

Vorlesungsskript  
Einführung in die Theoretische  
Informatik

Wintersemester 2011/12

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

30. Januar 2013

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>		
<b>2</b>	<b>Reguläre Sprachen</b>	<b>2</b>		
2.1	Endliche Automaten	2		
2.2	Nichtdeterministische endliche Automaten	4		
2.3	Reguläre Ausdrücke	7		
2.4	Relationalstrukturen	9		
2.4.1	Ordnungs- und Äquivalenzrelationen	13		
2.4.2	Abbildungen	16		
2.4.3	Homo- und Isomorphismen	17		
2.5	Minimierung von DFAs	19		
2.6	Das Pumping-Lemma	23		
2.7	Grammatiken	24		
<b>3</b>	<b>Kontextfreie Sprachen</b>	<b>27</b>		
3.1	Chomsky-Normalform	29		
3.2	Das Pumping-Lemma für kontextfreie Sprachen	33		
3.3	Der CYK-Algorithmus	34		
3.4	Kellerautomaten	35		
3.5	Deterministisch kontextfreie Sprachen	40		
<b>4</b>	<b>Kontextsensitive Sprachen</b>	<b>44</b>		
4.1	Kontextsensitive Grammatiken	44		
4.2	Turingmaschinen	45		
4.3	Linear beschränkte Automaten	47		
<b>5</b>	<b>Entscheidbare und semi-entscheidbare Sprachen</b>	<b>50</b>		
5.1	Unentscheidbarkeit des Halteproblems	53		
5.2	Der Satz von Rice	56		
5.3	Das Postsche Korrespondenzproblem	57		
5.4	Weitere Unentscheidbarkeitsresultate	60		
5.5	Die arithmetische Hierarchie	62		
5.6	GOTO-Berechenbarkeit	63		
5.7	WHILE- und LOOP-Berechenbarkeit	66		
<b>6</b>	<b>Komplexitätsklassen</b>	<b>68</b>		
6.1	Zeitkomplexität	68		
6.2	Das P-NP-Problem	69		
6.3	Platzkomplexität	71		
<b>7</b>	<b>NP-vollständige Probleme</b>	<b>72</b>		
7.1	Aussagenlogische Erfüllbarkeitsprobleme	72		

# 1 Einleitung

Rechenmaschinen spielen in der Informatik eine zentrale Rolle. In dieser Vorlesung beschäftigen wir uns mit mathematischen Modellen für Maschinentypen von unterschiedlicher Berechnungskraft. Unter anderem lernen wir das Rechenmodell der Turingmaschine (TM) kennen, mit dem sich alle anderen Rechenmodelle simulieren lassen. Ein weiteres wichtiges Thema der Vorlesung ist die Frage, welche Probleme algorithmisch lösbar sind und wo die Grenzen der Berechenbarkeit verlaufen.

Schließlich untersuchen wir die Komplexität von algorithmischen Problemen, indem wir den benötigten Rechenaufwand möglichst gut nach oben und unten abschätzen. Eine besondere Rolle spielen hierbei die NP-vollständigen Probleme, deren Komplexität bis heute offen ist.

## Themen der Vorlesung

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

In den theoretisch orientierten Folgeveranstaltungen wird es dagegen um folgende Themen gehen.

## Thema der Vorlesung Algorithmen und Datenstrukturen

- Wie lassen sich praktisch relevante Problemstellungen möglichst effizient lösen? (Algorithmik)

## Thema der Vorlesung Logik in der Informatik

- Mathematische Grundlagen der Informatik, Beweise führen, Modellierung (Aussagenlogik, Prädikatenlogik)

Der Begriff *Algorithmus* geht auf den persischen Gelehrten **Muhammed Al Chwarizmi** (8./9. Jhd.) zurück. Der älteste bekannte nicht-triviale Algorithmus ist der nach *Euklid* benannte Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen (300 v. Chr.). Von einem Algorithmus wird erwartet, dass er jede *Problemeingabe* nach endlich vielen Rechenschritten löst (etwa durch Produktion einer *Ausgabe*). Eine wichtige Rolle spielen Entscheidungsprobleme, bei denen jede Eingabe nur mit ja oder nein beantwortet wird. Problemeingaben können Zahlen, Formeln, Graphen etc. sein. Diese werden über einem *Eingabealphabet*  $\Sigma$  kodiert.

### Definition 1.

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

$$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n,$$

wobei  $\Sigma^n = \{x_1 \dots x_n \mid n \geq 0 \text{ und } x_i \in \Sigma \text{ für } i = 1, \dots, n\}$  alle Wörter der Länge  $n$  enthält.

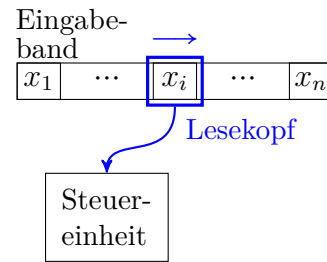
- Das (einzige) Wort der Länge  $n = 0$  ist das **leere Wort**, welches wir mit  $\varepsilon$  bezeichnen.
- Jede Teilmenge  $L \subseteq \Sigma^*$  heißt **Sprache** über dem Alphabet  $\Sigma$ .

Das zu einer Sprache  $L$  gehörige Entscheidungsproblem ist die Frage, ob ein gegebenes Wort  $x$  in  $L$  enthalten ist oder nicht.

## 2 Reguläre Sprachen

Wir betrachten zunächst Einschränkungen des TM-Modells, die vielfältige praktische Anwendungen haben, wie z.B. endliche Automaten (DFA, NFA), Kellerautomaten (PDA, DPDA) etc.

### 2.1 Endliche Automaten



Ein endlicher Automat führt bei einer Eingabe der Länge  $n$  nur  $n$  Rechenschritte aus. Um die gesamte Eingabe lesen zu können, muss der Automat also in jedem Schritt ein Zeichen der Eingabe verarbeiten.

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

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

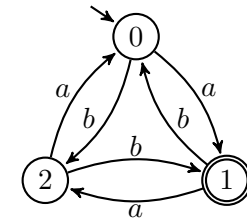
Die von  $M$  **akzeptierte** oder **erkannte Sprache** ist

$$L(M) = \left\{ x_1 \dots x_n \in \Sigma^* \mid \begin{array}{l} \exists q_1, \dots, q_{n-1} \in Z, q_n \in E: \\ \delta(q_i, x_{i+1}) = q_{i+1} \text{ f\u00fcr } i = 0, \dots, n-1 \end{array} \right\}.$$

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

$\delta$	0	1	2
a	1	2	0
b	2	0	1

Graphische Darstellung:



Der Startzustand wird meist durch einen Pfeil und Endzustände werden durch einen doppelten Kreis gekennzeichnet.  $\triangleleft$

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

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

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

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

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

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

schreiben.

**Behauptung 4.** Der DFA  $M$  aus Beispiel 3 akzeptiert die Sprache

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

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

*Beweis.* Da  $M$  nur den Endzustand 1 hat, ist  $L(M) = \{x \in \Sigma^* \mid \hat{\delta}(0, x) = 1\}$ , d.h. wir müssen folgende Äquivalenz zeigen:

$$\hat{\delta}(0, x) = 1 \Leftrightarrow \#_a(x) - \#_b(x) \equiv_3 1.$$

Hierzu reicht es, die Kongruenz

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

zu beweisen, wofür wir Induktion über die Länge  $n$  von  $x$  benutzen.

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

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

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

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

$$\begin{aligned} \delta(i, x_{n+1}) &\equiv_3 i + \#_a(x_{n+1}) - \#_b(x_{n+1}) \\ &\equiv_3 \#_a(x_1 \dots x_n) - \#_b(x_1 \dots x_n) + \#_a(x_{n+1}) - \#_b(x_{n+1}) \\ &= \#_a(x) - \#_b(x). \end{aligned}$$

Folglich ist

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

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

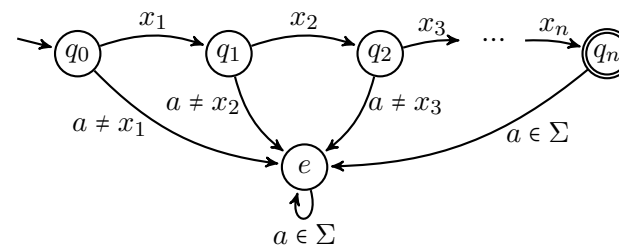
$$\text{REG} = \{L(M) \mid M \text{ ist ein DFA}\}.$$

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

**Frage:** Welche Sprachen gehören zu REG und welche nicht?

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

**Beobachtung 5.** Alle Sprachen, die aus einem einzigen Wort  $x = x_1 \dots x_n \in \Sigma^*$  bestehen (diese Sprachen werden auch als Singletonsprachen bezeichnet), sind regulär. Für folgenden DFA  $M_x$  gilt nämlich  $L(M_x) = \{x\}$ .



Formal ist  $M_x$  also das Tupel  $(Z, \Sigma, \delta, q_0, E)$  mit  $Z = \{q_0, \dots, q_n, e\}$ ,  $E = \{q_n\}$  und der Überföhrungsfunktion

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

Als nächstes betrachten wir Abschlusseigenschaften der Sprachklasse REG.

**Definition 6.** Ein **k-stelliger Sprachoperator** ist eine Abbildung  $op$ , die  $k$  Sprachen  $L_1, \dots, L_k$  auf eine Sprache  $op(L_1, \dots, L_k)$  abbildet.

**Beispiel 7.** Der Schnittoperator  $\cap$  bildet zwei Sprachen  $L_1$  und  $L_2$  auf die Sprache  $L_1 \cap L_2$  ab.  $\triangleleft$

**Definition 8.** Eine Sprachklasse  $\mathcal{K}$  heißt unter  $op$  **abgeschlossen**, wenn gilt:

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

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

**Beispiel 9.** Der Abschluss der Singletonsprachen unter Vereinigung besteht aus allen nichtleeren endlichen Sprachen.  $\triangleleft$

**Definition 10.** Für eine Sprachklasse  $\mathcal{C}$  bezeichne  $co\text{-}\mathcal{C}$  die Klasse  $\{\bar{L} \mid L \in \mathcal{C}\}$  aller Komplemente von Sprachen in  $\mathcal{C}$ .

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

**Beobachtung 11.** Mit  $L_1, L_2 \in \text{REG}$  sind auch die Sprachen  $\bar{L}_1 = \Sigma^* \setminus L_1$ ,  $L_1 \cap L_2$  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

$$\bar{M}_1 = (Z_1, \Sigma, \delta_1, q_0, Z_1 \setminus E_1)$$

das Komplement  $\bar{L}_1$  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  $L_1 \cup L_2 = \overline{(\bar{L}_1 \cap \bar{L}_2)}$  ist dann aber auch die Vereinigung von  $L_1$  und  $L_2$  regulär. (Wie sieht der zugehörige DFA aus?)

Aus Beobachtung 11 folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da die in Beispiel 3 betrachtete Sprache weder endlich noch co-endlich ist, haben wir damit allerdings noch nicht alle regulären Sprachen erfasst.

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

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

(auch **Verkettung** oder **Konkatenation** genannt) oder der Bildung der **Sternhülle**

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

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

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

Die **Plushülle** von  $L$  ist

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

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

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

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

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

## 2.2 Nichtdeterministische endliche Automaten

**Definition 12.** 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 Überföhrungsfunktion die Form

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

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

$$L(N) = \left\{ x_1 \dots x_n \in \Sigma^* \mid \begin{array}{l} \exists q_0 \in Q_0, q_1, \dots, q_{n-1} \in Z, q_n \in E: \\ q_{i+1} \in \delta(q_i, x_{i+1}) \text{ f\u00fcr } i = 0, \dots, n-1 \end{array} \right\}.$$

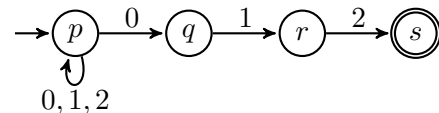
Ein NFA kann also nicht nur eine, sondern mehrere verschiedene Rechnungen ausföhren. Die Eingabe geh\u00f6rt bereits dann zu  $L(N)$ , wenn bei einer dieser Rechnungen nach Lesen des gesamten Eingabewortes ein Endzustand erreicht wird.

Im Gegensatz zu einem DFA, dessen \u00dcberf\u00f6hrungsfunktion 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 n\u00e4chste Eingabezeichen  $x_i$  wegen  $\delta(q, x_i) = \emptyset$  nicht gelesen werden kann.

**Beispiel 13.** 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 \u00dcberf\u00f6hrungsfunktion

$\delta$	$p$	$q$	$r$	$s$
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 W\u00f6rter, die mit dem Suffix 012 enden.  $\triangleleft$

**Beobachtung 14.** 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 k\u00f6nnen  $Z_1 \cap Z_2 = \emptyset$  annehmen. Dann akzeptiert der NFA

$$N = (Z_1 \cup Z_2, \Sigma, \delta_3, Q_1, E)$$

mit

$$\delta_3(p, a) = \begin{cases} \delta_1(p, a), & p \in Z_1 \setminus E_1, \\ \delta_1(p, a) \cup \bigcup_{q \in Q_2} \delta_2(q, a), & p \in E_1, \\ \delta_2(p, a), & \text{sonst} \end{cases}$$

und

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

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

$$N^* = (Z_1 \cup \{q_{neu}\}, \Sigma, \delta_4, Q_1 \cup \{q_{neu}\}, E_1 \cup \{q_{neu}\})$$

mit

$$\delta_4(p, a) = \begin{cases} \delta_1(p, a), & p \in Z_1 \setminus E_1, \\ \delta_1(p, a) \cup \bigcup_{q \in Q_1} \delta_1(q, a), & p \in E_1, \\ \emptyset, & \text{sonst} \end{cases}$$

die Sprache  $L(N_1)^*$ .

