzen wir 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:

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

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 $uv^iw \in L$ für $i \geq 0$, da wegen $\hat{\delta}(q, v) = q$

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

ist. ■

Beispiel 68. *Die Sprache*

$$L = \{x \in \{a, b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}$$

hat die Pumping-Zahl $l = 3$. Sei nämlich $x \in L$ beliebig mit $|x| \geq 3$. Dann lässt sich innerhalb des Präfixes von x der Länge drei ein nichtleeres Teilwort v finden, das gepumpt werden kann:

1. Fall: *x hat das Präfix ab (oder ba).*
Zerlege $x = uvw$ mit $u = \varepsilon$ und $v = ab$ (bzw. $v = ba$).

2. Fall: *x hat das Präfix aab (oder bba).*
Zerlege $x = uvw$ mit $u = a$ (bzw. $u = b$) und $v = ab$ (bzw. $v = ba$).

3. Fall: *x hat das Präfix aaa (oder bbb).*
Zerlege $x = uvw$ mit $u = \varepsilon$ und $v = aaa$ (bzw. $v = bbb$). ◁

Beispiel 69. *Eine endliche Sprache L hat die Pumping-Zahl*

$$l = \begin{cases} 0, & L = \emptyset, \\ \max\{|x| + 1 \mid x \in L\}, & \text{sonst.} \end{cases}$$

Tatsächlich lässt sich jedes Wort $x \in L$ der Länge $|x| \geq l$ „pumpen“ (da solche Wörter gar nicht existieren), weshalb die Pumping-Zahl höchstens l ist. Zudem gibt es im Fall $l > 0$ ein Wort $x \in L$ der Länge $|x| = l - 1$, das sich nicht „pumpen“ lässt, weshalb die Pumping-Zahl nicht kleiner als l sein kann. ◁

Wollen wir mit Hilfe des Pumping-Lemmas von einer Sprache L zeigen, dass sie nicht regulär ist, so 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 67 aufgeführten Eigenschaften verletzt ist.

Beispiel 70. *Die Sprache*

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

ist nicht regulär, da sich für jede Zahl $l \geq 0$ das Wort $x = a^l b^l$ der Länge $|x| = 2l \geq l$ in der Sprache L befindet, welches offensichtlich nicht in Teilwörter u, v, w mit $v \neq \varepsilon$ und $uv^2w \in L$ zerlegbar ist. ◁

Beispiel 71. *Die Sprache*

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

ist ebenfalls nicht regulär. Andernfalls müsste es nämlich eine Zahl l geben, so dass jede Quadratzahl $n^2 \geq l$ 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 $u + 2v + w = n^2 + v$ eine Quadratzahl sein, was wegen

$$n^2 < n^2 + v \leq n^2 + l < n^2 + 2l + 1 = (n + 1)^2$$

ausgeschlossen ist. ◁

Beispiel 72. *Auch die Sprache*

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

ist nicht regulär, da sich sonst jede Primzahl p einer bestimmten Mindestgröße l als Summe von natürlichen Zahlen $u + v + w$ darstellen ließe, so dass $v \geq 1$ und für alle $i \geq 0$ auch $u + iv + w = p + (i - 1)v$ prim ist. Dies ist jedoch für $i = p + 1$ wegen

$$p + (p + 1 - 1)v = p(1 + v)$$

nicht der Fall. ◁

Bemerkung 73. *Mit Hilfe des Pumping-Lemmas kann nicht für jede Sprache $L \notin \mathsf{REG}$ gezeigt werden, dass L nicht regulär ist, da seine Umkehrung falsch ist. So hat beispielsweise die Sprache*

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

die Pumping-Zahl 1 (d.h. jedes Wort $x \in L$ mit Ausnahme von ε kann „gepumpt“ werden). Dennoch ist L nicht regulär (siehe Übungen).

3 Kontextfreie Sprachen

Wie wir gesehen haben, ist die Sprache $L = \{a^n b^n \mid n \geq 0\}$ nicht regulär. Es ist aber leicht, eine kontextfreie Grammatik für L zu finden:

$$G = (\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S).$$

Damit ist klar, dass die Klasse der regulären Sprachen echt in der Klasse der kontextfreien Sprachen enthalten ist. Als nächstes wollen wir zeigen, dass die Klasse der kontextfreien Sprachen wiederum echt in der Klasse der kontextsensitiven Sprachen enthalten ist:

$$\mathsf{REG} \subsetneq \mathsf{CFL} \subsetneq \mathsf{CSL}.$$

Kontextfreie Grammatiken sind dadurch charakterisiert, dass sie nur Regeln der Form $A \to \alpha$ haben. Dies lässt die Verwendung von beliebigen ε-Regeln der Form $A \to \varepsilon$ zu. Eine kontextsensitive Grammatik darf dagegen höchstens die ε-Regel $S \to \varepsilon$ haben. Voraussetzung hierfür ist, dass S das Startsymbol ist und dieses nicht auf der rechten Seite einer Regel vorkommt. Daher sind nicht alle kontextfreien Grammatiken kontextsensitiv. Es lässt sich jedoch zu jeder kontextfreien Grammatik eine äquivalente kontextfreie Grammatik G' konstruieren, die auch kontextsensitiv ist. Hierzu zeigen wir zuerst, dass sich zu jeder kontextfreien Grammatik G, in der nicht das leere Wort ableitbar ist, eine äquivalente kontextfreie Grammatik G' ohne ε-Regeln konstruieren lässt.

Satz 74. *Zu jeder kontextfreien Grammatik G gibt es eine kontextfreie Grammatik G' ohne ε-Produktionen mit $L(G') = L(G) \setminus \{\varepsilon\}$.*

Beweis. Zuerst sammeln wir mit folgendem Algorithmus alle Variablen A, aus denen das leere Wort ableitbar ist. Diese werden auch als *ε-ableitbar* bezeichnet.

```
1  E' := {A ∈ V | A → ε}
2  repeat
3     E := E'
4     E' := E ∪ {A ∈ V | ∃ B_1, ..., B_k ∈ E : A → B_1 ··· B_k}
5  until E = E'
```

Nun konstruieren wir $G' = (V, \Sigma, P', S)$ wie folgt:

> Nehme zu P' alle Regeln $A \to \alpha'$ mit $\alpha' \neq \varepsilon$ hinzu, für die P eine Regel $A \to \alpha$ enthält, so dass α' aus α durch Entfernen von beliebig vielen Variablen $A \in E$ hervorgeht.

■

Beispiel 75. *Betrachte die Grammatik $G = (V, \Sigma, P, S)$ mit $V = \{S, T, U, X, Y, Z\}$, $\Sigma = \{a, b, c\}$ und den Regeln*

$$\begin{aligned} P: \quad & S \to aY, bX, Z; \quad Y \to bS, aYY; \quad T \to U; \\ & X \to aS, bXX; \quad Z \to \varepsilon, S, T, cZ; \quad U \to abc. \end{aligned}$$

Bei der Berechnung von $E = \{A \in V \mid A \Rightarrow^* \varepsilon\}$ ergeben sich der Reihe nach folgende Belegungen für die Mengenvariablen E und E':

E'	$\{Z\}$	$\{Z, S\}$
E	$\{Z, S\}$	$\{Z, S\}$

Um nun die Regelmenge P' zu bilden, entfernen wir aus P die einzige ε-Regel $Z \to \varepsilon$ und fügen die Regeln $X \to a$ (wegen $X \to aS$), $Y \to b$ (wegen $Y \to bS$) und $Z \to c$ (wegen $Z \to cZ$) hinzu:

$$\begin{aligned} P': \quad & S \to aY, bX, Z; \quad Y \to b, bS, aYY; \quad T \to U; \\ & X \to a, aS, bXX; \quad Z \to c, S, T, cZ; \quad U \to abc. \end{aligned}$$

◁

Als direkte Anwendung des obigen Satzes können wir die Inklusion der Klasse der Typ 2 Sprachen in der Klasse der Typ 1 Sprachen zeigen.

Korollar 76. $\mathsf{REG} \subsetneq \mathsf{CFL} \subseteq \mathsf{CSL} \subseteq \mathsf{RE}$.

Beweis. Die Inklusionen $\mathsf{REG} \subseteq \mathsf{CFL}$ und $\mathsf{CSL} \subseteq \mathsf{RE}$ sind klar. Wegen $\{a^n b^n | n \geq 0\} \in \mathsf{CFL} - \mathsf{REG}$ ist die Inklusion $\mathsf{REG} \subseteq \mathsf{CFL}$ auch echt. Also ist nur noch die Inklusion $\mathsf{CFL} \subseteq \mathsf{CSL}$ zu zeigen. Nach obigem Satz ex. zu $L \in \mathsf{CFL}$ eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ohne ε-Produktionen mit $L(G) = L \setminus \{\varepsilon\}$. Da G dann auch kontextsensitiv ist, folgt hieraus im Fall $\varepsilon \notin L$ unmittelbar $L(G) = L \in \mathsf{CSL}$. Im Fall $\varepsilon \in L$ erzeugt die kontextsensitive Grammatik

$$G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \to S, \varepsilon\}, S')$$

die Sprache $L(G') = L$, d.h. $L \in \mathsf{CSL}$. ■

Als nächstes zeigen wir folgende Abschlusseigenschaften der kontextfreien Sprachen.

Satz 77. *Die Klasse* CFL *ist abgeschlossen unter Vereinigung, Produkt und Sternhülle.*

Beweis. Seien $G_i = (V_i, \Sigma, P_i, S_i)$, $i = 1, 2$, kontextfreie Grammatiken für die Sprachen $L(G_i) = L_i$ mit $V_1 \cap V_2 = \emptyset$ und sei S eine neue Variable. Dann erzeugt die kontextfreie Grammatik

$$G_3 = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S_2\}, S)$$

die Vereinigung $L(G_3) = L_1 \cup L_2$. Die Grammatik

$$G_4 = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$$

erzeugt das Produkt $L(G_4) = L_1 L_2$ und die Sternhülle $(L_1)^*$ wird von der Grammatik

$$G_5 = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S, \varepsilon\}, S)$$

erzeugt. ■

Offen bleibt zunächst, ob die kontextfreien Sprachen auch unter Durchschnitt und Komplement abgeschlossen sind. Hierzu müssen wir für bestimmte Sprachen nachweisen, dass sie nicht kontextfrei sind. Dies gelingt mit einem Pumping-Lemma für kontextfreie Sprachen, für dessen Beweis wir Grammatiken in Chomsky-Normalform benötigen.

3.1 Chomsky-Normalform

Definition 78. *Eine Grammatik* (V, Σ, P, S) *ist in* ***Chomsky-Normalform*** *(**CNF**), falls* $P \subseteq V \times (V^2 \cup \Sigma)$ *ist, also alle Regeln die Form* $A \to BC$ *oder* $A \to a$ *haben.*

Um eine kontextfreie Grammatik in Chomsky-Normalform zu bringen, müssen wir neben den ε-Regeln $A \to \varepsilon$ auch sämtliche Variablenumbenennungen $A \to B$ loswerden.

Definition 79. *Regeln der Form* $A \to B$ *heißen* ***Variablenumbenennungen****.*

Satz 80. *Zu jeder kontextfreien Grammatik* G *ex. eine kontextfreie Grammatik* G' *ohne Variablenumbenennungen mit* $L(G') = L(G)$*.*

Beweis. Zuerst entfernen wir sukzessive alle Zyklen

$$A_1 \to A_2 \to \cdots \to A_k \to A_1,$$

indem wir diese Regeln aus P entfernen und alle übrigen Vorkommen der Variablen $A_2, \ldots, A_k$ durch A_1 ersetzen. Falls sich unter den entfernten Variablen $A_2, \ldots, A_k$ die Startvariable S befindet, sei A_1 die neue Startvariable.

Nun entfernen wir sukzessive die restlichen Variablenumbenennungen, indem wir

- eine Regel $A \to B$ wählen, so dass in P keine Variablenumbenennung $B \to C$ mit B auf der rechten Seite existiert,
- diese Regel $A \to B$ aus P entfernen und
- für jede Regel $B \to \alpha$ in P die Regel $A \to \alpha$ zu P hinzunehmen.

■

Beispiel 81. *Ausgehend von den Produktionen*

$$P\colon S \to aY, bX, Z; \quad Y \to b, bS, aYY; \quad T \to U;$$
$$X \to a, aS, bXX; \quad Z \to c, S, T, cZ; \quad U \to abc$$

entfernen wir den Zyklus $S \to Z \to S$, indem wir die Regeln $S \to Z$ und $Z \to S$ entfernen und dafür die Produktionen $S \to c, T, cS$ (wegen $Z \to c, T, cZ$) hinzunehmen:

$$S \to aY, bX, c, T, cS; \quad Y \to b, bS, aYY; \quad T \to U;$$
$$X \to a, aS, bXX; \quad U \to abc.$$

Nun entfernen wir die Regel $T \to U$ und fügen die Regel $T \to abc$ (wegen $U \to abc$) hinzu:

$$S \to aY, bX, c, T, cS; \quad Y \to b, bS, aYY; \quad T \to abc;$$
$$X \to a, aS, bXX; \quad U \to abc.$$

Als nächstes entfernen wir dann auch die Regel $S \to T$ und fügen die Regel $S \to abc$ (wegen $T \to abc$) hinzu:

$$S \to abc, aY, bX, c, cS; \quad Y \to b, bS, aYY; \quad T \to abc;$$
$$X \to a, aS, bXX; \quad U \to abc.$$

Da T und U nun nirgends mehr auf der rechten Seite vorkommen, können wir die Regeln $T \to abc$ und $U \to abc$ weglassen:

$$S \to abc, aY, bX, c, cS;\ Y \to b, bS, aYY;\ X \to a, aS, bXX.$$

◁

Nach diesen Vorarbeiten ist es nun leicht, eine gegebene kontextfreie Grammatik in Chomsky-Normalform umzuwandeln.

Satz 82. *Zu jeder kontextfreien Sprache $L \in \mathsf{CFL}$ gibt es eine CNF-Grammatik G' mit $L(G') = L \setminus \{\varepsilon\}$.*

Beweis. Aufgrund der beiden vorigen Sätze hat $L \setminus \{\varepsilon\}$ eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ ohne ε-Produktionen und ohne Variablenumbenennungen. Wir transformieren G wie folgt in eine CNF-Grammatik.

- Füge für jedes Terminalsymbol $a \in \Sigma$ eine neue Variable X_a zu V und eine neue Regel $X_a \to a$ zu P hinzu.
- Ersetze alle Vorkommen von a durch X_a, außer wenn a alleine auf der rechten Seite einer Regel steht.
- Ersetze jede Regel $A \to B_1 \cdots B_k$, $k \geq 3$, durch die $k-1$ Regeln

  $A \to B_1 A_1, A_1 \to B_2 A_2, \ \ldots\ , A_{k-3} \to B_{k-2} A_{k-2}, A_{k-2} \to B_{k-1} B_k,$

  wobei $A_1, \ldots, A_{k-2}$ neue Variablen sind. ■

Beispiel 83. *In der Produktionenmenge*

$$P\colon\ S \to abc, aY, bX, c, cS;\ \ X \to a, aS, bXX;\ \ Y \to b, bS, aYY$$

ersetzen wir die Terminalsymbole a, b und c durch die Variablen A, B und C (außer wenn sie alleine auf der rechten Seite einer Regel vorkommen) und fügen die Regeln $A \to a$, $B \to b$, $C \to c$ hinzu:

$$S \to c, ABC, AY, BX, CS;\ \ X \to a, AS, BXX;$$
$$Y \to b, BS, AYY;\ \ A \to a;\ \ B \to b;\ \ C \to c.$$

Ersetze nun die Regeln $S \to ABC$, $X \to BXX$ und $Y \to AYY$ durch die Regeln $S \to AS'$, $S' \to BC$, $X \to BX'$, $X' \to XX$ und $Y \to AY'$, $Y' \to YY$:

$$S \to c, AS', AY, BX, CS;\ \ S' \to BC;$$
$$X \to a, AS, BX';\ \ X' \to XX;\ \ Y \to b, BS, AY';\ \ Y' \to YY;$$
$$A \to a;\ B \to b;\ C \to c.$$

◁

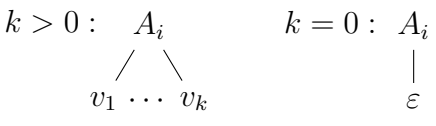
Für den Beweis des Pumping-Lemmas benötigen wir noch den Begriff des Syntaxbaums (auch **Ableitungsbaum** genannt, engl. *parse tree*).

Definition 84. *Wir ordnen einer Ableitung*

$$\underline{A_0} \Rightarrow l_1\underline{A_1}r_1 \Rightarrow \cdots \Rightarrow l_{m-1}\underline{A_{m-1}}r_{m-1} \Rightarrow \alpha_m.$$

den Syntaxbaum T_m zu, wobei die Bäume $T_0, \ldots, T_m$ induktiv wie folgt definiert sind:

- *T_0 besteht aus einem einzigen Knoten, der mit A_0 markiert ist.*
- *Wird im $(i+1)$-ten Ableitungsschritt die Regel $A_i \to v_1 \cdots v_k$ mit $v_j \in \Sigma \cup V$ für $j = 1, \ldots, k$ angewandt, so ensteht T_{i+1} aus T_i, indem wir das Blatt A_i in T_i durch folgenden Unterbaum ersetzen:*

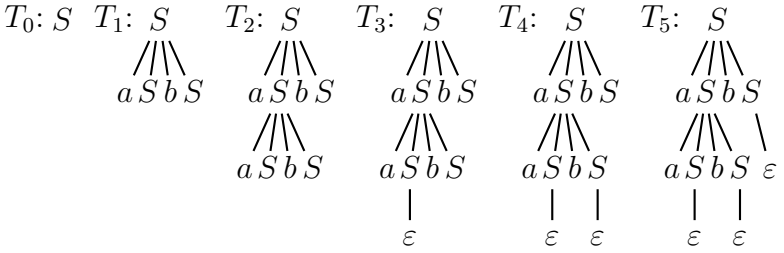


- *Hierbei stellen wir uns die Kanten von oben nach unten gerichtet und die Kinder $v_1 \cdots v_k$ von links nach rechts geordnet vor.*

Beispiel 85. *Betrachte die Grammatik $G = (\{S\}, \{a, b, c\}, \{S \to aSbS, \varepsilon\}, S)$ und die Ableitung*

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

Die zugehörigen Syntaxbäume sind dann



Die Satzform α_i ergibt sich aus T_i, indem wir die Blätter von T_i von links nach rechts zu einem Wort zusammensetzen. ◁

Es ist klar, dass Ableitungen, die sich nur in der Reihenfolge der Regelanwendungen unterscheiden, auf denselben Syntaxbaum führen. Dies bedeutet, dass aus einem Syntaxbaum die zugrunde liegende Ableitung nicht eindeutig rekonstruierbar ist. Dies ändert sich, wenn wir die Reihenfolge der Regelanwendungen festlegen.

Definition 86. *Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik.*

a) Eine Ableitung

$$\alpha_0 = l_0\underline{A_0}r_0 \Rightarrow l_1\underline{A_1}r_1 \Rightarrow \cdots \Rightarrow l_{m-1}\underline{A_{m-1}}r_{m-1} \Rightarrow \alpha_m.$$

*heißt **Linksableitung** von α (kurz $\alpha_0 \Rightarrow_L^* \alpha_m$), falls in jedem Ableitungsschritt die am weitesten links stehende Variable ersetzt wird, d.h. es gilt $l_i \in \Sigma^*$ für $i = 0, \ldots, m-1$.*

*b) **Rechtsableitungen** $\alpha_0 \Rightarrow_R^* \alpha_m$ sind analog definiert.*

*c) G heißt **mehrdeutig**, wenn es ein Wort $x \in L(G)$ gibt, das zwei verschiedene Linksableitungen $S \Rightarrow_L^* x$ hat.*

Es ist leicht zu sehen, dass für alle Wörter $x \in \Sigma^*$ folgende Äquivalenzen gelten:

$$x \in L(G) \;\Leftrightarrow\; S \Rightarrow^* x \;\Leftrightarrow\; S \Rightarrow_L^* x \;\Leftrightarrow\; S \Rightarrow_R^* x.$$

Beispiel 87. *Wir betrachten nochmals die Grammatik $G = (\{S\}, \{a, b, c\}, \{S \to aSbS, \varepsilon\}, S)$ aus dem letzten Beispiel. Es gibt insgesamt zehn Ableitungen, die auf den dort abgebildeten Syntaxbaum T_5 führen:*

$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSb\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSb\underline{S}bS \Rightarrow aaSbb\underline{S} \Rightarrow aa\underline{S}bb \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aab\underline{S}bS \Rightarrow aabb\underline{S} \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aabSb\underline{S} \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSbSb\underline{S} \Rightarrow aa\underline{S}bSb \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSbSb\underline{S} \Rightarrow aaSb\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb$$
$$\underline{S} \Rightarrow aSb\underline{S} \Rightarrow a\underline{S}b \Rightarrow aa\underline{S}bSb \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\underline{S} \Rightarrow aSb\underline{S} \Rightarrow a\underline{S}b \Rightarrow aaSb\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb.$$

Darunter sind genau eine Links- und genau eine Rechtsableitung, nämlich

$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aab\underline{S}bS \Rightarrow aabb\underline{S} \Rightarrow aabb$$

und

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

Die Grammatik G ist eindeutig. Dies liegt daran, dass in keiner Satzform von G die Variable S von einem a gefolgt wird. Daher muss jede Linksableitung eines Wortes $x \in L(G)$ die am weitesten links stehende Variable der aktuellen Satzform $\alpha S \beta$ genau dann nach $aSbS$ expandieren, falls das Präfix α in x von einem a gefolgt wird.

Dagegen ist die Grammatik $G' = (\{S\}, \{a, b, c\}, \{S \to aSbS, ab, \varepsilon\}, S)$ mehrdeutig, da das Wort $x = ab$ zwei verschiedene Linksableitungen hat:

$$\underline{S} \Rightarrow ab \text{ und } \underline{S} \Rightarrow a\underline{S}bS \Rightarrow ab\underline{S} \Rightarrow ab.$$

◁

Bemerkung 88.

