Vorlesungsskript

Theoretische Informatik II

Wintersemester 2007/2008

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

7. Februar 2008

Inhaltsverzeichnis

1 Reguläre Sprachen **1**

1.1 Endliche Automaten 1
1.2 Nichtdeterministische endliche Automaten 5
1.3 Reguläre Ausdrücke 8
1.4 Relationalstrukturen 11
1.4.1 Eigenschaften von Relationen 13
1.4.2 Äquivalenz- und Ordnungsrelationen 17
1.4.3 Abbildungen 22
1.4.4 Homo- und Isomorphismen 23
1.5 Minimierung von DFAs 25
1.6 Grammatiken 30
1.7 Das Pumping-Lemma 34

2 Kontextfreie Sprachen **37**

2.1 Kellerautomaten 37
2.2 Äquivalenz von kontextfreien Grammatiken und Kellerautomaten . . 39
2.3 Das Wortproblem für kontextfreie Grammatiken 45
2.3.1 Chomsky-Normalform 45
2.3.2 Der CYK-Algorithmus 49
2.4 Das Pumping-Lemma für kontextfreie Sprachen 50
2.5 Deterministisch kontextfreie Sprachen 53

3 Kontextsensitive Sprachen **60**

3.1 Kontextsensitive Grammatiken 60
3.2 Turingmaschinen 61
3.3 Linear beschränkte Automaten 64

4 Entscheidbare und semi-entscheidbare Sprachen **69**

4.1 Kodierung (Gödelisierung) von Turingmaschinen 74
4.2 Das Halteproblem 75
4.3 Das Postsche Korrespondenzproblem 79

5 Komplexitätsklassen und NP-Vollständigkeit **85**
5.1 Zeitkomplexität 85
5.2 Platzkomplexität 86
5.3 NP-Vollständigkeit und das P = NP-Problem 88
5.3.1 Aussagenlogische Erfüllbarkeitsprobleme 90
5.3.2 Cliquen, stabile Mengen und Knotenüberdeckungen 97
5.3.3 Das Wort- und das Äquivalenzproblem für reguläre Sprachen . 99
5.3.4 Färbung von Graphen 101
5.3.5 MAX-SAT Probleme 102
5.3.6 Matchings und der Heiratssatz 104

Kapitel 1

Reguläre Sprachen

1.1 Endliche Automaten

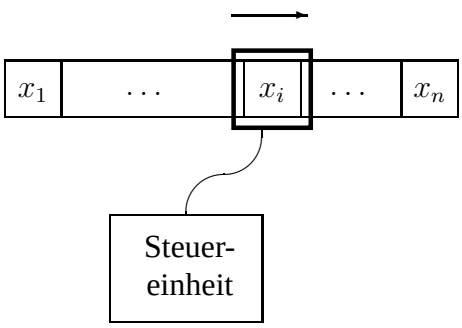
Rechenmaschinen spielen in der Informatik eine zentrale Rolle. Hier beschäftigen wir uns mit mathematischen Modellen für Maschinentypen von unterschiedlicher Berechnungskraft. In der Vorlesung Theoretische Informatik 1 wurde die Turingmaschine als ein universales Berechnungsmodell eingeführt. In dieser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

Definition 1.1 *Ein **Alphabet** ist eine geordnete endliche Menge $\Sigma = \{a_1, \ldots, a_m\}$ von **Zeichen**. Eine Folge $x = x_1 \ldots x_n \in \Sigma^n$ heißt **Wort** (der **Länge** n). Die Menge aller Wörter über Σ ist*

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

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

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



Definition 1.2 *Ein **endlicher Automat** (kurz: DFA; deterministic finite automaton) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, q_0, E)$ beschrieben, wobei*

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

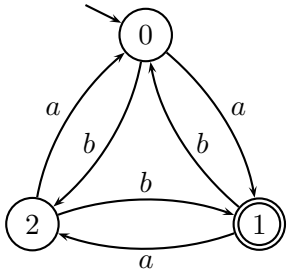
*Die von M **akzeptierte** oder **erkannte Sprache** ist*

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

Beispiel 1.3 Betrachte den DFA $M = (Z, \Sigma, \delta, 0, E)$ mit $Z = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $E = \{1\}$ und der Überführungsfunktion

δ	a	b
0	1	2
1	2	0
2	0	1

Graphische Darstellung:



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

Behauptung 1.4 *M 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. (Für $j \equiv k \pmod m$ schreiben wir im Folgenden auch kurz $j \equiv_m k$.)

Beweis 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

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

Da 1 der einzige Endzustand von M ist, reicht es, folgende Kongruenzgleichung zu zeigen:

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

Wir beweisen die Kongruenz induktiv über die Länge von x.

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

Induktionsschritt ($n \rightsquigarrow n+1$): Sei $x = x_1 \cdots x_{n+1}$ gegeben. Nach IV ist

$$\hat{\delta}(0, x_1 \cdots x_n) \equiv_3 \#_a(x_1 \cdots x_n) - \#_b(x_1 \cdots x_n).$$

Wegen

$$\delta(i, a) \equiv_3 i + 1 \text{ und } \delta(i, b) \equiv_3 i - 1$$

folgt daher sofort

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

□

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

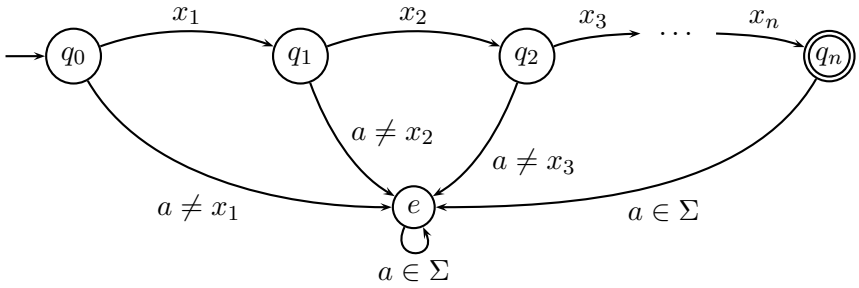
$$\mathsf{REG} = \{L(M) \mid M \text{ ist ein DFA}\}.$$

Um ein intuitives Verständnis für die Berechnungskraft von DFAs zu entwickeln, werden wir uns zunächst intensiv mit der Beantwortung folgender Frage beschäftigen.

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



Formal lässt sich M also durch das Tupel $M = (Z, \Sigma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots, q_n, e\}$, $E = \{q_n\}$ 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 1.6 *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 1.7 Der 2-stellige Schnittoperator bildet zwei Sprachen L_1 und L_2 auf die Sprache $L_1 \cap L_2$ ab. ◁

Definition 1.8 *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.*

Beobachtung 1.9 *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 1.9 folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da die in Beispiel 1.3 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

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

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

Beim Versuch, einen endlichen Automaten für das Produkt L_1L_2 zweier regulärer Sprachen zu konstruieren, stößt man auf die Schwierigkeit, den richtigen Zeitpunkt für den Übergang von (der Simulation von) M_1 zu M_2 zu finden. Unter Verwendung eines nichtdeterministischen Automaten lässt sich dieses Problem jedoch leicht beheben, da dieser den richtigen Zeitpunkt „erraten“ kann.

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

1.2 Nichtdeterministische endliche Automaten

Definition 1.10 *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*

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

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

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

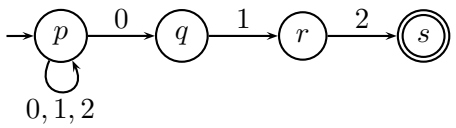
Ein NFA kann also nicht nur eine, sondern mehrere verschiedene Rechnungen ausführen. Die Eingabe gehört bereits dann zu $L(N)$, wenn bei einer dieser Rechnungen nach Lesen des gesamten Eingabewortes ein Endzustand erreicht wird.

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 1.11 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 1.12 *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

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

und

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

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

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

die Sprache $L(N_1)^*$.

Theorem 1.13 $\mathsf{REG} = \{L(N) \mid N \textit{ ist ein NFA}\}$.

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 mit $L(M) = L(N)$. Zunächst erweitern wir die Überführungsfunktion $\delta : Z \times \Sigma \to \mathcal{P}(Z)$ zu $\delta' : \mathcal{P}(Z) \times \Sigma \to \mathcal{P}(Z)$ mittels

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

und zeigen folgende Behauptung:

> $\hat{\delta}'(Q_0, x)$ enthält alle von N bei Eingabe x in $|x|$ Schritten erreichbaren Zustände.

Wir beweisen die Behauptung induktiv über die Länge von x.

Induktionsanfang ($|x| = 0$): 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 $n-1$ 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 n Schritten erreichen kann.

Nun ist leicht zu sehen, dass der DFA

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

mit

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

und

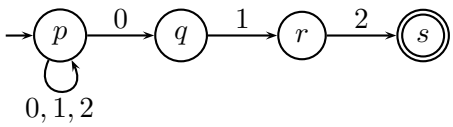
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \emptyset\}$$

äquivalent zu N ist, da für alle Wörter $x \in \Sigma^*$ gilt:

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

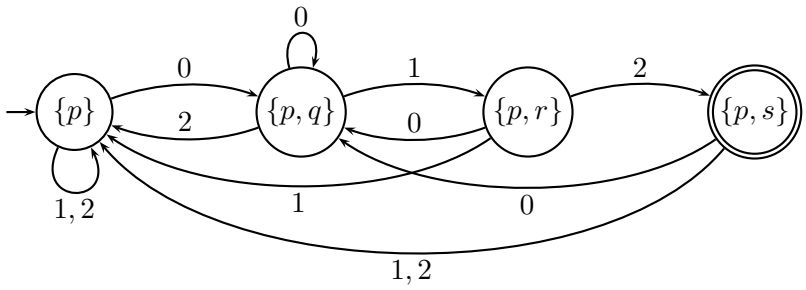
□

Beispiel 1.14 Für den NFA $N = (Z, \Sigma, \delta, Q_0, E)$ aus Beispiel 1.11



ergibt die Konstruktion des vorigen Satzes den folgenden DFA M (nach Entfernung 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 so genannten **Potenzmengenautomaten** M benötigt werden (siehe Übungen).

1.3 Reguläre Ausdrücke

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

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

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 einelementigen Sprachen bilden lassen.

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

Definition 1.15 *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 1.16 Über $\Sigma = \{0, 1\}$ sind ϵ^*, $\emptyset^*$, $(0|1)^*00$ und $(\epsilon 0|\emptyset 1^*)$ reguläre Ausdrücke, die folgende Sprachen beschreiben:

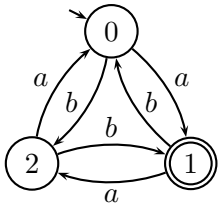
γ	ϵ^*	$\emptyset^*$	$(0\|1)^*00$	$(\epsilon 0\|\emptyset 1^*)$
$L(\gamma)$	$\{\varepsilon\}^* = \{\varepsilon\}$	$\emptyset^* = \{\varepsilon\}$	$\{x00 \mid x \in \Sigma^*\}$	$\{0\}$

◁

Bemerkung 1.17

- *Um Klammern zu sparen, definieren wir folgende **Präzedenzordnung**:*
  Der Sternoperator * bindet stärker als der Produktoperater und dieser wiederum stärker als der Vereinigungsoperator $|$.
- *Da der reguläre Ausdruck $\gamma\gamma^*$ die Sprache $L(\gamma)^+$ beschreibt, verwenden wir γ^+ als Abkürzung für den Ausdruck $\gamma\gamma^*$.*

Beispiel 1.18 Betrachte den 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 $L_1 = \{x \in \{a, b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 0\}$. Da sich L_1 durch den regulären Ausdruck

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

beschreiben lässt, erhalten wir für $L(M)$ den regulären Ausdruck $\gamma(ba)^*(a|bb)$. ◁

Theorem 1.19 $\mathsf{REG} = \{L(\gamma) \mid \gamma$ *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 1.9 und 1.12).

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

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

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

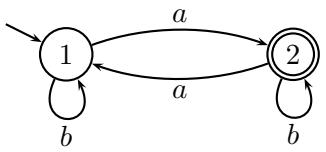
$$L^{r+1}_{p,q} \;=\; L^r_{p,q} \cup L^r_{p,r+1}(L^r_{r+1,r+1})^* L^r_{r+1,q}$$

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 1.20 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 einen regulären Ausdruck für $L(M) = L(b^*a(b|ab^*a)^*)$ zu erhalten, genügt es also, die regulären Ausdrücke $\gamma^0_{1,1}$, $\gamma^0_{1,2}$, $\gamma^0_{2,1}$, $\gamma^0_{2,2}$, $\gamma^1_{1,2}$, $\gamma^1_{2,2}$ und $\gamma^2_{1,2}$ 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	-	$\underbrace{b^*a \mid b^*a(\epsilon \mid b \mid ab^*a)^*(\epsilon \mid b \mid ab^*a)}_{b^*a(b \mid ab^*a)^*}$	-	-

◁

1.4 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.

Bemerkung 1.21

- *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 1.22

- (F, M) mit $F := \{f \mid f$ ist Fluss in Europa$\}$ und
  $M := \{(f, g) \in F \times F \mid f$ mündet in $g\}$.

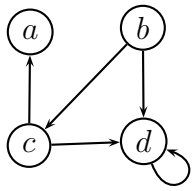
- (U, B) mit $U := \{x \mid x \text{ ist Berliner }\}$ und
  $B := \{(x, y) \in U \times U \mid x \text{ ist Bruder von } y\}$.
- $(\mathsf{P}(M), \subseteq)$, wobei $\mathsf{P}(M)$ die Potenzmenge einer beliebigen Menge M und $\subseteq$ die Inklusionsbeziehung auf den Teilmengen von M ist.
- (A, Id_A), 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 $\mathcal{F}ml := \{F \mid F \text{ ist aussagenlogische Formel}\}$ und
  $\Rightarrow = \{(F, G) \in \mathcal{F}ml \times \mathcal{F}ml \mid G \text{ ist aussagenlogische Folgerung von } F\}$.

◁

Graphische Darstellung von Relationen

Eine Relation R auf einer endlichen Menge A kann durch einen ***gerichteten Graphen*** $G = (V, E)$ 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 1.23 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 $d_{out}(x) = \|R(x)\|$, wobei $R(x) = \{y \in V \mid xRy\}$ der ***Nachbereich*** von x ist. Entsprechend ist $d_{in}(x) = \|\{y \in V \mid yRx\}\|$ der ***Eingangsgrad*** von x. Falls R symmetrisch ist, können die Pfeilspitzen auch weggelassen werden. In diesem Fall ist $d(x) = d_{in}(x) = d_{out}(x)$ der ***Grad*** von x. Ist R zudem irreflexiv, so erhalten wir einen (schleifenfreien) ***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

1.4.1 Eigenschaften von Relationen

Sei R eine Relation auf A. Dann heißt R

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

gilt.

Beispiel 1.24

- 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.

◁

Operationen auf Relationen

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

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

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:

$$\bigcap \mathcal{M} = \{(x,y) \mid \forall R \in \mathcal{M} : xRy\}$$

$$\bigcup \mathcal{M} = \{(x,y) \mid \exists R \in \mathcal{M} : xRy\}$$

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

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

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 $(\mathbb{R}, \leq^T) = (\mathbb{R}, \geq)$.
Eine wichtige zweistellige Operation auf der Menge $\mathsf{P}(A \times A)$ aller Relationen auf A ist das Relationenprodukt (auch Komposition genannt).
Das ***Produkt*** zweier Relationen R und S auf A ist

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

Übliche Bezeichnungen für das Relationenprodukt sind auch $R;S$ und $R \cdot S$ oder einfach RS. Für $\underbrace{R \circ \cdots \circ R}_{\text{n-mal}}$ wird auch R^n geschrieben. Dabei ist $R^0 = Id$.
Vorsicht: Das n-fache Relationenprodukt von R sollte nicht mit dem n-fachen kartesischen Produkt der *Menge* R verwechselt werden. Wir vereinbaren, dass R^n das n-fache Relationenprodukt bezeichnen soll, falls R eine Relation ist.

Beispiel 1.25 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 Relation "ist Onkel von".◁

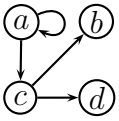
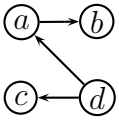
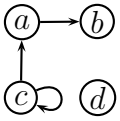
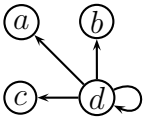
Sind $M_R = (r_{ij})$ und $M_S = (s_{ij})$ boolesche $n \times n$-Matrizen für R und S, so erhalten wir für $T = R \circ S$ die Matrix $M_T = (t_{ij})$ mit

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

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 1.26 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	$R \circ S$	$S \circ R$
Graph				
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.

Theorem 1.27 *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 1.28 *Sei R eine Relation.*

- *Die **reflexive Hülle** von R ist*

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

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

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

- *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*

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

Theorem 1.29 *Sei R eine Relation auf A.*

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

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

(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 $x_0 \cdots x_n$ einen ***Weg der Länge*** n ***von*** a ***nach*** b.

1.4.2 Äquivalenz- und Ordnungsrelationen

Die nachfolgende Tabelle gibt einen Überblick über die definierenden Eigenschaften der wichtigsten Relationalstrukturen.

	refl.	sym.	trans.	asym.	antisym.	konnex	semikon.
Äquivalenzrel.	✓	✓	✓				
(Halb-)Ordnung	✓		✓		✓		
Striktordnung			✓	✓			
lineare Ord.			✓		✓	✓	
lin. Striktord.			✓	✓			✓
schwache Ord.			✓			✓	
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 eine Reihe von Beispielen für ***Äquivalenzrelationen***, die durch die drei Eigenschaften reflexiv, symmetrisch und transitiv definiert sind.

Beispiel 1.30

- Auf der Menge aller Geraden im $\mathbb{R}^2$ die Parallelität.
- Auf der Menge aller Menschen "im gleichen Jahr geboren wie".
- Auf $\mathbb{Z}$ die Relation "gleicher Rest bei Division durch m".
- Auf der Menge der aussagenlogischen Formeln die semantische Äquivalenz. ◁

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.

Definition 1.31 *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 den selben Sachverhalt.

Theorem 1.32 *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.

□

Beispiel 1.33 Für die weiter oben betrachteten Äquivalenzrelationen erhalten wir folgende Klasseneinteilungen.