**Satz 15** (Rabin und Scott).

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

*Beweis.* Die Inklusion von links nach rechts ist klar, da jeder DFA auch als NFA aufgefasst werden kann. F\u00fcr die Gegenrichtung konstruieren wir zu einem NFA  $N = (Z, \Sigma, \delta, Q_0, E)$  einen DFA  $M = (\mathcal{P}(Z), \Sigma, \delta', Q_0, E')$  mit  $L(M) = L(N)$ . Wir definieren die \u00dcberf\u00f6hrungsfunktion  $\delta' : \mathcal{P}(Z) \times \Sigma \rightarrow \mathcal{P}(Z)$  von  $M$  mittels

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

Die Menge  $\delta'(Q, a)$  enthält also alle Zustände, in die  $N$  gelangen kann, wenn  $N$  ausgehend von einem beliebigen Zustand  $q \in Q$  das Zeichen  $a$  liest. Intuitiv bedeutet dies, dass der DFA  $M$  den NFA  $N$  simuliert, indem  $M$  in seinem aktuellen Zustand  $Q$  die Information speichert, in welchen Zuständen sich  $N$  momentan befinden könnte. Für die Erweiterung  $\hat{\delta}' : \mathcal{P}(Z) \times \Sigma^* \rightarrow \mathcal{P}(Z)$  von  $\delta'$  (siehe Seite 2) können wir nun folgende Behauptung zeigen:

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

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

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

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

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

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

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

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

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

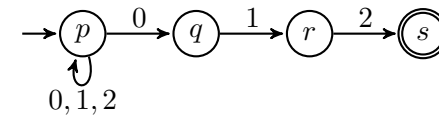
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \emptyset\},$$

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

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

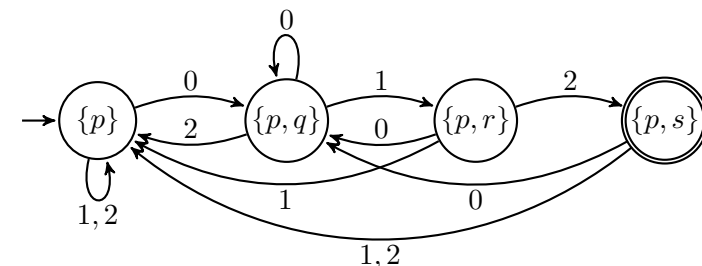
■

**Beispiel 16.** Für den NFA  $N = (Z, \Sigma, \delta, Q_0, E)$  aus Beispiel 13



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

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



◀



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

**Korollar 17.** Die Klasse REG der regulären Sprachen ist unter folgenden Operationen abgeschlossen:

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

### 2.3 Reguläre Ausdrücke

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

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

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

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

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

**Definition 18.** Die Menge der **regulären Ausdrücke**  $\gamma$  (über einem Alphabet  $\Sigma$ ) und die durch  $\gamma$  dargestellte Sprache  $L(\gamma)$  sind induktiv wie folgt definiert. Die Symbole  $\emptyset$ ,  $\epsilon$  und  $a$  ( $a \in \Sigma$ ) sind reguläre Ausdrücke, die

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

beschreiben. Sind  $\alpha$  und  $\beta$  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.

**Bemerkung 19.**

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

**Beispiel 20.** Die regulären Ausdrücke  $\epsilon^*$ ,  $\emptyset^*$ ,  $(0|1)^*00$  und  $(\epsilon|0|1)^*$  beschreiben folgende Sprachen:

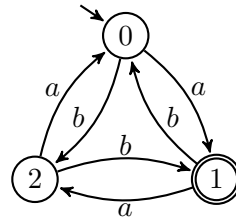
$\gamma$	$\epsilon^*$	$\emptyset^*$	$(0 1)^*00$	$(\epsilon 0 1)^*$
$L(\gamma)$	$\{\epsilon\}^* = \{\epsilon\}$	$\emptyset^* = \{\epsilon\}$	$\{x00 \mid x \in \{0,1\}^*\}$	$\{0\}$



**Beispiel 21.** Betrachte nebenstehenden DFA  $M$ .  
Um für die von  $M$  erkannte Sprache

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

einen regulären Ausdruck zu finden, betrachten wir zunächst die Sprache  $L_{0,0}$  aller Wörter  $x$ , die den DFA  $M$  ausgehend vom Zustand 0 in den Zustand 0 überführen. Weiter sei  $L_{0,0}^{\neq 0}$  die Teilsprache der Wörter  $y \neq \varepsilon$  in  $L_{0,0}$ , die dies tun ohne zwischendurch den Zustand 0 zu besuchen. Dann setzt sich jedes  $x \in L_{0,0}$  aus beliebig vielen Teilwörtern  $y_1, \dots, y_k \in L_{0,0}^{\neq 0}$  zusammen, d.h.  $L_{0,0} = (L_{0,0}^{\neq 0})^*$ .



Jedes  $y \in L_{0,0}^{\neq 0}$  beginnt entweder mit einem  $a$  (Übergang von 0 nach 1) oder mit einem  $b$  (Übergang von 0 nach 2). Im ersten Fall folgt eine beliebige Anzahl von Teilwörtern  $ab$  (Wechsel zwischen 1 und 2), an die sich entweder das Suffix  $aa$  (Rückkehr von 1 nach 0 über 2) oder das Suffix  $b$  (direkte Rückkehr von 1 nach 0) anschließt. Analog folgt im zweiten Fall eine beliebige Anzahl von Teilwörtern  $ba$  (Wechsel zwischen 2 und 1), an die sich entweder das Suffix  $a$  (direkte Rückkehr von 2 nach 0) oder das Suffix  $bb$  (Rückkehr von 2 nach 0 über 1) anschließt. Daher lässt sich  $L_{0,0}^{\neq 0}$  durch den regulären Ausdruck

$$\gamma_{0,0}^{\neq 0} = a(ab)^*(aa|b) \mid b(ba)^*(a|bb)$$

beschreiben. Eine ähnliche Überlegung zeigt, dass sich die Sprache  $L_{0,1}^{\neq 0}$  aller Wörter, die  $M$  ausgehend von 0 in den Zustand 1 überführen, ohne dass zwischendurch der Zustand 0 nochmals besucht wird, durch den regulären Ausdruck  $\gamma_{0,1}^{\neq 0} = (a|bb)(ab)^*$  beschreibbar ist. Somit erhalten wir für  $L(M)$  den regulären Ausdruck

$$\gamma_{0,1} = (\gamma_{0,0}^{\neq 0})^* \gamma_{0,1}^{\neq 0} = (a(ab)^*(aa|b) \mid b(ba)^*(a|bb))^* (a|bb)(ab)^*.$$

◀

**Satz 22.**  $\{L(\gamma) \mid \gamma \text{ ist ein regulärer Ausdruck}\} \subseteq \text{REG}$ .

*Beweis.* Die Inklusion von rechts nach links ist klar, da die Basisausdrücke  $\emptyset$ ,  $\epsilon$  und  $a$ ,  $a \in \Sigma^*$ , nur reguläre Sprachen beschreiben und die Sprachklasse REG unter Produkt, Vereinigung und Sternhülle abgeschlossen ist (siehe Beobachtungen 11 und 14).

Für die Gegenrichtung konstruieren wir zu einem DFA  $M$  einen regulären Ausdruck  $\gamma$  mit  $L(\gamma) = L(M)$ . Sei also  $M = (Z, \Sigma, \delta, q_0, E)$  ein DFA, wobei wir annehmen können, dass  $Z = \{1, \dots, m\}$  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

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

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

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

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

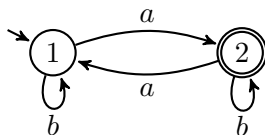
$$L_{p,q}^{r+1} = L_{p,q}^r \cup L_{p,r+1}^r (L_{r+1,r+1}^r)^* L_{r+1,q}^r$$

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

$$\gamma_{p,q}^{r+1} = \gamma_{p,q}^r \mid \gamma_{p,r+1}^r (\gamma_{r+1,r+1}^r)^* \gamma_{r+1,q}^r.$$

■

**Beispiel 23.** Betrachte den DFA



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

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

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

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

und erhalten

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

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

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

◁

**Korollar 24.** Sei  $L$  eine Sprache. Dann sind folgende Aussagen äquivalent:

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

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

## 2.4 Relationalstrukturen

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

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

**Beispiel 25.**

- $(F, M)$  mit  $F = \{f \mid f \text{ ist Fluss in Europa}\}$  und

$$M = \{(f, g) \in F \times F \mid f \text{ mündet in } g\}.$$

- $(U, B)$  mit  $U = \{x \mid x \text{ ist Berliner}\}$  und

$$B = \{(x, y) \in U \times U \mid x \text{ ist Bruder von } y\}.$$

- $(\mathcal{P}(M), \subseteq)$ , wobei  $\mathcal{P}(M)$  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.
- $(\mathbb{R}, \leq)$ .
- $(\mathbb{Z}, \mid)$ , wobei  $\mid$  die "teilt"-Relation bezeichnet (d.h.  $a \mid b$ , falls ein  $c \in \mathbb{Z}$  mit  $b = ac$  existiert).  $\triangleleft$

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

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

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

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

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

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

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

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

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

**Beispiel 26.** Ist  $B$  die Relation "ist Bruder von",  $V$  "ist Vater von",  $M$  "ist Mutter von" und  $E = V \cup M$  "ist Elternteil von", so ist  $B \circ E$  die Onkel-Relation.  $\triangleleft$

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

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

### Eigenschaften von Relationen

Sei  $R$  eine Relation auf  $A$ . Dann heißt  $R$

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

gilt.

Die nachfolgende Tabelle gibt einen Überblick über die wichtigsten Relationalstrukturen.

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

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

**Beispiel 27.**

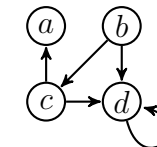
- Die Relation "ist Schwester von" ist zwar in einer reinen Damengesellschaft symmetrisch, i.a. jedoch weder symmetrisch noch asymmetrisch noch antisymmetrisch.
- Die Relation "ist Geschwister von" ist zwar symmetrisch, aber weder reflexiv noch transitiv und somit keine Äquivalenzrelation.
- $(\mathbb{R}, <)$  ist irreflexiv, asymmetrisch, transitiv und semikonnex und somit eine lineare Striktordnung.
- $(\mathbb{R}, \leq)$  und  $(\mathcal{P}(M), \subseteq)$  sind reflexiv, antisymmetrisch und transitiv und somit Ordnungen.
- $(\mathbb{R}, \leq)$  ist auch konnex und somit eine lineare Ordnung.
- $(\mathcal{P}(M), \subseteq)$  ist zwar im Fall  $\|M\| \leq 1$  konnex, aber im Fall  $\|M\| \geq 2$  weder semikonnex noch konnex. ◀

**Graphische Darstellung von Relationen**

Eine Relation  $R$  auf einer endlichen Menge  $A$  kann durch einen **gerichteten Graphen** (oder **Digraphen**)  $G = (V, E)$  mit **Knoten-**

**menge**  $V = A$  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 28.** 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  $\text{deg}^+(x) = \|R[x]\|$ , wobei  $R[x] = \{y \in V \mid xRy\}$  die Menge der **Nachfolger** von  $x$  ist. Entsprechend ist  $\text{deg}^-(x) = \|\{y \in V \mid yRx\}\|$  der **Eingangsgrad** von  $x$  und  $R^{-1}[x] = \{y \in V \mid yRx\}$  die Menge der **Vorgänger** von  $x$ . Falls  $R$  symmetrisch ist, werden die Pfeilspitzen meist weggelassen. In diesem Fall ist  $d(x) = \text{deg}^-(x) = \text{deg}^+(x)$  der **Grad** von  $x$  und  $R[x] = R^{-1}[x]$  heißt die **Nachbarschaft** von  $x$ . Ist  $R$  zudem irreflexiv, so ist  $G$  **schleifenfrei** und wir erhalten einen (**ungerichteten**) **Graphen**.

**Darstellung durch eine Adjazenzmatrix**

Eine Relation  $R$  auf einer endlichen (geordneten) Menge  $A = \{a_1, \dots, a_n\}$  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  $x \in A$  seine Nachfolgermenge  $R[x]$  in Form einer Liste zuordnet:

$x$	$R[x]$
$a$	-
$b$	$c, d$
$c$	$a, d$
$d$	$d$

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

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

Die Nachfolgermenge  $T[x]$  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 29.** Betrachte die Relationen  $R = \{(a, a), (a, c), (c, b), (c, d)\}$  und  $S = \{(a, b), (d, a), (d, c)\}$  auf der Menge  $A = \{a, b, c, d\}$ .

Relation	$R$	$S$	$R \circ S$	$S \circ R$
Digraph				
Adjazenzmatrix	1010 0000 0101 0000	0100 0000 0000 1010	0100 0000 1010 0000	0000 0000 0000 1111
Adjazenzliste	$a : a, c$ $b : -$ $c : b, d$ $d : -$	$a : b$ $b : -$ $c : -$ $d : a, c$	$a : b$ $b : -$ $c : a, c$ $d : -$	$a : -$ $b : -$ $c : -$ $d : a, b, c, d$

◁

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

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

**Satz 30.** 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 31.** Sei  $R$  eine Relation auf  $A$ .

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

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

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

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

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

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

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

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

- Die **Äquivalenzhülle** von  $R$  ist

$$h_{\ddot{a}q}(R) = \bigcap \{S \mid S \text{ ist eine Äquivalenzrelation auf } A \text{ und } R \subseteq S\}.$$

**Satz 32.** Sei  $R$  eine Relation auf  $A$ .

- (i)  $h_{refl}(R) = R \cup Id_A$ ,
- (ii)  $h_{sym}(R) = R \cup R^T$ ,
- (iii)  $R^+ = \bigcup_{n \geq 1} R^n$ ,
- (iv)  $R^* = \bigcup_{n \geq 0} R^n$ ,
- (v)  $h_{\ddot{a}q}(R) = (R \cup R^T)^*$ .

*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  $aR^n b$ , d.h. es gibt Elemente  $x_0, \dots, x_n \in A$  mit  $x_0 = a$ ,  $x_n = b$  und

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

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

### 2.4.1 Ordnungs- und Äquivalenzrelationen

Wir betrachten zunächst **Ordnungsrelationen**, die durch die drei Eigenschaften reflexiv, antisymmetrisch und transitiv definiert sind.

**Beispiel 33.**

- $(\mathcal{P}(M), \subseteq)$ ,  $(\mathbb{Z}, \leq)$ ,  $(\mathbb{R}, \leq)$  und  $(\mathbb{N}, |)$  sind Ordnungen.  $(\mathbb{Z}, |)$  ist keine Ordnung, aber eine Quasiordnung.
- Für jede Menge  $M$  ist die relationale Struktur  $(\mathcal{P}(M); \subseteq)$  eine Ordnung. Diese ist nur im Fall  $\|M\| \leq 1$  linear.
- Ist  $R$  eine Relation auf  $A$  und  $B \subseteq A$ , so ist  $R_B = R \cap (B \times B)$  die Einschränkung von  $R$  auf  $B$ .
- Einschränkungen von (linearen) Ordnungen sind ebenfalls (lineare) Ordnungen.