- *Aus einem Syntaxbaum ist die zugehörige Linksableitung eindeutig rekonstruierbar. Daher führen unterschiedliche Linksableitungen auch auf unterschiedliche Syntaxbäume. Linksableitungen und Syntaxbäume entsprechen sich also eineindeutig. Ebenso Rechtsableitungen und Syntaxbäume.*
- *Ist T Syntaxbaum einer CNF-Grammatik, so hat jeder Knoten in T höchstens zwei Kinder (d.h. T ist ein Binärbaum).*

3.2 Das Pumping-Lemma für kontextfreie Sprachen

In diesem Abschnitt beweisen wir das Pumping-Lemma für kontextfreie Sprachen. Dabei nützen wir die Tatsache aus, dass die Syntaxbäume einer CNF-Grammatik Binärbäume sind.

Definition 89. *Die **Tiefe** eines Baumes mit Wurzel w ist die maximale Pfadlänge von w zu einem Blatt.*

Lemma 90. *Ein Binärbaum B der Tiefe k hat höchstens 2^k Blätter.*

Beweis. Wir führen den Beweis durch Induktion über k.

$k = 0$: Ein Baum der Tiefe 0 kann nur einen Knoten haben.

$k \rightsquigarrow k+1$: Sei B ein Binärbaum der Tiefe $k+1$. Dann hängen an B's Wurzel maximal zwei Teilbäume. Da deren Tiefe $\leq k$ ist, haben sie nach IV höchstens 2^k Blätter. Also hat $B \leq 2^{k+1}$ Blätter. ■

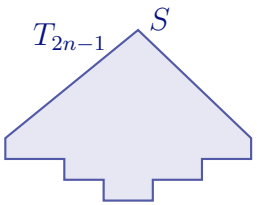
Korollar 91. *Ein Binärbaum B mit mehr als 2^{k-1} Blättern hat mindestens Tiefe k.*

Beweis. Würde B mehr als 2^{k-1} Blätter und eine Tiefe $\leq k-1$ besitzen, so würde dies im Widerspruch zu Lemma 90 stehen. ■

Satz 92 (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 $i \geq 0$.*

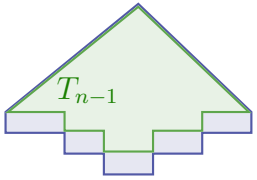
Beweis. Sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik für $L \setminus \{\varepsilon\}$. Dann existiert in G für jedes Wort $z = z_1 \cdots z_n \in L$ mit $n \geq 1$, eine Ableitung

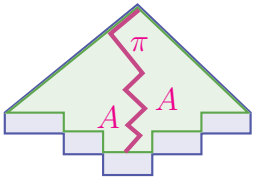


$$S = \alpha_0 \Rightarrow \alpha_1 \cdots \Rightarrow \alpha_m = z.$$

Da G in CNF ist, werden hierbei $n-1$ Regeln der Form $A \to BC$ und n Regeln der Form $A \to a$ angewandt, d.h. $m = 2n-1$ und z hat den Syntaxbaum T_{2n-1}.

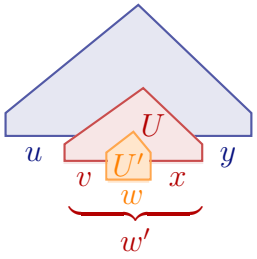
Wir können annehmen, dass zuerst alle Regeln der Form $A \to BC$ und danach die Regeln der Form $A \to a$ zur Anwendung kommen. Dann besteht die Satzform α_{n-1} aus n Variablen und der Syntaxbaum T_{n-1} hat ebenfalls n Blätter.



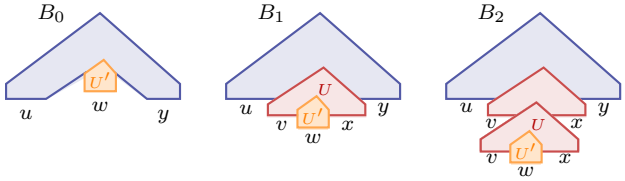


Setzen wir $l = 2^k$, wobei $k = \|V\|$ ist, so hat T_{n-1} im Fall $n \geq l$ mindestens $l = 2^k > 2^{k-1}$ Blätter und daher mindestens die Tiefe k. Sei π ein von der Wurzel ausgehender Pfad maximaler Länge in T_{n-1}. Dann hat π die Länge $\geq k$ und unter den letzten $k+1$ Knoten von π müssen zwei mit derselben Variablen A markiert sein.

Seien U und U' die von diesen Knoten ausgehenden Unterbäume des vollständigen Syntaxbaums T_{2n-1}. Nun zerlegen wir z wie folgt. w' ist das Teilwort von $z = uw'y$, das von U erzeugt wird und w ist das Teilwort von $w' = vwx$, das von U' erzeugt wird. Jetzt bleibt nur noch zu zeigen, dass diese Zerlegung die geforderten 3 Eigenschaften erfüllt.



- Da U mehr Blätter hat als U', ist $vx \neq \varepsilon$ (Bedingung 1).
- Da der Baum $U^* = U \cap T_{n-1}$ die Tiefe $\leq k$ hat (andernfalls wäre π nicht maximal), hat U^* höchstens $2^k = l$ Blätter. Da U^* genau $|vwx|$ Blätter hat, folgt $|vwx| \leq l$ (Bedingung 2).
- Für den Nachweis von Bedingung 3 lassen sich schließlich Syntaxbäume B^i für die Wörter uv^iwx^iy, $i \geq 0$, wie folgt konstruieren:



B^0 entsteht also aus $B^1 = T_{2n-1}$, indem wir U durch U' ersetzen, und B^{i+1} entsteht aus B^i, indem wir U' durch U ersetzen. ■

Beispiel 93. *Betrachte die Sprache $L = \{a^nb^n | n \geq 0\}$. Dann lässt sich jedes Wort $z = a^nb^n$ mit $|z| \geq 2$ pumpen: Zerlege $z = uvwxy$ mit $u = a^{n-1}$, $v = a$, $w = \varepsilon$, $x = b$ und $y = b^{n-1}$.* ◁

Beispiel 94. *Die Sprache $\{a^nb^nc^n \mid n \geq 0\}$ ist nicht kontextfrei. Für eine vorgegebene Zahl $l \geq 0$ hat nämlich $z = a^lb^lc^l$ die Länge $|z| = 3l \geq l$. Dieses Wort lässt sich aber nicht pumpen, da für jede Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$ und $|vwx| \leq l$ das Wort $z' = uv^2wx^2y$ nicht zu L gehört:*

- *Wegen $vx \neq \varepsilon$ ist $|z| < |z'|$.*
- *Wegen $|vwx| \leq l$ kann in vx nicht jedes der drei Zeichen a, b, c vorkommen.*
- *Kommt aber in vx beispielsweise kein a vor, so ist*

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

also kann z' nicht zu L gehören. ◁

Satz 95. *Die Klasse* CFL *ist nicht abgeschlossen unter Durchschnitt und Komplement.*

Beweis. Die beiden Sprachen

$$L_1 = \{a^nb^mc^m \mid n, m \geq 0\} \quad \text{und} \quad L_2 = \{a^nb^nc^m \mid n, m \geq 0\}$$

sind kontextfrei. Nicht jedoch $L_1 \cap L_2 = \{a^nb^nc^n \mid n \geq 0\}$. Also ist CFL nicht unter Durchschnitt abgeschlossen.

Da CFL zwar unter Vereinigung aber nicht unter Schnitt abgeschlossen ist, kann CFL wegen de Morgan nicht unter Komplementbildung abgeschlossen sein. ■

3.3 Der CYK-Algorithmus

In diesem Abschnitt stellen wir einen effizienten Algorithmus zur Lösung des Wortproblems für kontextfreie Grammatiken vor, das wie folgt definiert ist.

Wortproblem für kontextfreie Grammatiken:

Gegeben: Eine kontextfreie Grammatik G und ein Wort x.

Gefragt: Ist $x \in L(G)$?

Wir lösen das Wortproblem, indem wir G zunächst in Chomsky-Normalform bringen und dann den nach seinen Autoren Cocke, Younger und Kasami benannten CYK-Algorithmus anwenden, welcher auf dem Prinzip der Dynamischen Programmierung beruht.

Satz 96. *Das Wortproblem für kontextfreie Grammatiken ist effizient entscheidbar.*

Beweis. Seien eine Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $x = x_1 \cdots x_n$ gegeben. Falls $x = \varepsilon$ ist, können wir effizient prüfen, ob $S \Rightarrow^* \varepsilon$ gilt. Andernfalls transformieren wir G in eine CNF-Grammatik G' für die Sprache $L(G) \setminus \{\varepsilon\}$. Chomsky-Normalform. Es lässt sich leicht verifizieren, dass die nötigen Umformungsschritte effizient ausführbar sind. Nun setzen wir den CYK-Algorithmus auf das Paar (G', x) an, der die Zugehörigkeit von x zu $L(G')$ wie folgt entscheidet.
Bestimme für $l = 1, \ldots, n$ und $k = 1, \ldots, n - l + 1$ die Menge

$$V_{l,k}(x) = \{A \in V \mid A \Rightarrow^* x_k \cdots x_{k+l-1}\}$$

aller Variablen, aus denen das mit x_k beginnende Teilwort $x_k \cdots x_{k+l-1}$ von x der Länge l ableitbar ist. Dann gilt offensichtlich $x \in L(G') \Leftrightarrow S \in V_{n,1}(x)$.
Für $l = 1$ ist

$$V_{1,k}(x) = \{A \in V \mid A \to x_k\}$$

und für $l = 2, \cdots, n$ ist

$$V_{l,k}(x) = \{A \in V \mid \exists l' < l \, \exists B \in V_{l',k}(x) \, \exists C \in V_{l-l',k+l'}(x) \colon A \to BC\}.$$

Eine Variable A gehört also genau dann zu $V_{l,k}(x)$, $l \geq 2$, falls eine Zahl $l' \in \{1, \ldots, l-1\}$ und eine Regel $A \to BC$ existieren, so dass $B \in V_{l',k}(x)$ und $C \in V_{l-l',k+l'}(x)$ sind.
Da der Zeitaufwand für die Berechnung der Menge $V_{l,k}(x)$ durch $O(l|G'|)$ beschränkt ist, und insgesamt $n(n+1)/2$ solche Mengen zu bestimmen sind, lässt sich die Zeitkomplexität durch $O(n^3|G'|)$ abschätzen ■

Algorithmus `CYK`(G, x)

```
1  Input: CNF-Grammatik G = (V, Σ, P, S) und ein Wort
       x = x_1 ··· x_n
2  for k := 1 to n do
3    V_{1,k} := {A ∈ V | A → x_k ∈ P}
4  for l := 2 to n do
5    for k := 1 to n − l + 1 do
6      V_{l,k} := ∅
7      for l' := 1 to l − 1 do
8        for all A → BC ∈ P do
9          if B ∈ V_{l',k} and C ∈ V_{l−l',k+l'} then
10           V_{l,k} := V_{l,k} ∪ {A}
11 if S ∈ V_{n,1} then accept else reject
```

Der CYK-Algorithmus lässt sich leicht dahingehend modifizieren, dass er im Fall $x \in L(G)$ auch einen Syntaxbaum T von x ausgibt. Hierzu genügt es, zu jeder Variablen A in $V_{l,k}$ den Wert von l' und die Regel $A \to BC$ zu speichern, die zur Aufnahme von A in $V_{l,k}$ geführt haben. Im Fall $S \in V_{n,1}(x)$ lässt sich dann mithilfe dieser Information leicht ein Syntaxbaum T von x konstruieren.

Beispiel 97. *Betrachte die CNF-Grammatik mit den Produktionen*

$S \to AS', AY, BX, CS, c$; $S' \to BC$; $X \to AS, BX', a$; $X' \to XX$; $Y \to BS, AY', b$; $Y' \to YY$; $A \to a$; $B \to b$; $C \to c$.

Dann erhalten wir für das Wort $x = abb$ folgende Mengen $V_{l,k}$:

x_k:	a	b	b
l: 1	$\{X, A\}$	$\{Y, B\}$	$\{Y, B\}$
2	$\{S\}$	$\{Y'\}$	
3	$\{Y\}$		

Wegen $S \notin V_{3,1}(abb)$ ist $x \notin L(G)$. Dagegen gehört das Wort $y = aababb$ wegen $S \in V_{6,1}(aababb)$ zu $L(G)$:

a	a	b	a	b	b
$\{X, A\}$	$\{X, A\}$	$\{Y, B\}$	$\{X, A\}$	$\{Y, B\}$	$\{Y, B\}$
$\{X'\}$	$\{S\}$	$\{S\}$	$\{S\}$	$\{Y'\}$	
$\{X\}$	$\{X\}$	$\{Y\}$	$\{Y\}$		
$\{X'\}$	$\{S\}$	$\{Y'\}$			
$\{X\}$	$\{Y\}$				
$\{S\}$					

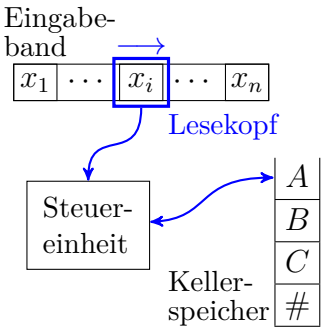
◁

3.4 Kellerautomaten

In diesem Abschnitt befassen wir uns mit der Frage, wie sich das Maschinenmodell des DFA erweitern lässt, um die Sprache $L = \{a^n b^n \mid n \geq 0\}$ und alle anderen kontextfreien Sprachen erkennen zu können. Dass ein DFA die Sprache $L = \{a^n b^n \mid n \geq 0\}$ nicht erkennen kann, liegt an seinem beschränkten Speichervermögen, das zwar von L aber nicht von der Eingabe abhängen darf.

Um L erkennen zu können, reicht bereits ein so genannter Kellerspeicher (Stapel, engl. *stack, pushdown memory*) aus. Dieser erlaubt nur den Zugriff auf die höchste belegte Speicheradresse. Ein Kellerautomat

- verfügt über einen Kellerspeicher,
- kann ε-Übergänge machen,
- liest in jedem Schritt das aktuelle Eingabezeichen und das oberste Kellersymbol,
- kann das oberste Kellersymbol entfernen (durch eine **pop-Operation**) und
- danach beliebig viele Symbole einkellern (mittels **push-Operationen**).



Für eine Menge M bezeichne $\mathcal{P}_e(M)$ die Menge aller endlichen Teilmengen von M, d.h.

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

Definition 98. *Ein **Kellerautomat** (kurz: PDA; pushdown automaton) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ beschrieben, wobei*

- *$Z \neq \emptyset$ eine endliche Menge von **Zuständen**,*
- *Σ das **Eingabealphabet**,*
- *Γ das **Kelleralphabet**,*
- *$\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ die **Überführungsfunktion**,*
- *$q_0 \in Z$ der **Startzustand** 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,

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

Hierfür sagen wir auch, M führt die **Anweisung** $quA \to pB_1 \cdots B_k$ aus. 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 (A_1 steht oben).

Eine Anweisung $quA_1 \to pB_1 \cdots B_k$ (mit $u \in \{\varepsilon, x_i\}$) überführt die Konfiguration K in die **Folgekonfiguration**

$$K' = (p, x_j \cdots x_n, B_1 \cdots B_k A_2 \cdots A_l) \text{ mit } j = i + |u|.$$

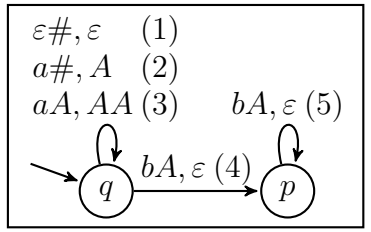
Hierfür schreiben wir auch kurz $K \vdash K'$. Eine *Rechnung* von M bei Eingabe x ist eine Folge von Konfigurationen $K_0, K_1, K_2 \ldots$ mit $K_0 = (q_0, x, \#)$ und $K_0 \vdash K_1 \vdash K_2 \cdots$. K_0 heißt **Startkonfiguration** von M bei Eingabe x. Die reflexive, transitive Hülle von $\vdash$ bezeichnen wir wie üblich mit $\vdash^*$. Die von M **akzeptierte** oder **erkannte Sprache** ist

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

Ein Wort x wird also genau dann von M akzeptiert, wenn es eine Rechnung gibt, bei der M das gesamte Eingabewort bis zum Ende liest und den Keller leert. Man beachte, dass bei leerem Keller kein weiterer Übergang mehr möglich ist.

Beispiel 99. *Sei $M = (Z, \Sigma, \Gamma, \delta, q, \#)$ ein PDA mit $Z = \{q, p\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$ und den Anweisungen*

$$\begin{array}{llll} \delta: & q\varepsilon\# \to q & (1) & qa\# \to qA \;\; (2) \\ & qaA \to qAA & (3) & qbA \to p \;\; (4) \\ & pbA \to p & (5) & \end{array}$$



Dann akzeptiert M die Eingabe aabb:

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

Allgemeiner akzeptiert M das Wort $x = a^n b^n$ mit folgender Rechnung:

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

$n \geq 1$: $(q, a^n b^n, \#) \underset{(2)}{\vdash} (q, a^{n-1} b^n, A) \overset{n-1}{\underset{(3)}{\vdash}} (q, b^n, A^n) \underset{(4)}{\vdash} (p, b^{n-1}, A^{n-1}) \overset{n-1}{\underset{(5)}{\vdash}} (p, \varepsilon, \varepsilon)$.

Dies zeigt $\{a^n b^n \mid n \geq 0\} \subseteq L(M)$. Als nächstes zeigen wir, dass jede von M akzeptierte Eingabe $x = x_1 \ldots x_n$ die Form $x = a^m b^m$ hat.

Ausgehend von der Startkonfiguration $(q, x, \#)$ sind nur die Anweisungen (1) oder (2) möglich. Falls M zuerst Anweisung (1) ausführt, wird der Keller geleert. Daher kann M in diesem Fall nur das leere Wort $x = \varepsilon = a^0 b^0$ akzeptieren.

Falls M mit Anweisung (2) beginnt, muss M später mittels Anweisung (4) in den Zustand p gelangen, da sonst der Keller nicht geleert wird. Dies geschieht, sobald M nach Lesen von $m \geq 1$ a's das erste b liest:

$$(q, x_1 \ldots x_n, \#) \underset{(2)}{\vdash} (q, x_2 \ldots x_n, A) \overset{m-1}{\underset{(3)}{\vdash}} (q, x_{m+1} \cdots x_n, A^m) \underset{(4)}{\vdash} (p, x_{m+2} \cdots x_n, A^{m-1})$$

mit $x_1 = x_2 = \cdots = x_m = a$ und $x_{m+1} = b$. Um den Keller leeren zu können, muss M nun noch genau $m - 1$ b's lesen, weshalb x auch in diesem Fall die Form $a^m b^m$ haben muss. ◁

Als nächstes zeigen wir, dass PDAs genau die kontextfreien Sprachen erkennen.

Satz 100. $\mathsf{CFL} = \{L(M) \mid M \textit{ ist ein PDA}\}$.

Beweis. Wir zeigen zuerst die Inklusion von links nach rechts.
Idee: Konstruiere zu einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ einen PDA $M = (\{q\}, \Sigma, \Gamma, \delta, q_0, S)$ mit $\Gamma = V \cup \Sigma$, so dass gilt:

$$S \Rightarrow_L^* x_1 \cdots x_n \text{ gdw. } (q, x_1 \cdots x_n, S) \vdash^* (q, \varepsilon, \varepsilon).$$

Hierzu fügen wir folgende Anweisungen zu δ hinzu:

$$\begin{array}{ll} \text{Für jede Regel } A \to_G \alpha\text{:} & q\varepsilon A \to q\alpha. \\ \text{Für jedes Zeichen } a \in \Sigma\text{:} & qaa \to q\varepsilon. \end{array}$$

M berechnet also nichtdeterministisch eine Linksableitung für die Eingabe x. Da M hierbei den Syntaxbaum von oben nach unten aufbaut, wird M als *Top-Down Parser* bezeichnet. Nun ist leicht zu sehen, dass sogar folgende Äquivalenz gilt:

$$S \Rightarrow_L^l x_1 \cdots x_n \text{ gdw. } (q, x_1 \cdots x_n, S) \vdash^{l+n} (q, \varepsilon, \varepsilon).$$

Daher folgt

$$x \in L(G) \Leftrightarrow S \Rightarrow_L^* x \Leftrightarrow (q, x, S) \vdash^* (q, \varepsilon, \varepsilon) \Leftrightarrow x \in L(M).$$

Als nächstes zeigen wir die Inklusion von rechts nach links.
Idee: Konstruiere zu einem PDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Variablen $X_{pAp'}$, $A \in \Gamma$, $p, p' \in Z$, so dass folgende Äquivalenz gilt:

$$(p, x, A) \vdash^* (p', \varepsilon, \varepsilon) \text{ gdw. } X_{pAp'} \Rightarrow^* x. \qquad (*)$$

Ein Wort x soll also genau dann in G aus $X_{pAp'}$ ableitbar sein, wenn M ausgehend vom Zustand p bei Lesen von x in den Zustand p' gelangen kann und dabei das Zeichen A aus dem Keller entfernt. Um dies zu erreichen, fügen wir für jede Anweisung $puA \to p_0 A_1 \cdots A_k$, $k \geq 0$, die folgenden $\|Z\|^k$ Regeln zu P hinzu:

$$\text{Für jede Zustandsfolge } p_1, \ldots, p_k\text{: } X_{pAp_k} \to u X_{p_0 A_1 p_1} \cdots X_{p_{k-1} A_k p_k}.$$

Um damit alle Wörter $x \in L(M)$ aus S ableiten zu können, benötigen wir jetzt nur noch für jeden Zustand $p \in Z$ die Regel $S \to X_{q_0 \# p}$. Die Variablenmenge von G ist also

$$V = \{S\} \cup \{X_{pAp'} \mid p, p' \in Z, A \in \Gamma\}$$

und P enthält neben den Regeln $S \to X_{q_0 \# p}$, $p \in Z$, für jede Anweisung $puA \to p_0 A_1 \cdots A_k$, $k \geq 0$, von M und jede Zustandsfolge $p_1, \ldots, p_k$ die Regel $X_{pAp_k} \to u X_{p_0 A_1 p_1} \cdots X_{p_{k-1} A_k p_k}$.
Unter der Voraussetzung, dass die Äquivalenz $(*)$ gilt, lässt sich nun leicht die Korrektheit von G zeigen. Es gilt

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

Wir müssen also nur noch die Gültigkeit von $(*)$ zeigen. Hierzu zeigen wir durch Induktion über m für alle $p, p' \in Z$, $A \in \Gamma$ und $x \in \Sigma^*$ folgende stärkere Behauptung:

$$(p, x, A) \vdash^m (p', \varepsilon, \varepsilon) \text{ gdw. } X_{pAp'} \Rightarrow^m x. \qquad (**)$$

$m = 0$**:** Da sowohl $(p, x, A) \vdash^0 (p', \varepsilon, \varepsilon)$ als auch $X_{pAp'} \Rightarrow^0 x$ falsch sind, ist die Äquivalenz $(**)$ für $m = 0$ erfüllt.