- Für die Parallelität auf der Menge aller Geraden im $\mathbb{R}^2$: alle Geraden mit derselben Richtung (oder Steigung) bilden jeweils eine Äquivalenzklasse.
- Für die Relation "im gleichen Jahr geboren wie" auf der Menge aller Menschen: jeder Jahrgang bildet eine Äquivalenzklasse.
- Für die Relation "gleicher Rest bei Division durch m" auf $\mathbb{Z}$: jede der m Restklassen $[0], [1], \ldots, [m-1]$ mit

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

  bildet eine Äquivalenzklasse. ◁

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 $h_{\text{äq}}(R)$ generierten Äquivalenzklassen auch die **schwachen Zusammenhangskomponenten** des gerichteten Graphen (A, R) genannt (siehe Übungen). Als nächstes betrachten wir Ordnungen.

Definition 1.34 (A, R) *heißt* ***Ordnung*** *(auch* ***Halbordnung*** *oder* ***partielle Ordnung****), wenn* R *eine reflexive, antisymmetrische und transitive Relation auf* A *ist.*

Beispiel 1.35

- $(\mathbb{Z}, \leq)$ 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 $\mathcal{A}(M)$. $\triangleleft$

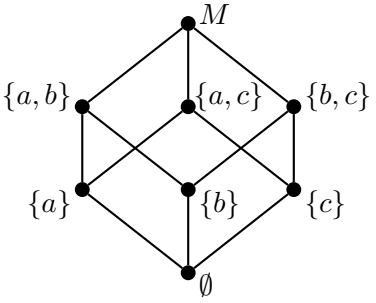
Ordnungen lassen sich sehr anschaulich durch Hasse-Diagramme darstellen. Sei $\leq$ eine Ordnung auf A und sei $<$ die Relation $\leq \cap \overline{Id}_A$. Um die Ordnung $\leq$ in einem ***Hasse-Diagramm*** darzustellen, wird nur der Graph der ***Nachbarrelation***

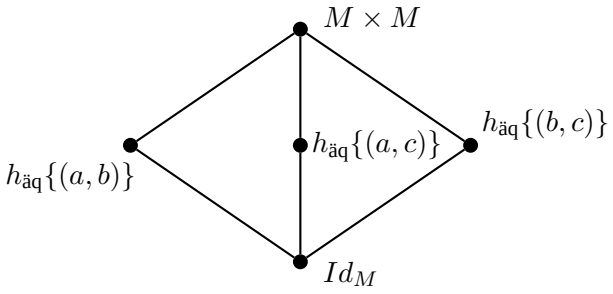
$$\lhd \; = \; < \setminus <^2, \text{ d.h. } x \lhd y \;\Leftrightarrow\; x < y \wedge \neg \exists z : x < z < y$$

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

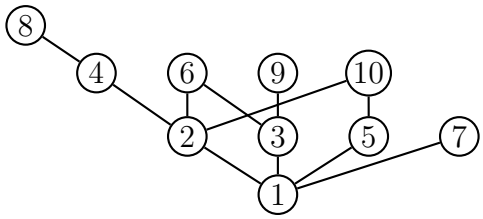
Beispiel 1.36

1. Die Inklusionsrelation auf der Potenzmenge $\mathcal{P}(M)$ von $M = \{a, b, c\}$ und die Verfeinerungsrelation auf der Menge $\mathcal{A}(M)$ aller Äquivalenzrelationen auf $M = \{a, b, c\}$ lassen sich durch folgende Hasse-Diagramme darstellen.





2. Die "teilt"-Relation auf $\{1, 2, \ldots, 10\}$ ist durch folgendes Hasse-Diagramme darstellbar.



◁

Definition 1.37 *Sei $\leq$ eine Ordnung auf A und sei $h \in H$ Element einer Teilmenge $H \subseteq A$.*

- *h heißt **kleinstes Element** oder **Minimum** von H ($h = \min H$), falls gilt:*

$$\forall h' \in H : h \leq h'.$$

- *h heißt **größtes Element** oder **Maximum** von H ($h = \max H$), falls gilt:*

$$\forall h' \in H : h' \leq h.$$

- *a heißt **minimal** in H, falls es in H kein kleineres Element gibt:*

$$\forall h' \in H : h' \leq h \Rightarrow h' = h.$$

- *a heißt **maximal** in H, falls es in H kein größeres Element gibt:*

$$\forall h' \in H : h \leq h' \Rightarrow h = h'.$$

Bemerkung 1.38 *Wegen der Antisymmetrie kann es in H höchstens ein kleinstes und höchstens ein größtes Element geben.*

Definition 1.39 *Sei $\leq$ eine Ordnung auf A und sei $H \subseteq A$.*

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

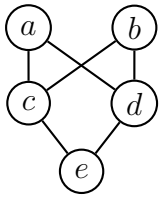
$$(\forall h \in H : h \geq i) \wedge [\forall u \in A : (\forall h \in H : h \geq u) \Rightarrow u \leq i].$$

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

$$(\forall h \in H : h \leq s) \wedge [\forall o \in A : (\forall h \in H : h \leq o) \Rightarrow s \leq o]$$

Bemerkung 1.40 *H kann nicht mehr als ein Supremum und ein Infimum haben.*

Beispiel 1.41 Betrachte nebenstehende Ordnung auf der Menge $A = \{a, b, c, d, e\}$. Die folgende Tabelle zeigt für verschiedene Teilmengen $H \subseteq A$ alle minimalen und maximalen Elemente in H, alle unteren und oberen Schranken, sowie Minimum, Maximum, Infimum und Supremum von H (falls existent).



H	minimal	maximal	Minimum	Maximum	unt. Schr.	ob. Schr.	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 1.42

- *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*

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

  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.*

1.4.3 Abbildungen

Definition 1.43 *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*

$$N(R) = \bigcup_{x \in M} R(x) \text{ und } V(R) = \bigcup_{x \in M} R^T(x).$$

- *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 1.44

- *Wie üblich werden wir Abbildungen meist mit kleinen Buchstaben* $f, g, h, ...$ *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 1.45

- *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 $V(f^{-1}) = N(f)$ nur dann gleich B ist, wenn f auch surjektiv, also eine Bijektion ist.

1.4.4 Homo- und Isomorphismen

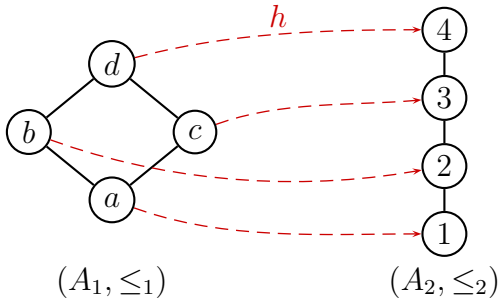
Definition 1.46 *Seien* (A_1, R_1) *und* (A_2, R_2) *Relationalstrukturen.*

- *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 1.47 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

$$2 \leq_2 3, \text{ aber } h^{-1}(2) = b \not\leq_1 c = h^{-1}(3).$$

◁

Definition 1.48 *Ein bijektiver Homomorphismus $h : A_1 \to A_2$, bei dem auch h^{-1} ein Homomorphismus ist, d.h. es gilt*

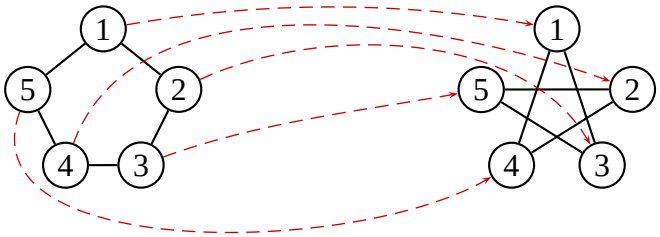
$$\forall a, b \in A_1 : aR_1b \Leftrightarrow h(a)R_2h(b).$$

*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 1.49

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

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



- Die Bijektion $h : x \mapsto e^x$ ist ein Ordnungsisomorphismus zwischen $(\mathbb{R}, \leq)$ und $((0, \infty), \leq)$.

- Für $n \in \mathbb{N}$ sei
$$T_n \;=\; \{k \in \mathbb{N} \mid k \text{ teilt } n\}$$
die Menge aller Teiler von n und $P_n \;=\; T_n \cap \text{Prim}$ die Menge aller Primteiler von n. Dann ist für quadratfreies n, d.h. es gibt kein $k \geq 2$, so dass k^2 die Zahl n teilt, die Abbildung
$$h : k \mapsto P_k$$
ein Ordnungsisomorphismus zwischen $(T_n, |)$ und $(\mathcal{P}(P_n), \subseteq)$.

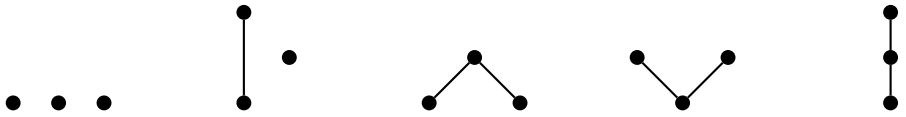
- Während auf der Knotenmenge $V = [3]$ insgesamt $2^3 = 8$ verschiedene Graphen existieren, gibt es auf dieser Menge nur 4 unterschiedliche nichtisomorphe Graphen:

◁

Bemerkung 1.50 *Auf der Knotenmenge $V = [n]$ existieren genau $2^{\binom{n}{2}}$ verschiedene Graphen. Sei $a(n)$ die Anzahl aller nichtisomorphen Graphen auf V. Da maximal $n!$ verschiedene Graphen auf V in einer Isomorphieklasse liegen können, ist $a(n) \geq 2^{\binom{n}{2}}/n!$.*

Tatsächlich ist $a(n)$ ***asymptotisch gleich*** *$g(n) = 2^{\binom{n}{2}}/n!$ (in Zeichen: $a(n) \sim g(n)$), d.h.*
$$\lim_{n\to\infty} a(n)/g(n) = 1.$$
Also gibt es auf $V = [n]$ nicht wesentlich mehr als $g(n)$ nichtisomorphe Graphen.

Beispiel 1.51 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. ◁

1.5 Minimierung von DFAs

Wie können wir feststellen, ob ein DFA $M = (Z, \Sigma, \delta, q_0, E)$ unnötige Zustände enthält? Zunächst einmal können alle Zustände entfernt werden, die nicht vom Startzustand aus erreichbar sind. Im folgenden gehen wir daher davon aus, dass M keine

unerreichbaren Zustände enthält. Offensichtlich können zwei Zustände q und p zu einem Zustand verschmolzen werden (kurz: $q \sim p$), wenn M von q und von p ausgehend jeweils dieselben Wörter akzeptiert. Bezeichnen wir den DFA $(Z, \Sigma, \delta, q, E)$ mit M_q und $L(M_q)$ mit L_q, so sind q und p genau dann verschmelzbar, wenn $L_q = L_p$ ist.

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 $M' = (\tilde{Z}, \Sigma, \delta', [q_0], \tilde{E})$ mit

$$\delta'([q], a) = [\delta(q, a)].$$

Im nächsten Satz zeigen wir, dass M' tatsächlich der gesuchte Minimalautomat für $L(M)$ ist. Für eine Teilmenge $Q \subseteq Z$ bezeichne $\tilde{Q}$ die Menge $\{\tilde{q} \mid q \in Q\}$ aller Äquivalenzklassen $\tilde{q}$, die einen Repräsentanten q in Q haben.

Theorem 1.52 *Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA, der nur Zustände enthält, die vom Startzustand q_0 aus erreichbar sind. Dann ist $M' = (\tilde{Z}, \Sigma, \delta', [q_0], \tilde{E})$ mit*

$$\delta'([q], a) = [\delta(q, a)]$$

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

Beweis

- Wir zeigen zuerst, dass δ' wohldefiniert ist, also der Wert von $\delta'(\tilde{q}, a)$ 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 $L(M') = L(M)$ 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

  $$\delta'([q_{i-1}], x_i) = [\delta(q_{i-1}, x_i)] = [q_i]$$

  durchläuft M' dann die Zustände

  $$[q_0], [q_1], \cdots, [q_n].$$

  Da aber q_n genau dann zu E gehört, wenn $[q_n] \in \tilde{E}$ ist, folgt $L(M') = L(M)$.

- Es bleibt zu zeigen, dass 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 $k = \|\tilde{Z}\| = \|\{L_q \mid q \in Z\}\|$ nicht von M, sondern nur von $L(M)$ abhängt. Für $x \in \Sigma^*$ sei

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

  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 $L(M)$ ab. □

Für die algorithmische Konstruktion von M' aus M steht man vor der Aufgabe, festzustellen, ob zwei Zustände p und q verschmelzbar sind oder nicht. Zur Beantwortung dieser Frage machen wir folgende Beobachtungen.

Beobachtung 1.53

- *Kein Endzustand $p \in E$ ist mit einem Zustand $q \in Z \setminus E$ verschmelzbar (wegen $\varepsilon \in L_p \Delta L_q$).*
- *Wenn $\delta(p,a)$ und $\delta(q,a)$ unverschmelzbar sind, dann sind auch p und q unverschmelzbar (wegen $p \sim q \Rightarrow \delta(p,a) \sim \delta(q,a)$).*
- *Wenn also D nur unverschmelzbare Zustandspaare enthält, dann sind auch alle Paare in der Menge*

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

  unverschmelzbar.

Daher berechnen wir 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

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

indem wir zu D_i alle Paare $\{q,p\}$ hinzufügen, für die eines der Paare $\{\delta(q,a), \delta(p,a)\}$, $a \in \Sigma$, 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):

$$p \sim q \Leftrightarrow \{p,q\} \notin D_j.$$

Daher kann nun M' durch Verschmelzen aller Zustände p, q mit $\{p,q\} \notin D_j$ gebildet werden. Damit erhalten wir folgenden Algorithmus zur Berechnung eines minimalen DFA.

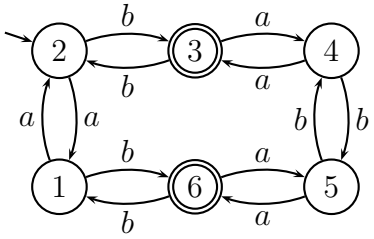
Algorithmus 1.54 MIN-DFA

1 **Eingabe:** DFA $M = (Z, \Sigma, \delta, q_0, E)$.
2 Entferne alle von q_0 aus unerreichbaren Zustände aus Z.
3 Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$.
4 Solange noch ein unmarkiertes Paar $\{z, z'\} \subseteq Z$ existiert, für das eines der Paare $\{\delta(z, a), \delta(z', a)\}$, $a \in \Sigma$, bereits markiert ist, markiere auch $\{z, z'\}$.
5 Bilde die Verschmelzungsmengen

$$\tilde{z} = \{z\} \cup \{z' \in Z \mid \{z, z'\} \text{ ist nicht markiert}\}, \ z \in Z.$$

6 **Ausgabe:** Minimal-DFA $M' = (\tilde{Z}, \Sigma, \delta', \tilde{z}_0, \tilde{E})$ mit $\tilde{Z} = \{\tilde{z} \mid z \in Z\}$, $\tilde{E} = \{\tilde{z} \mid z \in E\}$ und $\delta'(\tilde{z}, a) = \widetilde{\delta(z, a)}$.

Beispiel 1.55 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 eine Null markiert.

2					
3	0	0			
4	1	1	0		
5	1	1	0		
6	0	0		0	0
	1	2	3	4	5

Wegen

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

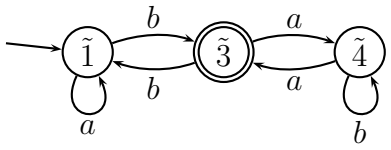
enthält D_1 zusätzlich die Paare $\{1,4\}$, $\{1,5\}$, $\{2,4\}$, $\{2,5\}$ (in obiger Matrix durch eine Eins markiert). Da die verbliebenen Paare $\{1,2\}$, $\{3,6\}$, $\{4,5\}$ wegen

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

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 Verschmelzungsmengen

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

die auf folgenden Minimal-DFA M' führen:



◁

Durch eine leichte Modifikation von M' ist es möglich, einen Minimalautomaten M_L direkt aus einer regulären Sprache L zu gewinnen (also ohne den Umweg über einen DFA M für L). Da nämlich zwei Eingaben x und y den DFA M' genau dann in denselben Zustand $[\hat{\delta}(q_0,x)] = [\hat{\delta}(q_0,y)]$ überführen, wenn $L_x = L_y$ ist, können wir den von M' bei Eingabe x erreichten Zustand $[\hat{\delta}(q_0,x)]$ auch mit der Sprache L_x bezeichnen. Dies führt auf den DFA $M_L = (Z_L, \Sigma, \delta_L, L_\varepsilon, E_L)$ mit

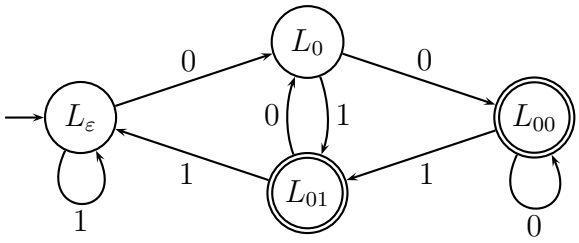
$$\begin{aligned} Z_L &= \{L_x \mid x \in \Sigma^*\}, \\ E_L &= \{L_x \mid x \in L\} \text{ und} \\ \delta_L(L_x, a) &= L_{xa}, \end{aligned}$$

welcher isomorph zu M' ist (also bis auf die Benennung der Zustände mit diesem identisch ist).

Beispiel 1.56 Für $L = \{x_1 \cdots x_n \mid x_i \in \{0,1\} \text{ für } i = 1, \ldots n \text{ und } x_{n-1} = 0\}$ ist

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

Somit erhalten wir den folgenden Minimalautomaten M_L:



◁

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

Um für eine reguläre Sprache L einen Minimalautomaten zu konstruieren, ist es nicht nötig, die Sprachen L_x zu bestimmen. Um die Überführungsfunktion aufzustellen, müssen wir nur herausfinden, welche Eigaben jeweils in denselben Zustand L_x führen. Wie die Sprachen L_x konkret aussehen, spielt dagegen keine Rolle. Daher werden häufig einfach die Äquivalenzklassen $[x] = \{y \mid x \, R_L \, y\}$ der durch

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

definierten Relation R_L als Zustände des Minimalautomaten verwendet. Dies führt auf den so genannten **Äquivalenzklassenautomaten** $M_{R_L} = (Z, \Sigma, \delta, [\varepsilon], E)$ mit

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

Die Relation R_L gibt uns eine Möglichkeit an die Hand, herauszufinden, ob eine gegebene Sprache L regulär ist oder nicht. Offensichtlich gibt es genau dann einen DFA für L, wenn R_L die Menge Σ^* in endlich viele Klassen zerlegt.