- Beispielsweise ist  $(\mathbb{Q}, \leq)$  die Einschränkung von  $(\mathbb{R}, \leq)$  auf  $\mathbb{Q}$  und  $(\mathbb{N}, |)$  die Einschränkung von  $(\mathbb{Z}, |)$  auf  $\mathbb{N}$ .  $\triangleleft$

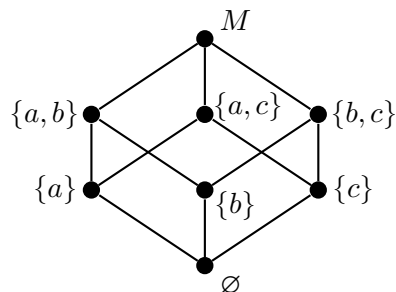
Ordnungen lassen sich sehr anschaulich durch Hasse-Diagramme darstellen. Sei  $\leq$  eine Ordnung auf  $A$  und sei  $<$  die Relation  $\leq \cap \overline{Id}_A$ . Um die Ordnung  $\leq$  in einem **Hasse-Diagramm** darzustellen, wird nur der Graph der Relation

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

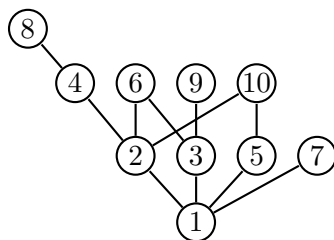
gezeichnet. Für  $x < y$  sagt man auch,  $y$  ist **oberer Nachbar** von  $x$ . Weiterhin wird im Fall  $x < y$  der Knoten  $y$  oberhalb vom Knoten  $x$  gezeichnet, so dass auf Pfeilspitzen verzichtet werden kann.

**Beispiel 34.**

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



Schränken wir die "teilt"-Relation auf die Menge  $\{1, 2, \dots, 10\}$  ein, so erhalten wir folgendes Hasse-Diagramm.



**Definition 35.** Sei  $\leq$  eine Ordnung auf  $A$  und sei  $b$  ein Element in einer Teilmenge  $B \subseteq A$ .

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

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

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

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

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

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

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

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

**Bemerkung 36.** Da Ordnungen antisymmetrisch sind, kann es in jeder Teilmenge  $B$  höchstens ein kleinstes und höchstens ein größtes Element geben. Die Anzahl der minimalen und maximalen Elemente in  $B$  kann dagegen beliebig groß sein.

**Definition 37.** Sei  $\leq$  eine Ordnung auf  $A$  und sei  $B \subseteq A$ .

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

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

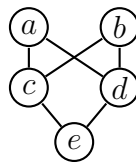


- Besitzt  $B$  eine kleinste obere Schranke  $s$ , d.h. besitzt die Menge  $O$  aller oberen Schranken von  $B$  ein kleinstes Element  $s$ , so heißt  $s$  das **Supremum** von  $B$  ( $s = \sup B$ ):

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

**Bemerkung 38.**  $B$  kann nicht mehr als ein Supremum und ein Infimum haben.

**Beispiel 39.** Betrachte nebenstehende Ordnung auf der Menge  $A = \{a, b, c, d, e\}$ . Die folgende Tabelle zeigt für verschiedene Teilmengen  $B \subseteq A$  alle minimalen und maximalen Elemente in  $B$  Minimum und Maximum, alle unteren und oberen Schranken, sowie Infimum und Supremum von  $B$  (falls existent).



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

◁

**Bemerkung 40.**

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

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

weder ein Supremum noch ein Infimum.

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

Als nächstes betrachten wir Äquivalenzrelationen, die durch die drei Eigenschaften reflexiv, symmetrisch und transitiv definiert sind.

Ist  $E$  eine Äquivalenzrelation, so nennt man die Nachbarschaft  $E[x]$  die **von  $x$  repräsentierte Äquivalenzklasse** und bezeichnet sie mit  $[x]_E$  oder einfach mit  $[x]$ . Eine Menge  $S \subseteq A$  heißt **Repräsentantensystem**, falls sie genau ein Element aus jeder Äquivalenzklasse enthält.

**Beispiel 41.**

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

$$[r] = \{a \in \mathbb{Z} \mid a \equiv r \pmod{m}\}, \quad r = 0, 1, \dots, m - 1.$$

Ein Repräsentantensystem wird beispielsweise durch die Reste  $0, 1, \dots, m - 1$  gebildet. ◁

**Definition 42.** Eine Familie  $\{B_i \mid i \in I\}$  von nichtleeren Teilmengen  $B_i \subseteq A$  heißt **Partition** der Menge  $A$ , falls gilt:

- die Mengen  $B_i$  **überdecken**  $A$ , d.h.  $A = \bigcup_{i \in I} B_i$  und
- die Mengen  $B_i$  sind **paarweise disjunkt**, d.h. für je zwei verschiedene Mengen  $B_i \neq B_j$  gilt  $B_i \cap B_j = \emptyset$ .

Die Äquivalenzklassen einer Äquivalenzrelation  $E$  bilden eine Partition  $\{[x] \mid x \in A\}$  von  $A$  (siehe Satz 43). Diese Partition wird auch **Quotienten-** oder **Faktormenge** genannt und mit  $A/E$  bezeichnet. Die Anzahl der Äquivalenzklassen von  $E$  wird auch als der **Index** von  $E$  bezeichnet. Wie der nächste Satz zeigt, beschreiben Äquivalenzrelationen auf  $A$  und Partitionen von  $A$  denselben Sachverhalt.

**Satz 43.** 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

$$xEy \Leftrightarrow E[x] = E[y] \quad (*)$$

(iii) Es gibt eine Partition  $\{B_i \mid i \in I\}$  von  $A$  mit

$$xEy \Leftrightarrow \exists i \in I : x, y \in B_i.$$

*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  $y \in E[y] = E[x]$  enthalten ist, und somit  $xEy$ . Dies zeigt, dass  $E$  die Äquivalenz  $(*)$  erfüllt.

(ii)  $\Rightarrow$  (iii) Wir zeigen, dass die Äquivalenzklassen  $E[x]$ ,  $x \in A$ , die Menge  $A$  partitionieren, falls  $E$  die Bedingung  $(*)$  erfüllt.

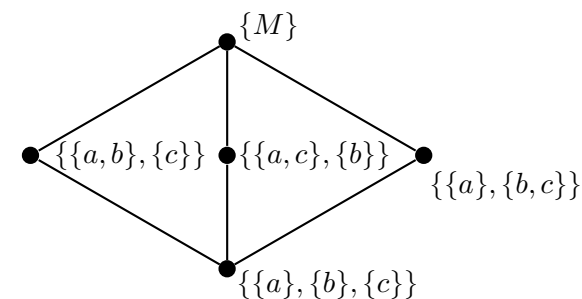
Wegen  $E[x] = E[x]$  folgt  $xEx$  und somit  $x \in E[x]$ . Folglich überdecken die Mengen  $E[x]$  die Menge  $A$ .

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  $E[x] = E[z] = E[y]$ .

(iii)  $\Rightarrow$  (i) Existiert schließlich eine Partition  $\{B_i \mid i \in I\}$  von  $A$  mit  $xEy \Leftrightarrow \exists i \in I : x, y \in B_i$ , so ist  $E$  reflexiv, da zu jedem  $x \in A$  eine Menge  $B_i$  mit  $x \in B_i$  existiert. Zudem ist  $E$  symmetrisch, da aus  $x, y \in B_i$  auch  $y, x \in B_i$  folgt. Und  $E$  ist transitiv, da aus  $x, y \in B_i$  und  $y, z \in B_j$  wegen  $z \in B_i \cap B_j$  die Gleichheit  $B_i = B_j$  und somit  $x, z \in B_i$  folgt. ■

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  $E$ .  $E$  bewirkt also eine **feinere** Partitionierung als  $E'$ . Demnach ist die Identität die **feinste** und die Allrelation die **gröbste** Äquivalenzrelation.



Die feiner-Relation auf der Menge aller Partitionen von  $M = \{a, b, c\}$  hat das folgende Hasse-Diagramm:

### 2.4.2 Abbildungen

**Definition 44.** Sei  $R$  eine binäre Relation auf einer Menge  $M$ .

- $R$  heißt **rechtseindeutig**, falls für alle  $x, y, z \in M$  gilt:

$$xRy \wedge xRz \Rightarrow y = z.$$

- $R$  heißt **linkseindeutig**, falls für alle  $x, y, z \in M$  gilt:

$$xRz \wedge yRz \Rightarrow x = y.$$

- Der **Nachbereich**  $N(R)$  und der **Vorbereich**  $V(R)$  von  $R$  sind

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

- Eine rechtseindeutige Relation  $R$  mit  $V(R) = A$  und  $N(R) \subseteq B$  heißt **Abbildung** oder **Funktion von A nach B** (kurz  $R: A \rightarrow B$ ).

**Bemerkung 45.**

- Wie üblich werden wir Abbildungen meist mit kleinen Buchstaben  $f, g, h, \dots$  bezeichnen und für  $(x, y) \in f$  nicht  $xfy$  sondern  $f(x) = y$  oder  $f: x \mapsto y$  schreiben.
- Ist  $f: A \rightarrow B$  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 46.**

- 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**.
- Ist  $f$  injektiv, so ist auch  $f^{-1}: N(f) \rightarrow A$  eine Abbildung, die als die zu  $f$  inverse Abbildung bezeichnet wird.

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

**2.4.3 Homo- und Isomorphismen**

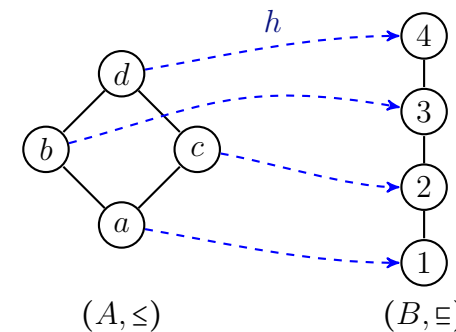
**Definition 47.** Seien  $(A_1, R_1)$  und  $(A_2, R_2)$  Relationalstrukturen.

- Eine Abbildung  $h: A_1 \rightarrow A_2$  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 48.** Folgende Abbildung  $h: A_1 \rightarrow A_2$  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

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

◁

**Definition 49.** Ein bijektiver Homomorphismus  $h : A_1 \rightarrow A_2$ , 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:  $(A_1, R_1) \cong (A_2, R_2)$ ).

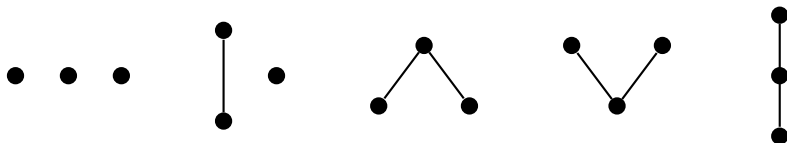
**Beispiel 50.**

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

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

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

- Es existieren genau 5 nichtisomorphe Ordnungen mit 3 Elementen:



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

- Für  $n \in \mathbb{N}$  sei

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

die Menge aller Teiler von  $n$  und

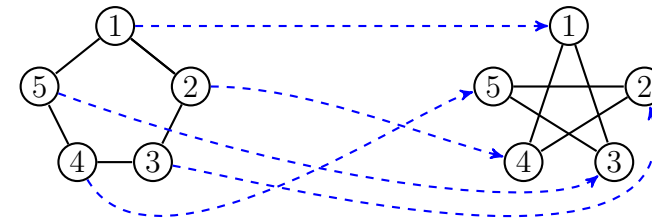
$$P_n = \{p \in T_n \mid p \text{ ist prim}\}$$

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

$$h : k \mapsto P_k$$

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

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



$G = (V, E)$		$G' = (V, E')$
$v$	1 2 3 4 5	
$h_1(v)$	1 3 5 2 4	
$h_2(v)$	1 4 2 5 3	

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



◁

**Bemerkung 51.** Auf der Knotenmenge  $V = \{1, \dots, n\}$  existieren genau  $2^{\binom{n}{2}}$  verschiedene Graphen. Sei  $a(n)$  die Anzahl aller nichtisomorphen Graphen auf  $V$ . Da jede Isomorphieklasse mindestens einen und höchstens  $n!$  verschiedene Graphen enthält, ist  $2^{\binom{n}{2}}/n! \leq a(n) \leq 2^{\binom{n}{2}}$ . Tatsächlich ist  $a(n)$  **asymptotisch gleich**  $u(n) = 2^{\binom{n}{2}}/n!$  (in Zeichen:  $a(n) \sim u(n)$ ), d.h.

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

Also gibt es auf  $V = \{1, \dots, n\}$  nicht wesentlich mehr als  $u(n)$  nicht-isomorphe Graphen.

## 2.5 Minimierung von DFAs

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

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

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

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

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

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

**Satz 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  $M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E})$  mit*

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

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

*Beweis.* Wir zeigen zuerst, dass  $\delta'$  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  $x = x_1 \dots x_n$  eine Eingabe und seien

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

die von  $M$  beim Abarbeiten von  $x$  durchlaufenen Zustände. Wegen

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

durchläuft  $M'$  dann die Zustände

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

Da aber  $q_n$  genau dann zu  $E$  gehört, wenn  $\tilde{q}_n \in \tilde{E}$  ist, folgt  $L(M') = L(M)$  (man beachte, dass  $\tilde{q}_n$  entweder nur Endzustände oder nur Nicht-Endzustände enthält, vgl. Beobachtung 53).

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  $k = \|\tilde{Z}\| = \|\{L_q \mid q \in Z\}\|$  der Zustände von  $M'$  nicht von  $M$ , sondern nur von  $L = L(M)$  abhängt. Für  $x \in \Sigma^*$  sei

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

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

Eine interessante Folgerung aus obigem Beweis ist, dass für eine reguläre Sprache  $L \subseteq \Sigma^*$  die Menge  $\{L_x \mid x \in \Sigma^*\}$  nur endlich viele verschiedene Sprachen enthält, und somit die durch

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

auf  $\Sigma^*$  definierte Äquivalenzrelation  $R_L$  endlichen Index hat.