$m \rightsquigarrow m+1$**:** Wir zeigen zuerst die Implikation von links nach rechts. Für eine gegebene Rechnung

$$(p, x, A) \vdash (p_0, x', A_1 \cdots A_k) \vdash^m (p', \varepsilon, \varepsilon)$$

der Länge $m + 1$ sei $puA \to p_0 A_1 \cdots A_k$, $k \geq 0$, die im ersten Rechenschritt ausgeführte Anweisung (d.h. $x = ux'$). Zudem

sei p_i für $i = 1, \ldots, k$ der Zustand, in den M mit Kellerinhalt $A_{i+1} \cdots A_k$ gelangt (d.h. $p_k = p'$). Dann enthält P die Regel $X_{pAp_k} \to uX_{p_0A_1p_1} \cdots X_{p_{k-1}A_kp_k}$. Weiter sei u_i für $i = 1, \ldots, k$ das Teilwort von x', das M zwischen den Besuchen von p_{i-1} und p_i liest.

Dann gibt es Zahlen $m_i \geq 1$ mit $m_1 + \cdots + m_k = m$ und

$$(p_{i-1}, u_i, A_i) \vdash^{m_i} (p_i, \varepsilon, \varepsilon)$$

für $i = 1, \ldots, k$. Nach IV gibt es daher Ableitungen

$$X_{p_{i-1}A_ip_i} \Rightarrow^{m_i} u_i,\ i = 1, \ldots, k,$$

die wir zu der gesuchten Ableitung zusammensetzen können:

$$\begin{array}{lll} X_{pAp_k} & \Rightarrow & uX_{p_0A_1p_1} \cdots X_{p_{k-2}A_{k-1}p_{k-1}}X_{p_{k-1}A_kp_k} \\ & \Rightarrow^{m_1} & uu_1X_{p_1A_2p_2} \cdots X_{p_{k-2}A_{k-1}p_{k-1}}X_{p_{k-1}A_kp_k} \\ & \vdots & \\ & \Rightarrow^{m_{k-1}} & uu_1 \cdots u_{k-1}X_{p_{k-1}A_kp_k} \\ & \Rightarrow^{m_k} & uu_1 \cdots u_k = x. \end{array}$$

Zuletzt zeigen wir den Induktionsschritt für die Implikation von rechts nach links von $(**)$. Gelte also umgekehrt $X_{pAp'} \Rightarrow^{m+1} x$ und sei α die im ersten Schritt abgeleitete Satzform, d.h.

$$X_{pAp'} \Rightarrow \alpha \Rightarrow^m x.$$

Wegen $X_{pAp'} \to_G \alpha$ gibt es eine Anweisung $puA \to p_0A_1 \cdots A_k$, $k \geq 0$, und Zustände $p_1, \ldots, p_k \in Z$ mit

$$\alpha = u\, X_{p_0A_1p_1} \cdots X_{p_{k-1}A_kp_k},$$

wobei $p_k = p'$ ist. Wegen $\alpha \Rightarrow^m x$ ex. eine Zerlegung $x = uu_1 \cdots u_k$ und Zahlen $m_i \geq 1$ mit $m_1 + \cdots + m_k = m$ und

$$X_{p_{i-1}A_ip_i} \Rightarrow^{m_i} u_i\ (i = 1, \ldots, k).$$

Nach IV gibt es somit Rechnungen

$$(p_{i-1}, u_i, A_i) \vdash^{m_i} (p_i, \varepsilon, \varepsilon),\ i = 1, \ldots, k,$$

aus denen sich die gesuchte Rechnung der Länge $m + 1$ zusammensetzen lässt:

$$\begin{array}{rll} (p, uu_1 \cdots u_k, A) \vdash & (p_0, u_1 \cdots u_k, A_1 \cdots A_k) \\ \vdash^{m_1} & (p_1, u_2 \cdots u_k, A_2 \cdots A_k) \\ \vdots & & \\ \vdash^{m_{k-1}} & (p_{k-1}, u_k, A_k) \\ \vdash^{m_k} & (p_k, \varepsilon, \varepsilon). \end{array}$$

■

Beispiel 101. *Sei $G = (\{S\}, \{a, b\}, P, S)$ mit*

$$P\colon\ S \to aSbS,\ (1)\quad S \to a.\ (2)$$

Der zugehörige PDA besitzt dann die Anweisungen

$$\begin{array}{lllll} \delta\colon & qaa \to q\varepsilon, & (0) & qbb \to q\varepsilon, & (0') \\ & q\varepsilon S \to qaSbS, & (1') & q\varepsilon S \to qa. & (2') \end{array}$$

Der Linksableitung

$$\underline{S} \underset{(1)}{\Rightarrow} a\underline{S}bS \underset{(2)}{\Rightarrow} aab\underline{S} \underset{(2)}{\Rightarrow} aaba$$

in G entspricht beispielsweise die akzeptierende Rechnung

$$\begin{array}{l} (q, aaba, S) \underset{(1')}{\vdash} (q, aaba, aSbS) \underset{(0)}{\vdash} (q, aba, SbS) \\ \qquad \underset{(2')}{\vdash} (q, aba, abS) \underset{(0)}{\vdash} (q, ba, bS) \\ \qquad \underset{(0')}{\vdash} (q, a, S) \underset{(2')}{\vdash} (q, a, a) \underset{(0)}{\vdash} (q, \varepsilon, \varepsilon) \end{array}$$

von M und umgekehrt. ◁

Beispiel 102. *Sei M der PDA $(\{p,q\},\{a,b\},\{A,\#\},\delta,p,\#)$ mit*

$$\begin{aligned}\delta: p\varepsilon\# &\to q\varepsilon, \quad (1) \qquad paA \to pAA, \quad (3) \qquad qbA \to q\varepsilon. \quad (5)\\ pa\# &\to pA, \quad (2) \qquad pbA \to q\varepsilon, \quad (4)\end{aligned}$$

Dann erhalten wir die Grammatik $G=(V,\Sigma,P,S)$ mit der Variablenmenge

$$V=\{S, X_{p\#p}, X_{p\#q}, X_{q\#p}, X_{q\#q}, X_{pAp}, X_{pAq}, X_{qAp}, X_{qAq}\}.$$

Die Regelmenge P enthält neben den beiden Startregeln

$$S \to X_{p\#p}, X_{p\#q} \quad (0,0')$$

die folgenden Produktionen:

Anweisung $puA \to p_0A_1\cdots A_k$		k	$p_1,\ldots,p_k$	*zugehörige Regel* $X_{pAp_k} \to uX_{p_0A_1p_1}\cdots X_{p_{k-1}A_kp_k}$	
$p\varepsilon\# \to q\varepsilon$	(1)	0	-	$X_{p\#q} \to \varepsilon$	$(1')$
$pa\# \to pA$	(2)	1	p	$X_{p\#p} \to aX_{pAp}$	$(2')$
			q	$X_{p\#q} \to aX_{pAq}$	$(2'')$
$paA \to pAA$	(3)	2	p,p	$X_{pAp} \to aX_{pAp}X_{pAp}$	$(3')$
			p,q	$X_{pAq} \to aX_{pAp}X_{pAq}$	$(3'')$
			q,p	$X_{pAp} \to aX_{pAq}X_{qAp}$	$(3''')$
			q,q	$X_{pAq} \to aX_{pAq}X_{qAq}$	$(3'''')$
$pbA \to q\varepsilon$	(4)	0	-	$X_{pAq} \to b$	$(4')$
$qbA \to q\varepsilon$	(5)	0	-	$X_{qAq} \to b$	$(5')$

Der akzeptierenden Rechnung

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

von M entspricht dann die Ableitung

$$S \underset{(0')}{\Rightarrow} X_{p\#q} \underset{(2'')}{\Rightarrow} aX_{pAq} \underset{(3'''')}{\Rightarrow} aaX_{pAq}X_{qAq} \underset{(4')}{\Rightarrow} aabX_{qAq} \underset{(5')}{\Rightarrow} aabb$$

in G und umgekehrt. ◁

3.5 Deterministisch kontextfreie Sprachen

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

Definition 103. *Ein Kellerautomat heißt **deterministisch**, falls $\vdash$ eine rechtseindeutige Relation ist:*

$$K \vdash K_1 \wedge K \vdash K_2 \Rightarrow K_1 = K_2.$$

Äquivalent hierzu ist, dass die Überführungsfunktion δ für alle $(q,a,A) \in Z \times \Sigma \times \Gamma$ folgende Bedingung erfüllt (siehe Übungen):

$$\|\delta(q,a,A)\| + \|\delta(q,\varepsilon,A)\| \leq 1.$$

Beispiel 104. *Der PDA $M=(\{q_0,q_1,q_2\},\{a,b,c\},\{A,B,\#\},\delta,q_0,\#)$ mit der Überführungsfunktion*

$$\begin{aligned}\delta:\ & q_0a\# \to q_0A\# \quad q_0b\# \to q_0B\# \quad q_0aA \to q_0AA \quad q_0bA \to q_0BA\\ & q_0aB \to q_0AB \quad q_0bB \to q_0BB \quad q_0cA \to q_1A \quad q_0cB \to q_1B\\ & q_1aA \to q_1 \quad q_1bB \to q_1 \quad q_1\varepsilon\# \to q_2\end{aligned}$$

erkennt die Sprache $L(M)=\{xcx^R \mid x \in \{a,b\}^+\}$. Um auf einen Blick erkennen zu können, ob M deterministisch ist, empfiehlt es sich, δ in Form einer Tabelle darzustellen:

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

Man beachte, dass jedes Tabellenfeld höchstens eine Anweisung enthält und jede Spalte, die einen ε-Eintrag in der ersten Zeile hat, sonst keine weiteren Einträge enthält. Daher ist für alle $(q, a, A) \in Z \times \Sigma \times \Gamma$ die Bedingung

$$\|\delta(q, a, A)\| + \|\delta(q, \varepsilon, A)\| \leq 1$$

erfüllt. ◁

Verlangen wir von einem deterministischen Kellerautomaten, dass er seine Eingabe durch Leeren des Kellers akzeptiert, so können nicht alle regulären Sprachen von deterministischen Kellerautomaten erkannt werden. Um beispielsweise die Sprache $L = \{a, aa\}$ zu erkennen, muss der Keller von M nach Lesen von a geleert werden. Daher ist es M nicht mehr möglich, die Eingabe aa zu akzeptieren. Deterministische Kellerautomaten können also durch Leeren des Kellers nur **präfixfreie** Sprachen L akzeptieren (d.h. kein Wort $x \in L$ ist Präfix eines anderen Wortes in L).

Wir können das Problem aber einfach dadurch lösen, dass wir deterministischen Kellerautomaten erlauben, ihre Eingabe durch Erreichen eines Endzustands zu akzeptieren.

Definition 105.

- *Ein **Kellerautomat mit Endzuständen** wird durch ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ beschrieben. Dabei sind die Komponenten $Z, \Sigma, \Gamma, \delta, q_0, \#$ dieselben wie bei einem PDA und zusätzlich ist $E \subseteq Z$ eine Menge von **Endzuständen**.*
- *Die von M **akzeptierte** oder **erkannte Sprache** ist*

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

- *M ist ein **deterministischer Kellerautomat mit Endzuständen** (kurz: **DPDA**), falls M zusätzlich für alle $(q, a, A) \in Z \times \Sigma \times \Gamma$ folgende Bedingung erfüllt:*

$$\|\delta(q, a, A)\| + \|\delta(q, \varepsilon, A)\| \leq 1.$$

- *Die Klasse der deterministisch kontextfreien Sprachen ist definiert durch*

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

Als nächstes zeigen wir, dass DCFL unter Komplementbildung abgeschlossen ist. Versuchen wir, die End- und Nichtendzustände eines DPDA M einfach zu vertauschen, um einen DPDA $\overline{M}$ für $\overline{L(M)}$ zu erhalten, so ergeben sich folgende Schwierigkeiten:

1. Falls M eine Eingabe x nicht zu Ende liest, wird x weder von M noch von $\overline{M}$ akzeptiert.
2. Falls M nach dem Lesen von x noch ε-Übergänge ausführt und dabei End- und Nichtendzustände besucht, wird x von M und von $\overline{M}$ akzeptiert.

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

Satz 106. *Jede Sprache $L \in \mathsf{DCFL}$ wird von einem DPDA M' erkannt, der alle Eingaben zu Ende liest.*

Beweis. Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA mit $L(M) = L$. Falls M eine Eingabe $x = x_1 \cdots x_n$ nicht zu Ende liest, muss einer der folgenden drei Gründe vorliegen:

1. M gerät in eine Konfiguration $(q, x_i \cdots x_n, \varepsilon)$, $i \leq n$, mit leerem Keller.
2. M gerät in eine Konfiguration $(q, x_i \cdots x_n, A\gamma)$, $i \leq n$, in der wegen $\delta(q, x_i, A) = \delta(q, \varepsilon, A) = \emptyset$ keine Anweisung ausführbar ist.
3. M gerät in eine Konfiguration $(q, x_i \cdots x_n, A\gamma)$, $i \leq n$, so dass M ausgehend von der Konfiguration (q, ε, A) eine unendliche Folge von ε-Anweisungen ausführt.

Die erste Ursache schließen wir aus, indem wir ein neues Zeichen $\square$ auf dem Kellerboden platzieren:

(a) $s\varepsilon\# \to q_0\#\square$ (dabei sei s der neue Startzustand).

Die zweite Ursache schließen wir durch Hinzunahme eines Fehlerzustands f sowie folgender Anweisungen aus (hierbei ist $\Gamma' = \Gamma \cup \{\square\}$):

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

(c) $faA \to fA$, für alle $a \in \Sigma$ und $A \in \Gamma'$.

Als nächstes verhindern wir die Ausführung einer unendlichen Folge von ε-Übergängen. Dabei unterscheiden wir die beiden Fälle, ob M hierbei auch Endzustände besucht oder nicht. Falls ja, sehen wir einen Umweg über den neuen Endzustand e vor.

(d) $q\varepsilon A \to fA$, für alle $q \in Z$ und $A \in \Gamma$, so dass M ausgehend von der Konfiguration (q, ε, A) unendlich viele ε-Übergänge ausführt ohne dabei einen Endzustand zu besuchen.

(e) $q\varepsilon A \to eA$
$e\varepsilon A \to fA$, für alle $q \in Z$ und $A \in \Gamma$, so dass M ausgehend von der Konfiguration (q, ε, A) unendlich viele ε-Übergänge ausführt und dabei auch Endzustände besucht.

Schließlich übernehmen wir von M die folgenden Anweisungen:

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

Zusammenfassend transformieren wir M in den DPDA

$$M' = (Z \cup \{s, e, f\}, \Sigma, \Gamma', \delta', s, \#, E \cup \{e\})$$

mit $\Gamma' = \Gamma \cup \{\square\}$, wobei δ' die unter (a) bis (f) genannten Anweisungen enthält. ■

Beispiel 107. *Wenden wir diese Konstruktion auf den DPDA*

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

mit der Überführungsfunktion

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

an, so erhalten wir den DPDA

$$M' = (\{q_0, q_1, q_2, s, e, f\}, \{a, b, c\}, \{A, B, \#, \square\}, \delta', s, \#, \{q_2, e\})$$

mit folgender Überführungsfunktion δ':

δ'	$s, \#$	s, A	s, B	$s, \square$	$q_0, \#$	q_0, A	q_0, B	$q_0, \square$
ε	$q_0\#\square$	–	–	–	–	–	–	–
a	–	–	–	–	$q_0A\#$	q_0AA	q_0AB	$f\square$
b	–	–	–	–	$q_0B\#$	q_0BA	q_0BB	$f\square$
c	–	–	–	–	$f\#$	q_1A	q_1B	$f\square$
Typ	(a)				(f, b)	(f)	(f)	(b)

	$q_1, \#$	q_1, A	q_1, B	$q_1, \square$	$q_2, \#$	q_2, A	q_2, B	$q_2, \square$
ε	q_2	–	–	–	$e\#$	–	–	–
a	–	q_1	fB	$f\square$	–	fA	fB	$f\square$
b	–	fA	q_1	$f\square$	–	fA	fB	$f\square$
c	–	fA	fB	$f\square$	–	fA	fB	$f\square$
Typ	(f)	(f, b)	(f, b)	(b)	(e)	(b)	(b)	(b)

	$e,\#$	e,A	e,B	$e,\square$	$f,\#$	f,A	f,B	$f,\square$
ε	$f\#$	–	–	–	–	–	–	–
a	–	–	–	–	$f\#$	fA	fB	$f\square$
b	–	–	–	–	$f\#$	fA	fB	$f\square$
c	–	–	–	–	$f\#$	fA	fB	$f\square$
Typ	(e)				(c)	(c)	(c)	(c)

◁

Satz 108. *Die Klasse* DCFL *ist unter Komplement abgeschlossen, d.h. es gilt* DCFL = co-DCFL.

Beweis. Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA, der alle Eingaben zu Ende liest, und sei $L(M) = L$. Wir konstruieren einen DPDA $\overline{M}$ für $\overline{L}$.

Die Idee dabei ist, dass sich $\overline{M}$ in seinem Zustand (q, i) neben dem aktuellen Zustand q von M in der Komponente i merkt, ob M nach Lesen des letzten Zeichens (bzw. seit Rechnungsbeginn) einen Endzustand besucht hat ($i = 2$) oder nicht ($i = 1$). Möchte M das nächste Zeichen lesen und befindet sich $\overline{M}$ im Zustand $(q, 1)$, so macht $\overline{M}$ noch einen Umweg über den Endzustand $(q, 3)$.

Konkret erhalten wir $\overline{M} = (Z \times \{1,2,3\}, \Sigma, \Gamma, \delta', s, \#, Z \times \{3\})$ mit

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

indem wir zu δ' für jede Anweisung $q\varepsilon A \to_M p\gamma$ die Anweisungen

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

sowie für jede Anweisung $qaA \to_M p\gamma$ die Anweisungen

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

hinzufügen. ■

Eine nützliche Eigenschaft von $\overline{M}$ ist, dass $\overline{M}$ in einem Endzustand keine ε-Übergänge macht.

Beispiel 109. *Angenommen, ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ führt bei der Eingabe $x = a$ folgende Rechnung aus:*

$$(q_0, a, \#) \vdash (q_1, \varepsilon, \gamma_1) \vdash (q_2, \varepsilon, \gamma_2).$$

Dann würde $\overline{M}$ im Fall $E = \{q_0, q_2\}$ (d.h. $x \in L(M)$) die Rechnung

$$((q_0, 2), a, \#) \vdash ((q_1, 1), \varepsilon, \gamma_1) \vdash ((q_2, 2), \varepsilon, \gamma_2)$$

ausführen. Da $(q_1, 1), (q_2, 2) \notin Z \times \{3\}$ sind, verwirft also $\overline{M}$ das Wort a. Dagegen würde $\overline{M}$ im Fall $E = \{q_0\}$ (d.h. $x \notin L(M)$) die Rechnung

$$((q_0, 2), a, \#) \vdash ((q_1, 1), \varepsilon, \gamma_1) \vdash ((q_2, 1), \varepsilon, \gamma_2) \vdash ((q_2, 3), \varepsilon, \gamma_2)$$

ausführen. Da $(q_2, 3) \in Z \times \{3\}$ ein Endzustand von $\overline{M}$ ist, würde $\overline{M}$ nun also das Wort a akzeptieren. ◁

Satz 110. *Die Klasse* DCFL *ist nicht abgeschlossen unter Durchschnitt, Vereinigung, Produkt und Sternhülle.*

Beweis. Die beiden Sprachen

$$L_1 = \{a^n b^m c^m \mid n, m \geq 0\} \quad \text{und} \quad L_2 = \{a^n b^n c^m \mid n, m \geq 0\}$$

sind deterministisch kontextfrei (siehe Übungen). Da der Schnitt $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ nicht kontextfrei ist, liegt er auch nicht in DCFL, also ist DCFL nicht unter Durchschnitt abgeschlossen.

Da DCFL unter Komplementbildung abgeschlossen ist, kann DCFL wegen de Morgan dann auch nicht unter Vereinigung abgeschlossen sein. Beispielsweise sind folgende Sprachen deterministisch kontextfrei:

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

Ihre Vereinigung $L_3 \cup L_4 = \{a^i b^j c^k \mid i \neq j \text{ oder } j \neq k\}$ gehört aber nicht zu DCFL, d.h. $L_3 \cup L_4 \in \mathsf{CFL} \setminus \mathsf{DCFL}$. DCFL ist nämlich unter Schnitt mit regulären Sprachen abgeschlossen (siehe Übungen). Daher wäre mit $L_3 \cup L_4$ auch die Sprache

$$(\overline{L_3 \cup L_4}) \cap L(a^*b^*c^*) = \{a^n b^n c^n \mid n \geq 0\}$$

(deterministisch) kontextfrei. Als nächstes zeigen wir, dass DCFL nicht unter Produktbildung abgeschlossen ist. Wir wissen bereits, dass $L_3 \cup L_4 \notin \mathsf{DCFL}$ ist. Sei $L_0 = \{0\}$. Dann ist auch die Sprache

$$L_5 = L_0(L_3 \cup L_4) = \{0a^i b^j c^k \,|\, i \neq j \vee j \neq k\} \notin \mathsf{DCFL},$$

da sich ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ für L_5 leicht zu einem DPDA für $L_3 \cup L_4$ umbauen ließe. Sei nämlich (p, ε, γ) die Konfiguration, die M nach Lesen der Eingabe 0 erreicht. Dann erkennt der DPDA $M' = (Z \cup \{s\}, \Sigma, \Gamma, \delta', s, \#, E)$ die Sprache $L_3 \cup L_4$, wobei δ' wie folgt definiert ist:

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

Es ist leicht zu sehen, dass die beiden Sprachen L_0^* und $L = L_0 L_3 \cup L_4$ in DCFL sind (siehe Übungen). Ihr Produkt $L_0^* L$ gehört aber nicht zu DCFL. Da DCFL unter Schnitt mit regulären Sprachen abgeschlossen ist (siehe Übungen), wäre andernfalls auch

$$L_0^* L \cap L_0 L(a^*b^*c^*) = \{0a^i b^j c^k \,|\, i \neq j \vee j \neq k\} = L_0(L_3 \cup L_4) = L_5$$

in DCFL, was wir bereits ausgeschlossen haben. ■

Dass DCFL auch nicht unter Sternhüllenbildung abgeschlossen ist, lässt sich ganz ähnlich zeigen (siehe Übungen). Wir fassen die bewiesenen Abschlusseigenschaften der Klassen REG, DCFL und CFL in folgender Tabelle zusammen:

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

Die Klasse der deterministisch kontextfreien Sprachen lässt sich auch mit Hilfe von speziellen kontextfreien Grammatiken charakterisieren, den so genannten $LR(k)$-Grammatiken. Der erste Buchstabe L steht für die Leserichtung bei der Syntaxanalyse, d.h. das Eingabewort x wird von links (nach rechts) gelesen. Der zweite Buchstabe R bedeutet, dass bei der Syntaxanalyse eine Rechtsableitung entsteht. Schließlich gibt der Parameter k an, wieviele Zeichen man in der Eingabe vorauslesen muss, damit die nächste anzuwendende Regel eindeutig feststeht (k wird auch als *Lookahead* bezeichnet). Durch $LR(0)$-Grammatiken lassen sich nur die präfixfreien Sprachen in DCFL erzeugen. Dagegen erzeugen die $LR(k)$-Grammatiken für jedes $k \geq 1$ genau die Sprachen in DCFL. Daneben gibt es noch $LL(k)$-Grammatiken, die für wachsendes k immer mehr deterministisch kontextfreie Sprachen erzeugen.