Satz 1.57 (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. *Ist L regulär, so gibt es bis auf Isomorphie nur einen DFA für L mit einer minimalen Anzahl von Zuständen.*

Beispiel 1.58 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. ◁

1.6 Grammatiken

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

Beispiel 1.59 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:

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

◁

Definition 1.60 *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 1.61 *Seien $\alpha, \beta \in (V \cup \Sigma)^*$.*

*a) Wir sagen, β **ist aus** α **in** G **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) Die durch G **erzeugte Sprache** ist*

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

Das n-fache Produkt von $\Rightarrow_G$ ist $\Rightarrow_G^n$, d.h. $\alpha \Rightarrow_G^n \beta$ besagt, dass β aus α **in** n **Schritten ableitbar** ist. Die reflexiv-transitive Hülle von $\Rightarrow_G$ ist $\Rightarrow_G^*$, d.h. $\alpha \Rightarrow_G^* \beta$ besagt, dass β aus α (in endlich vielen Schritten) **ableitbar** ist. Ein Wort $\alpha \in (V \cup \Sigma)^*$ heißt **Satzform**, wenn es aus dem Startsymbol S ableitbar ist.

Definition 1.62 *Eine **Ableitung** von x ist eine Folge*

$$\sigma = (l_0, u_0, r_0), \ldots, (l_m, u_m, r_m)$$

von Tripeln (l_i, u_i, r_i) mit $(l_0, u_0, r_0) = (\varepsilon, S, \varepsilon)$, $l_m u_m r_m = x$ und

$$l_i \underline{u_i} r_i \Rightarrow l_{i+1} u_{i+1} r_{i+1} \text{ für } i = 0, \ldots, m-1.$$

*Die **Länge** von σ ist m. Wir notieren eine Ableitung σ wie oben auch in der Form*

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

Beispiel 1.63 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 1.64 *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 1.65 *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 1.66 $\mathsf{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}$.

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 $x = x_1 \cdots x_n \in \Sigma^*$:

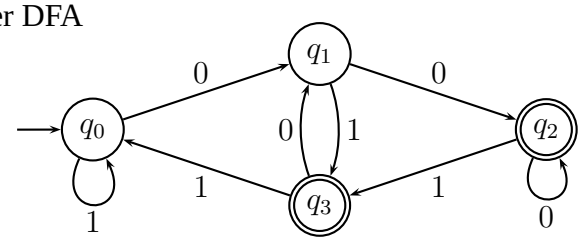
$$\begin{aligned} x \in L(M) &\Leftrightarrow \exists\, q_1, \ldots, q_{n-1} \in Z\ \exists\, q_n \in E : \delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \ldots, n \\ &\Leftrightarrow \exists\, q_1, \ldots, q_n \in V : q_{i-1} \to_G x_i q_i \text{ für } i = 1, \ldots, n \text{ und } q_n \to_G \varepsilon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

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 1.67 Der DFA



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

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

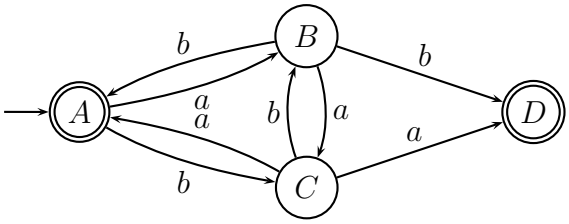
Umgekehrt führt die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ mit

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

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



◁

1.7 Das Pumping-Lemma

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

Theorem 1.68 (Pumping-Lemma für reguläre Sprachen) *Zu jeder regulären Sprache L gibt es eine Zahl l, so dass sich alle Wörter $x \in L$ mit $|x| \geq l$ in $x = uvw$ zerlegen lassen mit*

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

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

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

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

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

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

ist. □

Beispiel 1.69 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 1.70 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 1.68 aufgeführten Eigenschaften verletzt ist.

Beispiel 1.71

- 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.

- 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.

- Schließlich ist auch die Sprache
$$L = \{a^p \mid p \text{ prim }\}$$
nicht regulär, da sich sonst jede Primzahl p einer bestimmten Mindestgröße l als Summe von natürlichen Zahlen $u + v + w$ darstellen ließe, so dass $v \geq 1$, $u + v \leq l$ und für alle $i \geq 0$ auch $u + iv + w$ prim ist. Insbesondere müsste also $u + (u + w)v + w$ eine Primzahl sein, was wegen
$$u + (u+w)v + w = (u+w)(v+1) = \underbrace{(p-v)}_{\geq p-l}\underbrace{(v+1)}_{\geq 2}$$
für alle Primzahlen $p \geq l + 2$ ausgeschlossen ist. ◁

Bemerkung 1.72 *Mit dem Pumping-Lemma können nicht alle Sprachen $L \notin \mathsf{REG}$ als nicht regulär nachgewiesen werden, da seine Umkehrung falsch ist. Beispielsweise hat die Sprache*
$$L = \{a^i b^j c^k \mid i = 0 \text{ oder } j = k\}$$
die Pumping-Zahl 1 (jedes Wort $x \in L$ mit Ausnahme von ε kann also „gepumpt“ werden), obwohl L nicht regulär ist (siehe Übungen).

Kapitel 2

Kontextfreie Sprachen

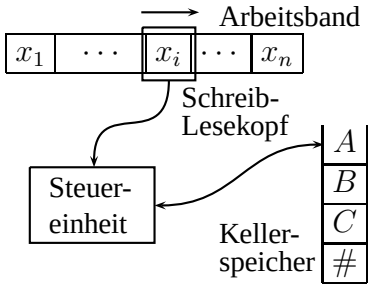
2.1 Kellerautomaten

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:

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

In diesem Abschnitt befassen wir uns mit der Frage, wie sich das Maschinenmodell des DFA erweitern lässt, um die Sprache L 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 (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 2.1 *Ein **Kellerautomat** (kurz: PDA; pushdown automaton) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ beschrieben, wobei*

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

Arbeitsweise eines PDA

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 (oberstes Kellerzeichen ist A_1).

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 $K \vdash K'$. 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 (Folge von Konfigurationen) von M bei Eingabe x gibt, die ausgehend von der **Startkonfiguration** $(q_0, x, \#)$ das gesamte Wort bis zum Ende liest und den Keller leert. Man beachte, dass bei leerem Keller kein weiterer Übergang mehr möglich ist.

Beispiel 2.2 Sei M der PDA $(\{q,p\},\{a,b\},\{A,\#\},\delta,q,\#)$ mit

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

Dann wird das Wort $x = aabb$ beispielsweise durch folgende Rechnung akzeptiert:

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

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

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

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

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

Ausgehend von der Startkonfiguration $(q, x, \#)$ sind nur die Anweisungen (1) oder (2) anwendbar. Da Anweisung (1) den Keller leert, ohne ein Zeichen zu lesen, muss x in diesem Fall das leere Wort $x = \varepsilon = a^0 b^0$ sein.

Falls M die Rechnung mit Anweisung (2) beginnt, muss M in den Zustand p gelangen, um den Keller leeren zu können. Da eine Rückkehr von p nach q nicht möglich ist, wechselt M den Zustand nur einmal, und zwar mittels Anweisung (4). Bis dahin kann M nur a's lesen, wobei für jedes gelesene a ein A eingekellert wird. Liest M bis zum Wechsel in den Zustand p insgesamt m a's, so muss die Rechnung wie folgt verlaufen:

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

mit $x_1 = x_2 = \cdots = x_m = a$ und $x_{m+1} = b$. Um den Keller zu leeren, muss M noch $m-1$ weitere b's lesen. Also muss $n = 2m$ und $x_{m+2} = \cdots = x_{2m} = b$ sein. x hat also auch in diesem Fall die Form $a^m b^m$. ◁

2.2 Äquivalenz von kontextfreien Grammatiken und Kellerautomaten

Als nächstes wollen wir zeigen, dass eine Sprache genau dann kontextfrei ist, wenn sie von einem PDA erkannt wird. Zuvor führen wir aber noch Links- und Rechtsableitungen sowie den Begriff des Syntaxbaums ein.

Definition 2.3 *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:

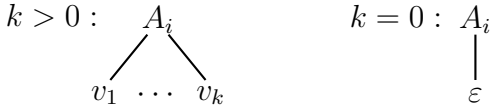
$$x \in L(G) \;\Leftrightarrow\; S \Rightarrow_L^* x \;\Leftrightarrow\; S \Rightarrow_R^* x.$$

Definition 2.4 *Sei G eine kontextfreie Grammatik. Dann ordnen wir 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:*

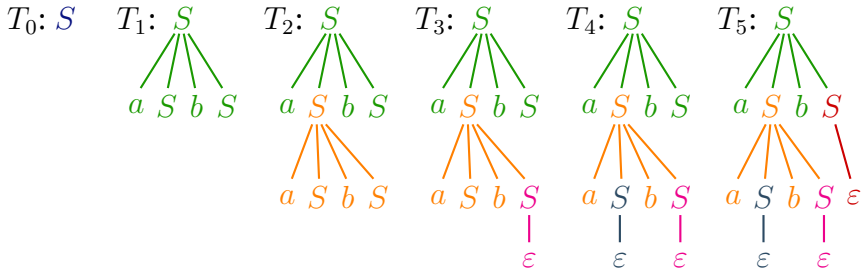


- *Hierbei stellen wir uns die Kanten von oben nach unten gerichtet und die Kinder $v_1 \cdots v_k$ von links nach rechts geordnet vor.*

Beispiel 2.5 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



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

◁

Bemerkung 2.6

- *Die Satzform α_i ergibt sich aus T_i, indem wir die Blätter von T_i von links nach rechts zu einem Wort zusammensetzen.*
- *Ableitungen, die sich nur in der Reihenfolge der Regelanwendungen unterscheiden, führen auf denselben Syntaxbaum.*
- *Aus einem Syntaxbaum ist die zugehörige Linksableitung ein- deutig rekonstruierbar.*
- *Daher führen unterschiedliche Linksableitungen auch auf unter- schiedliche Syntaxbäume.*
- *Linksableitungen und Syntaxbäume entsprechen sich also ein- eindeutig.*
- *Ebenso Rechtsableitungen und Syntaxbäume.*

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

Satz 2.7 $\mathsf{CFL} = \{L(M) \mid M \textit{ ist ein PDA}\}$.

Beweis Wir zeigen zuerst die Inklusion $\mathsf{CFL} \subseteq \{L(M) \mid M \text{ ist ein PDA}\}$.
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

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

gilt. Hierzu fügen wir die Anweisungen

$$\begin{aligned} q\varepsilon A &\to q\alpha, \quad A \to_G \alpha, \\ qaa &\to q\varepsilon, \quad a \in \Sigma. \end{aligned}$$

zu δ hinzu. Dann ist leicht zu sehen, dass sogar

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

gilt. 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 umgekehrte Inklusion $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathsf{CFL}$.
Idee: Konstruiere zu einem PDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Variablen X_{pAq}, $A \in \Gamma$, $p, q \in Z$, so dass

$$(p, x, A) \vdash^* (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^* x.$$

gilt. Hierzu fügen wir für jede Anweisung $puA \to q_1 B_1 \cdots B_k$, $k \geq 0$, die folgenden Regeln zu P hinzu:

$$X_{pAq_{k+1}} \to u X_{q_1 B_1 q_2} \cdots X_{q_k B_k q_{k+1}}, \quad q_2, \ldots, q_{k+1} \in Z.$$

Um damit alle Wörter $x \in L(M)$ aus S ableiten zu können, benötigen wir jetzt nur noch die Regeln $S \to X_{q_0 \# q}$, $q \in Z$. Damit hat M die Variablenmenge

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

und P enthält neben den Regeln $S \to X_{q_0 \# q}$, $q \in Z$, für jede Anweisung $puA \to q_1 B_1 \cdots B_k$ von M die folgenden Regeln:

$$X_{pAq_{k+1}} \to u X_{q_1 B_1 q_2} \cdots X_{q_k B_k q_{k+1}}, \; q_2, \ldots, q_{k+1} \in Z.$$

Dann folgt $L(G) = L(M)$ aus der Äquivalenz

$$(p, x, A) \vdash^* (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^* x,$$

deren Beweis wir gleich nachholen. Es gilt nämlich

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

Es bleibt zu zeigen, dass für alle $p, q \in Z$, $A \in \Gamma$ und $x \in \Sigma^*$ die Äquivalenz

$$(p, x, A) \vdash^* (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^* x.$$

gilt. Hierzu zeigen wir durch Induktion über m folgende stärkere Behauptung:

> Für alle $p, q \in Z$, $A \in \Gamma$, $x \in \Sigma^*$ und $m \geq 0$ gilt:
>
> $$(p, x, A) \vdash^m (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^m x.$$

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

$m \rightsquigarrow m + 1$**:** Wir zeigen unter der Induktionsvoraussetzung (IV), dass für alle $i \leq m$, $p, q \in Z$, $A \in \Gamma$ und $x \in \Sigma^*$ die Äquivalenz

$$(p, x, A) \vdash^i (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^i x$$

gilt, folgnde Induktionsbehauptung (IB):

Für alle $p, q \in Z$, $A \in \Gamma$ und $x \in \Sigma^*$ gilt

$$(p, x, A) \vdash^{m+1} (q, \varepsilon, \varepsilon) \text{ gdw. } X_{pAq} \Rightarrow^{m+1} x.$$

Wir beginnen mit dem Induktionsschritt für die Implikation von links nach rechts. Sei eine Rechnung $(p, x, A) \vdash^{m+1} (q, \varepsilon, \varepsilon)$ der Länge $m + 1$ gegeben und sei $puA \to q_1 B_1 \cdots B_k$, $k \geq 0$, die im ersten Rechenschritt ausgeführte Anweisung:

$$(p, x, A) \vdash (q_1, x', B_1 \cdots B_k) \vdash^m (q, \varepsilon, \varepsilon), \text{ wobei } x = ux' \text{ ist.}$$

Für $i = 2, \ldots, k$ sei q_i der Zustand von M, wenn B_i oberstes Kellerzeichen wird, und für $i = 1, \ldots, k$ sei u_i das Teilwort von x', das M zwischen den Besuchen von q_i und q_{i+1} verarbeitet (wobei $q_{k+1} = q$ ist).

Dann enthält P die Regel $X_{pAq} \to uX_{q_1 B_1 q_2} \cdots X_{q_k B_k q_{k+1}}$ und es gibt Zahlen $m_i \geq 1$ mit $m_1 + \cdots + m_k = m$ und

$$(q_i, u_i, B_i) \vdash^{m_i} (q_{i+1}, \varepsilon, \varepsilon) \text{ für } i = 1, \ldots, k.$$

Nach IV gibt es daher Ableitungen

$$X_{q_i B_i q_{i+1}} \Rightarrow^{m_i} u_i, \ i = 1, \ldots, k,$$

die wir zu der gesuchten Ableitung zusammensetzen können:

$$\begin{array}{lll} X_{pAq} & \Rightarrow & uX_{q_1 B_1 q_2} \cdots X_{q_{k-1} B_{k-1} q_k} X_{q_k B_k q} \\ & \Rightarrow^{m_1} & uu_1 X_{q_2 B_2 q_3} \cdots X_{q_{k-1} B_{k-1} q_k} X_{q_k B_k q} \\ & \vdots & \\ & \Rightarrow^{m_{k-1}} & uu_1 \cdots u_{k-1} X_{q_k B_k q} \\ & \Rightarrow^{m_k} & uu_1 \cdots u_k = x. \end{array}$$

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

$$X_{pAq} \Rightarrow \alpha \Rightarrow^m x.$$

Wegen $X_{pAq} \to_G \alpha$ gibt es eine Anweisung $puA \to q_1 B_1 \cdots B_k$, $k \geq 0$, und Zustände $q_1, \ldots, q_k \in Z$ mit

$$\alpha = uX_{q_1 B_1 q_2} \cdots X_{q_{k-1} B_{k-1} q_k} X_{q_k B_k q_{k+1}}, \text{ wobei } q_{k+1} = q \text{ ist.}$$

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

$$X_{q_i B_i q_{i+1}} \Rightarrow^{m_i} u_i \ (i = 1, \ldots, k), \text{ wobei } q_{k+1} = q \text{ ist.}$$

Nach IV gibt es somit Rechnungen

$$(q_i, u_i, B_i) \vdash^{m_i} (q_{i+1}, \varepsilon, \varepsilon), \ i = 1, \ldots, k,$$

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

$$\begin{aligned}
(q, uu_1 \cdots u_k, A) &\vdash \quad (q_1, u_1 \cdots u_k, B_1 \cdots B_k) \\
&\vdash^{m_1} \quad (q_2, u_2 \cdots u_k, B_2 \cdots B_k) \\
&\ \ \vdots \\
&\vdash^{m_{k-1}} (q_k, u_k, B_k) \\
&\vdash^{m_k} \quad (q, \varepsilon, \varepsilon).
\end{aligned}$$

□

Beispiel 2.8 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

$$\begin{aligned}
\delta:\ & qaa \to q\varepsilon,\ (0) \quad q\varepsilon S \to qaSbS,\ (1') \\
& qbb \to q\varepsilon,\ (0') \quad q\varepsilon S \to qa \qquad (2')
\end{aligned}$$

und der Linksableitung

$$S \underset{(1)}{\Rightarrow} aSbS \underset{(2)}{\Rightarrow} aabS \underset{(2)}{\Rightarrow} aaba$$

entspricht dann die Rechnung

$$\begin{aligned}
(q, aaba, S) &\underset{(1')}{\vdash} (q, aaba, aSbS) \underset{(0)}{\vdash} (q, aba, SbS) \\
&\underset{(2')}{\vdash} (q, aba, abS) \quad \underset{(0)}{\vdash} (q, ba, bS) \\
&\underset{(0')}{\vdash} (q, a, S) \ \underset{(2')}{\vdash} (q, a, a) \ \underset{(0)}{\vdash} (q, \varepsilon, \varepsilon).
\end{aligned}$$

◁

Beispiel 2.9 Sei M der PDA $(\{p,q\},\{a,b\},\{A,\#\},\delta,p,\#)$ mit

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

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

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

und den Regeln P:

$$\begin{array}{llll} S \to X_{p\#p}, X_{p\#q}, (0,0') & X_{pAp} \to aX_{pAp}X_{pAp}, \ (3') & X_{pAq} \to b, (4') \\ X_{p\#q} \to \varepsilon, \quad (1') & X_{pAp} \to aX_{pAq}X_{qAp}, \ (3'') & X_{qAq} \to b. \ (5') \\ X_{p\#p} \to aX_{pAp}, \quad (2') & X_{pAq} \to aX_{pAp}X_{pAq}, \ (3''') & \\ X_{p\#q} \to aX_{pAq}, \quad (2'') & X_{pAq} \to aX_{pAq}X_{qAq}, \ (3'''') & \end{array}$$

Der Rechnung

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

entspricht dann die Ableitung

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

2.3 Das Wortproblem für kontextfreie Grammatiken

Wie wir im letzten Abschnitt gesehen haben, können wir zwar zu jeder kontextfreien Grammatik G einen PDA M mit $L(M) = L(G)$ konstruieren. Dabei ist M bei Eingabe von G auch effizient berechenbar. Dennoch liefert dieser Ansatz keinen effizienten Algorithmus für das Wortproblem, da M nichtdeterministisch ist. 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 $x \in L(G)$?