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

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

**Beobachtung 53.**

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

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

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

zu. Wir können somit ausgehend von der Menge

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

eine Folge von Mengen

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

mittels der Vorschrift

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

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

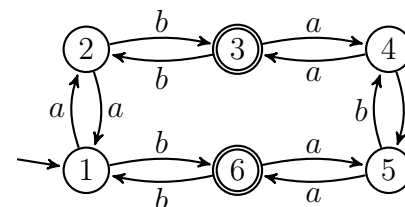
$$p \not\sim q \Leftrightarrow \{p, q\} \in D_j.$$

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

**Algorithmus min-DFA( $M$ )**

- 
- 1 **Input:** DFA  $M = (Z, \Sigma, \delta, q_0, E)$
  - 2 entferne alle nicht erreichbaren Zustände
  - 3  $D' := \{\{z, z'\} \mid z \in E, z' \notin E\}$
  - 4 **repeat**
  - 5      $D := D'$
  - 6      $D' := D \cup \{\{p, q\} \mid \exists a \in \Sigma : \{\delta(p, a), \delta(q, a)\} \in D\}$
  - 7 **until**  $D' = D$
  - 8 **Output:**  $M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E})$ , wobei für jeden Zustand  $z \in Z$  gilt:  $\tilde{z} = \{z\} \cup \{z' \in Z \mid \{z, z'\} \notin D\}$
- 

**Beispiel 54.** Betrachte den DFA  $M$



Dann enthält  $D_0$  die Paare

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

Die Paare in  $D_0$  sind in der folgenden Matrix durch den Unterscheider  $\varepsilon$  markiert.

2					
3	$\varepsilon$	$\varepsilon$			
4	$a$	$a$	$\varepsilon$		
5	$a$	$a$	$\varepsilon$		
6	$\varepsilon$	$\varepsilon$		$\varepsilon$	$\varepsilon$
	1	2	3	4	5

Wegen

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

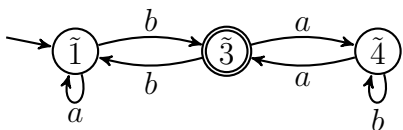
enthält  $D_1$  zusätzlich die Paare  $\{1, 4\}$ ,  $\{1, 5\}$ ,  $\{2, 4\}$ ,  $\{2, 5\}$  (in obiger Matrix durch den Unterscheider  $a$  markiert). Da die verbliebenen Paare  $\{1, 2\}$ ,  $\{3, 6\}$ ,  $\{4, 5\}$  wegen

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

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

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

die auf folgenden Minimal-DFA  $M'$  führen:



◁

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

$$\begin{aligned} \widehat{\delta}(q_0, x) = \widehat{\delta}(q_0, y) &\Leftrightarrow \widehat{\delta}(q_0, x) \sim \widehat{\delta}(q_0, y) \\ &\Leftrightarrow L_{\widehat{\delta}(q_0, x)} = L_{\widehat{\delta}(q_0, y)} \Leftrightarrow L_x = L_y \end{aligned}$$

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

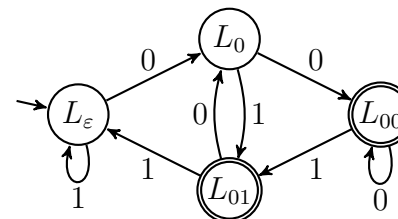
$$\begin{aligned} Z_L &= \{L_x \mid x \in \Sigma^*\}, \\ E_L &= \{L_x \mid x \in L\} \text{ und} \\ \delta_L(L_x, a) &= L_{xa}. \end{aligned}$$

Notwendig und hinreichend für die Existenz von  $M_L$  ist, dass  $R_L$  endlichen Index hat, also die Menge  $\{L_x \mid x \in \Sigma^*\}$  endlich ist.

**Beispiel 55.** Für  $L = \{x_1 \dots x_n \in \{0, 1\}^* \mid n \geq 2 \text{ und } x_{n-1} = 0\}$  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.

**Satz 56** (Myhill und Nerode).

1.  $\text{REG} = \{L \mid R_L \text{ hat endlichen Index}\}$ .
2. Sei  $L$  regulär und sei  $\text{index}(R_L)$  der Index von  $R_L$ . Dann gibt es für  $L$  bis auf Isomorphie genau einen Minimal-DFA. Dieser hat  $\text{index}(R_L)$  Zustände.

**Beispiel 57.** Sei  $L = \{a^i b^j \mid i \geq 0\}$ . Wegen  $b^i \in L_{a^i} \Delta L_{a^j}$  für  $i \neq j$  hat  $R_L$  unendlichen Index, d.h.  $L$  ist nicht regulär. ◁

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

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

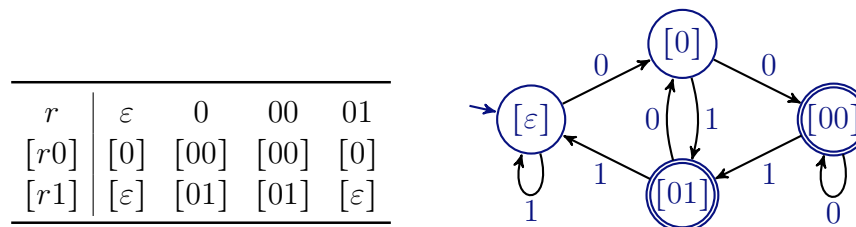
wird auch als **Äquivalenzklassenautomat** bezeichnet.

Die Konstruktion von  $M_{R_L}$  ist meist einfacher als die von  $M_L$ , da die Bestimmung der Sprachen  $L_x$  entfällt. Um die Überföhrungsfunktion von  $M_{R_L}$  aufzustellen, reicht es, ausgehend von  $r_1 = \varepsilon$  eine Folge  $r_1, \dots, r_k$  von paarweise bzgl.  $R_L$  inäquivalenten Wörtern zu bestimmen, so dass zu jedem Wort  $r_i a$ ,  $a \in \Sigma$ , ein  $r_j$  mit  $r_i a R_L r_j$  existiert. In diesem Fall ist  $\delta([r_i], a) = [r_i a] = [r_j]$ .

**Beispiel 58.** Für die Sprache  $L = \{x_1 \dots x_n \in \{0, 1\}^* \mid x_{n-1} = 0\}$  lässt sich  $M_{R_L}$  wie folgt konstruieren:

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

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



◁

Wir fassen nochmals die wichtigsten Ergebnisse zusammen.

**Korollar 59.** Sei  $L$  eine Sprache. Dann sind folgende Aussagen äquivalent:

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

Wir werden im nächsten Abschnitt noch eine weitere Methode kennenlernen, mit der man beweisen kann, dass eine Sprache nicht regulär ist, nämlich das Pumping-Lemma.



## 2.6 Das Pumping-Lemma

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

**Satz 60** (Pumping-Lemma für reguläre Sprachen).

Zu jeder regulären Sprache  $L$  gibt es eine Zahl  $l$ , so dass sich alle Wörter  $x \in L$  mit  $|x| \geq l$  in  $x = uvw$  zerlegen lassen mit

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

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

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

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

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

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

ist. ■

**Beispiel 61.** 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 62.** 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 60 aufgeführten Eigenschaften verletzt ist.

**Beispiel 63.** 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  $uv^2 w \in L$  zerlegbar ist. ◁

**Beispiel 64.** Die Sprache

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

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

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

ausgeschlossen ist. ◁

**Beispiel 65.** Auch die Sprache

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

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

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

nicht der Fall. ◁

**Bemerkung 66.** Mit Hilfe des Pumping-Lemmas kann nicht für jede Sprache  $L \notin \text{REG}$  gezeigt werden, dass  $L$  nicht regulär ist, da seine Umkehrung falsch ist. So hat beispielsweise die Sprache

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

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

## 2.7 Grammatiken

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

**Beispiel 67.** Die Sprache  $RA$  aller regulären Ausdrücke über einem Alphabet  $\Sigma = \{a_1, \dots, a_k\}$  lässt sich aus dem Symbol  $R$  durch wiederholte Anwendung folgender Regeln erzeugen:

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

◁

**Definition 68.** Eine **Grammatik** ist ein 4-Tupel  $G = (V, \Sigma, P, S)$ , wobei

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

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

**Definition 69.** Seien  $\alpha, \beta \in (V \cup \Sigma)^*$ .

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

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

Hierfür schreiben wir auch  $\underline{lur} \Rightarrow_G lvr$ . (Man beachte, dass durch Unterstreichen von  $u$  in  $\alpha$  sowohl die benutzte Regel als

auch die Stelle in  $\alpha$ , an der  $u$  durch  $v$  ersetzt wird, eindeutig erkennbar sind.)

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

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

Die **Länge** von  $\sigma$  ist  $m$  und wir notieren  $\sigma$  auch in der Form

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

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

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

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

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

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

**Beispiel 70.** Wir betrachten nochmals die Grammatik  $G = (\{R\}, \Sigma \cup \{\emptyset, \epsilon, (, ), *, | \}, P, R)$ , die die Menge der regulären Ausdrücke über dem Alphabet  $\Sigma$  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:

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

Man unterscheidet vier verschiedene Typen von Grammatiken.

**Definition 71.** Sei  $G = (V, \Sigma, P, S)$  eine Grammatik.

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

**$\epsilon$ -Sonderregel:** In einer kontextsensitiven Grammatik  $G = (V, \Sigma, P, S)$  kann auch die verkürzende Regel  $S \rightarrow \epsilon$  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

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

(context free languages) und

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

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

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

Die Sprachklassen

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

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

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

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

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

**Satz 73.**  $\text{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}.$

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

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

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

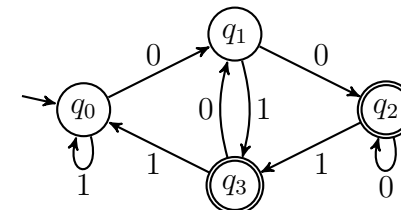
$$\begin{aligned} x \in L(M) &\Leftrightarrow \exists q_1, \dots, q_{n-1} \in Z \exists q_n \in E : \\ &\delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \dots, n \\ &\Leftrightarrow \exists q_1, \dots, q_n \in V : \\ &q_{i-1} \rightarrow_G x_i q_i \text{ für } i = 1, \dots, n \text{ und } q_n \rightarrow_G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

Für die entgegengesetzte Inklusion sei nun  $G = (V, \Sigma, P, S)$  eine reguläre Grammatik, die keine Produktionen der Form  $A \rightarrow a$  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

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

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

**Beispiel 74.** *Der DFA*



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

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

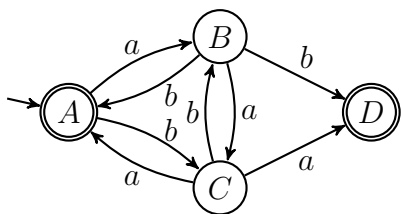
Umgekehrt führt die Grammatik  $G = (\{A, B, C\}, \{a, b\}, P, A)$  mit

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

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

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

auf den NFA



◁

### 3 Kontextfreie Sprachen

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

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

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

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

Kontextfreie Grammatiken sind dadurch charakterisiert, dass sie nur Regeln der Form  $A \rightarrow \alpha$  haben. Dies lässt die Verwendung von beliebigen  $\varepsilon$ -Regeln der Form  $A \rightarrow \varepsilon$  zu. Eine kontextsensitive Grammatik darf dagegen höchstens die  $\varepsilon$ -Regel  $S \rightarrow \varepsilon$  haben. Voraussetzung hierfür ist, dass  $S$  das Startsymbol ist und dieses nicht auf der rechten Seite einer Regel vorkommt. Daher sind nicht alle kontextfreien Grammatiken kontextsensitiv. Beispielsweise ist die Grammatik  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow \varepsilon\}, S)$  nicht kontextsensitiv, da sie die Regel  $S \rightarrow \varepsilon$  enthält, obwohl  $S$  auf der rechten Seite der Regel  $S \rightarrow aSb$  vorkommt.

Es lässt sich jedoch zu jeder kontextfreien Grammatik eine äquivalente kontextfreie Grammatik  $G'$  konstruieren, die auch kontextsensitiv ist. Hierzu zeigen wir zuerst, dass sich zu jeder kontextfreien Grammatik  $G$ , in der nicht das leere Wort ableitbar ist, eine äquivalente kontextfreie Grammatik  $G'$  ohne  $\varepsilon$ -Regeln konstruieren lässt.

**Satz 75.** *Zu jeder kontextfreien Grammatik  $G$  gibt es eine kontextfreie Grammatik  $G'$  ohne  $\varepsilon$ -Produktionen mit  $L(G') = L(G) \setminus \{\varepsilon\}$ .*

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

---

```

1  $E' := \{A \in V \mid A \rightarrow \varepsilon\}$ 
2 repeat
3    $E := E'$ 
4    $E' := E \cup \{A \in V \mid \exists B_1, \dots, B_k \in E : A \rightarrow B_1 \dots B_k\}$ 
5 until  $E = E'$ 

```

---

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

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

■

**Beispiel 76.** Betrachte die Grammatik  $G = (V, \Sigma, P, S)$  mit  $V = \{S, T, U, X, Y, Z\}$ ,  $\Sigma = \{a, b, c\}$  und den Regeln

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

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

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

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

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

◁

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 77.**  $\text{REG} \not\subseteq \text{CFL} \subseteq \text{CSL} \subseteq \text{RE}$ .

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

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

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

■

Als nächstes zeigen wir folgende Abschlusseigenschaften der kontextfreien Sprachen.

**Satz 78.** Die Klasse CFL ist abgeschlossen unter Vereinigung, Produkt und Sternhülle.

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

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

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

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

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

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

erzeugt.

■

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

**Satz** (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.  $uw^iwx^iy \in L$  für alle  $i \geq 0$ .

**Beispiel 79.** Betrachte die Sprache  $L = \{a^n b^n \mid n \geq 0\}$ . Dann lässt sich jedes Wort  $z = a^n b^n$  mit  $|z| \geq 2$  pumpen: Zerlege  $z = uvwxy$  mit  $u = a^{n-1}$ ,  $v = a$ ,  $w = \varepsilon$ ,  $x = b$  und  $y = b^{n-1}$ .  $\triangleleft$

**Beispiel 80.** Die Sprache  $\{a^n b^n c^n \mid n \geq 0\}$  ist nicht kontextfrei. Für eine vorgegebene Zahl  $l \geq 0$  hat nämlich  $z = a^l b^l c^l$  die Länge  $|z| = 3l \geq l$ . Dieses Wort lässt sich aber nicht pumpen, 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.  $\triangleleft$

Die Chomsky-Normalform ist auch Grundlage für einen effizienten Algorithmus zur Lösung des Wortproblems für kontextfreie Grammatiken, das wie folgt definiert ist.

**Wortproblem für kontextfreie Grammatiken:**

**Gegeben:** Eine kontextfreie Grammatik  $G$  und ein Wort  $x$ .

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

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

### 3.1 Chomsky-Normalform

**Definition 81.** Eine Grammatik  $(V, \Sigma, P, S)$  ist in **Chomsky-Normalform (CNF)**, falls  $P \subseteq V \times (V^2 \cup \Sigma)$  ist, also alle Regeln die Form  $A \rightarrow BC$  oder  $A \rightarrow a$  haben.