4 Kontextsensitive Sprachen

In diesem Kapitel führen wir das Maschinenmodell des linear beschränkten Automaten (LBA) ein und zeigen, dass LBAs genau die kontextsensitiven Sprachen erkennen. Die Klasse CSL ist unter Komplementbildung abgeschlossen. Es ist jedoch offen, ob die Klasse DCSL der von einem deterministischen LBA erkannten Sprachen eine echte Teilklasse von CSL ist (diese Frage ist als *LBA-Problem* bekannt).

4.1 Kontextsensitive Grammatiken

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

Das nächste Beispiel zeigt, dass die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ von einer kontextsensitiven Grammatik erzeugt wird. Da L nicht kontextfrei ist, ist also die Klasse CFL echt in der Klasse CSL enthalten.

Beispiel 111. *Betrachte die kontextsensitive Grammatik* $G = (V, \Sigma, P, S)$ *mit* $V = \{S, B, C\}$, $\Sigma = \{a, b, c\}$ *und den Regeln*

$$P: \quad \begin{array}{llll} S \to aSBC, & aBC, & (1,2) & CB \to BC, \ (3) \quad aB \to ab, \ (4) \\ bB \to bb, & & (5) & C \to c. \quad (6) \end{array}$$

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

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

Allgemein gilt für alle $n \geq 1$*:*

$$\begin{aligned} S &\underset{(1)}{\Rightarrow}{}^{n-1} a^{n-1}S(BC)^{n-1} \underset{(2)}{\Rightarrow} a^n(BC)^n \underset{(3)}{\Rightarrow}{}^{\binom{n}{2}} a^n B^n C^n \\ &\underset{(4)}{\Rightarrow} a^n bB^{n-1}C^n \underset{(5)}{\Rightarrow}{}^{n-1} a^n b^n C^n \underset{(6)}{\Rightarrow}{}^{n} a^n b^n c^n \end{aligned}$$

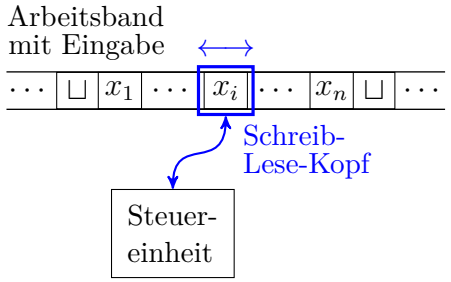
Also gilt $a^n b^n c^n \in L(G)$ *für alle* $n \geq 1$*. Umgekehrt folgt durch Induktion über die Ableitungslänge, dass jede Satzform* u *mit* $S \Rightarrow^* u$ *die folgenden Bedingungen erfüllt:*

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

Daraus ergibt sich, dass in G *nur Wörter der Form* $w = a^n b^n c^n$ *ableitbar sind.* ◁

4.2 Turingmaschinen

Um ein geeignetes Maschinenmodell für die kontextsensitiven Sprachen zu finden, führen wir zunächst das Rechenmodell der nichtdeterministischen Turingmaschine (NTM) ein. Eine NTM erhält ihre Eingabe auf einem nach links und rechts unbegrenzten Band. Während ihrer Rechnung kann sie den Schreib-Lese-Kopf auf dem Band in beide Richtungen bewegen und dabei die besuchten Bandfelder lesen sowie gelesenen Zeichen gegebenenfalls überschreiben.



Es gibt mehrere Arten von Turingmaschinen (u.a. mit einseitig unendlichem Band oder mit mehreren Schreib-Lese-Köpfen auf dem Band). Wir verwenden folgende Variante der Mehrband-Turingmaschine.

Definition 112. *Sei $k \geq 1$.*

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

- *Z eine endliche Menge von Zuständen,*
- *Σ das Eingabealphabet (wobei $\sqcup \notin \Sigma$),*
- *Γ das Arbeitsalphabet (wobei $\Sigma \cup \{\sqcup\} \subseteq \Gamma$),*
- *$\delta\colon Z \times \Gamma^k \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ die Überführungsfunktion,*
- *q_0 der Startzustand und*
- *$E \subseteq Z$ die Menge der Endzustände ist.*

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

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

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

Eine solche Anweisung ist ausführbar, falls

- q der aktuelle Zustand von M ist und
- sich für $i = 1, \ldots, k$ der Lesekopf des i-ten Bandes auf einem mit a_i beschrifteten Feld befindet.

Ihre Ausführung bewirkt, dass M

- vom Zustand q 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 113. *Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM.*

*a) Eine **Konfiguration** von M ist ein $(3k+1)$-Tupel*

$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \in Z \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$$

und besagt, dass

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

Im Fall $k = 1$ schreiben wir für eine Konfiguration (q, u, a, v) auch kurz $uqav$.

*b) Die **Startkonfiguration** von M bei Eingabe $x = x_1 \cdots x_n \in \Sigma^*$ ist*

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

*c) Eine Konfiguration $K' = (q, u'_1, a'_1, v'_1, \ldots, u'_k, a'_k, v'_k)$ heißt **Folgekonfiguration** von $K = (p, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ (kurz $K \vdash K'$), falls eine Anweisung*

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

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

im Fall $D_i = N$:	*$D_i = R$:*	*$D_i = L$:*
K: $u_i \boxed{a_i} v_i$	K: $u_i \boxed{a_i} v_i$	K: $u_i \boxed{a_i} v_i$
K': $u_i \boxed{b_i} v_i$	K': $u_i\ b_i \boxed{a'_i} v'_i$	K': $u'_i \boxed{a'_i}\ b_i\ v_i$
$u'_i = u_i$, $a'_i = b_i$ *und* $v'_i = v_i$.	$u'_i = u_i b_i$ *und* $a'_i v'_i = \begin{cases} v_i, & v_i \neq \varepsilon, \\ \sqcup, & \text{sonst.} \end{cases}$	$u'_i a'_i = \begin{cases} u_i, & u_i \neq \varepsilon, \\ \sqcup, & \text{sonst} \end{cases}$ *und* $v'_i = b_i v_i$.

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

d) *Eine* ***Rechnung*** *von M bei Eingabe x ist eine Folge von Konfigurationen $K_0, K_1, K_2 \dots$ mit $K_0 = K_x$ und $K_0 \vdash K_1 \vdash K_2 \cdots$.*

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

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

M akzeptiert also eine Eingabe x (hierfür sagen wir kurz $M(x)$ akzeptiert), falls es eine Rechnung von $M(x)$ gibt, bei der ein Endzustand erreicht wird.

Beispiel 114. *Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \dots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{A, B, \sqcup\}$, $E = \{q_4\}$, wobei δ folgende Anweisungen emthält:*

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

Dann führt M bei Eingabe aabb folgende Rechnung aus:

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

Ähnlich läßt sich $a^n b^n \in L(M)$ für ein beliebiges $n \geq 1$ zeigen. Andererseits führt die Eingabe abb auf die Rechnung

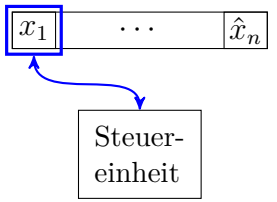
$$q_0abb \underset{(1)}{\vdash} Aq_1bb \underset{(4)}{\vdash} q_2ABb \underset{(7)}{\vdash} Aq_0Bb \underset{(8)}{\vdash} ABq_3b,$$

die nicht weiter fortsetzbar ist. Da M deterministisch ist, kann $M(abb)$ auch nicht durch eine andere Rechnung den Endzustand q_4 erreichen. D.h. abb gehört nicht zu $L(M)$. Tatsächlich lässt sich durch Betrachtung der übrigen Fälle ($x = a^n b^m$, $n > m$, $x = a^n b^m a^k$, $m, k \geq 1$, etc.) zeigen, dass M nur Eingaben der Form $a^n b^n$ akzeptiert, und somit $L(M) = \{a^n b^n \mid n \geq 1\}$ ist. ◁

Es ist leicht zu sehen, dass jede Typ-0 Sprache von einer NTM M akzeptiert wird, die ausgehend von x eine Rückwärtsableitung (Reduktion) auf das Startsymbol sucht. Ist $x \neq \varepsilon$ und markieren wir das letzte Zeichen von x, so kann M das Ende der Eingabe erkennen, ohne darüber hinaus lesen zu müssen. Zudem ist im Fall einer Typ-1 Sprache die linke Seite einer Regel höchstens so lang wie die rechte Seite. Deshalb muss M beim Erkennen von kontextsensitiven Sprachen den Bereich der Eingabe während der Rechnung nicht verlassen.

4.3 Linear beschränkte Automaten

Eine 1-NTM M, die bei keiner Eingabe $x \neq \varepsilon$, deren letztes Zeichen markiert ist, den Bereich der Eingabe verlässt, wird als LBA (linear beschränkter Automat) bezeichnet. Ein LBA darf also bei Eingaben der Länge $n > 0$ während der Rechnung nur die n mit der Eingabe beschrifteten Bandfelder besuchen und überschreiben. Tatsächlich lässt sich zeigen, dass jede k-NTM, die bei Eingaben der Länge n höchstens linear viele (also $cn+c$ für eine



Konstante c) Bandfelder besucht, von einem LBA simuliert werden kann.

In diesem Abschnitt zeigen wir, dass LBAs genau die kontextsensitiven Sprachen erkennen.

Definition 115.

a) Für ein Alphabet Σ und ein Wort $x = x_1 \cdots x_n \in \Sigma^*$ bezeichne $\hat{x}$ das Wort

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

über dem Alphabet $\hat{\Sigma} = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$.

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

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

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

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

Beispiel 116. *Es ist nicht schwer, die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ aus Beispiel 114 mit der Überführungsfunktion*

$$\begin{array}{llllll} \delta\colon & q_0a \to q_1AR & (1) & q_1a \to q_1aR & (2) & q_1B \to q_1BR \quad (3) \\ & q_1b \to q_2BL & (4) & q_2a \to q_2aL & (5) & q_2B \to q_2BL \quad (6) \\ & q_2A \to q_0AR & (7) & q_0B \to q_3BR & (8) & q_3B \to q_3BR \quad (9) \\ & q_3\sqcup \to q_4\sqcup N & (10) & & & \end{array}$$

*in einen deterministischen LBA (kurz **DLBA**) M' für die Sprache $\{a^nb^n \mid n \geq 1\}$ umzuwandeln. Ersetze hierzu*

- *Σ durch $\hat{\Sigma} = \{a, b, \hat{a}, \hat{b}\}$,*
- *Γ durch $\Gamma' = \hat{\Sigma} \cup \{A, B, \hat{B}, \sqcup\}$ sowie*
- *die Anweisung $q_3\sqcup \to q_4\sqcup N$ (10) durch $q_3\hat{B} \to q_4\hat{B}N$ (10′)*

und füge die Anweisungen $q_1\hat{b} \to q_2\hat{B}L$ (4a) und $q_0\hat{B} \to q_4\hat{B}N$ (8a) hinzu. Dann erhalten wir den DLBA $M' = (Z, \hat{\Sigma}, \Gamma', \delta', q_0, E)$ mit der Überführungsfunktion

$$\begin{array}{llllll} \delta'\colon & q_0a \to q_1AR & (1) & q_1\hat{b} \to q_2\hat{B}L & (4a) & q_0B \to q_3BR \quad (8) \\ & q_1a \to q_1aR & (2) & q_2a \to q_2aL & (5) & q_0\hat{B} \to q_4\hat{B}N \quad (8a) \\ & q_1B \to q_1BR & (3) & q_2B \to q_2BL & (6) & q_3B \to q_3BR \quad (9) \\ & q_1b \to q_2BL & (4) & q_2A \to q_0AR & (7) & q_3\hat{B} \to q_4\hat{B}N \quad (10') \end{array}$$

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

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

◁

Definition 117. *Die Klasse der **deterministisch kontextsensitiven** Sprachen ist definiert als*

$$\mathsf{DCSL} = \{L(M) \mid M \text{ ist ein DLBA}\}.$$

Der DLBA M' für die Sprache $A = \{a^nb^n \mid n \geq 1\}$ aus dem letzten Beispiel lässt sich leicht in einen DLBA für die Sprache $B = \{a^nb^nc^n \mid n \geq 1\}$ transformieren (siehe Übungen), d.h. $B \in \mathsf{DCSL} \setminus \mathsf{CFL}$. Die Inklusion von CFL in DCSL wird in den Übungen gezeigt.

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

Satz 118. $\mathsf{CSL} = \{L(M) \mid M \text{ ist ein LBA}\}$.

Beweis. Wir zeigen zuerst die Inklusion von links nach rechts. Sei $G = (V, \Sigma, P, S)$ eine kontextsensitive Grammatik. Dann wird $L(G)$ von folgendem LBA M akzeptiert (o.B.d.A. sei $\varepsilon \notin L(G)$):

Arbeitsweise von M bei Eingabe $x = x_1 \cdots x_n$ mit $n > 0$:

1. Markiere das erste Eingabezeichen x_1
2. Wähle eine beliebige Regel $\alpha \to \beta$ aus P
3. Wähle ein beliebiges Vorkommen von β auf dem Band (falls β nicht vorkommt, halte ohne zu akzeptieren)
4. Ersetze die ersten $|\alpha|$ Zeichen von β durch α
5. Falls das erste (oder letzte) Zeichen von β markiert war, markiere auch das erste (letzte) Zeichen von α
6. Verschiebe die Zeichen rechts von β um $|\beta| - |\alpha|$ Positionen nach links und überschreibe die frei werdenden Bandfelder mit Blanks
7. Enthält das Band außer Blanks nur das (markierte) Startsymbol, so halte in einem Endzustand
8. Gehe zurück zu Schritt 2

Nun ist leicht zu sehen, dass M wegen $|\beta| \geq |\alpha|$ tatsächlich ein LBA ist. M akzeptiert eine Eingabe x, falls es gelingt, eine Ableitung für x in G zu finden (in umgekehrter Reihenfolge, d.h. M ist ein nichtdeterministischer *Bottom-Up Parser*). Da sich genau für die Wörter in $L(G)$ eine Ableitung finden lässt, folgt $L(M) = L(G)$.

Für den Beweis der umgekehrten Inklusion sei ein LBA $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ gegeben (o.B.d.A. sei $\varepsilon \notin L(M)$). Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit

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

die für alle $a, b \in \Sigma$ und $c, d \in \Gamma$ folgende Regeln enthält:

$$\begin{array}{lll}
P\colon & S \to A(\hat{a}, a),\ (q_0\hat{a}, a) & \text{(S) „Startregeln“} \\
& A \to A(a, a),\ (q_0 a, a) & \text{(A) „A-Regeln“} \\
& (c, a) \to a & \text{(F) „Finale Regeln“} \\
& (qc, a) \to a, \qquad \text{falls } q \in E & \text{(E) „E-Regeln“}
\end{array}$$

$$\begin{array}{lll}
(qc, a) \to (q'c', a), & \text{falls } qc \to_M q'c'N & \text{(N) „N-Regeln“} \\
(qc, a)(d, b) \to (c', a)(q'd, b), & \text{falls } qc \to_M q'c'R & \text{(R) „R-Regeln“} \\
(d, a)(qc, b) \to (q'd, a)(c', b), & \text{falls } qc \to_M q'c'L & \text{(L) „L-Regeln“}
\end{array}$$

Durch Induktion über m lässt sich nun leicht für alle $a_1, \ldots, a_n \in \Gamma$ und $q \in Z$ die folgende Äquivalenz beweisen:

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

Ist also $q_0 x_1 \cdots x_{n-1}\hat{x}_n \;\vdash^m a_1 \cdots a_{i-1} q a_i \cdots a_n$ mit $q \in E$ eine akzeptierende Rechnung von $M(x_1 \cdots x_{n-1}\hat{x}_n)$, so folgt

$$\begin{aligned}
S \;& \underset{(S,A)}{\Rightarrow}^n (q_0 x_1, x_1)(x_2, x_2) \cdots (x_{n-1}, x_{n-1})(\hat{x}_n, x_n) \\
& \underset{(N,L,R)}{\Rightarrow}^m (a_1, x_1) \cdots (a_{i-1}, x_{i-1})(qa_i, x_i) \cdots (a_n, x_n) \\
& \underset{(F,E)}{\Rightarrow}^n x_1 \cdots x_n
\end{aligned}$$

Die Inklusion $L(G) \subseteq L(M)$ folgt analog. ■

Eine einfache Modifikation des Beweises zeigt, dass 1-NTMs genau die Sprachen vom Typ 0 akzeptieren (siehe Übungen).

Beispiel 119. *Betrachte den LBA $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \hat{a}, \hat{b}, A, B, \hat{B}, \sqcup\}$ und $E = \{q_4\}$, sowie*

$$\begin{array}{llll}
\delta\colon & q_0 a \to q_1 AR & q_1\hat{b} \to q_2\hat{B}L & q_0 B \to q_3 BR \\
& q_1 a \to q_1 aR & q_2 a \to q_2 aL & q_0\hat{B} \to q_4\hat{B}N \\
& q_1 B \to q_1 BR & q_2 B \to q_2 BL & q_3 B \to q_3 BR \\
& q_1 b \to q_2 BL & q_2 A \to q_0 AR & q_3\hat{B} \to q_4\hat{B}N
\end{array}$$

Die zugehörige kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ enthält dann neben den Start- und A-Regeln

$$S \to A(\hat{a}, a),\ A(\hat{b}, b),\ (q_0\hat{a}, a),\ (q_0\hat{b}, b) \qquad (S_1\text{-}S_4)$$
$$A \to A(a, a),\ A(b, b),\ (q_0 a, a),\ (q_0 b, b) \qquad (A_1\text{-}A_4)$$

für jedes Zeichen $c \in \Gamma$ folgende F- und E-Regeln (wegen $E = \{q_4\}$):

$$(c, a) \to a \text{ und } (c, b) \to b \qquad (F_1\text{-}F_{16})$$
$$(q_4 c, a) \to a \text{ und } (q_4 c, b) \to b \qquad (E_1\text{-}E_{16})$$

Daneben enthält P beispielsweise für die Anweisung $q_3\hat{B} \to q_4\hat{B}N$ folgende zwei N-Regeln:

$$(q_3\hat{B}, a) \to (q_4\hat{B}, a), \quad (q_3\hat{B}, b) \to (q_4\hat{B}, b).$$

Für die Anweisung $q_1 b \to q_2 B L$ kommen für jedes $d \in \Gamma$ die vier L-Regeln

$$(d, a)(q_1 b, a) \to (q_2 d, a)(B, a), \quad (d, b)(q_1 b, a) \to (q_2 d, b)(B, a)$$
$$(d, a)(q_1 b, b) \to (q_2 d, a)(B, b), \quad (d, b)(q_1 b, b) \to (q_2 d, b)(B, b)$$

zu P hinzu und die Anweisung $q_0 a \to q_1 A R$ bewirkt für jedes $d \in \Gamma$ die Hinzunahme folgender vier R-Regeln:

$$(q_0 a, a)(d, a) \to (A, a)(q_1 d, a), \quad (q_0 a, a)(d, b) \to (A, a)(q_1 d, b)$$
$$(q_0 a, b)(d, a) \to (A, b)(q_1 d, a), \quad (q_0 a, b)(d, b) \to (A, b)(q_1 d, b)$$

◁

Folgende Tabelle gibt einen Überblick über die Abschlusseigenschaften der Klassen REG, DCFL, CFL, DCSL, CSL und RE. In der Vorlesung Komplexitätstheorie wird gezeigt, dass die Klasse CSL unter Komplementbildung abgeschlossen ist. Im nächsten Kapitel werden wir sehen, dass die Klasse RE nicht unter Komplementbildung abgeschlossen ist. Die übrigen Abschlusseigenschaften in folgender Tabelle werden in den Übungen bewiesen.

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

5 Entscheidbare und semi-entscheidbare Sprachen

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

Definition 120.

*a) Eine NTM M **hält** bei Eingabe x, falls alle Rechnungen von $M(x)$ eine endliche Länge haben.*

*b) Eine NTM M **entscheidet** eine Eingabe x, falls $M(x)$ hält oder eine Konfiguration mit einem Endzustand erreicht.*

*c) Eine Sprache $L \subseteq \Sigma^*$ heißt **entscheidbar**, falls eine DTM M mit $L(M) = L$ existiert, die jede Eingabe $x \in \Sigma^*$ entscheidet.*

*d) Jede von einer DTM M erkannte Sprache heißt **semi-entscheidbar**.*

Bemerkung 121.

- *Die von einer DTM M akzeptierte Sprache $L(M)$ wird als semi-entscheidbar bezeichnet, da M zwar alle (positiven) Eingaben $x \in L$ entscheidet, aber möglicherweise nicht alle (negativen) Eingaben $x \in \bar{L}$.*
- *Wir werden später sehen, dass genau die Typ-0 Sprachen semi-entscheidbar sind.*

Wir wenden uns nun der Berechnung von Funktionen zu.