2.3.1 Chomsky-Normalform

Um das Wortproblem für kontextfreie Grammatiken zu lösen, transformieren wir die gegebene Grammatik zunächst in Chomsky-Normalform, die wie folgt definiert ist.

Definition 2.10 *Eine Grammatik (V, Σ, P, S) ist in **Chomsky-Normalform (CNF)**, falls alle Regeln die Form $A \to BC$ oder $A \to a$ haben.*

Außer einer effizienten Lösung des Wortproblems für kontextfreie Sprachen ermöglichen CNF-Grammatiken den Beweis des Pumping-Lemmas für kontextfreie Sprachen, mit dem sich viele Sprachen als nicht kontextfrei nachweisen lassen. Um eine gegebene Grammatik in Chomsky-Normalform zu bringen, entfernen wir zuerst alle ε-Produktionen.

Satz 2.11 *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.

Algorithmus 2.12 BESTIMMUNG VON $E = \{A \in V \mid A \Rightarrow^* \varepsilon\}$

Eingabe: $G = (V, \Sigma, P, S)$
  $E \leftarrow \emptyset$
  $E' \leftarrow \{A \in V \mid A \to_G \varepsilon\}$
  while $E' \neq E$ **do**
    $E \leftarrow E'$
    $E' \leftarrow E \cup \{A \in V \mid \exists B_1, \ldots, B_k \in E : A \to_G B_1 \cdots B_k\}$
  end
Ausgabe: 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 2.13 Betrachte die Grammatik $G = (\{S, T, U, X, Y, Z\}, \{a, b, c\}, P, S)$ mit

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

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

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

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:

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

◁

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 2.14 $\mathsf{REG} \subsetneq \mathsf{CFL} \subseteq \mathsf{CSL} \subseteq \mathsf{RE}$.

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 entfernen wir sämtliche Variablenumbenennungen.

Definition 2.15 *Regeln der Form $A \to B$ heißen **Variablenumbenennungen**.*

Satz 2.16 *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 2.17 Ausgehend von den Produktionen

$$\begin{array}{lll} P\colon S \to aY, bX, Z; & Y \to b, bS, aYY; & T \to U; \\ \quad X \to a, aS, bXX; & Z \to c, S, T, cZ; & U \to abc \end{array}$$

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:

$$\begin{array}{lll} S \to aY, bX, c, T, cS; & Y \to b, bS, aYY; & T \to U; \\ X \to a, aS, bXX; & U \to abc. & \end{array}$$

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

$$\begin{array}{lll} S \to aY, bX, c, T, cS; & Y \to b, bS, aYY; & T \to abc; \\ X \to a, aS, bXX; & U \to abc. & \end{array}$$

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

$$\begin{array}{lll} S \to abc, aY, bX, c, cS; & Y \to b, bS, aYY; & T \to abc; \\ X \to a, aS, bXX; & U \to abc. & \end{array}$$

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 2.18 *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 2.19 In der Produktionenmenge

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

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:

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

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 $Y \to AY'$, $Y' \to YY$:

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

◁

2.3.2 Der CYK-Algorithmus

Als erste Anwendung der Chomsky-Normalform stellen wir einen effizienten Algorithmus zur Entscheidung des Wortproblems für kontextfreie Grammatiken vor.

Satz 2.20 *Das Wortproblem für kontextfreie Grammatiken ist effizient entscheidbar.*

Beweis Sei 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 bringen wir G in Chomsky-Normalform und setzen den nach seinen Autoren Cocke, Younger und Kasami benannten CYK-Algorithmus auf das Paar (G, x) an.

Der CYK-Algorithmus bestimmt für $l = 1, \ldots, n$ die Mengen

$$V_{l,k} = \{A \in V \mid A \Rightarrow^* x_k \cdots x_{k+l-1}\},\ k = 1, \ldots, n-l+1.$$

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

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

und für $l = 2, \cdots, n$ ist

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

Eine Variable A gehört also genau dann zu $V_{l,k}$, falls eine Zahl $l' \in \{1, \ldots, l-1\}$ und eine Regel $A \to BC$ existieren, so dass $B \in V_{l',k}$ und $C \in V_{l-l',k+l'}$ ist. Da der Zeitaufwand für die Berechnung der Menge $V_{l,k}$ 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

Beispiel 2.21 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 $V_{l,k}$:

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

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

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

◁

2.4 Das Pumping-Lemma für kontextfreie Sprachen

Als zweite Anwendung der Chomsky-Normalform beweisen wir in diesem Abschnitt das Pumping-Lemma für kontextfreie Sprachen. Dabei nützen wir die Tatsache aus, dass die Syntaxbäume einer CNF-Grammatik Binärbäume sind (d.h. jeder Knoten hat maximal 2 Kinder).

Definition 2.22 *Die* Tiefe *eines Baumes ist die maximale Pfadlänge von der Wurzel zu einem Blatt.*

Lemma 2.23 *Ein Binärbaum B der Tiefe $\leq k$ 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 $\leq k+1$. 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 höchstens 2^{k+1} Blätter. □

Korollar 2.24 *Ein Binärbaum B mit mehr als 2^{k-1} Blättern hat mindestens Tiefe k.*

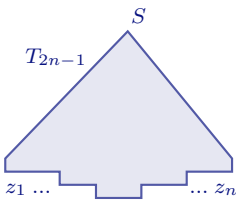
Beweis Würde B mehr als 2^{k-1} Blätter und eine Tiefe $\leq k-1$ besitzen, so würde dies im Widerspruch zu Lemma 2.23 stehen.

Satz 2.25 (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*

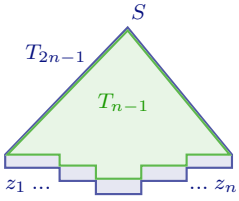
1. $vx \neq \varepsilon$,
2. $|vwx| \leq l$ *und*
3. $uv^iwx^iy \in L$ *für alle* $i \geq 0$.

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

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

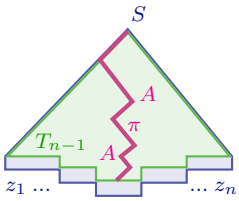


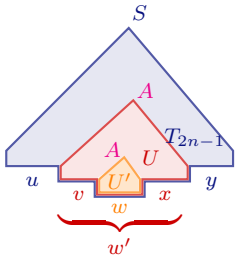
Da G in CNF ist, werden hierbei $n-1$ Regeln der Form $A \to BC$ und n Regeln der Form $A \to a$ angewandt. Folglich ist $m = 2n-1$ und z hat den Syntaxbaum T_{2n-1}.



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 $n \geq l$ wegen $l = 2^k > 2^{k-1}$ 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:

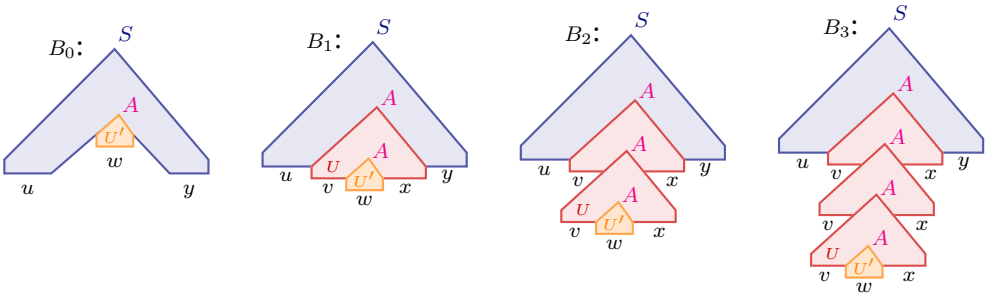




w' ist das Teilwort von $z = uw'y$, das von U erzeugt wird und w ist das Teilwort von $w' = vwx$, das von U' erzeugt wird. Jetzt bleibt nur noch zu zeigen, dass diese Zerlegung die geforderten 3 Eigenschaften erfüllt.

- Da U mehr Blätter hat als U', ist $vx \neq \varepsilon$, also ist Bedingung 1 erfüllt.
- 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 $|vwx| \leq l$, also ist auch Bedingung 2 erfüllt.

- 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 2.26 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 2.27 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. ◁

Theorem 2.28 *Die Klasse* CFL *ist abgeschlossen unter*

- *Vereinigung,*
- *Produkt und*
- *Sternhülle,*

aber nicht unter

- *Durchschnitt und*

- *Komplement.*

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. Die beiden Sprachen

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

sind kontextfrei. Nicht jedoch $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$. 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. □

2.5 Deterministisch kontextfreie Sprachen

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

Definition 2.29 *Ein Kellerautomat heißt* deterministisch, *falls die Relation* $\vdash$ *rechtseindeutig 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 2.30 Der PDA $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#)$ mit der Überführungsfunktion

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

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.

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

Definition 2.31 *Ein* deterministischer Kellerautomat *(kurz:* DPDA*) 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.
- *Zudem muss die Überführungsfunktion δ für alle $(q,a,A) \in Z \times \Sigma \times \Gamma$ folgende Bedingung erfüllen:*

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

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

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

Bemerkung 2.32 *Die Klasse* $\mathsf{DCFL} = \{L(M) \mid M$ *ist ein DPDA*$\}$ *(deterministic context free languages) ist eine echte Teilklasse von* CFL *(siehe Übungen).*

Als nächstes zeigen wir, dass DCFL unter Komplementbildung abgeschlossen ist. Beim Versuch, die End- und Nichtendzustände eines DPDA M einfach zu vertauschen, um einen DPDA $\overline{M}$ für $\overline{L(M)}$ zu erhalten, 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 2.33 *Jede Sprache $L \in$ 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 $L(M) = L$. Es gibt drei Gründe, die M daran hindern können, eine Eingabe zu Ende zu lesen:

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

Den 1. Grund schließen wir aus, indem wir ein neues Zeichen $\Box$ auf dem Kellerboden platzieren (dabei sei s der neue Startzustand):

$$\text{(a)}\quad s\varepsilon\# \to q_0\#\Box,$$

Den 2. und 3. Hinderungsgrund beseitigen wir durch Einführen eines neuen Fehlerzustands f (hierbei ist $\Gamma' = \Gamma \cup \{\Box\}$):

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

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

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

Zudem sehen wir im Falle einer unendlichen Folge von ε-Übergängen, bei denen ein Endzustand besucht wird, einen Umweg über den neuen Endzustand e vor:

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

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' neben den unter (a) bis (e) genannten Anweisungen noch

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

□

Beispiel 2.34 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)
	$f, \#$	f, A	f, B	$f, \square$	$e, \#$	e, A	e, B	$e, \square$
ε	–	–	–	–	$f\#$	–	–	–
a	$f\#$	fA	fB	$f\square$	–	–	–	–
b	$f\#$	fA	fB	$f\square$	–	–	–	–
c	$f\#$	fA	fB	$f\square$	–	–	–	–
Typ	(d)	(d)	(d)	(d)	(e)			

◁

Definition 2.35 *Für eine Sprachklasse $\mathcal{C}$ bezeichne co-$\mathcal{C}$ die Klasse $\{\bar{L} \mid L \in \mathcal{C}\}$ aller Komplemente von Sprachen in $\mathcal{C}$.*

Satz 2.36 $\mathsf{DCFL} = \mathrm{co}\text{-}\mathsf{DCFL}$, *d.h.* DCFL *ist unter Komplement abgeschlossen.*

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 $\overline{M}$ vorher noch einen ε-Übergang in 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

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

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

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

hinzufügen. □

Beispiel 2.37 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 $q_0, q_2 \in E$ und $q_1 \notin E$ (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 $q_0 \in E$ und $q_1, q_2 \notin E$ (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 $\overline{M}$ in diesem Fall das Wort a akzeptieren. ◁

Satz 2.38 *Die Klasse* DCFL *ist nicht abgeschlossen unter Durchschnitt, Vereinigung, Produkt und Sternhülle.*

Beweis Die beiden Sprachen

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

sind sogar deterministisch kontextfrei. Da $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ nicht kontextfrei ist, liegt diese Sprache natürlich erst recht 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 gehört aber nicht zu DCFL, d.h.

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

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. Betrachte hierzu die Sprachen

$$L_0 = \{0\},\ L_3 = \{a^i b^j c^k \mid i \neq j\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k\}.$$

Wir zeigen zuerst, dass die Sprache

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

Wäre nämlich $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA für L_5, so könnten wir M in einen DPDA $M' = (Z \cup \{s\}, \Sigma, \Gamma, \delta', s, \#, E)$ für die Sprache $L_3 \cup L_4$ transformieren (was wegen $L_3 \cup L_4 \notin$ DCFL nicht möglich ist).

Sei p der Zustand und γ der Kellerinhalt von M nach Lesen des Zeichens 0. Dann können wir δ' wie folgt definieren:

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

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

Bemerkung 2.39 *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

Kapitel 3

Kontextsensitive Sprachen

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

3.1 Kontextsensitive Grammatiken

Zur Erinnerung: Eine Grammatik $G = (V, \Sigma, P, S)$ heißt **kontextsensitiv**, falls für alle Regeln $\alpha \to \beta$ gilt: $|\beta| \geq |\alpha|$. 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.

Wie wir gesehen haben, ist die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ nicht kontextfrei. Das nächste Beispiel zeigt jedoch, dass L von einer kontextsensitiven Grammatik erzeugt werden kann. Daher ist die Klasse CFL echt in der Klasse CSL enthalten.

Beispiel 3.1 Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit $V = \{S, B, C\}$, $\Sigma = \{a, b, c\}$ und den Regeln

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

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

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

Allgemein gilt für alle $n \geq 1$:

$$S \underset{(1)}{\Rightarrow^n} a^{n-1}S(BC)^{n-1} \underset{(2)}{\Rightarrow} a^n(BC)^n \underset{(3)}{\Rightarrow^{\binom{n}{2}}} a^nB^nC^n$$
$$\underset{(4)}{\Rightarrow} a^nbB^{n-1}C^n \underset{(5)}{\Rightarrow^{n-1}} a^nb^nC^n \underset{(6)}{\Rightarrow} a^nb^ncC^{n-1} \underset{(7)}{\Rightarrow^{n-1}} a^nb^nc^n$$

Also gilt $a^nb^nc^n \in L(G)$ 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 $w = a^nb^nc^n$ ableitbar sind. ◁

3.2 Turingmaschinen

Um ein geeignetes Maschinenmodell für die kontextsensitiven Sprachen zu finden, führen wir zunächst das Rechenmodell der Turingmaschine (TM) ein. Eine TM darf während ihrer Rechnung ihre Eingabe überschreiben und beliebig viele Bandfelder als Speicher benutzen. Es ist leicht zu sehen, dass jede kontextsensitive Sprache von einer nichtdeterministischen 1-Band Turingmaschine (1-NTM) M akzeptiert wird. Dabei kann die Rechnung von M sogar auf den Bereich der Eingabe beschränkt werden (siehe Satz 3.9).

Definition 3.2

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

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

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

$$\|\delta(q, a_1, \ldots a_k)\| \leq 1.$$

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 3.3

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

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

und besagt, dass

- *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.*

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

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

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

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

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

Definition 3.4 *Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM und sei $x = x_1 \cdots x_n \in \Sigma^*$ eine Eingabe.*

*a) Die zugehörige **Startkonfiguration** ist*

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

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

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

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

Beispiel 3.5 Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{A, B, \sqcup\}$, $E = \{q_4\}$ und den Anweisungen

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

Dann führt M bei Eingabe $aabb$ folgende Rechnung aus:

$$\begin{aligned} q_0 aabb &\underset{(1)}{\vdash} A q_1 abb \underset{(2)}{\vdash} A a q_1 bb \underset{(4)}{\vdash} A q_2 a B b \\ &\underset{(5)}{\vdash} q_2 A a B b \underset{(7)}{\vdash} A q_0 a B b \underset{(1)}{\vdash} A A q_1 B b \\ &\underset{(3)}{\vdash} A A B q_1 b \underset{(4)}{\vdash} A A q_2 B B \underset{(6)}{\vdash} A q_2 A B B \\ &\underset{(7)}{\vdash} A A q_0 B B \underset{(8)}{\vdash} A A B q_3 B \underset{(9)}{\vdash} A A B B q_3 \sqcup \underset{(10)}{\vdash} A A B B q_4 \sqcup \end{aligned}$$

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

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

Da diese Rechnung nicht fortsetzbar ist und da M deterministisch ist, muss jede Rechnung von $M(abb)$ ein Anfangsstück dieser Rechnung sein. M kann bei Eingabe abb somit keinen Endzustand erreichen, weshalb abb nicht zu $L(M)$ gehört. 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. ◁

3.3 Linear beschränkte Automaten

Eine 1-NTM M, die nicht mehr Platz benötigt als ihr durch die Eingabe bereitgestellt wird, wird als LBA (linear beschränkter Automat) bezeichnet. Dass M bei Eingabe x nur $n = |x|$ Bandfelder besuchen darf, scheint auf den ersten Blick der Begriffsbildung „linear beschränkt“ zu widersprechen. Man kann jedoch zeigen, dass jede k-NTM, die bei Eingaben der Länge n nur linear viele (also $O(n)$) Bandfelder besucht, von einem LBA simuliert werden kann.