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

**Definition 82.** Regeln der Form  $A \rightarrow B$  heißen **Variablenumbenennungen**.

**Satz 83.** Zu jeder kontextfreien Grammatik  $G$  ex. eine kontextfreie Grammatik  $G'$  ohne Variablenumbenennungen mit  $L(G') = L(G)$ .

*Beweis.* Zuerst entfernen wir sukzessive alle Zyklen

$$A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_k \rightarrow A_1,$$

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

Nun entfernen wir sukzessive die restlichen Variablenumbenennungen, indem wir

- eine Regel  $A \rightarrow B$  wählen, so dass in  $P$  keine Variablenumbenennung  $B \rightarrow C$  mit  $B$  auf der rechten Seite existiert,

- diese Regel  $A \rightarrow B$  aus  $P$  entfernen und
- für jede Regel  $B \rightarrow \alpha$  in  $P$  die Regel  $A \rightarrow \alpha$  zu  $P$  hinzunehmen.

■

**Beispiel 84.** Ausgehend von den Produktionen

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

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

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

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

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

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

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

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

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

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

**Satz 85.** Zu jeder kontextfreien Sprache  $L \in \text{CFL}$  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  $\varepsilon$ -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  $X_a \rightarrow a$  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  $A \rightarrow B_1 \dots B_k$ ,  $k \geq 3$ , durch die  $k - 1$  Regeln

$$A \rightarrow B_1 A_1, \quad A_1 \rightarrow B_2 A_2, \quad \dots, \quad A_{k-3} \rightarrow B_{k-2} A_{k-2}, \quad A_{k-2} \rightarrow B_{k-1} B_k,$$

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

**Beispiel 86.** In der Produktionsmenge

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

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  $A \rightarrow a$ ,  $B \rightarrow b$ ,  $C \rightarrow c$  hinzu:

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

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

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



Eine interessante Frage ist, ob in einer kontextfreien Grammatik  $G$  jedes Wort  $x \in L(G)$  "eindeutig" ableitbar ist. Es ist klar, dass in diesem Kontext Ableitungen, die sich nur in der Reihenfolge der Regelanwendungen unterscheiden, nicht als verschieden betrachtet werden sollten. Dies erreichen wir dadurch, dass wir die Reihenfolge der Regelanwendungen festlegen.

**Definition 87.** Sei  $G = (V, \Sigma, P, S)$  eine kontextfreie Grammatik.

a) Eine Ableitung

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

heißt **Linksableitung** von  $\alpha$  (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  $i = 0, \dots, m-1$ .

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. Andernfalls heißt  $G$  **eindeutig**.

Offenbar gelten für alle Wörter  $x \in \Sigma^*$  folgende Äquivalenzen:

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

**Beispiel 88.** Wir betrachten die Grammatik  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSbS, \varepsilon\}, S)$ . Offenbar hat das Wort  $aabb$  in  $G$  zehn verschiedene Ableitungen, die sich allerdings nur in der Reihenfolge der Regelanwendungen unterscheiden:

$$\begin{aligned} \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}bS \Rightarrow aa\underline{S}bb\underline{S} \Rightarrow aa\underline{S}bb \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}bS \Rightarrow aab\underline{S}bS \Rightarrow aabb\underline{S} \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}bS \Rightarrow aab\underline{S}b\underline{S} \Rightarrow aab\underline{S}b \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}b\underline{S} \Rightarrow aa\underline{S}b\underline{S}b \Rightarrow aab\underline{S}b \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}b\underline{S} \Rightarrow aa\underline{S}b\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}b\underline{S} \Rightarrow a\underline{S}b \Rightarrow aa\underline{S}b\underline{S}b \Rightarrow aab\underline{S}b \Rightarrow aabb \\ \underline{S} &\Rightarrow a\underline{S}b\underline{S} \Rightarrow a\underline{S}b \Rightarrow aa\underline{S}b\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb. \end{aligned}$$

Darunter sind genau eine Links- und genau eine Rechtsableitung:

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

und

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

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

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

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

Wir gehen an dieser Stelle kurz der Frage nach, welche Sprache von der Grammatik  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSbS, \varepsilon\}, S)$  erzeugt wird. Zunächst einmal ist klar, dass  $L(G)$  nur Wörter  $x \in \{a, b\}^*$  mit der Eigenschaft  $\#_a(x) = \#_b(x)$  (\*) enthält. Allerdings sind nicht alle Wörter mit dieser Eigenschaft in  $L(G)$  enthalten, da beispielsweise  $ba \notin L(G)$  ist. Damit ein Wort  $x$  in  $G$  ableitbar ist, muss zudem für jedes Präfix  $u$  von  $x$  gelten, dass  $\#_a(u) \geq \#_b(u)$  (\*\*) ist.

Wir zeigen durch Induktion über die Ableitungslänge  $l$ , dass jede in  $G$  ableitbare Satzform  $\alpha \in \{a, b, S\}^*$  die Bedingungen (\*, \*\*) erfüllt.

$l = 0$ : Klar, da  $\alpha = S$  beide Bedingungen erfüllt.

$l \rightsquigarrow l + 1$ : Gelte  $S \Rightarrow^l \alpha \Rightarrow \beta$ .

- Falls  $\beta$  aus  $\alpha$  durch Anwendung der Regel  $S \rightarrow \varepsilon$  entsteht, ist dies ebenfalls klar.
- Entsteht  $\beta$  aus  $\alpha$  durch die Regel  $S \rightarrow aSbS$ , so folgt  $\#_a(\beta) = \#_a(\alpha) + 1 = \#_b(\alpha) + 1 = \#_b(\beta)$ , also (\*). Zudem entspricht jedem Präfix  $u$  von  $\beta$  ein Präfix  $u'$  von  $\alpha$  mit  $\#_a(u) - \#_b(u) \geq \#_a(u') - \#_b(u')$ , wodurch sich (\*\*) von  $\alpha$  auf  $\beta$  überträgt.

Tatsächlich sind in  $G$  alle Wörter  $x$  ableitbar, die die Bedingungen  $(*, **)$  erfüllen. Dazu zeigen wir durch Induktion über  $n$  folgende Behauptung.

**Behauptung 89.** *Alle Wörter  $x \in \{a, b\}^*$  der Länge  $\leq n$ , die die Bedingungen  $(*, **)$  erfüllen, sind in  $G$  ableitbar.*

$n = 0$ : Klar, da  $x = \varepsilon$  aus  $S$  ableitbar ist.

$n \rightsquigarrow n+1$ : Sei  $x$  ein Wort der Länge  $n+1$ , das die Bedingungen  $(*, **)$  erfüllt und sei  $u$  das kürzeste Präfix von  $x$  mit  $\#_a(u) = \#_b(u) > 1$ .

- Dann muss  $u$  die Form  $u = avb$  haben, wobei  $v$  die Bedingungen  $(*, **)$  erfüllt. Nach IV gilt daher  $S \Rightarrow^* v$ .
- Zudem hat  $x$  die Form  $x = uw$ , wobei auch  $w$  die Bedingungen  $(*, **)$  erfüllt. Nach IV gilt daher  $S \Rightarrow^* w$ .
- Nun ist  $x$  aus  $S$  wie folgt ableitbar:  $S \Rightarrow aSbS \Rightarrow^* avbS = uS \Rightarrow^* uw = x$ .

Ableitungen in einer kontextfreien Grammatik lassen sich graphisch sehr gut durch einen Syntaxbaum (auch **Ableitungsbaum** genannt, engl. *parse tree*) veranschaulichen.

**Definition 90.** *Sei  $G = (V, E)$  ein Digraph.*

- Ein  $v_0$ - $v_k$ -**Weg** in  $G$  ist eine Folge von Knoten  $v_0, \dots, v_k$  mit  $(v_i, v_{i+1}) \in E$  für  $i = 0, \dots, k-1$ . Seine **Länge** ist  $k$ .
- Ein Weg heißt **einfach** oder **Pfad**, falls alle seine Knoten paarweise verschieden sind.
- Ein  $u$ - $v$ -Weg der Länge  $\geq 1$  mit  $u = v$  heißt **Zyklus**.
- $G$  heißt **azyklisch**, wenn es in  $G$  keinen Zyklus gibt.
- $G$  heißt **gerichteter Wald**, wenn  $G$  azyklisch ist und jeder Knoten  $v \in V$  Eingangsgrad  $\deg^-(v) \leq 1$  hat.
- Ein Knoten  $u \in V$  vom Ausgangsgrad  $\deg^+(u) = 0$  heißt **Blatt**.
- Ein Knoten  $w \in V$  heißt **Wurzel** von  $G$ , falls alle Knoten  $v \in V$  von  $w$  aus erreichbar sind (d.h. es gibt einen  $w$ - $v$ -Weg in  $G$ ).

- Ein **gerichteter Wald**, der eine Wurzel hat, heißt **gerichteter Baum**.
- Da die Kantenrichtungen durch die Wahl der Wurzel eindeutig bestimmt sind, kann auf ihre Angabe verzichtet werden. Man spricht dann auch von einem **Wurzelbaum**.

**Definition 91.** *Wir ordnen einer Ableitung*

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

den Syntaxbaum  $T_m$  zu, wobei die Bäume  $T_0, \dots, T_m$  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  $A_i \rightarrow v_1 \dots v_k$  mit  $v_j \in \Sigma \cup V$  für  $j = 1, \dots, k$  angewandt, so entsteht  $T_{i+1}$  aus  $T_i$ , indem wir das Blatt  $A_i$  in  $T_i$  durch folgenden Unterbaum ersetzen:

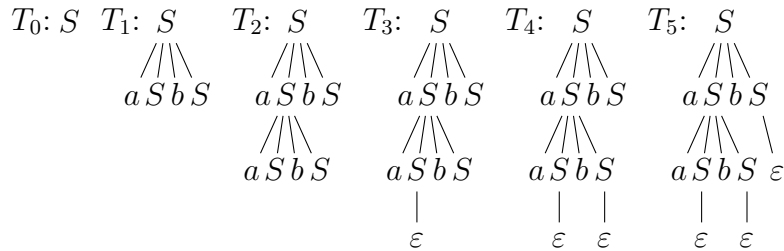
$$k > 0: \begin{array}{c} A_i \\ / \quad \backslash \\ v_1 \quad \dots \quad v_k \end{array} \qquad k = 0: \begin{array}{c} A_i \\ | \\ \varepsilon \end{array}$$

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

**Beispiel 92.** *Betrachte die Grammatik  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSbS, \varepsilon\}, S)$  und die Ableitung*

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

Die zugehörigen Syntaxbäume sind dann



Die Satzform  $\alpha_i$  ergibt sich aus  $T_i$ , indem wir die Blätter von  $T_i$  von links nach rechts zu einem Wort zusammensetzen.  $\triangleleft$

**Bemerkung 93.**

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

### 3.2 Das Pumping-Lemma für kontextfreie Sprachen

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

**Definition 94.** Die **Tiefe** eines Baumes mit Wurzel  $w$  ist die maximale Pfadlänge von  $w$  zu einem Blatt.

**Lemma 95.** Ein Binärbaum  $B$  der Tiefe  $k$  hat höchstens  $2^k$  Blätter.

*Beweis.* Wir führen den Beweis durch Induktion über  $k$ .  
 $k = 0$ : Ein Baum der Tiefe 0 kann nur einen Knoten haben.

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

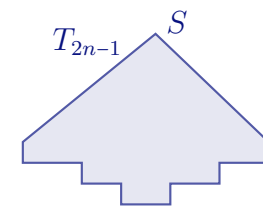
**Korollar 96.** Ein Binärbaum  $B$  mit mehr als  $2^{k-1}$  Blättern hat mindestens Tiefe  $k$ .

*Beweis.* Würde  $B$  mehr als  $2^{k-1}$  Blätter und eine Tiefe  $\leq k - 1$  besitzen, so würde dies im Widerspruch zu Lemma 95 stehen.  $\blacksquare$

**Satz 97** (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  $z = uvwx$  zerlegen lassen mit

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

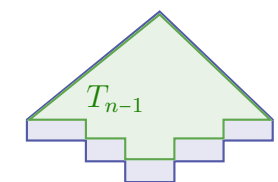
*Beweis.* Sei  $G = (V, \Sigma, P, S)$  eine CNF-Grammatik für  $L \setminus \{\epsilon\}$ . Dann gibt es in  $G$  für jedes Wort  $z = z_1 \dots z_n \in L$  mit  $n \geq 1$ , eine Ableitung

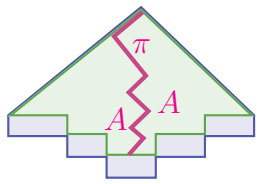


$$S = \alpha_0 \Rightarrow \alpha_1 \dots \Rightarrow \alpha_m = z.$$

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

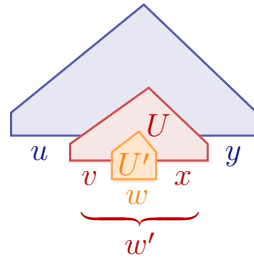
dass zuerst alle Regeln der Form  $A \rightarrow BC$  und danach die Regeln der Form  $A \rightarrow a$  zur Anwendung kommen. Dann besteht die Satzform  $\alpha_{n-1}$  aus  $n$  Variablen und der Syntaxbaum  $T_{n-1}$  hat ebenfalls  $n$  Blätter. Setzen wir  $l = 2^k$ , wobei  $k = \|V\|$  ist, so hat  $T_{n-1}$  im Fall  $n \geq l$  mindestens



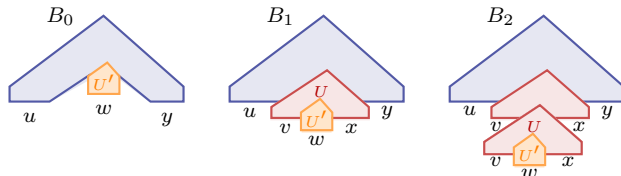


$l = 2^k > 2^{k-1}$  Blätter und daher mindestens die Tiefe  $k$ . Sei  $\pi$  ein von der Wurzel ausgehender Pfad maximaler Länge in  $T_{n-1}$ . Dann hat  $\pi$  die Länge  $\geq k$  und unter den letzten  $k + 1$  Knoten von  $\pi$  müssen zwei mit derselben Variablen  $A$  markiert sein.

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



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



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

**Satz 98.** Die Klasse CFL ist nicht abgeschlossen unter Durchschnitt und Komplement.

*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 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. ■

### 3.3 Der CYK-Algorithmus

In diesem Abschnitt stellen wir den bereits angekündigten effizienten Algorithmus zur Lösung des Wortproblems für kontextfreie Grammatiken vor.