Definition 122. *Eine k-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine Funktion $f : \Sigma^* \to \Gamma^*$, falls M bei jeder Eingabe $x \in \Sigma^*$ in einer Konfiguration*

$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \in Z \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$$

*mit $u_k = f(x)$ hält (d.h. $K_x \vdash^* K$ und K hat keine Folgekonfiguration). Hierfür sagen wir auch, M gibt bei Eingabe x das Wort $f(x)$ aus und schreiben $M(x) = f(x)$. f heißt **Turing-berechenbar** (oder einfach **berechenbar**), falls es eine k-DTM M mit $M(x) = f(x)$ für alle $x \in \Sigma^*$ gibt.*

Um eine Funktion $f : \Sigma^* \to \Gamma^*$ zu berechnen, muss M also bei jeder Eingabe x den Funktionswert $f(x)$ auf das k-te Band schreiben und danach halten. Falls M nicht bei allen Eingaben hält, berechnet M keine totale, sondern eine partielle Funktion.

Definition 123.

*a) Eine **partielle Funktion** hat die Form $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$.*

*b) Für $f(x) = \uparrow$ sagen wir auch $f(x)$ ist **undefiniert**.*

*c) Der **Definitionsbereich** (engl. domain) von f ist*

$$dom(f) = \{x \in \Sigma^* \mid f(x) \neq \uparrow\}.$$

*d) Das **Bild** (engl. image) von f ist*

$$img(f) = \{f(x) \mid x \in dom(f)\}.$$

*e) Eine DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine partielle Funktion $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$, falls $M(x)$ für alle $x \in dom(f)$ das Wort $f(x)$ ausgibt und für alle $x \notin dom(f)$ keine Ausgabe berechnet (d.h. $M(x)$ darf im Fall $x \notin dom(f)$ nicht halten).*

Aus historischen Gründen werden die berechenbaren Funktionen und die entscheidbaren Sprachen auch **rekursiv** (engl. *recursive*) genannt. Wir fassen die (semi-) entscheidbaren Sprachen und die (partiellen) berechenbaren Funktionen in folgenden Klassen zusammen:

$$\begin{aligned}\mathsf{REC} &= \{L(M) \mid M \text{ ist eine DTM, die jede Eingabe entscheidet}\},\\ \mathsf{FREC} &= \{f \mid f \text{ ist eine (totale) berechenbare Funktion}\},\\ \mathsf{FREC}_p &= \{f \mid f \text{ ist eine partielle berechenbare Funktion}\}.\end{aligned}$$

Dann gilt $\mathsf{FREC} \subsetneq \mathsf{FREC}_p$ und

$$\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DCSL} \subseteq \mathsf{CSL} \subsetneq \mathsf{REC} \subsetneq \mathsf{RE}.$$

Wir wissen bereits, dass die Inklusionen $\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DCSL}$ echt sind. In diesem Abschnitt werden wir die Echtheit der Inklusion $\mathsf{REC} \subsetneq \mathsf{RE}$ zeigen. Dass CSL eine echte Teilklasse von REC ist, wird in den Übungen gezeigt.

Beispiel 124. *Bezeichne x^+ den **lexikografischen Nachfolger** von $x \in \Sigma^*$. Für $\Sigma = \{0,1\}$ ergeben sich beispielsweise folgende Werte:*

x	ε	0	1	00	01	10	11	000	...
x^+	0	1	00	01	10	11	000	001	...

Betrachte die auf Σ^* definierten Funktionen f_1, f_2, f_3, f_4 mit

$$f_1(x) = 0, \qquad f_2(x) = x, \qquad f_3(x) = x^+ \qquad \text{und} \qquad f_4(x) = \begin{cases}\uparrow, & x = \varepsilon,\\ y, & x = y^+.\end{cases}$$

Da diese vier Funktionen alle berechenbar sind, gehören die totalen Funktionen f_1, f_2, f_3 zu FREC, während die partielle Funktion f_4 zu FREC_p gehört. ◁

Wie der nächste Satz zeigt, lässt sich jedes Entscheidungsproblem auf ein funktionales Problem zurückführen.

Satz 125.

*(i) Eine Sprache $A \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn ihre **charakteristische Funktion** $\chi_A : \Sigma^* \to \{0,1\}$ berechenbar ist. Diese ist wie folgt definiert:*

$$\chi_A(x) = \begin{cases}1, & x \in A,\\ 0, & x \notin A.\end{cases}$$

*(ii) Eine Sprache $A \subseteq \Sigma^*$ ist genau dann semi-entscheidbar, falls die **partielle charakteristische Funktion** $\hat{\chi}_A : \Sigma^* \to \{0,1,\uparrow\}$ berechenbar ist. Letztere ist wie folgt definiert:*

$$\hat{\chi}_A(x) = \begin{cases}1, & x \in A,\\ \uparrow, & x \notin A.\end{cases}$$

Beweis. Siehe Übungen. ■

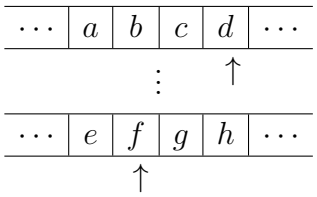
Definition 126. *Eine Sprache $A \subseteq \Sigma^*$ heißt **rekursiv aufzählbar**, falls A entweder leer oder das Bild $\mathrm{img}(f)$ einer berechenbaren Funktion $f : \Gamma^* \to \Sigma^*$ für ein beliebiges Alphabet Γ ist.*

Satz 127. *Folgende Eigenschaften sind äquivalent:*

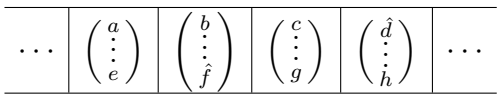
1. *A ist semi-entscheidbar (d.h. A wird von einer DTM akzeptiert),*
2. *A wird von einer 1-DTM akzeptiert,*
3. *A wird von einer 1-NTM akzeptiert,*
4. *A ist vom Typ 0,*
5. *A wird von einer NTM akzeptiert,*
6. *A ist rekursiv aufzählbar.*

Beweis. 1) ⇒ 2)**:** Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-DTM, die A akzeptiert. Wir konstruieren eine 1-DTM $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$

mit $L(M') = A$. M' simuliert M, indem sie jede Konfiguration K von M der Form



durch eine Konfiguration K' folgender 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 $K_x = (q_0, \varepsilon, x, \varepsilon, \sqcup, \ldots, \varepsilon, \sqcup)$ von M bei Eingabe x entsprechende Konfiguration

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

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

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

2) ⇒ 3)**:** Klar.

3) ⇒ 4) ⇒ 5)**:** Diese beiden Implikationen lassen sich ganz ähnlich wie die Charakterisierung der Typ-1 Sprachen durch LBAs zeigen (siehe Übungen).

5) ⇒ 6)**:** Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM, die eine Sprache $A \neq \emptyset$ akzeptiert. Kodieren wir eine Konfiguration $K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ von M durch das Wort

$$code(K) = \#q\#u_1\#a_1\#v_1\#\cdots\#u_k\#a_k\#v_k\#$$

über dem Alphabet $\tilde{\Gamma} = Z \cup \Gamma \cup \{\#\}$ und eine Rechnung $K_0 \vdash \cdots \vdash K_t$ durch $code(K_0) \cdots code(K_t)$, so lassen sich die Wörter von A durch folgende Funktion $f : \tilde{\Gamma}^* \to \Sigma^*$ aufzählen (dabei ist x_0 ein beliebiges Wort in A):

$$f(x) = \begin{cases} y, & x \text{ kodiert eine Rechnung } K_0 \vdash \cdots \vdash K_t \text{ von} \\ & M \text{ mit } K_0 = K_y \text{ und } K_t \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k \\ x_0, & \text{sonst.} \end{cases}$$

Da f berechenbar ist, ist $A = img(f)$ rekursiv aufzählbar.

6) ⇒ 1)**:** Sei $f : \Gamma^* \to \Sigma^*$ eine Funktion mit $A = img(f)$ und sei M eine k-DTM, die f berechnet. Dann akzeptiert folgende $(k+1)$-DTM M' die Sprache A.

> M' berechnet bei Eingabe x auf dem 2. Band der Reihe nach für alle Wörter $y \in \Gamma^*$ den Wert $f(y)$ durch Simulation von $M(y)$ und akzeptiert, sobald sie ein y mit $f(y) = x$ findet. ■

Satz 128. *A ist genau dann entscheidbar, wenn A und $\bar{A}$ semi-entscheidbar sind, d.h.* $\mathsf{REC} = \mathsf{RE} \cap \mathsf{co\text{-}RE}$.

Beweis. Sei A entscheidbar. Es ist leicht zu sehen, dass dann auch $\bar{A}$ entscheidbar ist. Also sind dann A und $\bar{A}$ auch semi-entscheidbar. Für die Rückrichtung seien $f_1, f_2 : \Gamma^* \to \Sigma^*$ Turing-berechenbare Funktionen mit $img(f_1) = A$ und $img(f_2) = \bar{A}$. Wir betrachten folgende k-DTM M, die bei Eingabe x

- für jedes $y \in \Gamma^*$ die beiden Werte $f_1(y)$ und $f_2(y)$ bestimmt,
- x akzeptiert, sobald ein y auf den Wert $f_1(y) = x$ führt und
- x verwirft, sobald ein y auf den Wert $f_2(y) = x$ führt.

Da jede Eingabe x entweder in $img(f_1) = A$ oder in $img(f_2) = \bar{A}$ enthalten ist, entscheidet M alle Eingaben. ■

5.1 Unentscheidbarkeit des Halteproblems

Eine für die Programmverifikation sehr wichtige Fragestellung ist, ob ein gegebenes Programm bei allen Eingaben nach endlich vielen Rechenschritten stoppt. In diesem Abschnitt werden wir zeigen, dass es zur Lösung dieses Problems keinen Algorithmus gibt, nicht einmal dann, wenn wir die Eingabe fixieren. Damit Turingmaschinen die Eingabe für Turingmaschinenprogramme bilden können, müssen wir eine geeignete Kodierung von Turingmaschinen vereinbaren (diese wird auch Gödelisierung genannt).

Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine 1-DTM mit Zustandsmenge $Z = \{q_0, \ldots, q_m\}$ (o.B.d.A. sei $E = \{q_m\}$) und Eingabealphabet $\Sigma = \{0, 1, \#\}$. Das Arbeitsalphabet sei $\Gamma = \{a_0, \ldots, a_l\}$, wobei wir o.B.d.A. $a_0 = 0$, $a_1 = 1$, $a_2 = \#$, $a_3 = \sqcup$ annehmen. Dann können wir jede Anweisung der Form $q_i a_j \to q_{i'} a_{j'} D$ durch das Wort

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#b_D\#$$

kodieren. Dabei ist $bin(n)$ die Binärdarstellung von n und $b_N = 0$, $b_L = 1$, sowie $b_R = 10$. M lässt sich nun als ein Wort über dem Alphabet $\{0, 1, \#\}$ kodieren, indem wir die Anweisungen von M in kodierter Form auflisten. Kodieren wir die Zeichen $0, 1, \#$ binär (z.B. $0 \mapsto 00$, $1 \mapsto 11$, $\# \mapsto 10$), so gelangen wir zu einer Binärkodierung w_M von M.

Die Binärzahl w_M wird auch die **Gödel-Nummer** von M genannt (tatsächlich kodierte Kurt Gödel Turingmaschinen durch natürliche Zahlen und nicht durch Binärstrings). Die Maschine M_w ist durch die Angabe von w bis auf die Benennung ihrer Zustände und Arbeitszeichen eindeutig bestimmt. Ganz analog lassen sich auch DTMs mit einer beliebigen Anzahl von Bändern (sowie NTMs, Konfigurationen oder Rechnungen von TMs) kodieren.

Umgekehrt können wir jedem Binärstring $w \in \{0, 1\}^*$ eine DTM M_w wie folgt zuordnen:

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

Hierbei ist M_0 eine beliebige DTM.

Definition 129.

*a) Das **Halteproblem** ist die Sprache*

$$\mathrm{H} = \left\{ w\#x \,\middle|\, \begin{array}{l} w, x \in \{0,1\}^* \text{ und die DTM} \\ M_w \text{ hält bei Eingabe } x \end{array} \right\}$$

*b) Das **spezielle Halteproblem** ist*

$$\mathrm{K} = \left\{ w \in \{0,1\}^* \,\middle|\, \begin{array}{l} \text{die DTM } M_w \\ \text{hält bei Eingabe } w \end{array} \right\}$$

χ_H	x_1	x_2	x_3	$\cdots$
w_1	1	1	0	$\cdots$
w_2	0	0	1	$\cdots$
w_3	1	1	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

χ_K				
w_1	1			
w_2		0		
w_3			1	
$\vdots$				$\ddots$

Der Werteverlauf der charakteristischen Funktion χ_K von K bildet also die Diagonale der als Matrix dargestellten charakteristischen Funktion χ_H von H.

Satz 130. $\mathrm{K} \in \mathsf{RE} \setminus \mathsf{REC}$.

Beweis. Wir zeigen zuerst $\mathrm{K} \in \mathsf{RE}$. Sei w_0 die Kodierung einer DTM, die bei jeder Eingabe (sofort) hält und betrachte die Funktion

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

Da f berechenbar und $img(f) = \mathrm{K}$ ist, folgt $\mathrm{K} \in \mathsf{RE}$. Um zu zeigen, dass K unentscheidbar ist, führen wir die Annahme $\mathrm{K} \in \mathsf{REC}$ auf einen Widerspruch. Angenommen, die Sprache

$$\mathrm{K} = \{w \mid M_w(w) \text{ hält}\} \qquad (*)$$

wäre durch eine DTM M_K entscheidbar. Betrachte die DTM $\hat{M}$, die bei Eingabe $w \in \{0,1\}^*$ die DTM $M_K(w)$ simuliert und genau dann hält, wenn $M_K(w)$ verwirft:

$$\hat{M}(x) \text{ hält } \Leftrightarrow x \notin \mathrm{K} \qquad (**)$$

χ_H	x_1	x_2	x_3	$\cdots$
w_1	**1**	1	0	$\cdots$
w_2	0	**0**	1	$\cdots$
w_3	1	1	**1**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\hat{w}$	0	1	0	$\cdots$

Für die Kodierung $\hat{w}$ von $\hat{M}$ folgt dann aber

$$\hat{w} \in \mathrm{K} \overset{(*)}{\Leftrightarrow} M_{\hat{w}}(\hat{w}) \text{ hält} \overset{(**)}{\Leftrightarrow} \hat{w} \notin \mathrm{K} \lightning \text{ (Widerspruch!)} \qquad \blacksquare$$

Das Argument in obigem Beweis wird als **Diagonalisierung** bezeichnet. Wir benutzen die Annahme, dass K entscheidbar ist zur Konstruktion einer DTM $\hat{M}$, die sich bei Eingabe w anders verhält als die DTM M_w: $\hat{M}$ hält bei Eingabe w genau dann, wenn M_w dies nicht tut. Da sich aber $\hat{M}$ nicht anders als sie selbst verhalten kann, folgt der gewünschte Widerspruch.

Durch ein ähnliches Diagonalisierungsargument lässt sich auch eine entscheidbare Sprache definieren, die sich von jeder kontextsensitiven Sprache unterscheidet (siehe Übungen).

Korollar 131.

(i) $\mathsf{REC} \subsetneq \mathsf{RE}$,

(ii) $\mathrm{K} \in \mathsf{RE} \setminus \mathsf{co\text{-}RE}$ *(d.h.* $\mathsf{RE} \neq \mathsf{co\text{-}RE}$*).*

Beweis.

(i) $\mathsf{REC} \subsetneq \mathsf{RE}$: klar da $\mathrm{K} \in \mathsf{RE} - \mathsf{REC}$.

(ii) $\mathrm{K} \notin \mathsf{co\text{-}RE}$: Aus der Annahme $\mathrm{K} \in \mathsf{co\text{-}RE}$ würde wegen $\mathrm{K} \in \mathsf{RE}$ folgen, dass K entscheidbar ist (Widerspruch). $\blacksquare$

Damit ist $\bar{K}$ eine Sprache in $\mathsf{co\text{-}RE}$, die nicht semi-entscheidbar ist.

Definition 132.

a) Eine Sprache $A \subseteq \Sigma^*$ heißt auf $B \subseteq \Gamma^*$ ***reduzierbar*** *(kurz: $A \leq B$), falls eine berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ ex., so dass gilt:*

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

b) Eine Sprachklasse $\mathcal{C}$ heißt ***unter $\leq$ abgeschlossen****, wenn für alle Sprachen A, B gilt:*

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

c) Eine Sprache A heißt ***hart*** *für eine Sprachklasse $\mathcal{C}$ (kurz:* ***$\mathcal{C}$-hart*** *oder* ***$\mathcal{C}$-schwer****), falls jede Sprache $L \in \mathcal{C}$ auf A reduzierbar ist:*

$$\forall L \in \mathcal{C} : L \leq A.$$

d) Eine $\mathcal{C}$-harte Sprache A, die zu $\mathcal{C}$ gehört, heißt ***$\mathcal{C}$-vollständig****.*

Beispiel 133. *Es gilt* $\mathrm{K} \leq \mathrm{H}$ *mittels* $f : w \mapsto w\#w$*, da für alle* $w \in \{0,1\}^*$ *gilt:*

$$\begin{aligned} w \in K &\Leftrightarrow M_w \text{ ist eine DTM, die bei Eingabe } w \text{ hält} \\ &\Leftrightarrow w\#w \in H. \end{aligned}$$

Das Halteproblem H ist sogar RE*-hart, da sich jede semi-entscheidbare Sprache A auf H reduzieren lässt. Die Reduktion $A \leq H$ leistet beispielsweise die Funktion $x \mapsto w\#x$, wobei w die Kodierung einer DTM M_w ist, die die partielle charakteristische Funktion $\hat{\chi}_A$ von A berechnet.* ◁

Satz 134. *Die Klasse* REC *ist unter $\leq$ abgeschlossen.*

Beweis. Gelte $A \leq B$ mittels f und sei M eine DTM, die χ_B berechnet. Betrachte folgende 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

$$M'(x) = M(f(x)) = \chi_B(f(x)) = \chi_A(x).$$

Also berechnet M' die Funktion χ_A, d.h. $A \in$ REC. ■

Der Abschluss von RE unter $\leq$ folgt analog (siehe Übungen).

Korollar 135.

1. $A \leq B \wedge A \notin \mathsf{REC} \Rightarrow B \notin \mathsf{REC}$.
2. $A \leq B \wedge A \notin \mathsf{RE} \Rightarrow B \notin \mathsf{RE}$.

Beweis. Aus der Annahme, dass B entscheidbar (bzw. semi-entscheidbar) ist, folgt wegen $A \leq B$, dass dies auch auf A zutrifft (Widerspruch). ■

Wegen K $\leq$ H überträgt sich die Unentscheidbarkeit von K auf H.

Korollar 136. H $\notin$ REC.

Definition 137. *Das* ***Halteproblem bei leerem Band*** *ist die Sprache*

$$\mathrm{H}_0 = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l} \text{die DTM } M_w \text{ hält} \\ \text{bei Eingabe } \varepsilon \end{array} \right\}$$

χ_{H_0}	x_1 $(= \varepsilon)$
w_1	1
w_2	1
w_3	0
$\vdots$	$\vdots$

Satz 138. H_0 *ist* RE*-vollständig.*

Beweis. Da die Funktion $w \mapsto w\#\varepsilon$ die Sprache H_0 auf H reduziert und da $H \in$ RE ist, folgt $H_0 \in$ RE.

Für den Beweis, dass H_0 RE-hart ist sei $A \in$ RE beliebig und sei w die Kodierung einer DTM, die $\hat{\chi}_A$ berechnet. Um A auf H_0 zu reduzieren, transformieren wir $x \in \{0,1\}^*$ auf die Kodierung einer DTM M_{w_x}, die zunächst ihre Eingabe durch x ersetzt und dann $M_w(x)$ simuliert. Dann gilt

$$x \in A \Leftrightarrow w_x \in H_0$$

und somit $A \leq H_0$ mittels der Reduktionsfunktion $x \mapsto w_x$. ■

Insbesondere folgt also $K \leq \mathrm{H}_0$, d.h. H_0 ist unentscheidbar.

5.2 Der Satz von Rice

Frage. *Kann man einer beliebig vorgegebenen TM ansehen, ob die von ihr berechnete Funktion (bzw. die von ihr akzeptierte Sprache) eine gewisse Eigenschaft hat? Kann man beispielsweise entscheiden, ob eine gegebene DTM eine totale Funktion berechnet?*

Antwort. *Nein (es sei denn, die fragliche Eigenschaft ist trivial, d.h. keine oder jede berechenbare Funktion hat sie).*

Definition 139.

*a) Eine Eigenschaft $\mathcal{F}$ von Funktionen heißt **trivial**, wenn $\mathcal{F}$ entweder alle oder keine partielle berechenbare Funktionen auf $\{0,1,\#\}^*$ enthält.*

b) Zu $\mathcal{F}$ definieren wir die Sprache

$$L_{\mathcal{F}} = \left\{ w \in \{0,1\}^* \;\middle|\; M_w \text{ berechnet eine Funktion in } \mathcal{F} \right\}.$$

$\mathcal{F}$ ist also nicht trivial, wenn $L_{\mathcal{F}} \neq \emptyset$ und $L_{\mathcal{F}} \neq \{0,1\}^*$ ist. Der Satz von Rice besagt, dass $L_{\mathcal{F}}$ in diesem Fall unentscheidbar ist:

$$\forall \mathcal{F} : L_{\mathcal{F}} \neq \emptyset \wedge L_{\mathcal{F}} \neq \{0,1\}^* \Rightarrow L_{\mathcal{F}} \notin \mathsf{REC}.$$

Satz 140 (Satz von Rice)**.**
Für jede nicht triviale Eigenschaft $\mathcal{F}$ ist $L_{\mathcal{F}}$ unentscheidbar.

Beweis. Wir reduzieren H_0 (bzw. $\overline{\mathrm{H}_0}$) auf $L_{\mathcal{F}}$. Die Idee besteht darin, für eine gegebene DTM M_w eine DTM $M_{w'}$ zu konstruieren mit

$$w \in \mathrm{H}_0 \Leftrightarrow M_{w'} \text{ berechnet eine Funktion in } \mathcal{F}.$$

Hierzu lassen wir $M_{w'}$ bei Eingabe x zunächst einmal die DTM M_w bei Eingabe ε simulieren. Falls $w \notin \mathrm{H}_0$ ist, berechnet $M_{w'}$ also die überall undefinierte Funktion u mit $u(x) = \uparrow$ für alle $x \in \Sigma^*$.