Damit ein LBA das Ende der Eingabe erkennen kann (und nicht versehentlich darüber hinausliest), muss das letzte Zeichen der Eingabe markiert werden. Das erste Zeichen der Eingabe braucht dagegen nicht markiert zu werden, da dies der LBA im ersten Rechenschritt selbst tun kann.

Definition 3.6

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 3.7 Es ist nicht schwer, die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ aus Beispiel 3.5 mit

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

in einen deterministischen LBA (kurz: **DLBA**) $M' = (Z, \hat{\Sigma}, \Gamma', \delta', q_0, E)$ für die Sprache $\{a^nb^n \,|\, n \geq 1\}$ 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 Anweisung $q_1\hat{b} \to q_2\hat{B}L\ (4a)$ hinzu:

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

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

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

◁

Definition 3.8 *Die Klasse der **deterministisch kontextsensitiven** Sprachen ist definiert als*

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

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. $B \in \mathsf{DCSL} \setminus \mathsf{CFL}$. Dass CFL in DCSL enthalten ist, wird in den Übungen gezeigt. Als nächstes beweisen wir, dass LBAs genau die kontextsensitiven Sprachen erkennen.

Satz 3.9 $\mathsf{CSL} = \{L(M) \mid M \textit{ ist ein LBA}\}$.

Beweis Wir zeigen zuerst die Inklusion $\mathsf{CSL} \subseteq \{L(M) \mid M \text{ ist ein LBA}\}$. Sei $G = (V, \Sigma, P, S)$ eine kontextsensitive Grammatik. Dann wird $L(G)$ von folgendem LBA M akzeptiert (o.B.d.A. sei $\varepsilon \notin L(G)$):

1 Markiere das erste Eingabezeichen.
2 Wähle (nichtdeterministisch) eine Regel $\alpha \to \beta$ aus P.

3 Wähle ein beliebiges Vorkommen von β auf dem Band. (Falls β auf dem Band 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 noch das (markierte) Startsymbol, so halte in einem Endzustand; sonst gehe zurück zu Schritt 2.

Nun ist leicht zu sehen, dass M wegen $|\beta| \geq |\alpha|$ tatsächlich linear beschränkt 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 von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathsf{CSL}$ 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:

$$\begin{array}{lrll}
P\colon & S & \to A(\hat{a}, a), & (1)\\
& A & \to A(a, a), & (2)\\
& A & \to (q_0 a, a), & (3)\\
& (qc, a) & \to (q'c', a), \quad \text{falls } qc \to_M q'c'N & (4)\\
& (qc, a)(d, b) & \to (c', a)(q'd, b), \quad \text{falls } qc \to_M q'c'R & (5)\\
& (d, a)(qc, b) & \to (q'd, a)(c', b), \quad \text{falls } qc \to_M q'c'L & (6)\\
& (qc, a) & \to a, \quad \text{falls } q \in E & (7)\\
& (c, a) & \to a. & (8)
\end{array}$$

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:

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

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

$$\begin{aligned}
S &\underset{(1,2,3)}{\Rightarrow^n} (q_0 x_1, x_1)(x_2, x_2) \cdots (x_{n-1}, x_{n-1})(\hat{x}_n, x_n)\\
&\underset{(4,5,6)}{\Rightarrow^m} (a_1, x_1) \cdots (a_{i-1}, x_{i-1})(q a_i, x_i) \cdots (a_n, x_n)\\
&\underset{(7,8)}{\Rightarrow^n} x_1 \cdots x_n
\end{aligned}$$

Ganz ähnlich folgt auch $L(G) \subseteq L(M)$. □

Bemerkung 3.10

- *Eine einfache Modifikation des Beweises zeigt, dass 1-NTMs genau die Sprachen vom Typ 0 akzeptieren (siehe Übungen). Im nächsten Abschnitt werden wir sehen, dass sogar für alle $k \geq 1$ folgende Gleichheiten gelten:*

$$\begin{aligned}\mathsf{RE} &= \{L(M) \,|\, M \textit{ ist eine } k\textit{-NTM}\} = \{L(G) \,|\, G \textit{ ist eine Typ-0 Grammatik}\} \\ &= \{L(M) \,|\, M \textit{ ist eine } k\textit{-DTM}\} = \{L \,|\, L \textit{ ist semi-entscheidbar}\}.\end{aligned}$$

- *Bis heute ungelöst ist die Frage, ob jede kontextsensitive Sprache von einem DLBA erkannt wird, d.h. ob* DCSL *eine echte Teilklasse von* CSL *ist oder nicht. Diese Frage ist als* ***LBA-Problem*** *bekannt.*

Zum Schluss dieses Abschnitts betrachten wir (neben dem Wortproblem) die folgenden Problemstellungen für (durch Grammatiken oder Automaten) gegebene Sprachen L, L_1 und L_2:

- **Leerheitsproblem:** Ist $L = \emptyset$?
- **Schnittproblem:** Ist $L_1 \cap L_2 = \emptyset$?
- **Äquivalenzproblem:** Ist $L_1 = L_2$?

Solange es nur um die Frage geht, ob diese Probleme entscheidbar sind oder nicht, ist es unerheblich, ob beispielsweise eine reguläre Sprache durch einen endlichen Automaten, durch eine Grammatik oder durch einen regulären Ausdruck gegeben ist. Dies liegt daran, dass die verschiedenen Darstellungsformen *effektiv* ineinander transformierbar sind. Dagegen kann die gewählte Darstellungsform einen beträchtlichen Einfluss auf die *Komplexität* dieser Probleme haben.

Die folgende Tabelle zeigt, welche dieser Probleme entscheidbar sind und welche nicht.

Sprachklasse	Wort-problem	Leerheits-problem	Äquivalenz-problem	Schnitt-problem
regulär (DFA, NFA etc.)	ja	ja	ja	ja
det. kontextfrei (DPDA)	ja	ja	ja[a]	nein
kontextfrei (PDA, kfr. Gr.)	ja	ja	nein	nein
kontextsensitiv (LBA)	ja	nein	nein	nein
semi-entscheidbar (DTM, NTM)	nein	nein	nein	nein

[a] Bewiesen in 1997 von Géraud Sénizergues (Univ. Bordeaux).

Eine Sonderrolle spielen hierbei die entscheidbaren Sprachen, da sich diese nicht syntaktisch mittels Automaten oder Grammatiken charakterisieren lassen. Vielmehr sind

die entscheidbaren Sprachen nur semantisch (etwa durch Turingmaschinen, die bei jeder Eingabe halten) charakterisierbar.

Folgende Tabelle gibt einen Überblick über die Abschlusseigenschaften der Klassen REG, DCFL, CFL, DCSL, CSL und RE. In der VL 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

Kapitel 4

Entscheidbare und semi-entscheidbare Sprachen

In diesem Kapitel beschäftigen wir uns mit der Klasse RE, die alle Typ-0 Sprachen enthält. 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 4.1

a) Eine DTM M ***hält*** *bei Eingabe x, falls $M(x)$ eine Konfiguration K erreicht, die keine Folgekonfiguration hat.*

b) Eine DTM M ***entscheidet*** *eine Eingabe x, falls $M(x)$ nach endlich vielen Schritten hält oder in einen Endzustand gelangt.*

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) Eine Sprache L heißt ***semi-entscheidbar****, falls eine DTM M mit $L(M) = L$ existiert.*

Bemerkung 4.2

- *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 $x \notin L$.*
- *Wir werden später sehen, dass genau die Typ-0 Sprachen semi-entscheidbar sind.*

Aus historischen Gründen werden die entscheidbaren Sprachen auch **rekursiv** (engl. *recursive*) genannt. Wir fassen die entscheidbaren Sprachen in der Klasse REC zusammen:

$$\mathsf{REC} = \{L(M) \mid M \text{ ist eine DTM, die jede Eingabe entscheidet}\}.$$

Dann gilt

$$\mathsf{REG} \subseteq \mathsf{DCFL} \subseteq \mathsf{CFL} \subseteq \mathsf{DCSL} \subseteq \mathsf{CSL} \subseteq \mathsf{REC} \subseteq \mathsf{RE}.$$

Wir wissen bereits, dass die ersten drei Inklusionen $\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DCSL}$ echt sind. In diesem Abschnitt werden wir zeigen, dass auch die Inklusion $\mathsf{REC} \subsetneq \mathsf{RE}$ echt ist. Dass CSL eine echte Teilklasse von REC ist, wird in den Übungen gezeigt. Wie bereits erwähnt, ist die Frage, ob alle kontextsensitiven Sprachen auch deterministisch kontextsensitiv sind, bis heute ungelöst (LBA-Problem). Wir wenden uns nun der Berechnung von Funktionen zu.

Definition 4.3

*a) 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 $M(x) = f(x)$.

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

Aus historischen Gründen werden berechenbare Funktionen auch **rekursiv** genannt. 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 nur eine partielle Funktion.

Definition 4.4

*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 für alle $x \in \Sigma^*$ gilt:*

- *im Fall $f(x) = \uparrow$ hält M bei Eingabe x nicht,*
- *andernfalls muss $M(x) = f(x)$ (siehe Definition 4.3) gelten.*

Wir fassen die (partiellen) berechenbaren Funktionen in folgenden Klassen zusammen:

$$\mathsf{FREC}_t = \{f \mid f \text{ ist eine totale berechenbare Funktion}\},$$
$$\mathsf{FREC}_p = \{f \mid f \text{ ist eine partielle berechenbare Funktion}\}.$$

Dann gilt

$$\mathsf{FREC}_t \subsetneq \mathsf{FREC}_p.$$

Beispiel 4.5 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

$$f_1(x) = 0, \quad f_2(x) = x, \quad f_3(x) = x^+ \qquad \text{und} \qquad f_4(x) = \begin{cases} \uparrow, & x = \varepsilon, \\ y, & x = y^+. \end{cases}$$

Da f_1, f_2, f_3, f_4 berechenbar sind, gehören die totalen Funktionen f_1, f_2, f_3 zu FREC_t, 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 4.6

*(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 ihre **partielle charakteristische Funktion** $\hat{\chi}_A : \Sigma^* \to \{0, 1, \uparrow\}$ berechenbar ist. Letztere ist wie folgt definiert:*

$$\hat{\chi}_A(x) \;=\; \begin{cases} 1, & x \in A, \\ \uparrow, & x \notin A. \end{cases}$$

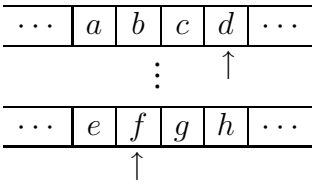
Beweis Siehe Übungen.

Definition 4.7 *Eine Sprache $A \subseteq \Sigma^*$ heißt **rekursiv aufzählbar**, falls $A = \emptyset$ oder das Bild $img(f)$ einer berechenbaren Funktion $f : \Gamma^* \to \Sigma^*$ für ein beliebiges Alphabet Γ ist.*

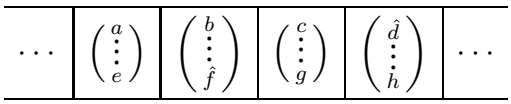
Satz 4.8 *Folgende Eigenschaften sind äquivalent:*

1. *A ist semi-entscheidbar (d.h. A wird von einer DTM akzeptiert),*
2. *A wird von einer 1-DTM akzeptiert,*
3. *A ist vom Typ 0,*
4. *A wird von einer NTM akzeptiert,*
5. *A ist rekursiv aufzählbar.*

Beweis $1) \Rightarrow 2)$**:** Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-DTM mit $L(M) = A$. Wir konstruieren eine 1-DTM $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$ für 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 $\hat{}$ markierten Zeichen (z.B. $\hat{a}_1, \ldots, \hat{a}_k$) gefunden hat.
- Diese Zeichen speichert M' in ihrem Zustand.
- Anschließend geht M' wieder nach links und realisiert dabei die durch $\delta(q, a_1, \ldots, a_k)$ vorgegebene Anweisung von M.
- Den aktuellen Zustand q von M speichert M' ebenfalls in ihrem Zustand.

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

$2) \Rightarrow 3)$**:** Ähnlich wie beim Beweis der Inklusion

$$\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \{L(G) \mid G \text{ ist eine Typ-1 Grammatik}\}$$

lässt sich zu einer 1-NTM (und damit auch zu einer 1-DTM) M eine Typ-0 Grammatik G mit $L(G) = L(M)$ konstruieren (siehe Übungen).

$3) \Rightarrow 4)$**:** Ähnlich wie beim Beweis der Inklusion

$$\{L(G) \mid G \text{ ist eine Typ-1 Grammatik}\} \subseteq \{L(M) \mid M \text{ ist ein LBA}\}$$

lässt sich zu einer Typ-0 Grammatik G eine 1-NTM M mit $L(M) = L(G)$ konstruieren (siehe Übungen).

$4) \Rightarrow 5)$**:** Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM und sei $A = L(M) \neq \emptyset$. Kodieren wir eine Konfiguration $K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ 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.

$5) \Rightarrow 1)$**:** 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 4.9 *A ist genau dann entscheidbar, wenn A und $\bar{A}$ semi-entscheidbar sind, d.h.* $\mathsf{REC} = \mathsf{RE} \cap \text{co-RE}$.

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

4.1 Kodierung (Gödelisierung) von Turingmaschinen

Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine 1-DTM mit Eingabealphabet $\Sigma = \{0, 1, \#\}$, Arbeitsalphabet $\Gamma = \{a_0, \ldots, a_l\}$ (o.B.d.A. sei $a_0 = 0$, $a_1 = 1$, $a_2 = \#$, $a_3 = \sqcup$), sowie Zustandsmenge $Z = \{q_0, \ldots, q_m\}$ (o.B.d.A. sei $E = \{q_m\}$). Dann können wir jede Anweisung der Form $q_i a_j \to q_{i'} a_{j'} D$ durch folgendes Wort kodieren

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#b_D\#.$$

Dabei ist $bin(n)$ die Binärdarstellung von n und

$$b_D = \begin{cases} 0, & D = N, \\ 1, & D = L, \\ 10, & D = R. \end{cases}$$

Über dem Alphabet $\{0, 1, \#\}$ lässt sich nun M durch die Folge ihrer kodierten Anweisungen kodieren. Wenn wir die Zeichen $0, 1, \#$ binär kodieren (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.

Ganz analog lassen sich auch k-NTMs sowie Konfigurationen und Rechnungen (Konfigurationsfolgen) von k-NTMs kodieren. Umgekehrt können wir jedem Binärstring $w \in \{0,1\}^*$ eine TM M_w wie folgt zuordnen (M_0 sei eine beliebige 1-DTM):

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

M_w ist durch Angabe von w bis auf die Benennung ihrer Zustände und Arbeitszeichen eindeutig bestimmt.

4.2 Das Halteproblem

Definition 4.10

*a) Das **Halteproblem** ist die Sprache*

$$\mathrm{H} = \left\{ w\#x \;\middle|\; \begin{array}{l} w, x \in \{0,1\}^* \textit{ und } M_w \textit{ ist eine} \\ \textit{1-DTM, die bei Eingabe } x \textit{ hält.} \end{array} \right\}$$

χ_{H}	x_1	x_2	x_3	$\cdots$
w_1	0	1	0	$\cdots$
w_2	0	1	1	$\cdots$
w_3	1	1	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

*b) Das **spezielle Halteproblem** ist die Sprache*

$$\mathrm{K} = \left\{ w \;\middle|\; \begin{array}{l} w \in \{0,1\}^* \textit{ und } M_w \textit{ ist eine} \\ \textit{1-DTM, die bei Eingabe } w \textit{ hält.} \end{array} \right\}$$

χ_{K}				
w_1	**0**			
w_2		**1**		
w_3			**0**	
$\vdots$				$\ddots$

Satz 4.11 $\mathrm{K} \in \mathsf{RE} \setminus \mathsf{REC}$.

Beweis Wir zeigen zuerst $\mathrm{K} \in \mathsf{RE}$. Sei w_0 die Kodierung einer 1-DTM, die bei jeder Eingabe (sofort) hält und betrachte die Funktion

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

Da f berechenbar und $img(f) = \mathrm{K}$ ist, folgt $\mathrm{K} \in \mathsf{RE}$. Um $\mathrm{K} \notin \mathsf{REC}$ zu beweisen, 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 entscheidbar. Dann ex. eine 1-DTM $\hat{M}$, die bei Eingabe x genau dann hält, wenn $x \notin \mathrm{K}$ ist:

$$\hat{M}(x) \text{ hält } \Leftrightarrow x \notin \mathrm{K} \qquad (**)$$

χ_{H}	x_1	x_2	x_3	$\cdots$
w_1	0	1	0	$\cdots$
w_2	0	1	1	$\cdots$
w_3	1	1	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\hat{w}$	**1**	**0**	**1**	$\cdots$

Für die Kodierung $\hat{w}$ von $\hat{M}$ folgt dann aber

$$\hat{w} \in \mathrm{K} \overset{(*)}{\Leftrightarrow} M_{\hat{w}}(\hat{w}) \text{ hält} \overset{(**)}{\Leftrightarrow} \hat{w} \notin \mathrm{K} \;\lightning \text{ (Widerspruch!)}$$

□

Korollar 4.12

(i) $\mathsf{REC} \subsetneq \mathsf{RE}$,

(ii) $\mathsf{RE} \neq \text{co-}\mathsf{RE}$,

(iii) $\mathrm{K} \notin \text{co-}\mathsf{RE}$.

Beweis

(i) $\mathsf{REC} \subsetneq \mathsf{RE}$: klar da $\mathrm{K} \in \mathsf{RE} - \mathsf{REC}$.