#### Wortproblem für kontextfreie Grammatiken:

**Gegeben:** Eine kontextfreie Grammatik  $G$  und ein Wort  $x$ .  
**Gefragt:** Ist  $x \in L(G)$ ?

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

**Satz 99.** Das Wortproblem für kontextfreie Grammatiken ist effizient entscheidbar.

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

Bestimme für  $l = 1, \dots, n$  und  $k = 1, \dots, n - l + 1$  die Menge

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

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

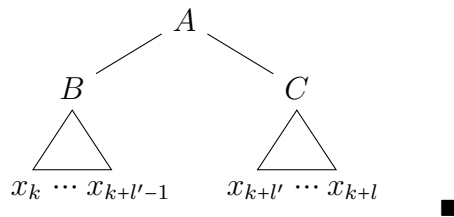
Für  $l = 1$  ist

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

und für  $l = 2, \dots, n$  ist

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

Eine Variable  $A$  gehört also genau dann zu  $V_{l,k}(x)$ ,  $l \geq 2$ , falls eine Zahl  $l' \in \{1, \dots, l-1\}$  und eine Regel  $A \rightarrow BC$  existieren, so dass  $B \in V_{l',k}(x)$  und  $C \in V_{l-l',k+l'}(x)$  sind.



**Algorithmus CYK( $G, x$ )**

```

1 Input: CNF-Grammatik  $G = (V, \Sigma, P, S)$  und ein Wort  $x = x_1 \dots x_n$ 
2 for  $k := 1$  to  $n$  do
3    $V_{1,k} := \{A \in V \mid A \rightarrow x_k \in P\}$ 
4 for  $l := 2$  to  $n$  do
5   for  $k := 1$  to  $n - l + 1$  do
6      $V_{l,k} := \emptyset$ 
7     for  $l' := 1$  to  $l - 1$  do
8       for all  $A \rightarrow BC \in P$  do
9         if  $B \in V_{l',k}$  and  $C \in V_{l-l',k+l'}$  then
10           $V_{l,k} := V_{l,k} \cup \{A\}$ 
11 if  $S \in V_{n,1}$  then accept else reject

```

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

Im Fall  $S \in V_{n,1}(x)$  lässt sich dann mithilfe dieser Information leicht ein Syntaxbaum  $T$  von  $x$  konstruieren.

**Beispiel 100.** Betrachte die CNF-Grammatik mit den Produktionen

$$S \rightarrow AS', AY, BX, CS, c; \quad S' \rightarrow BC; \quad X \rightarrow AS, BX', a; \quad X' \rightarrow XX;$$

$$Y \rightarrow BS, AY', b; \quad Y' \rightarrow YY; \quad A \rightarrow a; \quad B \rightarrow b; \quad C \rightarrow c.$$

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

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

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

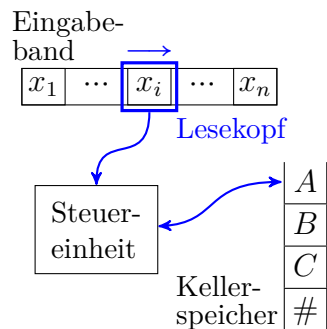
	$a$	$a$	$b$	$a$	$b$	$b$
	$\{X, A\}$	$\{X, A\}$	$\{Y, B\}$	$\{X, A\}$	$\{Y, B\}$	$\{Y, B\}$
	$\{X'\}$	$\{S\}$	$\{S\}$	$\{S\}$	$\{Y'\}$	
	$\{X\}$	$\{X\}$	$\{Y\}$	$\{Y\}$		
	$\{X'\}$	$\{S\}$	$\{Y'\}$			
	$\{X\}$	$\{Y\}$				
	$\{S\}$					

**3.4 Kellerautomaten**

Wie müssen wir das Maschinenmodell des DFA erweitern, damit die Sprache  $L = \{a^n b^n \mid n \geq 0\}$  und alle anderen kontextfreien Sprachen erkannt werden können? Dass ein DFA die Sprache  $L = \{a^n b^n \mid n \geq 0\}$  nicht erkennen kann, liegt an seinem beschränkten Speichervermögen, das zwar von  $L$  aber nicht von der Eingabe abhängen darf.

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

- verfügt über einen Kellerspeicher,
- kann  $\varepsilon$ -Übergänge machen,
- liest in jedem Schritt das aktuelle Eingabezeichen und das oberste Kellersymbol,
- kann das oberste Kellersymbol entfernen (durch eine **pop-Operation**) und
- durch beliebig viele Symbole ersetzen (durch eine **push-Operation**).



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

- $Z \neq \emptyset$  eine endliche Menge von **Zuständen**,
- $\Sigma$  das **Eingabealphabet**,
- $\Gamma$  das **Kelleralphabet**,
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_e(Z \times \Gamma^*)$  die **Überföhrungsfunktion**,
- $q_0 \in Z$  der **Startzustand** und
- $\# \in \Gamma$  das **Kelleranfangszeichen** ist.

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

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

Hierfür sagen wir auch,  $M$  führt die **Anweisung**  $quA \rightarrow pB_1 \dots B_k$  aus. Da im Fall  $u = \varepsilon$  kein Eingabezeichen gelesen wird, spricht man auch von einem **spontanen** Übergang (oder  **$\varepsilon$ -Übergang**). Eine **Konfiguration** wird durch ein Tripel

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

beschrieben und besagt, dass

- $q$  der momentane Zustand,
- $x_i \dots x_n$  der ungelesene Rest der Eingabe und
- $A_1 \dots A_l$  der aktuelle Kellerinhalt ist ( $A_1$  steht oben).

Eine Anweisung  $quA_1 \rightarrow pB_1 \dots B_k$  (mit  $u \in \{\varepsilon, x_i\}$ ) überföhrt die Konfiguration  $K$  in die **Folgekonfiguration**

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

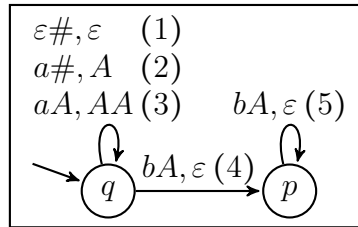
Hierfür schreiben wir auch kurz  $K \vdash K'$ . Eine **Rechnung** von  $M$  bei Eingabe  $x$  ist eine Folge von Konfigurationen  $K_0, K_1, K_2 \dots$  mit  $K_0 = (q_0, x, \#)$  und  $K_0 \vdash K_1 \vdash K_2 \dots$ .  $K_0$  heißt **Startkonfiguration** von  $M$  bei Eingabe  $x$ . Die reflexive,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

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

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

**Beispiel 102.** Sei  $M = (Z, \Sigma, \Gamma, \delta, q, \#)$  ein PDA mit  $Z = \{q, p\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{A, \#\}$  und den Anweisungen

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



Dann akzeptiert  $M$  die Eingabe  $aabb$ :

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

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

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

$$\begin{aligned} n \geq 1: (q, a^n b^n, \#) & \stackrel{(2)}{\vdash} (q, a^{n-1} b^n, A) \stackrel{(3)}{\vdash} (q, b^n, A^n) \\ & \stackrel{(4)}{\vdash} (p, b^{n-1}, A^{n-1}) \stackrel{(5)}{\vdash} (p, \varepsilon, \varepsilon). \end{aligned}$$

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

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

Falls die akzeptierende Rechnung mit Anweisung (2) beginnt, muss  $x_1 = a$  sein. Danach ist nur Anweisung (3) ausführbar, bis  $M$  das erste  $b$  liest:

$$\begin{aligned} (q, x_1 \dots x_n, \#) & \stackrel{(2)}{\vdash} (q, x_2 \dots x_n, A) \stackrel{(3)}{\vdash} (q, x_{m+1} \dots x_n, A^m) \\ & \stackrel{(4)}{\vdash} (p, x_{m+2} \dots x_n, A^{m-1}) \end{aligned}$$

mit  $x_1 = x_2 = \dots = x_m = a$  und  $x_{m+1} = b$ . Damit  $M$  den Keller leeren kann, müssen jetzt noch genau  $m - 1$   $b$ 's kommen, weshalb  $x$  auch in diesem Fall die Form  $a^m b^m$  hat.  $\triangleleft$

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

**Satz 103.**  $CFL = \{L(M) \mid M \text{ ist ein PDA}\}$ .

*Beweis.* Wir zeigen zuerst die Inklusion von links nach rechts.

*Idee:* Konstruiere zu einer kontextfreien Grammatik  $G = (V, \Sigma, P, S)$  einen PDA  $M = (\{q\}, \Sigma, \Gamma, \delta, q_0, S)$  mit  $\Gamma = V \cup \Sigma$ , so dass gilt:

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

Hierzu fügen wir für jede Regel  $A \rightarrow_G \alpha$  in  $P$  die Anweisung  $q\varepsilon A \rightarrow q\alpha$  und für jedes  $a \in \Sigma$  die Anweisung  $qaa \rightarrow q\varepsilon$  zu  $\delta$  hinzu.

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

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

Daher folgt

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

Als nächstes zeigen wir die Inklusion von rechts nach links.

*Idee:* Konstruiere zu einem PDA  $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$  eine kontextfreie Grammatik  $G = (V, \Sigma, P, S)$  mit Variablen  $X_{pAp'}$ ,  $A \in \Gamma$ ,  $p, p' \in Z$ , so dass folgende Äquivalenz gilt:

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

Ein Wort  $x$  soll also genau dann in  $G$  aus  $X_{pAp'}$  ableitbar sein, wenn  $M$  ausgehend vom Zustand  $p$  bei Lesen von  $x$  in den Zustand  $p'$  gelangen kann und dabei das Zeichen  $A$  aus dem Keller entfernt. Um

dies zu erreichen, fügen wir für jede Anweisung  $puA \rightarrow p_0A_1 \dots A_k$ ,  $k \geq 0$ , die folgenden  $\|Z\|^k$  Regeln zu  $P$  hinzu:

Für jede Zustandsfolge  $p_1, \dots, p_k$ :  $X_{pAp_k} \rightarrow uX_{p_0A_1p_1} \dots X_{p_{k-1}A_kp_k}$ .

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

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

und  $P$  enthält neben den Regeln  $S \rightarrow X_{q_0\#p}$ ,  $p \in Z$ , für jede Anweisung  $puA \rightarrow p_0A_1 \dots A_k$ ,  $k \geq 0$ , von  $M$  und jede Zustandsfolge  $p_1, \dots, p_k$  die Regel  $X_{pAp_k} \rightarrow uX_{p_0A_1p_1} \dots X_{p_{k-1}A_kp_k}$ .

Unter der Voraussetzung, dass die Äquivalenz  $(*)$  gilt, lässt sich nun leicht die Korrektheit von  $G$  zeigen. Es gilt

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

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

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

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

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

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

der Länge  $m + 1$  sei  $puA \rightarrow p_0A_1 \dots A_k$ ,  $k \geq 0$ , die im ersten Rechenschritt ausgeführte Anweisung (d.h.  $x = ux'$ ). Zudem sei  $p_i$  für  $i = 1, \dots, k$  der Zustand, in den  $M$  mit Kellerinhalt  $A_{i+1} \dots A_k$  gelangt (d.h.  $p_k = p'$ ). Dann enthält  $P$  die Regel  $X_{pAp_k} \rightarrow uX_{p_0A_1p_1} \dots X_{p_{k-1}A_kp_k}$ . Weiter sei  $u_i$  für  $i = 1, \dots, k$  das Teilwort von  $x'$ , das  $M$  zwischen den Besuchen von  $p_{i-1}$  und  $p_i$  liest.

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

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

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

$$X_{p_{i-1}A_i p_i} \Rightarrow^{m_i} u_i, \quad i = 1, \dots, k,$$

die wir zu der gesuchten Ableitung zusammensetzen können:

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

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

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

Wegen  $X_{pAp'} \rightarrow_G \alpha$  gibt es eine Anweisung  $puA \rightarrow p_0A_1 \dots A_k$ ,  $k \geq 0$ , und Zustände  $p_1, \dots, p_k \in Z$  mit

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

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

$$X_{p_{i-1}A_i p_i} \Rightarrow^{m_i} u_i \quad (i = 1, \dots, k).$$



Nach IV gibt es somit Rechnungen

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

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

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

**Beispiel 104.** Sei  $G = (\{S\}, \{a, b\}, P, S)$  mit

$$P: S \rightarrow aSbS, \quad (1) \quad S \rightarrow a. \quad (2)$$

Der zugehörige PDA besitzt dann die Anweisungen

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

Der Linksableitung

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

in  $G$  entspricht beispielsweise die akzeptierende Rechnung

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

von  $M$  und umgekehrt.  $\triangleleft$

**Beispiel 105.** Sei  $M$  der PDA  $(\{p, q\}, \{a, b\}, \{A, \#\}, \delta, p, \#)$  mit

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

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

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

Die Regelmengemenge  $P$  enthält neben den beiden Startregeln

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

die folgenden Produktionen:

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

Der akzeptierenden Rechnung

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

von  $M$  entspricht dann die Ableitung

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

in  $G$  und umgekehrt.  $\triangleleft$

### 3.5 Deterministisch kontextfreie Sprachen

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

**Definition 106.** Ein Kellerautomat heißt **deterministisch**, falls  $\vdash$  eine rechtseindeutige Relation ist:

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

Äquivalent hierzu ist, dass die Überföhrungsfunktion  $\delta$  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 107.** Der PDA  $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#)$  mit der Überföhrungsfunktion

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

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,  $\delta$  in Form einer Tabelle darzustellen:

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

Man beachte, dass jedes Tabellenfeld höchstens eine Anweisung enthält und jede Spalte, die einen  $\varepsilon$ -Eintrag in der ersten Zeile hat, sonst

keine weiteren Einträge enthält. Daher ist für alle  $(q, a, A) \in Z \times \Sigma \times \Gamma$  die Bedingung

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

erfüllt. ◁

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

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

**Definition 108.**

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

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

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

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

- Die Klasse der deterministisch kontextfreien Sprachen ist definiert durch

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

Die Klasse der deterministisch kontextfreien Sprachen lässt sich auch mit Hilfe von speziellen kontextfreien Grammatiken charakterisieren, den so genannten  $LR(k)$ -Grammatiken.

Der erste Buchstabe  $L$  steht für die Leserichtung bei der Syntaxanalyse, d.h. das Eingabewort  $x$  wird von links (nach rechts) gelesen. Der zweite Buchstabe  $R$  bedeutet, dass bei der Syntaxanalyse eine Rechtsableitung entsteht. Schließlich gibt der Parameter  $k$  an, wieviele Zeichen man über das aktuelle Eingabezeichen hinauslesen muss, damit der nächste Schritt eindeutig feststeht ( $k$  wird auch als *Lookahead* bezeichnet).