Für das Folgende nehmen wir an, dass $u \notin \mathcal{F}$ ist. Andernfalls zeigen wir für die komplementäre Eigenschaft $\mathcal{F}' = \overline{\mathcal{F}}$, dass $L_{\mathcal{F}'}$ unentscheidbar ist. Dies impliziert, dass auch $\overline{L}_{\mathcal{F}'} = \overline{L}_{\overline{\mathcal{F}}} = L_{\mathcal{F}}$ unentscheidbar ist.

Damit die Reduktion gelingt, müssen wir also nur dafür sorgen, dass $M_{w'}$ im Fall $w \in H_0$ eine Funktion $f \in \mathcal{F}$ berechnet.

Da $\mathcal{F}$ nicht trivial ist, gibt es eine DTM M_f, die eine partielle Funktion $f \in \mathcal{F}$ berechnet. Betrachte die Reduktionsfunktion $h : \{0,1\}^* \to \{0,1\}^*$ mit

$$h(w) = w', \text{ wobei } w' \text{ die Kodierung einer DTM ist, die bei Eingabe } x \text{ zunächst die DTM } M_w(\varepsilon) \text{ simuliert und im Fall, dass } M_w \text{ hält, mit der Simulation von } M_f(x) \text{ fortfährt.}$$

Dann ist $h : w \mapsto w'$ eine totale berechenbare Funktion und es gilt

$$w \in H_0 \Rightarrow M_{w'} \text{ berechnet } f \Rightarrow w' \in L_{\mathcal{F}},$$
$$w \notin H_0 \Rightarrow M_{w'} \text{ berechnet } u \Rightarrow w' \notin L_{\mathcal{F}}.$$

Dies zeigt, dass h das Problem H_0 auf $L_{\mathcal{F}}$ reduziert, und da H_0 unentscheidbar ist, muss auch $L_{\mathcal{F}}$ unentscheidbar sein. ■

Beispiel 141. *Die Sprache*

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

ist unentscheidbar. Dies folgt aus dem Satz von Rice, da $L = L_{\mathcal{F}}$ für folgende nicht triviale Eigenschaft ist:

$$\mathcal{F} = \{f \in \mathsf{FREC}_p \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}.$$

Es gibt nämlich partielle berechenbare Funktionen mit dieser Eigenschaft wie z.B. die Funktion $f : \{0,1,\#\}^* \to \{0\}^* \cup \{\uparrow\}$ mit

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

als auch ohne diese Eigenschaft (wie etwa die konstante Funktion $g(x) = 0$). ◁

In den Übungen wird folgende Variante des Satzes von Rice bewiesen, wonach wir einer gegebenen TM nicht ansehen können, ob die von ihr akzeptierte Sprache eine gewisse Eigenschaft hat oder nicht.

Satz 142. *Sei $\mathcal{S}$ eine Sprachklasse, so dass semi-entscheidbare Sprachen $A, B \subseteq \{0,1,\#\}^*$ existieren mit $A \in \mathcal{S}$ und $B \notin \mathcal{S}$. Dann ist die Sprache*

$$L_{\mathcal{S}} = \{w \in \{0,1\}^* \mid L(M_w) \in \mathcal{S}\}$$

unentscheidbar.

5.3 Das Postsche Korrespondenzproblem

Definition 143. *Sei Σ ein beliebiges Alphabet mit $\# \notin \Sigma$. Das **Postsche Korrespondenzproblem** über Σ (kurz **PCP**$_\Sigma$) ist wie folgt definiert.*

Gegeben: *k Paare $(x_1, y_1), \ldots, (x_k, y_k)$ von Wörtern über Σ.*

Gefragt: *Gibt es eine Folge $\alpha = (i_1, \ldots, i_n)$, $n \geq 1$, von Indizes $i_j \in \{1, \ldots, k\}$ mit $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$?*

*Das **modifizierte PCP über** Σ (kurz **MPCP**$_\Sigma$) fragt nach einer Lösung $\alpha = (i_1, \ldots, i_n)$ mit $i_1 = 1$.*

Wir notieren eine PCP-Instanz meist in Form einer Matrix $\binom{x_1 \cdots x_k}{y_1 \cdots y_k}$ und kodieren sie durch das Wort $x_1\#y_1\#\cdots\#x_k\#y_k$.

Beispiel 144. *Die Instanz* $I = \binom{x_1\ x_2\ x_3}{y_1\ y_2\ y_3} = \binom{a\ \ ab\ caa}{aca\ bc\ aa}$ *besitzt wegen*

$$\begin{aligned} x_1x_3x_2x_3 &= acaaabcaa \\ y_1y_3y_2y_3 &= acaaabcaa \end{aligned}$$

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

Lemma 145. *Für jedes Alphabet Σ gilt* $\mathrm{PCP}_\Sigma \leq \mathrm{PCP}_{\{a,b\}}$.

Beweis. Sei $\Sigma = \{a_1, \ldots, a_m\}$. Für ein Zeichen $a_i \in \Sigma$ sei $\hat{a}_i = 01^{i-1}$ 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 $\mathrm{PCP}_\Sigma \leq \mathrm{PCP}_{\{a,b\}}$ mittels der Reduktionsfunktion

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

■

f reduziert z.B. die $\mathrm{PCP}_{\{0,1,2\}}$-Instanz $I = \binom{0\ \ 01\ 200}{020\ 12\ 00}$ auf die äquivalente $\mathrm{PCP}_{\{a,b\}}$-Instanz $f(I) = \binom{a\ \ aab\ abbaa}{aabba\ ababb\ aa}$.

Im Folgenden lassen wir im Fall $\Sigma = \{a,b\}$ den Index weg und schreiben einfach PCP (bzw. MPCP).

Satz 146. $\mathrm{MPCP} \leq \mathrm{PCP}$.

Beweis. Wir zeigen $\mathrm{MPCP} \leq \mathrm{PCP}_\Sigma$ für $\Sigma = \{a, b, \langle, |, \rangle\}$. Für ein Wort $w = w_1 \cdots w_n$ sei

$\overleftarrow{w}$	$\overleftarrow{w}$	$\overrightarrow{w}$	$\overrightarrow{w}$
$\langle w_1 \vert \cdots \vert w_n \vert$	$\langle w_1 \vert \cdots \vert w_n$	$\vert w_1 \vert \cdots \vert w_n$	$w_1 \vert \cdots \vert w_n \vert$

Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ:

$$f : \begin{pmatrix} x_1 \cdots x_k \\ y_1 \cdots y_k \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1}\ \overrightarrow{x_1} \cdots \overrightarrow{x_k}\ \rangle \\ \overleftarrow{y_1}\ \overrightarrow{y_1} \cdots \overrightarrow{y_k}\ \vert\rangle \end{pmatrix}.$$

Beispielsweise ist

$$f\begin{pmatrix} aa\ \ b\ \ bab\ bb \\ aab\ bb\ \ a\ \ \ b \end{pmatrix} = \begin{pmatrix} \langle a\vert a\vert\ \ a\vert a\vert\ \ b\vert\ \ b\vert a\vert b\vert\ \ b\vert b\vert\ \rangle \\ \langle a\vert a\vert b\ \ \vert a\vert a\vert b\ \ \vert b\vert b\ \ \ \vert a\ \ \ \ \vert b\ \ \vert\rangle \end{pmatrix}.$$

Da jede MPCP-Lösung $\alpha = (1, i_2, \ldots, i_n)$ für I auf eine PCP-Lösung $\alpha' = (1, i_2 + 1, \ldots, i_n + 1, k + 2)$ für $f(I)$ führt, folgt

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

Für die umgekehrte Implikation sei $\alpha' = (i_1, \ldots, i_n)$ eine PCP-Lösung für

$$f(I) = \begin{pmatrix} \overleftarrow{x_1}\ \overrightarrow{x_1} \cdots \overrightarrow{x_k}\ \rangle \\ \overleftarrow{y_1}\ \overrightarrow{y_1} \cdots \overrightarrow{y_k}\ \vert\rangle \end{pmatrix}.$$

Dann muss $i_1 = 1$ sein, da $(\overleftarrow{x_1}, \overleftarrow{y_1})$ das einzige Paar in $f(I)$ ist, bei dem beide Komponenten mit demselben Buchstaben anfangen. Zudem muss $i_n = k+2$ sein, da nur das Paar $(\rangle, |\rangle)$ mit demselben Buchstaben aufhört. Wählen wir α' von minimaler Länge, so ist $i_j \in \{2, \ldots, k+1\}$ für $j = 2, \ldots, n-1$. Dann ist aber

$$\alpha = (i_1, i_2 - 1, \ldots, i_{n-1} - 1)$$

eine MPCP-Lösung für I. ■

Satz 147. PCP *ist* RE*-vollständig und damit unentscheidbar.*

Beweis. Es ist leicht zu sehen, dass PCP $\in$ RE ist. Um zu zeigen, dass PCP RE-hart ist, sei A eine beliebige Sprache in RE und sei $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$ eine 1-DTM mit $L(M) = A$. Wir zeigen $A \leq \mathrm{MPCP}_{\Sigma'}$ für $\Sigma' = \Gamma \cup Z \cup \{\langle, |, \rangle\}$. Wegen $\mathrm{MPCP}_{\Sigma'} \leq \mathrm{PCP}$ folgt hieraus $A \leq \mathrm{PCP}$.

> *Idee:* Transformiere eine Eingabe $w \in \Sigma^*$ in eine Instanz $f(w) = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$, so dass $\alpha = (i_1, \ldots, i_n)$ genau dann eine MPCP-Lösung für $f(w)$ ist, wenn das zugehörige *Lösungswort* $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$ eine akzeptierende Rechnung von $M(w)$ kodiert. Dann gilt
>
> $$x \in A \Leftrightarrow f(w) \in \mathrm{MPCP}_{\Sigma'}.$$

Um dies zu erreichen, bilden wir $f(w)$ aus folgenden Wortpaaren:

1. $(\langle, \langle \mid z_0 w)$, „Startregel“
2. für alle $a \in \Gamma \cup \{|\}$: (a, a), „Kopierregeln“
3. für alle $a, a', b \in \Gamma$, $z, z' \in Z$: „Überführungsregeln“

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

4. für alle $e \in E$, $a \in \Gamma$: (ae, e), (ea, e), „Löschregeln“
5. sowie für alle $e \in E$: $(e\,|\rangle, \rangle)$. „Abschlussregeln“

Ist nun

$$K_0 = z_0 w \vdash K_1 \vdash \cdots \vdash K_t = uev$$

eine akzeptierende Rechnung von $M(w)$ mit $e \in E$, so lässt sich aus dieser leicht eine MPCP-Lösung α mit dem Lösungswort

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

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

Zudem ist leicht zu sehen, dass auch umgekehrt jedes Lösungswort $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$ eine akzeptierende Rechnung von M bei Eingabe w beinhaltet. Somit gilt

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

und da f offensichtlich berechenbar ist, folgt $A \leq \mathrm{MPCP}_{\Sigma'}$. ■

Beispiel 148. *Betrachte die 1-DTM* $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ *mit* $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, A, B, \sqcup\}$, $E = \{q_4\}$ *und den Anweisungen*

$$\begin{array}{llll}
\delta\colon & q_0 a \to q_1 AR \quad (1) & q_1 a \to q_1 aR \quad (2) & q_1 B \to q_1 BR \quad (3)\\
& q_1 b \to q_2 BL \quad (4) & q_2 a \to q_2 aL \quad (5) & q_2 B \to q_2 BL \quad (6)\\
& q_2 A \to q_0 AR \quad (7) & q_0 B \to q_3 BR \quad (8) & q_3 B \to q_3 BR \quad (9)\\
& q_3 \sqcup \to q_4 \sqcup N \quad (10) & &
\end{array}$$

Dann enthält die MPCP-Instanz $f(ab)$ für $u \in \Gamma$ die Wortpaare

Startregel	*Kopierregeln*	*Löschregeln*	*Abschlussregel*
$(\langle, \langle \mid z_0ab)$	(u,u), $(\mid,\mid)$	(q_4u, q_4), (uq_4, q_4)	$(q_4 \mid \rangle, \rangle)$

sowie folgende Überführungsregeln:

Anweisung		*zugehörige Überführungsregeln*
$q_0a \to q_1AR$	*(1)*	(q_0a, Aq_1)
$q_1a \to q_1aR$	*(2)*	(q_1a, aq_1)
$q_1B \to q_1BR$	*(3)*	(q_1B, Bq_1)
$q_1b \to q_2BL$	*(4)*	(uq_1b, q_2uB), $(\mid q_1b, \mid q_2\sqcup B)$
$q_2a \to q_2aL$	*(5)*	(uq_2a, q_2ua), $(\mid q_2a, \mid q_2\sqcup a)$
$q_2B \to q_2BL$	*(6)*	(uq_2B, q_2uB), $(\mid q_2B, \mid q_2\sqcup B)$
$q_2A \to q_0AR$	*(7)*	(q_2A, Aq_0)
$q_0B \to q_3BR$	*(8)*	(q_0B, Bq_3)
$q_3B \to q_3BR$	*(9)*	(q_3B, Bq_3)
$q_3\sqcup \to q_4\sqcup N$	*(10)*	$(q_3\sqcup, q_4\sqcup)$, $(q_3\mid, q_4\sqcup\mid)$

Der akzeptierenden Rechnung

$$q_0ab \underset{(1)}{\vdash} Aq_1b \underset{(4)}{\vdash} q_2AB \underset{(7)}{\vdash} Aq_0B \underset{(8)}{\vdash} ABq_3\sqcup \underset{(10)}{\vdash} ABq_4\sqcup$$

von $M(ab)$ entspricht dann das MPCP-Lösungswort

$$\langle \mid q_0ab \mid Aq_1b \mid q_2AB \mid Aq_0B \mid ABq_3 \mid ABq_4\sqcup \mid Aq_4\sqcup \mid q_4\sqcup \mid q_4 \mid \rangle$$
$$\langle \mid q_0ab \mid Aq_1b \mid q_2AB \mid Aq_0B \mid ABq_3 \mid ABq_4\sqcup \mid Aq_4\sqcup \mid q_4\sqcup \mid q_4 \mid \rangle$$

◁

5.4 Weitere Unentscheidbarkeitsresultate

In diesem Abschnitt leiten wir aus der Unentscheidbarkeit des Postschen Korrespondenzproblems eine Reihe von weiteren Unentscheidbarkeitsresultaten her. Wir zeigen zuerst, dass das Schnittproblem für kontextfreie Grammatiken unentscheidbar ist.

Schnittproblem für kontextfreie Grammatiken

Gegeben: Zwei kontextfreie Grammatiken G_1 und G_2.

Gefragt: Ist $L(G_1) \cap L(G_2) \neq \emptyset$?

Satz 149. *Das Schnittproblem für kontextfreie Grammatiken ist* RE*-vollständig.*

Beweis. Es ist leicht zu sehen, dass das Problem semi-entscheidbar ist. Wir reduzieren eine PCP-Instanz $I = \binom{x_1 \cdots x_k}{y_1 \cdots y_k}$ auf ein Grammatikpaar (G_1, G_2), so dass gilt:

$$I \in \text{PCP} \Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset.$$

Für $i = 1, 2$ sei $G_i = (\{S\}, \{a, b, 1, \ldots, k\}, P_i, S)$ mit den Regeln

$$P_1\colon S \to 1Sx_1, \ldots, kSx_k, 1x_1, \ldots, kx_k,$$
$$P_2\colon S \to 1Sy_1, \ldots, kSy_k, 1y_1, \ldots, ky_k.$$

Dann gilt

$$L(G_1) = \{i_n \cdots i_1 x_{i_1} \cdots x_{i_n} \mid 1 \leq n,\ 1 \leq i_1, \ldots, i_n \leq k\}$$

und

$$L(G_2) = \{i_n \cdots i_1 y_{i_1} \cdots y_{i_n} \mid 1 \leq n,\ 1 \leq i_1, \ldots, i_n \leq k\}.$$

Somit ist $L(G_1) \cap L(G_2)$ die Sprache

$$\{i_n \cdots i_1 x_{i_1} \cdots x_{i_n} \mid 1 \leq n,\ x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}\}.$$

Folglich ist $\alpha = (i_1, \ldots, i_n)$ genau dann eine Lösung für I, wenn das Wort

$$i_n \cdots i_1 x_{i_1} \cdots x_{i_n} \in L(M_1) \cap L(M_2)$$

ist. Also vermittelt $f : I \mapsto (G_1, G_2)$ eine Reduktion von PCP auf das Schnittproblem für CFL. ■

Beispiel 150. *Die PCP-Instanz*

$$I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & aab & abbaa \\ aabba & ababb & aa \end{pmatrix}$$

wird auf das Grammatikpaar (G_1, G_2) mit folgenden Regeln reduziert:

$$P_1\text{: } S \to 1Sa,\ 2Saab,\ 3Sabbaa,\ 1a,\ 2aab,\ 3abbaa,$$
$$P_2\text{: } S \to 1Saabba,\ 2Sababb,\ 3Saa,\ 1aabba,\ 2ababb,\ 3aa.$$

Der PCP-Lösung $\alpha = (1, 3, 2, 3)$ entspricht dann das Wort

$$3231x_1x_3x_2x_3 = 3231aabbaaaababbaa = 3231aabbaaaababbaa = 3231y_1y_3y_2y_3$$

im Schnitt $L(G_1) \cap L(G_2)$. ◁

Des weiteren erhalten wir die Unentscheidbarkeit des Schnitt- und des Inklusionsproblems für DPDAs.

Inklusionsproblem für DPDAs

Gegeben: Zwei DPDAs M_1 und M_2.

Gefragt: Ist $L(M_1) \subseteq L(M_2)$?

Korollar 151.

(i) Das Schnittproblem für DPDAs ist RE*-vollständig.*

(ii) Das Inklusionsproblem für DPDAs ist co-RE*-vollständig.*

Beweis.

(i) Es ist leicht, die kontextfreien Grammatiken G_1 und G_2 in obigem Beweis in äquivalente DPDAs M_1 und M_2 zu verwandeln (siehe Übungen).

(ii) Wir reduzieren das Schnittproblem für DPDAs auf das Inklusionsproblem für DPDAs. Es gilt

$$L_1 \cap L_2 = \emptyset \Leftrightarrow L_1 \subseteq \overline{L_2}.$$

Daher reduziert die Funktion $f : (M_1, M_2) \mapsto (M_1, \overline{M_2})$ das Komplement des Schnittproblems für DPDAs auf das Inklusionsproblem für DPDAs.

■

Für den obigen Beweis ist es wichtig, dass die Funktion $M \mapsto \overline{M}$, also die Transformation eines gegebenen DPDA M in den zugehörigen Komplementärautomaten $\overline{M}$, berechenbar ist (dies wurde in den Übungen gezeigt). Schließlich ergeben sich für CFL noch folgende Unentscheidbarkeitsresultate.

Korollar 152. *Für kontextfreie Grammatiken sind folgende Fragestellungen unentscheidbar:*

(i) Ist $L(G) = \Sigma^*$? *(Ausschöpfungsproblem)*

(ii) Ist $L(G_1) = L(G_2)$? *(Äquivalenzproblem)*

(iii) Ist G mehrdeutig? *(Mehrdeutigkeitsproblem)*

Beweis.

(i) Wir zeigen, dass das Ausschöpfungsproblem für kontextfreie Grammatiken co-RE-vollständig ist, indem wir das Komplement des Schnittproblems für DPDAs darauf reduzieren. Es gilt

$$L_1 \cap L_2 = \emptyset \Leftrightarrow \overline{L_1} \cup \overline{L_2} = \Sigma^*.$$

Daher vermittelt die Funktion $f : (M_1, M_2) \mapsto G$, wobei G eine kontextfreie Grammatik mit

$$L(G) = \overline{L(M_1)} \cup \overline{L(M_2)}$$

ist, die gewünschte Reduktion.

(ii) Wir zeigen, dass das Äquivalenzproblem für kontextfreie Grammatiken ebenfalls co-RE-vollständig ist, indem wir das Ausschöpfungsproblem für kontextfreie Grammatiken darauf reduzieren. Dies leistet beispielsweise die Funktion

$$f : G \mapsto (G, G_{all}),$$

wobei G_{all} eine kontextfreie Grammatik mit $L(G_{all}) = \Sigma^*$ ist.

(iii) Schließlich zeigen wir, dass das Mehrdeutigkeitsproblem RE-vollständig ist, indem wir PCP darauf reduzieren. Betrachte die Funktion $f : \binom{x_1 \cdots x_k}{y_1 \ldots y_k} \mapsto G$ mit

$$G = (\{S, A, B\}, \{a, b, 1, \ldots, k\}, P_1 \cup P_2 \cup \{S \to A, S \to B\}, S)$$

und den Regeln

$$P_1\colon A \to 1Ax_1, \ldots, kAx_k, 1x_1, \ldots, kx_k,$$
$$P_2\colon B \to 1By_1, \ldots, kBy_k, 1y_1, \ldots, ky_k.$$

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

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

Wie wir im Beweis der Unentscheidbarkeit des Schnittproblems für CFL gesehen haben, ist dies genau dann der Fall, wenn die PCP-Instanz $I = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ eine PCP-Lösung hat.

■

Als weitere Folgerung erhalten wir die Unentscheidbarkeit des Leerheitsproblems für DLBAs.

Leerheitsproblem für DLBAs

Gegeben: Ein DLBA M.

Gefragt: Ist $L(M) = \emptyset$?

Satz 153. *Das Leerheitsproblem für DLBAs ist* co-RE*-vollständig.*

Beweis. Wir reduzieren das Ausschöpfungsproblem für CFL auf das Leerheitsproblem für DLBAs. Eine kontextfreie Grammatik G lässt sich wie folgt in einen DLBA M mit $L(M) = \overline{L(G)}$ überführen (siehe Übungen):

Bestimme zunächst einen DLBA M mit $L(M) = L(G)$.
Konstruiere daraus einen DLBA $\overline{M}$ mit $L(\overline{M}) = \overline{L(M)}$.

Dann gilt $L(G) = \Sigma^* \Leftrightarrow L(M) = \emptyset$, d.h. die Funktion $f : G \mapsto \overline{M}$ berechnet die gewünschte Reduktion. ■

Dagegen ist es nicht schwer, für eine kontextfreie Grammatik G zu entscheiden, ob mindestens ein Wort in G ableitbar ist. Ebenso ist es möglich, für eine kontextsensitive Grammatik G und ein Wort x zu entscheiden, ob x in G ableitbar ist.