(ii) $\mathsf{RE} \neq \text{co-}\mathsf{RE}$: klar, da andernfalls wegen $\mathsf{REC} = \mathsf{RE} \cap \text{co-}\mathsf{RE}$ die Gleichheit $\mathsf{REC} = \mathsf{RE}$ folgen würde.

(iii) $\mathrm{K} \notin \text{co-}\mathsf{RE}$: Aus der Annahme $\mathrm{K} \in \text{co-}\mathsf{RE}$ würde wegen $\mathrm{K} \in \mathsf{RE}$ folgen, dass K entscheidbar ist (Widerspruch). □

Definition 4.13

*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 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:*

$$\forall L \in \mathcal{C} : L \leq A.$$

*c) Eine $\mathcal{C}$-harte Sprache A, die zu $\mathcal{C}$ gehört, heißt $\mathcal{C}$-**vollständig**.*

Beispiel 4.14 Es gilt $\mathrm{K} \leq \mathrm{H}$ mittels $f(w) = w\#w$:

$$\forall w \in \{0,1\}^* : w \in \mathrm{K} \Leftrightarrow w\#w \in \mathrm{H}.$$ ◁

Definition 4.15

*a) Eine Sprachklasse $\mathcal{C}$ heißt **unter $\leq$ abgeschlossen**, wenn für alle Sprachen A, B gilt:*

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

Satz 4.16 *Die Klasse* REC *ist unter $\leq$ abgeschlossen.*

Beweis Gelte $A \leq B$ mittels f und sei M eine 1-DTM, die χ_B berechnet. Betrachte folgende 1-DTM M':

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

Wegen

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

folgt

$$\chi_A(x) = \chi_B(f(x)) = M(f(x)) = M'(x).$$

Also berechnet M' die Funktion χ_A, d.h. $A \in \mathsf{REC}$. □

Der Abschluss von RE unter $\leq$ folgt analog (siehe Übungen). Wegen K $\leq$ H überträgt sich die Unentscheidbarkeit von K auf H.

Korollar 4.17 $\mathrm{H} \notin \mathsf{REC}$.

Beweis Aus der Annahme, dass H entscheidbar ist, folgt wegen K $\leq$ H, dass auch K entscheidbar ist (Widerspruch). □

Definition 4.18 *Das **Halteproblem bei leerem Band** ist die Sprache*

$$\mathrm{H}_0 = \left\{ w \;\middle|\; \begin{array}{l} w \in \{0,1\}^* \textit{ und} \\ M_w \textit{ ist eine 1-DTM,} \\ \textit{die bei Eing. } \varepsilon \textit{ hält.} \end{array} \right\}$$

χ_{H}	x_1	x_2	x_3	$\cdots$
w_1	**0**	1	0	$\cdots$
w_2	0	**1**	1	$\cdots$
w_3	1	1	**0**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

χ_{H_0}	
w_1	**0**
w_2	**0**
w_3	**1**
$\vdots$	$\vdots$

Satz 4.19 $\mathrm{H} \leq \mathrm{H}_0$.

Beweis Für eine 1-DTM M_w und ein Wort x sei $M_{w,x}$ eine 1-DTM, die bei Eingabe ε zuerst das Wort x auf das Band schreibt und dann $M_w(x)$ simuliert.
Sei $w_0 \notin \mathrm{H}_0$ und betrachte die Funktion

$$f(y) = \begin{cases} w', & y \text{ hat die Form } y = w\#x \text{ für eine 1-DTM } M_w \text{ und } w' \text{ ist} \\ & \text{die Kodierung von } M_{w,x}, \\ w_0, & \text{sonst.} \end{cases}$$

Offensichtlich ist f berechenbar und reduziert H auf H_0. □

Wegen $\mathrm{H} \leq \mathrm{H}_0$ überträgt sich die Unentscheidbarkeit von H auf H_0.

Korollar 4.20 $\mathrm{H}_0 \notin \mathsf{REC}$.

Beweis Wegen $\mathrm{H} \leq \mathrm{H}_0$ wäre mit H_0 auch H entscheidbar (Widerspruch). □

Frage 4.21 *Kann man einer TM ansehen, ob die von ihr berechnete Funktion eine gewisse Eigenschaft hat?*

Antwort 4.22 *Nein (es sei denn, die fragliche Eigenschaft ist trivial, d.h. keine oder jede berechenbare Funktion hat sie).*

Notation 4.23

1. *Mit* $\mathsf{FREC}_p(\Sigma)$ *bezeichnen wir die Klasse aller partiellen berechenbaren Funktionen* $f : \Sigma^* \to \Sigma^* \cup \{\uparrow\}$.
2. *Eine Eigenschaft* $\mathsf{F} \subseteq \mathsf{FREC}_p(\Sigma)$ *heißt* ***nichttrivial****, wenn es sowohl Funktionen* $f \in \mathsf{F}$ *mit dieser Eigenschaft als auch Funktionen* $g \in \mathsf{FREC}_p(\Sigma) - \mathsf{F}$ *ohne diese Eigenschaft gibt, d.h.* $\emptyset \subsetneq \mathsf{F} \subsetneq \mathsf{FREC}_p(\Sigma)$.

Satz 4.24 (Satz von Rice) *Sei* $\Sigma = \{0, 1, \#\}$. *Dann ist die Sprache*

$$L_{\mathsf{F}} = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l} M_w \textit{ ist eine DTM, die eine} \\ \textit{Funktion in } \mathsf{F} \textit{ berechnet} \end{array} \right\}$$

für jede nichttriviale Eigenschaft $\mathsf{F} \subseteq \mathsf{FREC}_p(\Sigma)$ *unentscheidbar.*

Beweis Wir reduzieren H_0 (bzw. $\overline{\mathrm{H}_0}$) auf L_{F}. Die Idee besteht darin, für eine gegebene 1-DTM M_w eine DTM $M_{w'}$ zu konstruieren mit

$$w \in \mathrm{H}_0 \text{ (bzw. } w \in \overline{\mathrm{H}_0}) \Leftrightarrow M_{w'} \text{ berechnet eine Funktion in } \mathsf{F}.$$

Hierzu lassen wir $M_{w'}$ bei Eingabe x zunächst einmal die 1-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 $x \in \Sigma^*$.
Damit die Reduktion gelingt, muss $M_{w'}$ im Fall $w \in \mathrm{H}_0$ eine Funktion $g \in \mathsf{FREC}_p(\Sigma)$ berechnen mit

$$g \in \mathsf{F} \Leftrightarrow u \notin \mathsf{F}.$$

Wegen $\emptyset \subsetneq \mathsf{F} \subsetneq \mathsf{FREC}_p(\Sigma)$ muss eine solche Funktion g existieren. Sei also M eine DTM, die g berechnet und betrachte die Reduktionsfunktion $h : \{0,1\}^* \to \{0,1\}^*$ mit

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

Dann ist h eine totale berechenbare Funktion und es gilt

$$\begin{aligned} w \in \mathrm{H}_0 &\Rightarrow M_{w'} \text{ berechnet } g, \\ w \notin \mathrm{H}_0 &\Rightarrow M_{w'} \text{ berechnet } u. \end{aligned}$$

Im Fall $g \in \mathsf{F}$ (d.h. $u \notin \mathsf{F}$) folgt daher

$$w \in \mathrm{H}_0 \Leftrightarrow w' \in L_{\mathsf{F}},$$

das heißt, h reduziert H_0 auf L_{F}. Im Fall $g \notin \mathsf{F}$ (d.h. $u \in \mathsf{F}$) gilt dagegen

$$w \in \mathrm{H}_0 \quad \Leftrightarrow \quad w' \notin L_{\mathsf{F}},$$

das heißt, h reduziert $\overline{\mathrm{H}_0}$ auf L_{F}. Da weder H_0 noch $\overline{\mathrm{H}_0}$ entscheidbar sind, folgt $L_{\mathsf{F}} \notin \mathsf{REC}$. □

Beispiel 4.25 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 die durch

$$\mathsf{F} = \{f \in \mathsf{FREC}_p(\Sigma) \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

beschriebene Eigenschaft nichttrivial ist. So hat beispielsweise die Funktion

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

diese Eigenschaft, nicht jedoch die konstante Funktion $g(x) = 0$. ◁

4.3 Das Postsche Korrespondenzproblem

Sei Σ ein beliebiges Alphabet mit $\# \notin \Sigma$.

Definition 4.26 *Das **Postsche Korrespondenzproblem** über Σ (**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 $\binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ und kodieren sie durch das Wort $x_1 \# y_1 \# \cdots \# x_k \# y_k$.

Beispiel 4.27 Die Instanz $I = \left(\begin{smallmatrix} a & ab & caa \\ aca & bc & aa \end{smallmatrix}\right)$ 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 4.28 *Für jedes Alphabet Σ gilt* $\text{PCP}_\Sigma \leq \text{PCP}_{\{0,1\}}$.

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

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

□

Beispiel 4.29 Die $\text{PCP}_{\{a,b,c\}}$-Instanz $I = \left(\begin{smallmatrix} a & ab & caa \\ aca & bc & aa \end{smallmatrix}\right)$ wird mittels f auf die äquivalente $\text{PCP}_{\{0,1\}}$-Instanz $f(I) = \left(\begin{smallmatrix} 0 & 010 & 11000 \\ 01100 & 10110 & 00 \end{smallmatrix}\right)$ reduziert. ◁

Satz 4.30 PCP *ist* RE*-vollständig und damit unentscheidbar.*

Beweis Sei $A \in \text{RE}$ und sei $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$ eine 1-DTM mit $L(M) = A$. Wir zeigen $A \leq \text{MPCP}_{\Sigma'}$ für $\Sigma' = \Gamma \cup Z \cup \{\langle, |, \rangle\}$. Wegen $\text{MPCP}_{\Sigma'} \leq \text{PCP}$ folgt hieraus $A \leq \text{PCP}$.

> *Idee:* Transformiere eine Eingabe $w \in \Sigma^*$ in eine Instanz $f(w) = \left(\begin{smallmatrix} x_1 \cdots x_k \\ y_1 \ldots y_k \end{smallmatrix}\right)$, 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.

Um dies zu erreichen, bilden wir $f(w)$ aus folgenden Wortpaaren:

1. $(\,\langle\,, \langle\,|\, z_0w)$, „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 $z_e \in E$, $a \in \Gamma$: (az_e, z_e), $(z_e a, z_e)$, „Löschregeln“
5. sowie für alle $z_e \in E$: $(z_e \,|\, \rangle, \rangle)$. „Abschlussregeln“

Ist nun

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

eine akzeptierende Rechnung von $M(w)$ mit $z_e \in E$, so lässt sich aus dieser leicht eine MPCP-Lösung α mit einem Lösungswort der Form

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

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

Zudem ist leicht zu sehen, dass sich auch umgekehrt zu jeder MPCP-Lösung α von $f(w)$ aus dem zugehörigen Lösungswort $x_{i_1} \cdots x_{i_n} = y_{i_1} \cdots y_{i_n}$ eine akzeptierende Rechnung von M bei Eingabe w ablesbar ist. Somit gilt

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

und da f offensichtlich berechenbar ist, folgt $A \leq \text{MPCP}_{\Sigma'}$. □

Beispiel 4.31 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

$$\begin{array}{l}\delta\colon q_0 a \to q_1 AR\ (1)\ \ q_1 a \to q_1 aR\ \ (2)\ \ q_1 B \to q_1 BR\ (3)\ q_1 b \to q_2 BL\ (4)\\ \quad\ q_2 a \to q_2 aL\ \ (5)\ q_2 B \to q_2 BL\ \ (6)\ \ q_2 A \to q_0 AR\ (7)\ q_0 B \to q_3 BR\ (8)\\ \quad\ q_3 B \to q_3 BR\ (9)\ q_3 \sqcup \to q_4 \sqcup N\ (10)\end{array}$$

Dann enthält $f(w)$ für alle $u \in \Gamma$ die Startregel $(\langle, \langle \,|\, z_0 w)$, die Kopierregeln (u, u), $(|, |)$, die Überführungsregeln

$$\begin{array}{ll}(q_0 a, A q_1), (q_1 a, a q_1), (q_1 B, B q_1), & (1, 2, 3)\\ (u q_1 b, q_2 u B), (|q_1 b, |q_2 \sqcup B), & (4)\\ (u q_2 a, q_2 u a), (|q_2 a, |q_2 \sqcup a), & (5)\\ (u q_2 B, q_2 u B), (|q_2 B, |q_2 \sqcup B), & (6)\\ (q_2 A, A q_0), (q_0 B, B q_3), (q_3 B, B q_3), & (7, 8, 9)\\ (q_3 \sqcup, q_4 \sqcup), (q_3 |, q_4 \sqcup |), & (10)\end{array}$$

die Löschregeln $(q_4 u, q_4)$, $(u q_4, q_4)$, und die Abschlussregel $(q_4 \,|\, \rangle, \rangle)$.

Der akzeptierenden Rechnung

$$q_0 ab \underset{(1)}{\vdash} A q_1 b \underset{(4)}{\vdash} q_2 AB \underset{(7)}{\vdash} A q_0 B \underset{(8)}{\vdash} AB q_3 \sqcup \underset{(10)}{\vdash} AB q_4 \sqcup$$

von $M(ab)$ entspricht dann das MPCP-Lösungswort

$$\begin{array}{l}\langle \,|\, q_0 ab \,|\, A q_1 b \,|\, q_2 AB \,|\, A q_0 B \,|\, AB q_3 \,|\, AB q_4 \sqcup \,|\, A q_4 \sqcup \,|\, q_4 \sqcup \,|\, q_4 \,|\, \rangle\\ \langle \,|\, q_0 ab \,|\, A q_1 b \,|\, q_2 AB \,|\, A q_0 B \,|\, AB q_3 \,|\, AB q_4 \sqcup \,|\, A q_4 \sqcup \,|\, q_4 \sqcup \,|\, q_4 \,|\, \rangle\end{array}$$

◁

Satz 4.32 *Das Schnittproblem für kontextfreie Grammatiken ist unentscheidbar.*

Beweis Wir reduzieren eine PCP-Instanz $I = \binom{x_1 \cdots x_k}{y_1 \ldots y_k}$ auf ein Grammatikpaar (G_1, G_2), so dass gilt:

$$I \in \text{PCP} \Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset.$$

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

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

Dann gilt

$$L(G_1) = \{0^{i_1} 10^{i_2} \cdots 10^{i_n} 11 x_{i_n} \cdots x_{i_1} \mid n \geq 1, 1 \leq i_1, \ldots, i_n \leq k\}$$

und

$$L(G_2) = \{0^{i_1} 10^{i_2} \cdots 10^{i_n} 11 y_{i_n} \cdots y_{i_1} \mid n \geq 1, 1 \leq i_1, \ldots, i_n \leq k\}.$$

Somit ist $L(G_1) \cap L(G_2)$ die Sprache

$$\{0^{i_1} 1 \cdots 10^{i_n} 11 x_{i_n} \cdots x_{i_1} \mid n \geq 1, x_{i_n} \cdots x_{i_1} = y_{i_n} \cdots y_{i_1}\}.$$

Folglich ist $\alpha = (i_n \cdots i_1)$ genau dann eine Lösung für I, wenn das Wort

$$0^{i_1} 1 \cdots 10^{i_n} 11 x_{i_n} \cdots x_{i_1} \in L(M_1) \cap L(M_2)$$

ist. Also vermittelt $f : I \mapsto (G_1, G_2)$ eine Reduktion von PCP auf das Schnittproblem für CFL. □

Korollar 4.33

(i) *Das Schnittproblem für DPDAs ist unentscheidbar.*

(ii) *Das Inklusionsproblem für DPDAs ist unentscheidbar.*

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. □

Korollar 4.34 *Für kontextfreie Grammatiken sind folgende Fragestellungen unentscheidbar:*

(i) *Ist G mehrdeutig?*

(ii) *Ist $L(G_1) = L(G_2)$?* *(Äquivalenzproblem)*

(iii) *Ist $L(G) = \Sigma^*$?* *(Ausschöpfungsproblem)*

Beweis

(i) Betrachte die Funktion $f(I) = G$ mit

$$G = (\{S, S_1, S_2\}, \{0, 1\}, P_1 \cup P_2 \cup \{S \to S_1, S_2\}, S),$$

wobei P_1 und P_2 die Regelmengen aus vorigem Beweis sind. Da alle von S_1 oder S_2 ausgehende Ableitungen eindeutig sind, ist G genau dann mehrdeutig, wenn es ein Wort $w \in L(G)$ gibt mit

$$S \Rightarrow S_1 \Rightarrow^* w \text{ und } S \Rightarrow S_2 \Rightarrow^* w.$$

Wie wir im 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. Also vermittelt die berechenbare Funktion $f : I \mapsto G$ eine Reduktion von PCP auf das Mehrdeutigkeitsproblem für CFLs.

(ii) Wir reduzieren das Inklusionsproblem für DPDAs auf das Äquivalenzproblem für kontextfreie Grammatiken. Es gilt

$$L_1 \subseteq L_2 \Leftrightarrow L_1 \cup L_2 = L_2.$$

Daher vermittelt die Funktion $f : (M_1, M_2) \mapsto (G, G')$, wobei G und G' kontextfreie Grammatiken mit $L(G) = L(M_1) \cup L(M_2)$ und $L(G') = L(M_2)$ sind, die gewünschte Reduktion.

(iii) Wir reduzieren das Komplement des Schnittproblems für DPDAs auf das Ausschöpfungsproblem für kontextfreie Grammatiken. 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 $L(G) = L(\overline{M}_1) \cup L(\overline{M}_2)$ ist, die gewünschte Reduktion. □

Theorem 4.35 *Das Leerheitsproblem für DLBAs ist unentscheidbar.*

Beweis Wir reduzieren das Komplement des Schnittproblems für CFL auf das Leerheitsproblem für DCSL. Zwei kontextfreie Grammatiken G_1 und G_2 lassen sich wie folgt in einen DLBA M mit $L(M) = L(G_1) \cap L(G_2)$ überführen (siehe Übungen):

Bestimme zunächst DLBAs M_i, $i = 1, 2$, mit $L(M_i) = L(G_i)$. Konstruiere daraus einen DLBA M für die Sprache $L(M_1) \cap L(M_2)$.