Durch  $LR(0)$ -Grammatiken lassen sich nur die präfixfreien Sprachen in DCFL erzeugen. Dagegen erzeugen die  $LR(k)$ -Grammatiken für jedes  $k \geq 1$  genau die Sprachen in DCFL.

Daneben gibt es noch  $LL(k)$ -Grammatiken, die für wachsendes  $k$  immer mehr deterministisch kontextfreie Sprachen erzeugen.

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

1. Falls  $M$  eine Eingabe  $x$  nicht zu Ende liest, wird  $x$  weder von  $M$  noch von  $\overline{M}$  akzeptiert.
2. Falls  $M$  nach dem Lesen von  $x$  noch  $\varepsilon$ -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 109.** *Jede Sprache  $L \in \text{DCFL}$  wird von einem DPDA  $M'$  erkannt, der alle Eingaben zu Ende liest.*

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

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

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

$$(a) \quad s\varepsilon\# \rightarrow q_0\#\square \quad (\text{dabei sei } s \text{ der neue Startzustand}).$$

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

$$(b) \quad qaA \rightarrow rA, \quad \text{für alle } (q, a, A) \in Z \times \Sigma \times \Gamma' \text{ mit } A = \square \text{ oder } \delta(q, a, A) = \delta(q, \varepsilon, A) = \emptyset,$$

$$(c) \quad raA \rightarrow rA, \quad \text{für alle } a \in \Sigma \text{ und } A \in \Gamma'.$$

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

$$(d) \quad q\varepsilon A \rightarrow rA, \quad \text{für alle } q \in Z \text{ und } A \in \Gamma, \text{ so dass } M \text{ ausgehend von der Konfiguration } (q, \varepsilon, A) \text{ unendlich viele } \varepsilon\text{-Übergänge ausführt ohne dabei einen Endzustand zu besuchen.}$$

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

Schließlich übernehmen wir von  $M$  die folgenden Anweisungen:

- (f) alle Anweisungen aus  $\delta$ , soweit sie nicht durch Anweisungen vom Typ (d) oder (e) überschrieben wurden.

Zusammenfassend transformieren wir  $M$  in den DPDA

$$M' = (Z \cup \{r, s, t\}, \Sigma, \Gamma', \delta', s, \#, E \cup \{t\})$$

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

**Beispiel 110.** 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 Überföhrungsfunktion

$\delta$	$q_0, \#$	$q_0, A$	$q_0, B$	$q_1, \#$	$q_1, A$	$q_1, B$	$q_2, \#$	$q_2, A$	$q_2, B$
$\varepsilon$	-	-	-	$q_2$	-	-	$q_2\#$	-	-
$a$	$q_0A\#$	$q_0AA$	$q_0AB$	-	$q_1$	-	-	-	-
$b$	$q_0B\#$	$q_0BA$	$q_0BB$	-	-	$q_1$	-	-	-
$c$	-	$q_1A$	$q_1B$	-	-	-	-	-	-

an, so erhalten wir den DPDA

$$M' = (\{q_0, q_1, q_2, r, s, t\}, \{a, b, c\}, \{A, B, \#, \square\}, \delta', s, \#, \{q_2, t\})$$

mit folgender Überföhrungsfunktion  $\delta'$ :

$\delta'$	$q_0, \#$	$q_0, A$	$q_0, B$	$q_0, \square$	$q_1, \#$	$q_1, A$	$q_1, B$	$q_1, \square$	$q_2, \#$	$q_2, A$	$q_2, B$	$q_2, \square$
$\varepsilon$	-	-	-	-	$q_2$	-	-	-	$t\#$	-	-	-
$a$	$q_0A\#$	$q_0AA$	$q_0AB$	$r\square$	-	$q_1$	$rB$	$r\square$	-	$rA$	$rB$	$r\square$
$b$	$q_0B\#$	$q_0BA$	$q_0BB$	$r\square$	-	$rA$	$q_1$	$r\square$	-	$rA$	$rB$	$r\square$
$c$	$r\#$	$q_1A$	$q_1B$	$r\square$	-	$rA$	$rB$	$r\square$	-	$rA$	$rB$	$r\square$
Typ	$(f, b)$	$(f)$	$(f)$	$(b)$	$(f)$	$(f, b)$	$(f, b)$	$(b)$	$(e)$	$(b)$	$(b)$	$(b)$
	$s, \#$	$s, A$	$s, B$	$s, \square$	$r, \#$	$r, A$	$r, B$	$r, \square$	$t, \#$	$t, A$	$t, B$	$t, \square$
$\varepsilon$	$q_0\#\square$	-	-	-	-	-	-	-	$r\#$	-	-	-
$a$	-	-	-	-	$r\#$	$rA$	$rB$	$r\square$	-	-	-	-
$b$	-	-	-	-	$r\#$	$rA$	$rB$	$r\square$	-	-	-	-
$c$	-	-	-	-	$r\#$	$rA$	$rB$	$r\square$	-	-	-	-
Typ	$(a)$				$(c)$	$(c)$	$(c)$	$(c)$	$(e)$			

◁

**Satz 111.** Die Klasse DCFL ist unter Komplement abgeschlossen, d.h. es gilt  $DCFL = \text{co-DCFL}$ .

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

Die Idee dabei ist, dass sich  $\bar{M}$  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  $\bar{M}$  im Zustand  $(q, 1)$ , so macht  $\bar{M}$  noch einen Umweg über den Endzustand  $(q, 3)$ .

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

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

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

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

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

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

hinzufügen. ■

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

**Beispiel 112.** Angenommen, ein DPDA  $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$  führt bei der Eingabe  $x = a$  folgende Rechnung aus:

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

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

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

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

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

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

**Satz 113.** Die Klasse DCFL ist nicht abgeschlossen unter Durchschnitt, Vereinigung, Produkt und Sternhülle.

*Beweis.* Die beiden Sprachen

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

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

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

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

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

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

(deterministisch) kontextfrei.

Als nächstes zeigen wir, dass DCFL nicht unter Produktbildung abgeschlossen ist. Wir wissen bereits, dass  $L = L_3 \cup L_4 \notin \text{DCFL}$  ist. Dann ist auch die Sprache

$$0L = 0L_3 \cup 0L_4 \notin \text{DCFL},$$

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

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

Es ist leicht zu sehen, dass die beiden Sprachen  $\{\varepsilon, 0\}$  und  $L_5 = L_3 \cup 0L_4$  in DCFL sind (siehe Übungen). Ihr Produkt  $\{\varepsilon, 0\}L_5 = L_5 \cup 0L_5 = L_3 \cup 0L_4 \cup 0L_3 \cup 00L_4$  gehört aber nicht zu DCFL. Da DCFL unter Schnitt mit regulären Sprachen abgeschlossen ist (siehe Übungen), wäre andernfalls auch

$$\{\varepsilon, 0\}L_5 \cap L(0a^*b^*c^*) = 0L_3 \cup 0L_4$$

in DCFL, was wir bereits ausgeschlossen haben. ■

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

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

## 4 Kontextsensitive Sprachen

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

### 4.1 Kontextsensitive Grammatiken

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

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

**Beispiel 114.** Betrachte die kontextsensitive Grammatik  $G = (V, \Sigma, P, S)$  mit  $V = \{S, B\}$ ,  $\Sigma = \{a, b, c\}$  und den Regeln

$$P: S \rightarrow aSBc, abc \quad (1) \quad cB \rightarrow Bc \quad (3) \quad bB \rightarrow bb \quad (4)$$

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

$$S \xRightarrow{(1)} aSBc \xRightarrow{(2)} aabcBc \xRightarrow{(3)} aabBcc \xRightarrow{(4)} aabbcc$$

Allgemein gilt für alle  $n \geq 1$ :

$$S \xRightarrow{(1)} a^{n-1} S (Bc)^{n-1} \xRightarrow{(2)} a^n bc (Bc)^{n-1} \xRightarrow{(3)} a^n b B^{n-1} c^n \xRightarrow{(4)} a^n b^n c^n$$

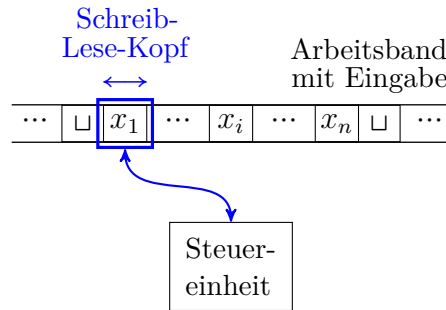
Also gilt  $a^n b^n c^n \in L(G)$  für alle  $n \geq 1$ . Umgekehrt folgt durch Induktion über die Ableitungslänge  $m$ , dass jede Satzform  $u$  mit  $S \Rightarrow^m \alpha$  die folgenden Bedingungen erfüllt:

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

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

## 4.2 Turingmaschinen

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



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

**Definition 115.** Sei  $k \geq 1$ .

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

- $\Sigma$  das Eingabealphabet (wobei  $\sqcup \notin \Sigma$ ),
- $\Gamma$  das Arbeitsalphabet (wobei  $\Sigma \cup \{\sqcup\} \subseteq \Gamma$ ),
- $\delta: Z \times \Gamma^k \rightarrow \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$  die Überföhrungsfunktion,
- $q_0$  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  $(q, a_1, \dots, a_k) \in Z \times \Gamma^k$  die Ungleichung  $\|\delta(q, a_1, \dots, a_k)\| \leq 1$  gilt.

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

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

Eine solche Anweisung ist ausführbar, falls

- $q$  der aktuelle Zustand von  $M$  ist und
- sich für  $i = 1, \dots, k$  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 116.** Sei  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  eine  $k$ -NTM.

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

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

und besagt, dass

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

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

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

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

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

$$(q, a_1, \dots, a_k) \rightarrow (q', b_1, \dots, b_k, D_1, \dots, D_k)$$

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

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

Man beachte, dass sich die Länge der Bandinschrift  $u_i a_i v_i$  beim Übergang von  $K$  zu  $K'$  genau dann um 1 erhöht, wenn in  $K'$  zum ersten Mal ein neues Feld auf dem  $i$ -ten Band besucht wird. Andernfalls bleibt die Länge von  $u_i a_i v_i$  unverändert. Die Länge von  $u_i a_i v_i$  entspricht also genau der Anzahl der auf dem  $i$ -ten Band besuchten Felder (inkl. Eingabezeichen im Fall  $i = 1$ ).

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

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

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

$M$  akzeptiert also eine Eingabe  $x$  (hierfür sagen wir kurz  $M(x)$  akzeptiert), falls es eine Rechnung  $K_x = K_0 \vdash K_1 \vdash K_2 \dots \vdash K_l$  von  $M(x)$  gibt, bei der ein Endzustand erreicht wird.

**Beispiel 117.** Betrachte die 1-DTM  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  mit  $Z = \{q_0, \dots, q_4\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \Sigma \cup \{A, B, \sqcup\}$ ,  $E = \{q_4\}$ , wobei  $\delta$  folgende Anweisungen enthält:

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

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

$$\begin{array}{llll} q_0 aabb \vdash Aq_1abb & \vdash Aaq_1bb & \vdash Aq_2aBb & \\ (1) & (2) & (4) & \\ \vdash q_2AaBb & \vdash Aq_0aBb & \vdash AAq_1Bb & \\ (5) & (7) & (1) & \\ \vdash AABq_1b & \vdash AAq_2BB & \vdash Aq_2ABB & \\ (3) & (4) & (6) & \\ \vdash AAq_0BB & \vdash AABq_3B & \vdash AABBq_3\sqcup & \vdash AABBq_4\sqcup \\ (7) & (8) & (9) & (10) \end{array}$$



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

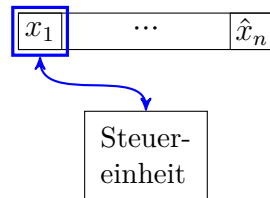
$$q_0abb \stackrel{(1)}{\vdash} Aq_1bb \stackrel{(4)}{\vdash} q_2ABb \stackrel{(7)}{\vdash} Aq_0Bb \stackrel{(8)}{\vdash} ABq_3b,$$

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

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

### 4.3 Linear beschränkte Automaten

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



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

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

#### Definition 118.

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

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

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 119.** Es ist nicht schwer, die 1-DTM  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  aus Beispiel 117 mit der Überföhrungsfunktion

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

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

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

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

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

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

$$q_0aabb\hat{b} \vdash^* AABq_1\hat{b} \xrightarrow{(4a)} AAq_2B\hat{B} \vdash^* AABq_3\hat{B} \xrightarrow{(10')} AABq_4\hat{B}$$

◁

**Definition 120.** Die Klasse der **deterministisch kontextsensitiven Sprachen** ist definiert als

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

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

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

**Satz 121.**  $\mathbf{CSL} = \{L(M) \mid M \text{ ist ein LBA}\}.$

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

Arbeitsweise von  $M$  bei Eingabe  $x = x_1 \dots x_{n-1} \hat{x}_n$  mit  $n > 0$ :

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

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

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

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

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

$$\begin{aligned} P: \quad S \rightarrow A(\hat{a}, a), (q_0\hat{a}, a) & \quad (S) \text{ „Startregeln“} \\ A \rightarrow A(a, a), (q_0a, a) & \quad (A) \text{ „A-Regeln“} \\ (c, a) \rightarrow a & \quad (F) \text{ „Finale Regeln“} \\ (qc, a) \rightarrow a, \quad \text{falls } q \in E & \quad (E) \text{ „E-Regeln“} \end{aligned}$$

$$\begin{aligned}
 (qc, a) &\rightarrow (q'c', a), & \text{falls } qc \rightarrow_M q'c'N & \quad (\text{N}) \text{ „N-Regeln“} \\
 (qc, a)(d, b) &\rightarrow (c', a)(q'd, b), & \text{falls } qc \rightarrow_M q'c'R & \quad (\text{R}) \text{ „R-Regeln“} \\
 (d, a)(qc, b) &\rightarrow (q'd, a)(c', b), & \text{falls } qc \rightarrow_M q'c'L & \quad (\text{L}) \text{ „L-Regeln“}
 \end{aligned}$$

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

$$\begin{aligned}
 q_0x_1 \dots x_{n-1}\hat{x}_n \vdash^m a_1 \dots a_{i-1}qa_i \dots a_n &\iff \\
 (q_0x_1, x_1) \dots (\hat{x}_n, x_n) &\xRightarrow{(N,R,L)}^m (a_1, x_1) \dots (qa_i, x_i) \dots (a_n, x_n)
 \end{aligned}$$