Satz 154.

(i) Das Leerheitsproblem für kfr. Grammatiken ist entscheidbar.

(ii) Das Wortproblem für kontextsensitive Grammatiken ist entscheidbar.

Die folgende Tabelle gibt an, welche Probleme für Sprachen in den verschiedenen Stufen der Chomsky-Hierarchie entscheidbar sind.

	Wort-problem $x \in L$?	Leerheits-problem $L = \emptyset$?	Aus-schöpfung $L = \Sigma^*$?	Äquivalenz-problem $L_1 = L_2$?	Inklusions-problem $L_1 \subseteq L_2$	Schnitt-problem $L_1 \cap L_2 \neq \emptyset$?
REG	ja	ja	ja	ja	ja	ja
DCFL	ja	ja	ja	ja[a]	nein	nein
CFL	ja	ja	nein	nein	nein	nein
DCSL	ja	nein	nein	nein	nein	nein
CSL	ja	nein	nein	nein	nein	nein
RE	nein	nein	nein	nein	nein	nein

[a] Bewiesen in 1997 von Géraud Sénizergues (Univ. Bordeaux).

5.5 Die arithmetische Hierarchie

Mit Hilfe der arithmetischen Hierarchie lassen sich unentscheidbare Problemen entsprechend dem Grad ihrer Unentscheidbarkeit einordnen.

Definition 155. *Sei $A \subseteq \Sigma^*$ eine Sprache und $\mathcal{K}$ eine Sprachklasse. Dann ist*

a) $\exists A = \{x \in \Sigma^* \mid \exists y \in \{0,1\}^* : x\#y \in A\}$ *und*

b) $\exists \mathcal{K} = \mathcal{K} \cup \{\exists A \mid A \in \mathcal{K}\}$.

c) Weiter sei $\Sigma_0 = \Pi_0 = \mathsf{REC}$ *und* $\Sigma_i = \exists \Pi_{i-1}$ *sowie* $\Pi_i = \mathsf{co}\text{-}\Sigma_i$ *für* $i \geq 1$.

d) Die Sprachklassen Σ_i, Π_i, $i \geq 0$, *bilden die Stufen der* ***arithmetischen Hierarchie****.*

Proposition 156.

(i) $\mathsf{RE} = \Sigma_1$.

(ii) $\Sigma_i \cup \Pi_i \subseteq \Sigma_{i+1} \cap \Pi_{i+1}$ *für* $i \geq 0$.

Beweis.

(i) Die Inklusion $\exists\mathsf{REC} \subseteq \mathsf{RE}$ ist klar, da eine DTM im Fall $B \in \mathsf{REC}$ die Sprache $A = \exists B$ akzeptiert, indem sie bei Eingabe x systematisch nach einem String y mit $x\#y \in B$ sucht.

Für den Beweis von $\mathsf{RE} \subseteq \exists\mathsf{REC}$ sei $A \subseteq \Sigma^*$ eine beliebige Sprache in RE und M sei eine DTM mit $L(M) = A$. Dann gilt $A = \exists B$ für die Sprache

$$B = \{x\#y \mid y \text{ kodiert eine akz. Rechnung von } M(x)\}.$$

Da B offensichtlich entscheidbar ist, folgt $A \in \exists\mathsf{REC}$.

(ii) Die Inklusion $\Pi_i \subseteq \Sigma_{i+1}$ ist klar, da $\Pi_i \subseteq \exists\Pi_i = \Sigma_{i+1}$ ist. Wegen $\mathcal{K} \subseteq \mathcal{K}' \Leftrightarrow \mathsf{co}\text{-}\mathcal{K} \subseteq \mathsf{co}\text{-}\mathcal{K}'$ und $\Sigma_j = \mathsf{co}\text{-}\Pi_j$ für $j \geq 0$ folgt hieraus auch $\Sigma_i \subseteq \Pi_{i+1}$.

Die Inklusionen $\Sigma_{i-1} \subseteq \Sigma_i$ lassen sich induktiv wie folgt zeigen.

$i = 1$**:** $\Sigma_0 = \mathsf{REC} \subseteq \mathsf{RE} = \Sigma_1$.

$i \rightsquigarrow i+1$**:** Nach IV gilt $\Sigma_{i-1} \subseteq \Sigma_i$. Folglich ist $\Pi_{i-1} \subseteq \Pi_i$, was wegen der Monotonie des $\exists$-Operators (d.h. $\mathcal{K} \subseteq \mathcal{K}' \Rightarrow \exists\mathcal{K} \subseteq \exists\mathcal{K}'$) wiederum $\Sigma_i = \exists\Pi_{i-1} \subseteq \exists\Pi_i = \Sigma_{i+1}$ impliziert.

Durch Übergang zu den co-Klassen erhalten wir auch $\Pi_{i-1} \subseteq \Pi_i$. ■

Mittels Diagonalisierung kann man zeigen, dass die Inklusionen $\Sigma_{i-1} \subseteq \Sigma_i$ echt sind, und dass $\Sigma_i \neq \Pi_i$ für alle $i \geq 1$ gilt.

Die betrachteten Entscheidungsprobleme für die verschiedenen Stufen der Chomsky-Hierarchie lassen sich nun wie folgt in die arithmetische Hierarchie einordnen.

	Wort-problem $x \in L$?	Leerheits-problem $L = \emptyset$?	Aus-schöpfung $L = \Sigma^*$?	Äquivalenz-problem $L_1 = L_2$?	Inklusions-problem $L_1 \subseteq L_2$	Schnitt-problem $L_1 \cap L_2 \neq \emptyset$?
REG	REC	REC	REC	REC	REC	REC
DCFL	REC	REC	REC	REC	co -RE	RE
CFL	REC	REC	co -RE	co -RE	co -RE	RE
DCSL	REC	co -RE	co -RE	co -RE	co -RE	RE
CSL	REC	co -RE	co -RE	co -RE	co -RE	RE
RE	RE	co -RE	Π_2	Π_2	Π_2	RE

Tatsächlich sind alle betrachteten Entscheidungsprobleme sogar vollständig für die angegebenen Sprachklassen.

6 Komplexitätsklassen

6.1 Zeitkomplexität

Die Laufzeit $time_M(x)$ einer NTM M bei Eingabe x ist die maximale Anzahl an Rechenschritten, die $M(x)$ ausführt.

Definition 157.

a) Die ***Laufzeit*** *einer* NTM M *bei Eingabe* x *ist definiert als*

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

wobei $\max \mathbb{N} = \infty$ *ist.*

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

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

Die Zeitschranke $t(n)$ beschränkt also die Laufzeit bei allen Eingaben der Länge n (worst-case Komplexität).

Wir fassen alle Sprachen und Funktionen, die in einer vorgegebenen Zeitschranke $t(n)$ entscheidbar bzw. berechenbar sind, in folgenden Komplexitätsklassen zusammen.

Definition 158.

a) Die in deterministischer Zeit $t(n)$ *entscheidbaren Sprachen bilden die Sprachklasse*

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

b) Die in nichtdeterministischer Zeit $t(n)$ entscheidbaren Sprachen bilden die Sprachklasse

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

c) Die in deterministischer Zeit $t(n)$ berechenbaren Funktionen bilden die Funktionenklasse

$$\mathbf{FTIME}(t(n)) = \{f \mid \exists\ t(n)\text{-zeitb. DTM } M, \text{ die } f \text{ berechnet}\}.$$

Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

$$\begin{aligned}
\mathsf{LINTIME} &= \bigcup_{c\geq 1} \mathsf{DTIME}(cn+c) && \text{„Linearzeit“}\\
\mathsf{P} &= \bigcup_{c\geq 1} \mathsf{DTIME}(n^c+c) && \text{„Polynomialzeit“}\\
\mathsf{E} &= \bigcup_{c\geq 1} \mathsf{DTIME}(2^{cn+c}) && \text{„Lineare Exponentialzeit“}\\
\mathsf{EXP} &= \bigcup_{c\geq 1} \mathsf{DTIME}(2^{n^c+c}) && \text{„Exponentialzeit“}
\end{aligned}$$

Die nichtdeterministischen Klassen $\mathsf{NLINTIME}, \mathsf{NP}, \mathsf{NE}, \mathsf{NEXP}$ und die Funktionenklassen $\mathsf{FP}, \mathsf{FE}, \mathsf{FEXP}$ sind analog definiert.

Für eine Funktionenklasse $\mathcal{F}$ sei $\mathsf{DTIME}(\mathcal{F}) = \bigcup_{t\in\mathcal{F}} \mathsf{DTIME}(t(n))$ (die Klassen $\mathsf{NTIME}(\mathcal{F})$ und $\mathsf{FTIME}(\mathcal{F})$ seien analog definiert).

Asymptotische Laufzeit und Landau-Notation

Definition 159. *Seien f und g Funktionen von $\mathbb{N}$ nach $\mathbb{R}^+$. Wir schreiben $f(n) = O(g(n))$, falls es Zahlen n_0 und c gibt mit*

$$\forall n \geq n_0 : f(n) \leq c \cdot g(n).$$

Die Bedeutung der Aussage $f(n) = O(g(n))$ ist, dass f „nicht wesentlich schneller“ als g wächst. Formal bezeichnet der Term $O(g(n))$ die Klasse aller Funktionen f, die obige Bedingung erfüllen. Die Gleichung $f(n) = O(g(n))$ drückt also in Wahrheit eine Element-Beziehung $f \in O(g(n))$ aus. O-Terme können auch auf der linken Seite vorkommen. In diesem Fall wird eine Inklusionsbeziehung ausgedrückt. So steht $n^2 + O(n) = O(n^2)$ für die Aussage $\{n^2 + f \mid f \in O(n)\} \subseteq O(n^2)$.

Beispiel 160.

- $7\log(n) + n^3 = O(n^3)$ *ist richtig.*
- $7\log(n)n^3 = O(n^3)$ *ist falsch.*
- $2^{n+O(1)} = O(2^n)$ *ist richtig.*
- $2^{O(n)} = O(2^n)$ *ist falsch (siehe Übungen).* ◁

Mit der O-Notation lassen sich die Klassen $\mathsf{LINTIME}$, P, E und EXP durch $\mathsf{DTIME}(O(n))$, $\mathsf{DTIME}(n^{O(1)})$, $\mathsf{DTIME}(2^{O(n)})$ und $\mathsf{DTIME}(2^{n^{O(1)}})$ definieren.

6.2 Platzkomplexität

Als nächstes definieren wir den Platzverbrauch von NTMs. Intuitiv ist dies die Anzahl aller während einer Rechnung benutzten Bandfelder. Wollen wir auch sublinearen Platz sinnvoll definieren, so dürfen wir hierbei das Eingabeband offensichtlich nicht berücksichtigen. Um sicherzustellen, dass eine NTM M das Eingabeband nicht als Speicher benutzt, verlangen wir, dass M die Felder auf dem Eingabeband nicht verändert und sich nicht mehr als ein Feld von der Eingabe entfernt.

Definition 161. *Eine* NTM M *heißt* ***Offline*-NTM** *(oder* NTM *mit* ***Eingabeband****), falls für jede von M bei Eingabe x erreichbare Konfiguration*

$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$$

gilt, dass $u_1a_1v_1$ ein Teilwort von $\sqcup x\sqcup$ ist.

Definition 162. *Der **Platzverbrauch** einer Offline-*NTM *M bei Eingabe x ist definiert als*

$$space_M(x) = \max\left\{ s \geq 1 \;\middle|\; \begin{array}{l} \exists K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \\ \textit{mit } K_x \vdash^* K \textit{ und } s = \sum_{i=2}^{k} |u_i a_i v_i| \end{array} \right\}.$$

*Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $s(n)$**-platzbeschränkt**, falls für alle Eingaben x gilt:*

$$space_M(x) \leq s(|x|).$$

Wir fassen alle Sprachen, die in einer vorgegebenen Platzschranke $s(n)$ entscheidbar sind, in folgenden **Platzkomplexitätsklassen** zusammen.

Definition 163. *Die auf deterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse*

$$\mathbf{DSPACE}(s(n)) = \{L(M) \mid M \textit{ ist eine } s(n)\textit{-platzb. Offline-}\text{DTM}\}.$$

Die auf nichtdeterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse

$$\mathbf{NSPACE}(s(n)) = \{L(M) \mid M \textit{ ist eine } s(n)\textit{-platzb. Offline-}\text{NTM}\}.$$

Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$\mathbf{L}$	$= \mathsf{DSPACE}(O(\log n))$	„Logarithmischer Platz“
$\mathbf{LINSPACE}$	$= \mathsf{DSPACE}(O(n))$	„Linearer Platz“
$\mathbf{PSPACE}$	$= \mathsf{DSPACE}(n^{O(1)})$	„Polynomieller Platz“

Die nichtdeterministischen Klassen **NL**, **NLINSPACE** und **NPSPACE** sind analog definiert.

Zwischen den verschiedenen Zeit- und Platzklassen bestehen die folgenden elementaren Inklusionsbeziehungen. Für jede Funktion $s(n) \geq \log n$ gilt

$$\mathsf{DSPACE}(s) \subseteq \mathsf{NSPACE}(s) \subseteq \mathsf{DTIME}(2^{O(s)})$$

und für jede Funktion $t(n) \geq n + 2$ gilt

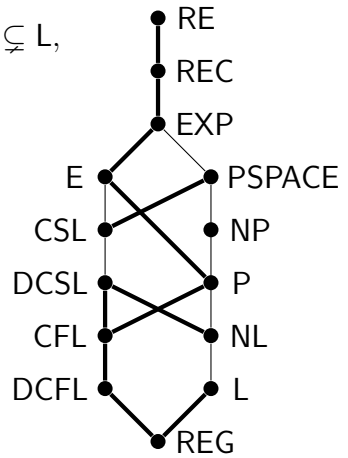
$$\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t).$$

Eine unmittelbare Konsequenz hiervon sind folgende Inklusionen:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{EXPSPACE}.$$

Die Klassen der Chomsky-Hierarchie lassen sich wie folgt einordnen (eine dicke Linie deutet an, dass die Inklusion als echt nachgewiesen werden konnte):

$$\begin{aligned} \mathsf{REG} &= \mathsf{DSPACE}(O(1)) = \mathsf{NSPACE}(O(1)) \subsetneq \mathsf{L}, \\ \mathsf{DCFL} &\subsetneq \mathsf{LINTIME} \cap \mathsf{CFL} \subsetneq \mathsf{LINTIME} \subsetneq \mathsf{P}, \\ \mathsf{CFL} &\subsetneq \mathsf{NLINTIME} \cap \mathsf{DTIME}(n^3) \subsetneq \mathsf{P}, \\ \mathsf{DCSL} &= \mathsf{LINSPACE} \subseteq \mathsf{CSL}, \\ \mathsf{CSL} &= \mathsf{NLINSPACE} \subseteq \mathsf{PSPACE} \cap \mathsf{E}, \\ \mathsf{REC} &= \bigcup \mathsf{DSPACE}(f(n)) \\ &= \bigcup \mathsf{NSPACE}(f(n)) \\ &= \bigcup \mathsf{DTIME}(f(n)) \\ &= \bigcup \mathsf{NTIME}(f(n)), \end{aligned}$$



wobei f alle berechenbaren (oder äquivalent: alle) Funktionen $f : \mathbb{N} \to \mathbb{N}$ durchläuft.

Die Klasse L ist nicht in CFL enthalten, da beispielsweise die Sprache $\{a^n b^n c^n \mid n \geq 0\}$ in logarithmischem Platz (und linearer Zeit) entscheidbar ist. Ob P in CSL enthalten ist, ist nicht bekannt. Auch nicht

ob DCSL $\subseteq$ P gilt. Man kann jedoch zeigen, dass CSL $\neq$ P $\neq$ DCSL ist. Ähnlich verhält es sich mit den Klassen E und PSPACE: Man kann zwar zeigen, dass sie verschieden sind, aber ob eine in der anderen enthalten ist, ist nicht bekannt.

7 NP-vollständige Probleme

Wie wir im letzten Kapitel gesehen haben (siehe Satz 127), sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden. Die Frage, wieviel Zeit eine DTM zur Simulation einer NTM benötigt, ist eines der wichtigsten offenen Probleme der Informatik. Wegen $\mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t) \subseteq \mathsf{NSPACE}(t) \subseteq \mathsf{DTIME}(2^{O(t)})$ erhöht sich die Laufzeit im schlimmsten Fall exponentiell. Insbesondere die Klasse NP enthält viele für die Praxis überaus wichtige Probleme, für die kein Polynomialzeitalgorithmus bekannt ist. Da jedoch nur Probleme in P als effizient lösbar gelten, hat das so genannte **P-NP-Problem**, also die Frage, ob alle NP-Probleme effizient lösbar sind, eine immense praktische Bedeutung. Die analoge Frage für den Platzverbrauch wurde bereits 1970 gelöst.

Satz 164 (Satz von Savitch, 1970)**.**
Jede $s(n)$-platzbeschränkte NTM kann von einer $s^2(n)$-platzbeschränkten DTM simuliert werden (ohne Beweis).

Definition 165. *Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen.*

*a) A ist auf B **in Polynomialzeit reduzierbar** ($A \leq^p B$), falls eine Funktion $f : \Sigma^* \to \Gamma^*$ in* FP *existiert mit*

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

*b) A heißt $\leq^p$**-hart** für eine Sprachklasse $\mathcal{C}$ (kurz: $\mathcal{C}$**-hart** oder $\mathcal{C}$**-schwer**), falls gilt:*

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

*c) Eine $\mathcal{C}$-harte Sprache, die zu $\mathcal{C}$ gehört, heißt $\mathcal{C}$**-vollständig**. Die Klasse aller* NP*-vollständigen Sprachen bezeichnen wir mit* **NPC**.

Aus $A \leq^p B$ folgt offenbar $A \leq B$ und wie die Relation $\leq$ ist auch $\leq^p$ reflexiv und transitiv (s. Übungen). In diesem Kapitel verlangen wir also von einer $\mathcal{C}$-vollständigen Sprache A, dass jede Sprache $L \in \mathcal{C}$ auf A in Polynomialzeit reduzierbar ist. Es ist leicht zu sehen, dass alle im letzten Kapitel als RE-vollständig nachgewiesenen Sprachen (wie z.B. $\mathrm{K}, \mathrm{H}, \mathrm{H}_0, \mathrm{PCP}$ etc.) sogar $\leq^p$-vollständig für RE sind.

Satz 166. *Die Klassen* P *und* NP *sind unter* $\leq^p$ *abgeschlossen.*

Beweis. Sei $B \in \mathsf{P}$ und gelte $A \leq^p B$ mittels einer Funktion $f \in \mathsf{FP}$. Seien M und T DTMs mit $L(M) = B$ und $T(x) = f(x)$. Weiter seien p und q polynomielle Zeitschranken für M und T. Betrachte die DTM M', die bei Eingabe x zuerst T simuliert, um $f(x)$ zu berechnen, und danach M bei Eingabe $f(x)$ simuliert. Dann gilt

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow f(x) \in L(M) \Leftrightarrow x \in L(M').$$

Also ist $L(M') = A$ und wegen

$$time_{M'}(x) \leq time_T(x) + time_M(f(x)) \leq q(|x|) + p(q(|x|))$$

ist M' polynomiell zeitbeschränkt und somit A in P. Den Abschluss von NP unter $\leq^p$ zeigt man vollkommen analog. ■

Satz 167.

(i) $A \leq^p B$ *und* A *ist* NP*-schwer* $\Rightarrow$ B *ist* NP*-schwer.*

(ii) $A \leq^p B$, A *ist* NP*-schwer und* $B \in \mathsf{NP}$ $\Rightarrow$ $B \in \mathsf{NPC}$.

(iii) $\mathsf{NPC} \cap \mathsf{P} \neq \emptyset$ $\Rightarrow$ $\mathsf{P} = \mathsf{NP}$.

Beweis.

(i) Sei $L \in \mathsf{NP}$ beliebig. Da A NP-schwer ist, folgt $L \leq^p A$. Da zudem $A \leq^p B$ gilt und $\leq^p$ transitiv ist, folgt $L \leq^p B$.

(ii) Klar, da mit *(i)* folgt, dass B NP-schwer und B nach Voraussetzung in NP ist.

(iii) Sei $A \in \mathsf{P}$ eine NP-vollständige Sprache und sei $L \in \mathsf{NP}$ beliebig. Dann folgt $L \leq^p A$ und da P unter $\leq^p$ abgeschlossen ist, folgt $L \in \mathsf{P}$. ■

Satz 168. *Folgende Sprache ist* NP*-vollständig:*

$$L = \left\{ w\#x\#0^m \;\middle|\; \begin{array}{l} w, x \in \{0,1\}^* \text{ und } M_w \text{ ist eine NTM,} \\ \text{die } x \text{ in} \leq m \text{ Schritten akzeptiert} \end{array} \right\}$$

Beweis. Folgende NTM M entscheidet L in NP:

> M simuliert bei Eingabe $w\#x\#0^m$ die NTM M_w bei Eingabe x für höchstens m Schritte. Falls M_w hierbei akzeptiert, akzeptiert M ebenfalls, andernfalls verwirft M.

Sei nun A eine beliebige NP-Sprache. Dann ist A in Polynomialzeit auf eine Sprache $B \subseteq \{0,1\}^*$ in NP reduzierbar (siehe Übungen). Sei M_w eine durch ein Polynom p zeitbeschränkte NTM für B. Dann reduziert folgende FP-Funktion f die Sprache B auf L:

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

■

7.1 Aussagenlogische Erfüllbarkeitsprobleme

Definition 169.