Dann gilt $L(G_1) \cap L(G_2) = \emptyset \Leftrightarrow L(M) = \emptyset$, d.h. die Funktion $f : (G_1, G_2) \mapsto M$ leistet 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 4.36

(i) Das Leerheitsproblem für kontextfreie Grammatiken ist entscheidbar.

(ii) Das Wortproblem für kontextsensitive Grammatiken ist entscheidbar.

Beweis

(i) Betrachte folgenden Algorithmus:

```
1  Eingabe: eine kontextfreie Grammatik G = (V, Σ, P, S)
2    U' ← ∅
3    repeat
4      U ← U'
5      U' ← {A | es gibt eine Regel A → α ∈ P mit α ∈ (Σ ∪ U)*}
6    until (U = U')
7    if S ∈ U then
8      output „L(G) ist nicht leer“ else
9      output „L(G) ist leer“ end
```

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

(ii) Siehe Übungen. □

Wir fassen die wichtigsten (Un-)Entscheidbarkeitsresultate nochmals zusammen.

Sprachklasse	Wortproblem $x \in L$?	Leerheitsproblem $L = \emptyset$?	Ausschöpfung $L = \Sigma^*$?	Äquivalenzproblem $L_1 = L_2$?	Schnittproblem $L_1 \cap L_2 \neq \emptyset$?	Inklusionsproblem $L_1 \subseteq L_2$
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).

Kapitel 5

Komplexitätsklassen und NP-Vollständigkeit

5.1 Zeitkomplexität

Der Laufzeit $time_M(x)$ einer NTM M bei Eingabe x ist die maximale Anzahl an Rechenschritten, die $M(x)$ ausführt, bevor sie hält.

Definition 5.1 *Sei M eine* NTM *und sei x eine Eingabe. Dann ist die **Laufzeit** von M bei Eingabe x definiert als*

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

*wobei $\max \mathbb{N} = \infty$ ist. Sei $t : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $t(n)$-**zeitbeschränkt**, falls für alle Eingaben x gilt:*

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

Wir fassen alle Sprachen und Funktionen, die in einer vorgegebenen Zeitschranke $t(n)$ entscheidbar bzw. berechenbar sind, in folgenden **Komplexitätsklassen** zusammen.

Definition 5.2 *Die Klasse* $\mathsf{DTIME}(t(n))$ *enthält alle in deterministischer Zeit $t(n)$ entscheidbaren Sprachen, d.h.*

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

Die Klasse $\mathsf{NTIME}(t(n))$ *enthält alle in nichtdeterministischer Zeit $t(n)$ entscheidbaren Sprachen, d.h.*

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

Die Klasse $\mathsf{FTIME}(t(n))$ *enthält alle in deterministischer Zeit $t(n)$ berechenbaren Funktionen, d.h.*

$$\mathsf{FTIME}(t(n)) = \{f \mid \exists\, t(n)\text{-zeitb. DTM } M, \text{ die } f \text{ berechnet}\}.$$

Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

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

Die nichtdeterministischen Klassen $\mathsf{NP}, \mathsf{NE}, \mathsf{NEXP}$ und die Funktionenklassen $\mathsf{FP}, \mathsf{FE}, \mathsf{FEXP}$ sind analog definiert.

5.2 Platzkomplexität

Als nächstes definieren wir den Platzverbrauch von NTMs. Intuitiv ist dies die maximale Anzahl aller während einer Rechnung besuchten Bandfelder. Wollen wir auch sublinearen Platz sinnvoll definieren, so muss das Eingabeband offensichtlich unberücksichtigt bleiben. Um sicherzustellen, dass eine NTM M das Eingabeband nicht als Speicher benutzt, verlangen wir, dass M die Eingabe nicht verändert und sich nicht mehr als ein Feld von ihr entfernt.

Definition 5.3 *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 5.4 *Sei M eine Offline-*NTM *und sei x eine Eingabe. Dann ist der* ***Platzverbrauch*** *von M bei Eingabe x definiert als*

$$\mathrm{space}_M(x) = \max\left\{ s \geq 1 \;\middle|\; \begin{array}{l} \exists K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \\ \text{mit } K_x \vdash^* K \text{ und } s = \sum_{i=2}^{k} |u_i a_i v_i| \end{array} \right\}.$$

*Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion. Dann ist M $s(n)$-**platzbeschränkt**, falls für alle Eingaben x gilt:*

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

Wir fassen alle Sprachen, die in einer vorgegebenen Platzschranke $s(n)$ entscheidbar sind, in folgenden **Platzkomplexitätsklassen** zusammen.

Definition 5.5 *Die Klasse* $\mathsf{DSPACE}(s(n))$ *enthält alle in deterministischem Platz* $s(n)$ *entscheidbaren Sprachen, d.h.*

$$\mathsf{DSPACE}(s(n)) = \{L(M) \mid M \textit{ ist eine } s(n)\textit{-platzb. Offline-}\mathrm{DTM}\}.$$

Die Klasse $\mathsf{NSPACE}(s(n))$ *enthält alle in nichtdeterministischem Platz* $s(n)$ *entscheidbaren Sprachen, d.h.*

$$\mathsf{NSPACE}(s(n)) = \{L(M) \mid M \textit{ ist eine } s(n)\textit{-platzb. Offline-}\mathrm{NTM}\}.$$

Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$$\begin{aligned}
\mathsf{L} &= \mathsf{DSPACE}(O(\log n)) = \bigcup_{c>0} \mathsf{DSPACE}(c \log n),\\
\mathsf{LINSPACE} &= \mathsf{DSPACE}(O(n)) = \bigcup_{c\geq 1} \mathsf{DSPACE}(cn),\\
\mathsf{PSPACE} &= \mathsf{DSPACE}(n^{O(1)}) = \bigcup_{c\geq 1} \mathsf{DSPACE}(n^c).
\end{aligned}$$

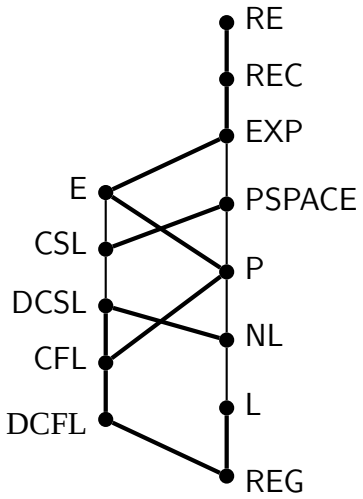
Die nichtdeterministischen Klassen NL, NLINSPACE und NPSPACE sind analog definiert, wobei letztere wegen

$$\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}(s^2(n))$$

mit PSPACE zusammenfällt (dies wird in der VL Komplexitätstheorie gezeigt).

Die Klassen der Chomsky-Hierarchie lassen sich wie folgt einordnen (eine dicke Linie deutet an, dass die Inklusion als echt nachgewiesen werden konnte):

$$\begin{aligned}
\mathsf{REG} &= \mathsf{DSPACE}(O(1)) = \mathsf{NSPACE}(O(1)) \subsetneq \mathsf{L},\\
\mathsf{DCFL} &\subsetneq \mathsf{LINTIME} \cap \mathsf{CFL} \subsetneq \mathsf{LINTIME} \subsetneq \mathsf{P},\\
\mathsf{CFL} &\subsetneq \mathsf{NLINTIME} \cap \mathsf{DTIME}(n^3) \subsetneq \mathsf{P},\\
\mathsf{DCSL} &= \mathsf{LINSPACE} \subseteq \mathsf{CSL},\\
\mathsf{CSL} &= \mathsf{NLINSPACE} \subseteq \mathsf{PSPACE} \cap \mathsf{E},\\
\mathsf{REC} &= \bigcup_f \mathsf{DSPACE}(f(n)) = \bigcup_f \mathsf{NSPACE}(f(n))\\
&= \bigcup_f \mathsf{DTIME}(f(n)) = \bigcup_f \mathsf{NTIME}(f(n)),
\end{aligned}$$



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 $\mathsf{DCSL} \subseteq \mathsf{P}$ gilt. Man kann jedoch zeigen, dass $\mathsf{CSL} \neq \mathsf{P} \neq \mathsf{DCSL}$ 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.

5.3 NP-Vollständigkeit und das P = NP-Problem

Wie wir im letzten Kapitel gesehen haben (siehe Satz 4.8), sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden. Die Frage, wieviel Zeit eine NTM gegenüber einer DTM einsparen kann, ist eines der wichtigsten offenen Probleme der Informatik. So enthält die Klasse NP viele für die Praxis überaus wichtige Probleme, von denen nicht bekannt ist, ob sie auch zu P gehören. Da jedoch nur Probleme in P als effizient lösbar gelten, hat das so genannte **P = NP-Problem**, also die Frage, ob alle NP-Probleme effizient lösbar sind, eine immense praktische Bedeutung.

Wie bereits erwähnt, ist diese Frage für den Platzverbrauch bereits gelöst:

Satz 5.6 (Satz von Savitch) *Jede $s(n)$-platzbeschränkte NTM kann von einer $s^2(n)$-platzbeschränkten DTM simuliert werden (ohne Beweis).*

Definition 5.7

a) Eine Sprache $A \subseteq \Sigma^*$ ist auf $B \subseteq \Gamma^*$ ***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) Eine Sprache 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 A, die zu $\mathcal{C}$ gehört, heißt $\mathcal{C}$-**vollständig**.*

d) NPC *bezeichnet die Klasse aller* NP*-vollständigen Sprachen.*

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. $\mathrm{K}, \mathrm{H}, \mathrm{H}_0$, PCP etc. sogar $\leq^p$-vollständig für RE sind.

Lemma 5.8

(i) Aus $A \leq^p B$ folgt $A \leq B$.

(ii) Die Reduktionsrelation $\leq^p$ ist reflexiv und transitiv (s. Übungen).

Satz 5.9 *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 5.10

(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 5.11 *Die Sprache*

$$A = \{w\#x\#0^m \mid \textit{die NTM } M_w \textit{ akzeptiert } x \textit{ in} \leq m \textit{ Schritten}\}$$

ist NP*-vollständig.*

Beweis Zunächst ist klar, dass $A \in \mathsf{NP}$ mittels folgender NTM M gilt:

> M simuliert bei Eingabe $w\#x\#0^m$ die NTM M_w bei Eingabe x für höchstens m Schritte. Falls M_w in dieser Zeit akzeptiert, akzeptiert M ebenfalls, andernfalls verwirft M.

Sei nun L eine beliebige NP-Sprache und sei M_w eine durch ein Polynom p zeitbeschränkte NTM mit $L(M_w) = L$. Dann reduziert folgende FP-Funktion f die Sprache L auf A:

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

□

5.3.1 Aussagenlogische Erfüllbarkeitsprobleme

Definition 5.12

*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 $a = a_1 \ldots a_n \in \{0,1\}^n$.*

*c) Der **Wert** $F(a)$ von F unter a ist induktiv wie folgt über den Aufbau von F definiert:*

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

*d) 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.*

*e) F heißt **erfüllbar**, falls es eine Belegung a mit $F(a) = 1$ gibt.*

Notation 5.13 *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. $(\neg G \vee H)$.*

Beispiel 5.14 (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

◁

Um Klammern zu sparen, vereinbaren wir folgende Präzedenzregeln:

- Der Junktor $\wedge$ bindet stärker als der Junktor $\vee$ und dieser wiederum stärker als der Junktor $\rightarrow$.
- Formeln der Form $(x_1 \circ (x_2 \circ (x_3 \circ \cdots \circ x_n) \cdots)))$, $\circ \in \{\wedge, \vee, \rightarrow\}$ kürzen wir durch $(x_1 \circ \cdots \circ x_n)$ ab.

Beispiel 5.15 (Formel für die mehrstellige Entweder-Oder Funktion)
Die Formel

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

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. $\triangleleft$

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 aussagenlogischen Erfüllbarkeitsproblems hat also große praktische Bedeutung.

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

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

Gefragt: Ist F erfüllbar?

Satz 5.16 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,

$$w \in L \Leftrightarrow F_w \in \text{SAT}.$$

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 die Anweisungen $q_m a \rightarrow q_m a N$ für alle $a \in \Gamma$ 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. Hierz bilden wir F_w über den Variablen

$$\begin{array}{ll} x_{t,q}, & \text{für } 0 \leq t \leq p(n), q \in Z, \\ y_{t,i}, & \text{für } 0 \leq t \leq p(n), -p(n) \leq i \leq p(n), \\ z_{t,i,a}, & \text{für } 0 \leq t \leq p(n), -p(n) \leq i \leq p(n), a \in \Gamma, \end{array}$$

die für folgende Aussagen stehen:

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

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:

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

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 übrigen Teilformeln sind

$$S = x_{0,q_0} \wedge y_{0,0} \wedge \bigwedge_{i=-p(n)}^{-1} z_{0,i,\sqcup} \wedge \bigwedge_{i=0}^{n-1} z_{0,i,w_{i+1}} \wedge \bigwedge_{i=n}^{p(n)} z_{0,i,\sqcup},$$

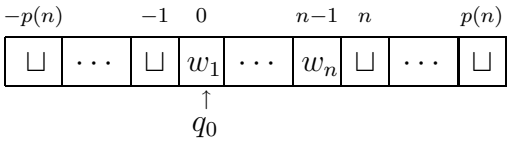
$$\ddot{U}_1 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \big(\neg y_{t,i} \wedge z_{t,i,a} \rightarrow z_{t+1,i,a}\big),$$

$$\ddot{U}_2 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \bigwedge_{p \in Z} \big(x_{t,p} \wedge y_{t,i} \wedge z_{t,i,a} \rightarrow \bigvee_{(q,b,D) \in \delta(p,a)} x_{t+1,q} \wedge y_{t+1,i+D} \wedge z_{t+1,i,b}\big),$$

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

wobei $i + D = i - 1$, falls $D = L$, $i + D = i$, falls $D = N$ und $i + D = i + 1$, falls $D = R$ ist. Die Formel S (wie Startbedingung) stellt also sicher, dass zum Zeitpunkt

0 tatsächlich die Startkonfiguration vorliegt:



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. Dagegen achtet $\ddot{U}_2$ darauf, dass sich bei jedem Übergang der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in δ verändern. Schließlich überprüft E, ob M zur Zeit $p(n)$ den Endzustand q_m erreicht hat.

Da der Aufbau der Formel $f(w) = F_w$ einem einfachen Bildungs- gesetz 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 spätestens bei Erreichen der Konfiguration $K_{p(n)}$ den Endzustand q_m an (wegen $E(a) = 1$).

Also gilt für alle $w \in \Sigma^*$ die Äquivalenz $w \in L(M) \Leftrightarrow F_w \in$ SAT, d.h. die FP-Funktion $f : w \mapsto F_w$ reduziert $L(M)$ auf SAT. □

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

Korollar 5.17 $\text{SAT} \in \mathsf{P} \Leftrightarrow \mathsf{P} = \mathsf{NP}$.

Als nächstes betrachten wir das Erfüllbarkeitsproblem für Boolesche Schaltkreise.

Definition 5.18

*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 $1 \leq j, k < l$.

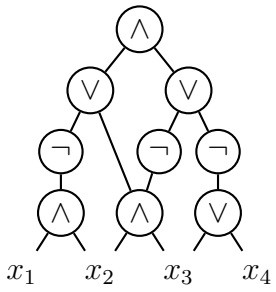
b) Die am Gatter g_l berechnete n-stellige Boolesche Funktion ist induktiv wie folgt definiert:

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 5.19 Der Schaltkreis $s = (x_1, x_2, x_3, x_4, (\wedge, 1, 2), (\wedge, 2, 3), (\vee, 3, 4), (\neg, 5), (\neg, 6), (\neg, 7), (\vee, 6, 8), (\vee, 9, 10), (\wedge, 11, 12))$ lässt sich graphisch wie folgt darstellen:



◁

Bemerkung 5.20

1. *Die Anzahl der Eingänge eines Gatters g wird als **Fanin** von g bezeichnet,*
2. *die Anzahl der Ausgänge von g (d.h. die Anzahl der Gatter, die g als Eingabe benutzen) als **Fanout**.*
3. *Boolesche Formeln entsprechen also den Booleschen Schaltkreisen mit (maximalem) Fanout 1 und umgekehrt.*
4. *Eine Boolesche Formel F kann somit leicht in einen äquivalenten Schaltkreis s mit $s(a) = F(a)$ für alle Belegungen a transformiert werden.*

Erfüllbarkeitsproblem für Boolesche Schaltkreise (CIRSAT):

Gegeben: Ein Boolescher Schaltkreis s.

Gefragt: Ist s erfüllbar?

Satz 5.21 CIRSAT *ist* NP*-vollständig.*

Bemerkung 5.22 *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 5.23

*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 Boolesche Formel F ist in **konjunktiver Normalform** (kurz **KNF**), falls F eine Konjunktion von Klauseln $C_1, \ldots, C_m$ ist,*

$$F = \bigwedge_{i=1}^{m} C_i.$$

*d) Enthält jede Klausel höchstens k Literale, so heißt F in k-**KNF**.*

Notation 5.24 *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.*

Als nächstes betrachten wir das Erfüllbarkeitsproblem für Formeln in konjunktiver Normalform.

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

Gegeben: Eine Boolesche Formel F in k-KNF.

Gefragt: Ist F erfüllbar?

Beispiel 5.25 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. ◁

Satz 5.26

(i) 1-SAT *und* 2-SAT *sind in* P *entscheidbar.*

(ii) 3-SAT *ist* NP*-vollständig.*

Beweis Es ist leicht zu sehen, dass 1-SAT und 2-SAT in P entscheidbar sind 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.

$$s \in \text{CIRSAT} \Leftrightarrow F_s \in \text{3-SAT}.$$

Zudem ist leicht zu sehen, dass die Reduktionsfunktion $s \mapsto F_s$ in FP berechenbar ist, womit CIRSAT $\leq^p$ 3-SAT folgt. □

Die Reduktion von CIRSAT auf 3-SAT lässt sich leicht zu einer Reduktion von CIRSAT zu folgender Variante von 3-SAT modifizieren.

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?