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

- *Jede Variable* x_i *ist eine boolesche Formel.*
- *Mit* G *und* H *sind auch die* ***Negation*** $\neg G$ *von* G *und die* ***Konjunktion*** $(G \wedge H)$ *von* G *und* H *boolesche Formeln.*

b) *Eine* ***Belegung*** *von $x_1, \ldots, x_n$ ist ein Wort $a = a_1 \ldots a_n \in \{0,1\}^n$. Der* ***Wert*** *$F(a)$ von F unter a ist induktiv über den Aufbau von F definiert:*

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

c) *Durch eine boolesche Formel F wird also eine n-**stellige boolesche Funktion** $F : \{0,1\}^n \to \{0,1\}$ definiert, die wir ebenfalls mit F bezeichnen.*

d) *F heißt* ***erfüllbar****, falls es eine Belegung a mit $F(a) = 1$ gibt.*

Beispiel 170 (Erfüllbarkeitstest mittels Wahrheitswerttabelle)**.** *Da die Formel $F = ((x_1 \vee x_2) \to (\neg x_2 \wedge x_3))$ für die Belegungen $a \in \{000, 001, 101\}$ den Wert $F(a) = 1$ animmt, ist sie erfüllbar:*

a	$(x_1 \vee x_2)$	$(\neg x_2 \wedge x_3)$	$((x_1 \vee x_2) \to (\neg x_2 \wedge x_3))$
000	0	0	1
001	0	1	1
010	1	0	0
011	1	0	0
100	1	0	0
101	1	1	1
110	1	0	0
111	1	0	0

◁

Notation. *Wir verwenden die* ***Disjunktion*** *$(G \vee H)$ und die* ***Implikation*** *$(G \to H)$ als Abkürzungen für die Formeln $\neg(\neg G \wedge \neg H)$ bzw. $(\neg G \vee H)$.*

Um Klammern zu sparen, vereinbaren wir folgende Präzedenzregeln:

- Der Junktor $\wedge$ bindet stärker als der Junktor $\vee$ und dieser wiederum stärker als der Junktor $\to$.
- Formeln der Form $(x_1 \circ (x_2 \circ (x_3 \circ \cdots \circ x_n) \cdots)))$, $\circ \in \{\wedge, \vee, \to\}$ kürzen wir durch $(x_1 \circ \cdots \circ x_n)$ ab.

Beispiel 171. *Die Formel*

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

nimmt unter einer Belegung $a = a_1 \cdots a_n$ genau dann den Wert 1 an, wenn $\sum_{i=1}^n a_i = 1$ ist. D.h. es gilt genau dann $G(a) = 1$, wenn genau eine Variable x_i mit dem Wert $a_i = 1$ belegt ist. Diese Formel wird im Beweis des nächsten Satzes benötigt. ◁

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 Erfüllbarkeitsproblems (engl. *satisfiability*) für boolesche Formeln hat also große praktische Bedeutung.

Aussagenlogisches Erfüllbarkeitsproblem (SAT):

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

Gefragt: Ist F erfüllbar?

Satz 172 (Cook, Karp, Levin)**.** SAT *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass SAT $\in$ NP ist, da eine NTM zunächst eine Belegung a für eine gegebene booleschen Formel F nichtdeterministisch raten und dann in Polynomialzeit testen kann, ob $F(a) = 1$ ist (*guess and verify* Strategie).

Es bleibt zu zeigen, dass SAT NP-hart ist. Sei L eine beliebige NP-Sprache und sei $M = (Z, \Sigma, \Gamma, \delta, q_0)$ eine durch ein Polynom p zeitbeschränkte k-NTM mit $L(M) = L$. Da sich eine $t(n)$-zeitbeschränkte k-NTM in Zeit $t^2(n)$ durch eine 1-NTM simulieren lässt, können wir

$k = 1$ annehmen. Unsere Aufgabe besteht nun darin, in Polynomialzeit zu einer gegebenen Eingabe $w = w_1 \cdots w_n$ eine Formel F_w zu konstruieren, die genau dann erfüllbar ist, wenn $w \in L$ ist,

$$w \in L \Leftrightarrow F_w \in \mathrm{SAT}.$$

Wir können o.B.d.A. annehmen, dass $Z = \{q_0, \ldots, q_m\}$, $E = \{q_m\}$ und $\Gamma = \{a_1, \ldots, a_l\}$ ist. Zudem können wir annehmen, dass δ für jedes Zeichen $a \in \Gamma$ die Anweisung $q_m a \to q_m a N$ enthält.

Die Idee besteht nun darin, die Formel F_w so zu konstruieren, dass sie unter einer Belegung a genau dann wahr wird, wenn a eine akzeptierende Rechnung von $M(w)$ beschreibt. Hierzu bilden wir F_w über den Variablen

$$\begin{aligned} &x_{t,q}, && \text{für } 0 \le t \le p(n), q \in Z, \\ &y_{t,i}, && \text{für } 0 \le t \le p(n), -p(n) \le i \le p(n), \\ &z_{t,i,a}, && \text{für } 0 \le t \le p(n), -p(n) \le i \le p(n), a \in \Gamma, \end{aligned}$$

die für folgende Aussagen stehen:

$$\begin{aligned} x_{t,q}&: \text{ zum Zeitpunkt } t \text{ befindet sich } M \text{ im Zustand } q, \\ y_{t,i}&: \text{ zur Zeit } t \text{ besucht } M \text{ das Feld mit der Nummer } i, \\ z_{t,i,a}&: \text{ zur Zeit } t \text{ steht das Zeichen } a \text{ auf dem } i\text{-ten Feld.} \end{aligned}$$

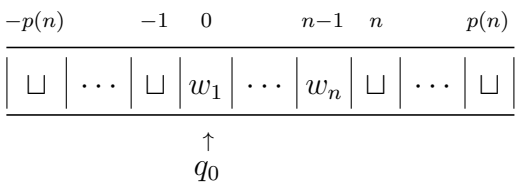
Konkret sei nun $F_w = R \wedge S \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$. Dabei stellt die Formel $R = \bigwedge_{t=0}^{p(n)} R_t$ (Randbedingungen) sicher, dass wir jeder erfüllenden Belegung eindeutig eine Folge von Konfigurationen $K_0, \ldots, K_{p(n)}$ zuordnen können:

$$\begin{aligned} R_t = {} & G(x_{t,q_0}, \ldots, x_{t,q_m}) \wedge G(y_{t,-p(n)}, \ldots, y_{t,p(n)}) \\ & \wedge \bigwedge_{i=-p(n)}^{p(n)} G(z_{t,i,a_1}, \ldots, z_{t,i,a_l}). \end{aligned}$$

Die Teilformel R_t sorgt also dafür, dass zum Zeitpunkt t

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

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



vorliegt:

$$S = x_{0,q_0} \wedge y_{0,0} \wedge \bigwedge_{i=-p(n)}^{-1} z_{0,i,\sqcup} \wedge \bigwedge_{i=0}^{n-1} z_{0,i,w_{i+1}} \wedge \bigwedge_{i=n}^{p(n)} z_{0,i,\sqcup}$$

Die Formel $\ddot{U}_1$ sorgt dafür, dass der Inhalt von nicht besuchten Feldern beim Übergang von K_t zu K_{t+1} unverändert bleibt:

$$\ddot{U}_1 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \left(\neg y_{t,i} \wedge z_{t,i,a} \to z_{t+1,i,a} \right)$$

Die Formel $\ddot{U}_2$ achtet darauf, dass sich bei jedem Übergang der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in δ verändern:

$$\begin{aligned} \ddot{U}_2 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \bigwedge_{p \in Z} \Big(& x_{t,p} \wedge y_{t,i} \wedge z_{t,i,a} \to \\ & \bigvee_{(q,b,D) \in \delta(p,a)} x_{t+1,q} \wedge y_{t+1,i+D} \wedge z_{t+1,i,b} \Big), \end{aligned}$$

wobei

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

ist. Schließlich überprüft E, ob M zur Zeit $p(n)$ den Endzustand q_m erreicht hat:

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

Da der Aufbau der Formel $f(w) = F_w$ einem einfachen Bildungsgesetz folgt und ihre Länge polynomiell in n ist, folgt $f \in \mathsf{FP}$. Es ist klar, dass F_w im Fall $w \in L(M)$ erfüllbar ist, indem wir die Variablen von F_w gemäß einer akz. Rechnung von $M(w)$ belegen. Umgekehrt führt eine Belegung a mit $F_w(a) = 1$ wegen $R(a) = 1$ eindeutig auf eine Konfigurationenfolge $K_0, \ldots, K_{p(n)}$, so dass gilt:

- K_0 ist Startkonfiguration von $M(w)$ (wegen $S(a) = 1$),
- $K_i \vdash K_{i+1}$ für $i = 0, \ldots, p(n) - 1$ (wegen $\ddot{U}_1(a) = \ddot{U}_2(a) = 1$),
- M nimmt spätestensin der Konfiguration $K_{p(n)}$ den Endzustand q_m an (wegen $E(a) = 1$).

Also gilt für alle $w \in \Sigma^*$ die Äquivalenz $w \in L(M) \Leftrightarrow F_w \in \mathrm{SAT}$, d.h. die FP-Funktion $f : w \mapsto F_w$ reduziert $L(M)$ auf SAT. ■

Korollar 173. $\mathrm{SAT} \in \mathsf{P} \Leftrightarrow \mathsf{P} = \mathsf{NP}$.

Gelingt es also, einen Polynomialzeit-Algorithmus für SAT zu finden, so lässt sich daraus leicht ein effizienter Algorithmus für jedes NP-Problem ableiten. Als nächstes betrachten wir das Erfüllbarkeitsproblem für boolesche Schaltkreise.

Definition 174.

*a) Ein **boolescher Schaltkreis** über den Variablen $x_1, \ldots, x_n$ ist eine Folge $s = (g_1, \ldots, g_m)$ von **Gattern***

$$g_l \in \{0, 1, x_1, \ldots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$$

mit $1 \le j, k < l$.

b) Die am Gatter g_l berechnete n-stellige boolesche Funktion ist induktiv wie folgt definiert:

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

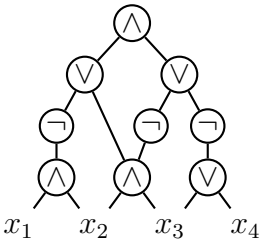
c) s berechnet die boolesche Funktion $s(a) = g_m(a)$.

*d) s heißt **erfüllbar**, wenn eine Eingabe $a \in \{0,1\}^n$ mit $s(a) = 1$ existiert.*

Beispiel 175. *Der Schaltkreis*

$$s = (x_1, x_2, x_3, x_4, (\wedge, 1, 2), (\wedge, 2, 3), (\vee, 3, 4), (\neg, 5), (\neg, 6), (\neg, 7), (\vee, 6, 8), (\vee, 9, 10), (\wedge, 11, 12))$$

ist nebenstehend graphisch dargestellt.



◁

Bemerkung 176.

- *Die Anzahl der Eingänge eines Gatters g wird als **Fanin** von g bezeichnet, die Anzahl der Ausgänge von g (d.h. 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 s.

Gefragt: Ist s erfüllbar?

Da eine boolesche Formel F leicht in einen äquivalenten Schaltkreis s mit $s(a) = F(a)$ für alle Belegungen a transformiert werden kann, folgt $\mathrm{SAT} \le^p \mathrm{CirSAT}$.

Korollar 177. CirSAT *ist* NP*-vollständig.*

Bemerkung 178. *Da* SAT NP*-vollständig ist, ist* CirSAT *in Polynomialzeit auf* SAT *reduzierbar. Dies bedeuted jedoch nicht, dass sich jeder Schaltkreis in Polynomialzeit in eine äquivalente Formel überführen lässt, sondern nur in eine erfüllbarkeitsäquivalente Formel.*

CirSAT ist sogar auf eine ganz spezielle SAT-Variante reduzierbar.

Definition 179.

a) Ein ***Literal*** *ist eine Variable x_i oder eine negierte Variable $\neg x_i$, die wir auch kurz mit $\bar{x}_i$ bezeichnen.*

b) Eine ***Klausel*** *ist eine Disjunktion $C = \bigvee_{j=1}^{k} l_j$ von Literalen.*

c) Eine Formel F ist in ***konjunktiver Normalform*** *(kurz* ***KNF****), falls F eine Konjunktion*

$$F = \bigwedge_{i=1}^{m} C_i$$

von Klauseln ist.

*d) Enthält jede Klausel höchstens k Literale, so heißt F in k-****KNF****.*

Notation. *Klauseln werden oft als Menge $C = \{l_1, \ldots, l_k\}$ ihrer Literale und KNF-Formeln als Menge $F = \{C_1, \ldots, C_m\}$ ihrer Klauseln dargestellt.*

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

Gegeben: Eine boolesche Formel F in k-KNF.

Gefragt: Ist F erfüllbar?

Folgende Variante von 3-SAT ist für den Nachweis weiterer NP-Vollständigkeitsresultate sehr nützlich.

Not-All-Equal-SAT (NaeSat):

Gegeben: Eine Formel F in 3-KNF.

Gefragt: Hat F eine (erfüllende) Belegung, unter der in keiner Klausel alle Literale denselben Wahrheitswert haben?

Beispiel 180. *Die 3-KNF 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 alternativ durch folgende Klauselmenge darstellbar:*

$$F = \{\{x_1, \bar{x}_2\}, \{\bar{x}_1, x_3\}, \{x_2, \bar{x}_3, x_4\}\}$$

Offenbar ist $F(1111) = 1$, d.h. $F \in$ 3-SAT. Da unter dieser Belegung in jeder Klausel von F nicht nur mindestens ein Literal wahr, sondern auch mindestens ein Literal falsch wird, ist F auch in NaeSat. ◁

Satz 181.

(i) 1-SAT *und* 2-SAT *sind in* P *entscheidbar.*

(ii) 3-SAT *ist* NP*-vollständig.*

Beweis. Es ist nicht schwer zu sehen, dass 1-SAT und 2-SAT in P entscheidbar sind (siehe Übungen) und dass 3-SAT in NP entscheidbar ist. Wir zeigen, dass 3-SAT NP-hart ist, indem wir CirSAT auf 3-SAT reduzieren. Hierzu transformieren wir einen Schaltkreis $s = (g_1, \ldots, g_m)$ mit n Eingängen in eine 3-KNF Formel F_s über den Variablen $x_1, \ldots, x_n, y_1, \ldots, y_m$. F_s enthält neben der Klausel $\{y_m\}$ für jedes Gatter g_i folgende Klauseln:

Gatter g_i	zugeh. Klauseln	Semantik
0	$\{\bar{y}_i\}$	$y_i = 0$
1	$\{y_i\}$	$y_i = 1$
x_j	$\{\bar{y}_i, x_j\}, \{\bar{x}_j, y_i\}$	$y_i \leftrightarrow x_j$
$(\neg, j)$	$\{\bar{y}_i, \bar{y}_j\}, \{y_j, y_i\}$	$y_i \leftrightarrow \bar{y}_j$
$(\wedge, j, k)$	$\{\bar{y}_i, y_j\}, \{\bar{y}_i, y_k\}, \{\bar{y}_j, \bar{y}_k, y_i\}$	$y_i \leftrightarrow y_j \wedge y_k$
$(\vee, j, k)$	$\{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}, \{\bar{y}_i, y_j, y_k\}$	$y_i \leftrightarrow y_j \vee y_k$

Nun ist leicht zu sehen, dass für alle $a \in \{0,1\}^n$ folgende Äquivalenz gilt:

$$s(a) = 1 \Leftrightarrow \exists b \in \{0,1\}^m : F_s(ab) = 1.$$

Ist nämlich $a \in \{0,1\}^n$ eine Eingabe mit $s(a) = 1$. Dann erhalten wir mit

$$b_l = g_l(a) \text{ für } l = 1, \ldots, m$$

eine erfüllende Belegung $ab_1 \cdots b_m$ für F_s. Ist umgekehrt $ab_1 \cdots b_m$ eine erfüllende Belegung für F_s, so folgt durch Induktion über $i = 1, \ldots, m$, dass

$$g_i(a) = b_i$$

ist. Insbesondere muss also $g_m(a) = b_m$ gelten, und da $\{y_m\}$ eine Klausel in F_s ist, folgt $s(a) = g_m(a) = b_m = 1$. Damit haben wir gezeigt, dass der Schaltkreis s und die 3-KNF-Formel F_s erfüllbarkeitsäquivalent sind, d.h.

$$s \in \textsc{CirSAT} \Leftrightarrow F_s \in \text{3-SAT}.$$

Zudem ist leicht zu sehen, dass die Reduktionsfunktion $s \mapsto F_s$ in FP berechenbar ist, womit $\textsc{CirSAT} \leq^p \text{3-SAT}$ folgt. ■

Die Reduktion von CirSAT auf 3-SAT lässt sich leicht zu einer Reduktion von CirSAT auf NaeSat modifizieren.

Satz 182. NaeSat *ist* NP*-vollständig.*

Beweis. Es ist klar, dass $\textsc{NaeSat} \in \mathsf{NP}$ liegt. Die Reduktion $s \mapsto F_s$ von CirSAT auf 3-SAT aus vorigem Beweis erfüllt bereits die folgenden Bedingungen:

- Ist $s(a) = 1$, so können wir a zu einer erfüllenden Belegung ab von F_s erweitern, d.h. unter ab wird in jeder Klausel von F_s ein Literal wahr.
- Tatsächlich wird unter der Belegung ab in jeder Dreierklausel von F_s auch bereits ein Literal falsch.

Letzteres ist leicht zu sehen, da ab für jedes Und-Gatter g_i nicht nur die Dreierklausel $\{\bar{y}_i, y_j, y_k\}$, sondern auch die Klauseln $\{\bar{y}_j, y_i\}$ und $\{\bar{y}_k, y_i\}$ erfüllt. Diese verhindern nämlich, dass ab alle Literale der Dreierklausel $\{\bar{y}_i, y_j, y_k\}$ erfüllt. Entsprechend verhindern die zu einem Oder-Gatter g_i gehörigen Klauseln $\{y_j, \bar{y}_i\}$ und $\{y_k, \bar{y}_j\}$, dass ab alle Literale der Dreierklausel $\{y_i, \bar{y}_j, \bar{y}_k\}$ erfüllt.

Um zu erreichen, dass auch in den übrigen Klauseln C mit $\|C\| < 3$ ein Literal falsch wird, können wir einfach eine neue Variable z zu diesen Klauseln hinzufügen und z mit dem Wert 0 belegen. Sei also F'_s die 3-KNF Formel über den Variablen $x_1, \ldots, x_n, y_1, \ldots, y_m, z$, die die Klausel $\{y_m, z\}$ und für jedes Gatter g_i die folgenden Klauseln enthält:

Gatter g_i	zugeh. Klauseln
0	$\{\bar{y}_i, z\}$
1	$\{y_i, z\}$
x_j	$\{\bar{y}_i, x_j, z\}, \{\bar{x}_j, y_i, z\}$
$(\neg, j)$	$\{\bar{y}_i, \bar{y}_j, z\}, \{y_j, y_i, z\}$
$(\wedge, j, k)$	$\{\bar{y}_i, y_j, z\}, \{\bar{y}_i, y_k, z\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$\{\bar{y}_j, y_i, z\}, \{\bar{y}_k, y_i, z\}, \{\bar{y}_i, y_j, y_k\}$

Wie wir gesehen haben, lässt sich dann jede Belegung $a \in \{0,1\}^n$ der x-Variablen mit $s(a) = 1$ zu einer Belegung $abc \in \{0,1\}^{n+m+1}$ für F'_s erweitern, unter der in jeder Klausel von F'_s mindestens ein Literal wahr und mindestens ein Literal falsch wird, d.h. es gilt

$$s \in \textsc{CirSAT} \Rightarrow F'_s \in \textsc{NaeSat}.$$

Für den Nachweis der umgekehrten Implikation sei nun $F'_s \in \textsc{NaeSat}$ angenommen. Dann existiert eine Belegung $abc \in \{0,1\}^{n+m+1}$ für F'_s, unter der in jeder Klausel ein wahres und ein falsches Literal vorkommen. Da dies auch unter der komplementären Belegung $\overline{abc}$ der Fall

ist, können wir $c = 0$ annehmen. Dann erfüllt aber die Belegung ab die Formel F_s und damit folgt $s(a) = 1$, also $s \in$ CirSAT. ■

7.2 Max-SAT Probleme

In manchen Anwendungen genügt es nicht, festzustellen, dass eine KNF-Formel F nicht erfüllbar ist. Vielmehr geht es darum, herauszufinden, wieviele Klauseln in F maximal erfüllbar sind. Die Schwierigkeit dieses Optimierungsproblems lässt sich durch folgendes Entscheidungsproblem charakterisieren.

Max-k-SAT:

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

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

Man beachte, dass hierbei die Klauseln in F auch mehrfach vorkommen können (d.h. F ist eine ***Multimenge*** von Klauseln).

Satz 183.

(i) Max-1-SAT $\in$ P.

(ii) Max-2-SAT $\in$ NPC.

Beweis. Wir reduzieren 3-SAT auf Max-2-SAT. Für eine Dreierklausel $C = \{l_1, l_2, l_3\}$, in der die Variable v nicht vorkommt, sei $G(l_1, l_2, l_3, v)$ die 2-KNF Formel, die aus folgenden 10 Klauseln besteht:

$$\{l_1\}, \{l_2\}, \{l_3\}, \{v\}, \{\overline{l_1}, \overline{l_2}\}, \{\overline{l_2}, \overline{l_3}\}, \{\overline{l_1}, \overline{l_3}\}, \{l_1, \overline{v}\}, \{l_2, \overline{v}\}, \{l_3, \overline{v}\}$$

Dann lassen sich folgende 3 Eigenschaften von G leicht verifizieren:

- Keine Belegung von G erfüllt mehr als 7 Klauseln von G.
- Jede Belegung a von C mit $C(a) = 1$ ist zu einer Belegung a' von G erweiterbar, die 7 Klauseln von G erfüllt.
- Keine Belegung a von C mit $C(a) = 0$ ist zu einer Belegung a' von G erweiterbar, die 7 Klauseln von G erfüllt.

Ist nun F eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$ mit m Klauseln, so können wir annehmen, dass $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$, $j = 1, \ldots, k$, die Dreier- und $C_{k+1}, \ldots, C_m$ die Einer- und Zweierklauseln von F sind. Wir transformieren F auf das Paar $g(F) = (F', m + 6k)$, wobei die 2-KNF Formel F' wie folgt aus F entsteht:

> Ersetze jede Dreierklausel $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$ in F durch die 10 Klauseln der 2-KNF-Formel $G_j = G(l_{j1}, l_{j2}, l_{j3}, v_j)$.

Dann ist leicht zu sehen, dass g in FP berechenbar ist. Außerdem gilt folgende Äquivalenz:

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

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

Für die Rückwärtsrichtung sei a eine Belegung, die mindestens $m+6k$ Klauseln von F' erfüllt. Da a in jeder 10er-Gruppe G_j, $j = 1, \ldots, k$, nicht mehr als 7 Klauseln erfüllen kann, muss a in jeder 10er-Gruppe genau 7 Klauseln und zudem alle Klauseln C_j für $j = k + 1, \ldots, m$ erfüllen. Dies ist aber nur möglich, wenn a alle Klauseln C_j von F erfüllt. ■