Satz 5.27 NAESAT *ist* NP*-vollständig.*

Beweis Dass NAESAT $\in$ NP liegt, ist klar. 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

$$s \in \text{CIRSAT} \Rightarrow F'_s \in \text{NAESAT}.$$

Für den Nachweis der umgekehrten Implikation sei nun $F'_s \in$ 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. □

5.3.2 Cliquen, stabile Mengen und Knotenüberdeckungen

Definition 5.28 *Sei $G = (V, E)$ ein (ungerichteter) Graph und sei $U \subseteq V$.*

a) U *heißt* ***Clique*** *in* G*, falls je zwei Knoten* $u, v \in U$ *durch eine Kante verbunden sind:*

$$\forall u, v \in U : u \neq v \Rightarrow \{u, v\} \in E.$$

b) U *heißt* ***unabhängig*** *(oder* ***stabil****), falls keine Kante in* E *zwei Knoten in* U *verbindet:*

$$\forall u, v \in U : \{u, v\} \notin E.$$

c) U *heißt* ***Knotenüberdeckung****, falls jede Kante in* E *mindestens einen Endpunkt in* U *hat:*

$$\forall e \in E : e \cap U \neq \emptyset.$$

Für einen gegebenen Graphen G und eine Zahl $k \geq 1$ betrachten wir die folgenden Fragestellungen:

CLIQUE: Hat G eine Clique der Größe k?

INDEPENDENT SET (IS): Hat G eine stabile Menge der Größe k?

NODE COVER (NC): Hat G eine Knotenüberdeckung der Größe k?

Satz 5.29 CLIQUE, IS *und* NC *sind* NP*-vollständig.*

Beweis Wir zeigen zuerst, dass IS ist NP-hart ist. Hierzu reduzieren wir 3-SAT auf IS. Sei $F = \{C_1, \ldots, C_m\}$ mit $C_i = \{l_{i,1}, \ldots, l_{i,k_i}\}$ für $i = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$. Betrachte den Graphen $G = (V, E)$ mit

$$V = \{v_{ij} \,|\, 1 \leq i \leq m, 1 \leq j \leq k_i\} \text{ und}$$
$$E = \left\{\{v_{s,t}, v_{u,v}\} \in \tbinom{V}{2} \,\middle|\, s = u \text{ oder } l_{st} \text{ ist komplementär zu } l_{uv}\right\}.$$

Dabei heißen zwei Literale **komplementär**, wenn das eine die Negation des anderen ist. Nun gilt

$$\begin{aligned} F \in \text{3-SAT} &\Leftrightarrow \text{es gibt eine Belegung, die in jeder Klausel } C_i \text{ mindestens ein Literal wahr macht} \\ &\Leftrightarrow \text{es gibt } m \text{ Literale } l_{1,j_1}, \ldots, l_{m,j_m}, \text{ die paarweise nicht komplementär sind} \\ &\Leftrightarrow \text{es gibt } m \text{ Knoten } v_{1,j_1}, \ldots, v_{m,j_m}, \text{ die nicht durch Kanten verbunden sind} \\ &\Leftrightarrow G \text{ besitzt eine stabile Knotenmenge der Größe } m. \end{aligned}$$

Als nächstes reduzieren wir IS auf CLIQUE. Es ist leicht zu sehen, dass jede Clique in einem Graphen $G = (V, E)$ eine stabile Menge in dem zu G komplementären Graphen $\bar{G} = (V, E')$ mit $E' = \binom{V}{2} \setminus E$ ist und umgekehrt. Daher lässt sich IS mittels

$$f : (G, k) \mapsto (\bar{G}, k)$$

auf CLIQUE reduzieren. Schließlich ist eine Menge I offenbar genau dann stabil, wenn ihr Komplement $V \setminus I$ eine Knotenüberdeckung ist. Daher lässt sich IS mittels

$$f : (G, k) \mapsto (G, n - k)$$

auf NC reduzieren, wobei $n = \|V\|$ die Anzahl der Knoten in G ist. □

5.3.3 Das Wort- und das Äquivalenzproblem für reguläre Sprachen

In diesem Abschnitt betrachten wir das Wortproblem für NFAs und für reguläre Ausdrücke. Wir werden sehen, dass beide Probleme effizient lösbar sind. Dagegen wird sich das Äquivalenzproblem für (sternfreie) reguläre Ausdrücke als co-NP-hart herausstellen.

Satz 5.30 *Das Wortproblem für NFAs,*

$$\mathrm{WP}_{\mathrm{NFA}} = \{N\#x \mid N \textit{ ist ein NFA und } x \in L(N)\},$$

ist in P *entscheidbar.*

Beweis Betrachte folgenden Algorithmus:

Algorithmus 5.31 P-ALGORITHMUS FÜR $\mathrm{WP}_{\mathrm{NFA}}$

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

Es ist klar, dass dieser Algorithmus korrekt arbeitet und auch in eine polynomiell zeitbeschränkte DTM für die Sprache $\mathrm{WP}_{\mathrm{NFA}}$ transformiert werden kann. □

Korollar 5.32 *Das Wortproblem für reguläre Ausdrücke ist in* P *entscheidbar:*

$$\mathrm{WP}_{\mathrm{RA}} = \{\alpha\#x \mid \alpha \textit{ ist ein regulärer Ausdruck und } x \in L(\alpha)\} \in \mathsf{P}.$$

Beweis Ein regulärer Ausdruck α lässt sich in Polynomialzeit in einen äquivalenten NFA N_α transformieren. Daher gilt $\mathrm{WP}_{\mathrm{RA}} \leq^p \mathrm{WP}_{\mathrm{NFA}}$ mittels $f : (\alpha\#x) \mapsto (N_\alpha\#x)$. Da nach vorigem Satz $\mathrm{WP}_{\mathrm{NFA}} \in \mathsf{P}$ ist, und da P unter $\leq^p$ abgeschlossen ist, folgt $\mathrm{WP}_{\mathrm{RA}} \in \mathsf{P}$. □

Ganz ähnlich folgt auch, dass das Leerheits- und das Schnittproblem für NFAs in Polynomialzeit lösbar sind (siehe Übungen).

Korollar 5.33 *Das Leerheitsproblem für NFAs,*

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

und das Schnittproblem für NFAs,

$$\text{SP}_{\text{NFA}} = \{N_1 \# N_2 \mid N_1 \text{ und } N_2 \text{ sind NFAs mit } L(N_1) \cap L(N_2) = \emptyset\}$$

sind in P *lösbar.*

Definition 5.34 *Ein regulärer Ausdruck α heißt **sternfrei**, falls in ihm kein Stern vorkommt.*

Satz 5.35 *Sei $\Sigma = \{0, 1\}$. Dann ist folgende Sprache* NP*-vollständig:*

$$\text{SF} = \{\alpha 10^n \,|\, \alpha \text{ ist ein sternfreier regulärer Ausdruck mit } L(\alpha) \neq \Sigma^n\}.$$

Beweis Wir zeigen zuerst, dass $\text{SF} \in \text{NP}$ enthalten ist. Anschließend reduzieren wir 3-SAT auf SF. Sei α ein sternfreier regulärer Ausdruck der Länge m. Es ist leicht zu zeigen (durch Induktion über den Aufbau von α), dass $L(\alpha) \subseteq \Sigma^{\leq m}$ gilt, wobei

$$\Sigma^{\leq m} = \bigcup_{i=0}^{m} \Sigma^i$$

ist. Daher akzeptiert folgender NP-Algorithmus die Sprache SF.

Algorithmus 5.36 NP-ALGORITHMUS FÜR DIE SPRACHE SF

1 **Eingabe:** $\alpha 10^n$, wobei α ein sternfreier regulärer Ausdruck der Länge m ist
2 **if** $n > m$ **then accept**
3 **rate** $x \in \Sigma^n$ und $y \in \Sigma^{\leq m} - \Sigma^n$
4 **if** $x \notin L(\alpha) \vee y \in L(\alpha)$ **then accept else reject end**

Als nächstes reduzieren wir 3-SAT auf SF. Sei eine 3-KNF Formel $F = \{C_1, \ldots, C_m\}$ gegeben. Betrachte den regulären Ausdruck $\alpha_F = (\alpha_1 | \cdots | \alpha_m)$ mit $\alpha_j = \beta_{j1} \cdots \beta_{jn}$ und

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

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

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

□

Korollar 5.37 *Das Äquivalenzproblem für sternfreie reguläre Ausdrücke,*

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

ist co-NP*-vollständig.*

Beweis Wir geben zuerst einen NP-Algorithmus für das Komplement von ÄP_{sfRA} an.

Algorithmus 5.38 NP-ALGORITHMUS FÜR DAS KOMPLEMENT VON ÄP_{sfRA}

1 **Eingabe:** sternfreie reguläre Ausdrücke α, β
2 **rate** $x \in \Sigma^{\leq \max(|\alpha|,|\beta|)}$
3 **if** $x \in L(\alpha)\Delta L(\beta)$ **then accept else reject end**

Es ist auch einfach, SF auf $\overline{\text{ÄP}}_{\text{sfRA}}$ zu reduzieren:

$$\alpha 10^n \mapsto \alpha\#\underbrace{(0|1)\cdots(0|1)}_{n\text{-mal}}\,.$$

□

Es ist nicht schwer, das Komplement von SF auch auf das Inklusionsproblem für sternfreie reguläre Ausdrücke zu reduzieren, d.h. IP_{sfRA} ist ebenfalls co-NP-vollständig. Folglich sind das Äquivalenz- und Inklusionsproblem für reguläre Ausdrücke (und somit für NFAs) co-NP-hart (siehe Übungen). Diese Probleme sind sogar PSPACE-vollständig (ohne Beweis).

5.3.4 Färbung von Graphen

Definition 5.39 *Sei $G = (V, E)$ ein Graph und sei $k \in \mathbb{N}$.*

*a) Eine k-**Färbung** von G ist eine Abbildung $c : V \to \{0, 1, \ldots, k-1\}$ mit $c(u) \neq c(v)$ für alle $\{u, v\} \in E$.*

*b) Falls eine solche Abbildung existiert, heißt G k-**färbbar**.*

k-Färbbarkeit (k-COLOR):

Gegeben: Ein Graph G.

Gefragt: Ist G k-färbbar?

Satz 5.40

(i) 1-$Color$ *und* 2-$Color$ *sind in* P *entscheidbar.*

(ii) 3-$Color$ *ist* NP*-vollständig.*

Beweis Es ist leicht zu sehen, dass 1-$Color$ und 2-$Color$ in P und 3-$Color$ in NP entscheidbar sind. Zum Nachweis, dass 3-$Color$ NP-hart ist, reduzieren wir NAESAT auf 3-$Color$.

Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j,1}, \ldots, l_{j,k_j}\}$ und $k_j \leq 3$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$. Wir können annehmen, dass F keine Einerklauseln enthält. Gesucht ist ein Graph G_F, der genau dann 3-färbbar ist, wenn $F \in$ NAESAT ist. Betrachte den Graphen $G_F = (V, E)$ mit

$$V = \{s\} \cup \{x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n\} \cup \{v_{jk} \,|\, 1 \leq i \leq n, 1 \leq k \leq k_j\},$$

und

$$E = \big\{\{s, x_i\}, \{s, \bar{x}_i\}, \{x_i, \bar{x}_i\} \big| 1 \leq i \leq n\big\} \cup \big\{\{s, v_{jk}\} \big| k_j = 2\big\} \cup \big\{\{v_{jk}, v_{jl}\} \big| k \neq l\big\} \cup \big\{\{v_{jk}, x_i\} \big| l_{jk} = \bar{x}_i\big\} \cup \big\{\{v_{jk}, \bar{x}_i\} \big| l_{jk} = x_i\big\}.$$

Sei $a = a_1 \cdots a_n$ eine Belegung für F, unter der in jeder Klausel $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ ein Literal wahr und eines falsch wird. Wir können annehmen, dass $l_{j1}(a) = 0$ und $l_{j2}(a) = 1$ ist. Dann lässt sich G_F wie folgt färben:

Knoten v	s	x_i	$\bar{x}_i$	v_{j1}	v_{j2}	$v_{j3}, k_j = 3$
Farbe $c(v)$	2	a_i	$\bar{a}_i$	0	1	2

Ist umgekehrt c eine 3-Färbung von G_F, dann können wir annehmen, dass $c(v) = 2$ ist. Dies hat zur Folge, dass $\{c(x_i), c(\bar{x}_i)\} = \{0, 1\}$ für $i = 1, \ldots, n$ ist. Zudem müssen die Knoten $v_{j1}, \ldots, v_{jk_j}$ im Fall $k_j = 2$ mit 0 und 1 und im Fall $k_j = 3$ mit allen drei Farben 0, 1 und 2 gefärbt sein. Wir können annehmen, dass $c(v_{j1}) = 0$ und $c(v_{j2}) = 1$ ist. Wegen $\{v_{jk}, \bar{l}_{jk}\} \in E$ muss $c(v_{jk}) \neq c(\bar{l}_{jk})$ für $k = 1, \ldots, k_j$ und daher $c(v_{jk}) = c(l_{jk})$ für $k = 1, 2$ gelten. Also macht die Belegung $a = c(x_1) \cdots c(x_n)$ die Literale l_{j1}, $j = 1, \ldots, m$, falsch und die Literale l_{j2}, $j = 1, \ldots, m$, wahr. Insgesamt gilt also

$$F \in \text{NAESAT} \Leftrightarrow G_F \in 3\text{-}Color.$$

□

5.3.5 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 5.41

(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. □

5.3.6 Matchings und der Heiratssatz

Beim Heiratsproblem ist eine Gruppe V von heiratswilligen Personen gegeben, wobei $u, w \in V$ durch eine Kante verbunden sind, falls aus Sicht von u und w die Möglichkeit einer Heirat zwischen u und w besteht. Sind Vielehen ausgeschlossen, so lässt sich jedes mögliche Heiratsarrangement durch ein Matching beschreiben.

Definition 5.42 *Sei $G = (V, E)$ ein Graph.*

*a) Zwei Kanten $e, e' \in E$ heißen **unabhängig**, falls $e \cap e' = \emptyset$ ist.*

*b) Eine Kantenmenge $M \subseteq E$ heißt **Matching** in G, falls die Kanten in M paarweise unabhängig sind.*

*c) Die **Matchingzahl** von G ist*

$$\mu(G) = \max\{\|M\| \mid M \text{ ist ein Matching in } G\}$$

*d) Ein Matching M heißt **maximal**, falls $\|M\| = \mu(G)$ ist, und **gesättigt**, falls es in keinem größeren Matching enthalten ist.*

e) Für eine Knotenmenge $U \subseteq V$ und eine Kantenmenge $F \subseteq E$ sei

$$U_F = \{u \in U \mid \exists v \in V : \{u, v\} \in F\}$$

die Menge der von F in U überdeckten Knoten.

*f) Ein Matching M heißt **perfekt**, falls $V_M = V$ ist.*

Eine Kantenmenge $M \subseteq E$ ist also genau dann ein Matching, wenn $\|V_M\| = 2\|M\|$ ist. In vielen Anwendungen geht es darum, ein möglichst großes Matching zu finden. Falls beim Heiratsproblem Homoehen ausgeschlossen sind, ist der resultierende Graph $G = (V, E)$ bipartit.

Definition 5.43 *Sei $G = (V, E)$ ein Graph.*

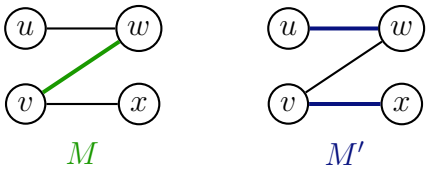
a) Für $U, W \subseteq V$ bezeichne $E(U, W)$ die Menge aller zwischen U und W verlaufenden Kanten,

$$E(U, W) = \{\{u, w\} \in E \mid u \in U, w \in W\}.$$

*b) G heißt **bipartit**, falls sich V in zwei Teilmengen U und W mit $E(U, W) = E$ zerlegen lässt.*

c) In diesem Fall notieren wir G auch in der Form $G = (U, W, E)$.

Beispiel 5.44 Ein gesättigtes Matching muss nicht maximal sein:

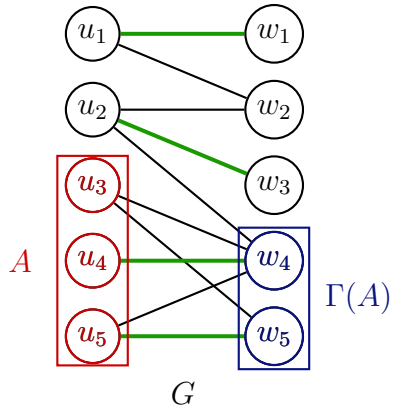


Das Matching $M = \{\{v, w\}\}$ ist gesättigt, da es sich nicht zu einem größeren Matching erweitern lässt. M ist jedoch nicht maximal, da $M' = \{\{v, x\}, \{u, w\}\}$ größer ist. Die Greedy-Methode, ausgehend von $M = \emptyset$ solange Kanten zu M hinzuzufügen, bis sich M nicht mehr zu einem größeren Matching erweitern lässt, funktioniert also nicht. ◁

Beispiel 5.45 Der bipartite Graph G enthält ein Matching M der Größe 4. Wegen

$$\Gamma(\underbrace{\{u_3, u_4, u_5\}}_{A}) = \{w_4, w_5\}$$

kann es in G offensichtlich kein Matching der Größe 5 geben. Folglich ist M maximal.



◁

Sei $A \subseteq U$ beliebig und sei M ein beliebiges Matching. Da M höchstens $\|\Gamma(A)\|$ Knoten in A überdecken kann, gilt

$$\|M\| \leq \|U - A\| + \|\Gamma(A)\| = \|U\| - (\|A\| - \|\Gamma(A)\|).$$

Folglich ist

$$\mu(G) \leq \|U\| - \max_{A \subseteq U} (\|A\| - \|\Gamma(A)\|)\,.$$

Frage 5.46 *Ist diese Schranke in jedem bipartiten Graphen scharf?*

Satz 5.47 (Heiratssatz von Hall) *Für einen bipartiten Graphen $G = (U, W, E)$ ist*

$$\mu(G) = \|U\| - \max_{A \subseteq U} (\|A\| - \|\Gamma(A)\|)\,.$$

Insbesondere hat G genau dann ein Matching M der Größe $\|M\| = \|U\|$, wenn für alle Teilmengen $A \subseteq U$ gilt: $\|\Gamma(A)\| \geq \|A\|$.