Ist also  $q_0x_1 \dots x_{n-1}\hat{x}_n \vdash^m a_1 \dots a_{i-1}qa_i \dots a_n$  mit  $q \in E$  eine akzeptierende Rechnung von  $M(x_1 \dots x_{n-1}\hat{x}_n)$ , so folgt

$$\begin{aligned}
 S &\xRightarrow{(S,A)}^n (q_0x_1, x_1)(x_2, x_2) \dots (x_{n-1}, x_{n-1})(\hat{x}_n, x_n) \\
 &\xRightarrow{(N,L,R)}^m (a_1, x_1) \dots (a_{i-1}, x_{i-1})(qa_i, x_i) \dots (a_n, x_n) \\
 &\xRightarrow{(F,E)}^n x_1 \dots x_n
 \end{aligned}$$

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

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

**Beispiel 122.** Betrachte den LBA  $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$  mit  $Z = \{q_0, \dots, q_4\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{a, b, \hat{a}, \hat{b}, A, B, \hat{B}, \sqcup\}$  und  $E = \{q_4\}$ , sowie

$$\begin{array}{lll}
 \delta: q_0a \rightarrow q_1AR & q_1\hat{b} \rightarrow q_2\hat{B}L & q_0B \rightarrow q_3BR \\
 q_1a \rightarrow q_1aR & q_2a \rightarrow q_2aL & q_0\hat{B} \rightarrow q_4\hat{B}N \\
 q_1B \rightarrow q_1BR & q_2B \rightarrow q_2BL & q_3B \rightarrow q_3BR \\
 q_1b \rightarrow q_2BL & q_2A \rightarrow q_0AR & q_3\hat{B} \rightarrow q_4\hat{B}N
 \end{array}$$

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

$$\begin{aligned}
 S &\rightarrow A(\hat{a}, a), A(\hat{b}, b), (q_0\hat{a}, a), (q_0\hat{b}, b) & (S_1-S_4) \\
 A &\rightarrow A(a, a), A(b, b), (q_0a, a), (q_0b, b) & (A_1-A_4)
 \end{aligned}$$

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

$$\begin{aligned}
 (c, a) &\rightarrow a \text{ und } (c, b) \rightarrow b & (F_1-F_{16}) \\
 (q_4c, a) &\rightarrow a \text{ und } (q_4c, b) \rightarrow b & (E_1-E_{16})
 \end{aligned}$$

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

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

Für die Anweisung  $q_1b \rightarrow q_2BL$  kommen für jedes  $d \in \Gamma$  die vier L-Regeln

$$\begin{aligned}
 (d, a)(q_1b, a) &\rightarrow (q_2d, a)(B, a), & (d, b)(q_1b, a) &\rightarrow (q_2d, b)(B, a) \\
 (d, a)(q_1b, b) &\rightarrow (q_2d, a)(B, b), & (d, b)(q_1b, b) &\rightarrow (q_2d, b)(B, b)
 \end{aligned}$$

zu  $P$  hinzu und die Anweisung  $q_0a \rightarrow q_1AR$  bewirkt für jedes  $d \in \Gamma$  die Hinzunahme folgender vier R-Regeln:

$$\begin{aligned}
 (q_0a, a)(d, a) &\rightarrow (A, a)(q_1d, a), & (q_0a, a)(d, b) &\rightarrow (A, a)(q_1d, b) \\
 (q_0a, b)(d, a) &\rightarrow (A, b)(q_1d, a), & (q_0a, b)(d, b) &\rightarrow (A, b)(q_1d, b)
 \end{aligned}$$

◁

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

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

## 5 Entscheidbare und semi-entscheidbare Sprachen

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

### Definition 123.

- Eine NTM  $M$  **hält** bei Eingabe  $x$ , falls alle Rechnungen von  $M(x)$  eine endliche Länge haben. Falls  $M(x)$  nicht hält, schreiben wir auch kurz  $M(x) = \uparrow$ .
- Eine NTM  $M$  **entscheidet** eine Eingabe  $x$ , falls  $M(x)$  hält oder eine Konfiguration mit einem Endzustand erreichen kann.
- Eine Sprache  $L \subseteq \Sigma^*$  heißt **entscheidbar**, falls eine DTM  $M$  mit  $L(M) = L$  existiert, die jede Eingabe  $x \in \Sigma^*$  entscheidet.
- Jede von einer DTM  $M$  erkannte Sprache heißt **semi-entscheidbar**.

### Bemerkung 124.

- Die von einer DTM  $M$  akzeptierte Sprache  $L(M)$  wird als semi-entscheidbar bezeichnet, da  $M$  zwar alle (positiven) Eingaben  $x \in L$  entscheidet, aber möglicherweise nicht alle (negativen) Eingaben  $x \in \bar{L}$ .
- Wir werden später sehen, dass genau die Typ-0 Sprachen semi-entscheidbar sind.

Wir wenden uns nun der Berechnung von Funktionen zu.

**Definition 125.** Eine  $k$ -DTM  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  **berechnet** eine Funktion  $f : \Sigma^* \rightarrow \Gamma^*$ , falls  $M$  bei jeder Eingabe  $x \in \Sigma^*$  in einer Konfiguration

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

mit  $u_k = f(x)$  hält (d.h.  $K_x \vdash^* K$  und  $K$  hat keine Folgekonfiguration). Hierfür sagen wir auch,  $M$  gibt bei Eingabe  $x$  das Wort  $f(x)$  aus und schreiben  $M(x) = f(x)$ .  $f$  heißt **Turing-berechenbar** (oder einfach **berechenbar**), falls es eine  $k$ -DTM  $M$  mit  $M(x) = f(x)$  für alle  $x \in \Sigma^*$  gibt.

Um eine Funktion  $f : \Sigma^* \rightarrow \Gamma^*$  zu berechnen, muss  $M$  also bei jeder Eingabe  $x$  den Funktionswert  $f(x)$  auf das  $k$ -te Band schreiben und danach halten. Falls  $M$  nicht bei allen Eingaben hält, berechnet  $M$  keine totale, sondern eine partielle Funktion.

**Definition 126.**

- a) Eine **partielle Funktion** hat die Form  $f : \Sigma^* \rightarrow \Gamma^* \cup \{\uparrow\}$ .
- b) Für  $f(x) = \uparrow$  sagen wir auch  $f(x)$  ist **undefiniert**.
- c) Der **Definitionsbereich** (engl. domain) von  $f$  ist

$$\text{dom}(f) = \{x \in \Sigma^* \mid f(x) \neq \uparrow\}.$$

- d) Das **Bild** (engl. image) von  $f$  ist

$$\text{img}(f) = \{f(x) \mid x \in \text{dom}(f)\}.$$

- e)  $f$  heißt **total**, falls  $\text{dom}(f) = \Sigma^*$  ist.
- f) Eine DTM  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  **berechnet** eine partielle Funktion  $f : \Sigma^* \rightarrow \Gamma^* \cup \{\uparrow\}$ , falls  $M(x)$  für alle  $x \in \text{dom}(f)$  das Wort  $f(x)$  ausgibt und für alle  $x \notin \text{dom}(f)$  keine Ausgabe berechnet (d.h.  $M(x) = \uparrow$ ).

Aus historischen Gründen werden die berechenbaren Funktionen und die entscheidbaren Sprachen auch **rekursiv** (engl. recursive) genannt. Wir fassen die entscheidbaren Sprachen und die (partiellen) berechenbaren Funktionen in folgenden Klassen zusammen:

$$\begin{aligned} \text{REC} &= \{L(M) \mid M \text{ ist eine DTM, die jede Eingabe entscheidet}\}, \\ \text{FREC} &= \{f \mid f \text{ ist eine berechenbare (totale) Funktion}\}, \\ \text{FREC}_p &= \{f \mid f \text{ ist eine berechenbare partielle Funktion}\}. \end{aligned}$$

Dann gilt  $\text{FREC} \not\subseteq \text{FREC}_p$  und

$$\text{REG} \not\subseteq \text{DCFL} \not\subseteq \text{CFL} \not\subseteq \text{DCSL} \subseteq \text{CSL} \not\subseteq \text{REC} \not\subseteq \text{RE}.$$

Wir wissen bereits, dass die Inklusionen  $\text{REG} \not\subseteq \text{DCFL} \not\subseteq \text{CFL} \not\subseteq \text{DCSL}$  echt sind. In diesem Abschnitt werden wir die Echtheit der Inklusion  $\text{REC} \not\subseteq \text{RE}$  zeigen. Dass  $\text{CSL}$  eine echte Teilklasse von  $\text{REC}$  ist, wird in den Übungen gezeigt.

**Beispiel 127.** Bezeichne  $x^+$  den **lexikografischen Nachfolger** von  $x \in \Sigma^*$ . Für  $\Sigma = \{0, 1\}$  ergeben sich beispielsweise folgende Werte:

$x$	$\varepsilon$	0	1	00	01	10	11	000	...
$x^+$	0	1	00	01	10	11	000	001	...

Betrachte die auf  $\Sigma^*$  definierten partiellen Funktionen  $f_1, f_2, f_3, f_4$  mit

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

Da diese vier partiellen Funktionen alle berechenbar sind, gehören die totalen Funktionen  $f_1, f_2, f_3$  zu  $\text{FREC}$ , während  $f_4$  zu  $\text{FREC}_p$  gehört. ◀

Wie der nächste Satz zeigt, lässt sich jedes Entscheidungsproblem auf ein funktionales Problem zurückführen.

**Satz 128.**

(i) Eine Sprache  $A \subseteq \Sigma^*$  ist genau dann entscheidbar, wenn ihre **charakteristische Funktion**  $\chi_A : \Sigma^* \rightarrow \{0, 1\}$  berechenbar ist. Diese ist wie folgt definiert:

$$\chi_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

(ii) Eine Sprache  $A \subseteq \Sigma^*$  ist genau dann semi-entscheidbar, falls die **partielle charakteristische Funktion**  $\hat{\chi}_A : \Sigma^* \rightarrow \{1, \uparrow\}$  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. ■

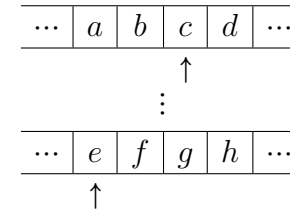
**Definition 129.** Eine Sprache  $A \subseteq \Sigma^*$  heißt **rekursiv aufzählbar**, falls  $A$  entweder leer oder das Bild  $\text{img}(f)$  einer berechenbaren Funktion  $f : \Gamma^* \rightarrow \Sigma^*$  für ein beliebiges Alphabet  $\Gamma$  ist.

**Satz 130.** Folgende Eigenschaften sind äquivalent:

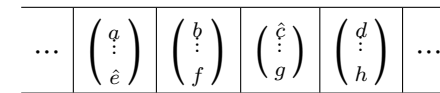
1.  $A$  ist semi-entscheidbar (d.h.  $A$  wird von einer DTM akzeptiert),
2.  $A$  wird von einer 1-DTM akzeptiert,
3.  $A$  wird von einer 1-NTM akzeptiert,
4.  $A$  ist vom Typ 0,
5.  $A$  wird von einer NTM akzeptiert,
6.  $A$  ist rekursiv aufzählbar.

*Beweis.* 1)  $\Rightarrow$  2): Sei  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  eine  $k$ -DTM, die  $A$  akzeptiert. Wir konstruieren eine 1-DTM  $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$

mit  $L(M') = A$ .  $M'$  simuliert  $M$ , indem sie jede Konfiguration  $K$  von  $M$  der Form



durch eine Konfiguration  $K'$  folgender Form nachbildet:



Das heißt,  $M'$  arbeitet mit dem Alphabet

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

und erzeugt bei Eingabe  $x = x_1 \dots x_n \in \Sigma^*$  zuerst die der Startkonfiguration  $K_x = (q_0, \varepsilon, x_1, x_2 \dots x_n, \varepsilon, \sqcup, \varepsilon, \dots, \varepsilon, \sqcup, \varepsilon)$  von  $M$  bei Eingabe  $x$  entsprechende Konfiguration

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

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

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

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

- 2)  $\Rightarrow$  3): Klar.
- 3)  $\Rightarrow$  4)  $\Rightarrow$  5): Diese beiden Implikationen lassen sich ganz ähnlich wie die Charakterisierung der Typ-1 Sprachen durch LBAs zeigen (siehe Übungen).
- 5)  $\Rightarrow$  6): Sei  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  eine  $k$ -NTM, die eine Sprache  $A \neq \emptyset$  akzeptiert. Kodieren wir eine Konfiguration  $K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$  von  $M$  durch das Wort

$$\text{code}(K) = \#q\#u_1\#a_1\#v_1\#\dots\#u_k\#a_k\#v_k\#$$

über dem Alphabet  $\tilde{\Gamma} = Z \cup \Gamma \cup \{\#\}$  und eine Rechnung  $K_0 \vdash \dots \vdash K_t$  durch  $\text{code}(K_0) \dots \text{code}(K_t)$ , so lassen sich die Wörter von  $A$  durch folgende Funktion  $f: \tilde{\Gamma}^* \rightarrow \Sigma^*$  aufzählen (dabei ist  $x_0$  ein beliebiges Wort in  $A$ ):

$$f(x) = \begin{cases} y, & x \text{ kodiert eine Rechnung } K_0 \vdash \dots \vdash K_t \text{ von } M \\ & \text{mit } K_0 = K_y \text{ und } K_t \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k \\ x_0, & \text{sonst.} \end{cases}$$

Da  $f$  berechenbar ist, ist  $A = \text{img}(f)$  rekursiv aufzählbar.

- 6)  $\Rightarrow$  1): Sei  $f: \Gamma^* \rightarrow \Sigma^*$  eine Funktion mit  $A = \text{img}(f)$  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 131.** *A ist genau dann entscheidbar, wenn A und  $\bar{A}$  semi-entscheidbar sind, d.h.  $\text{REC} = \text{RE} \cap \text{co-RE}$ .*

*Beweis.* Sei  $A$  entscheidbar. Es ist leicht zu sehen, dass dann auch  $\bar{A}$  entscheidbar ist. Also sind dann  $A$  und  $\bar{A}$  auch semi-entscheidbar. Für

die Rückrichtung seien  $f_1, f_2: \Gamma^* \rightarrow \Sigma^*$  Turing-berechenbare Funktionen mit  $\text{img}(f_1) = A$  und  $\text{img}(f_2) = \bar{A}$ . 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 und im Fall

- $f_1(y) = x$  in einem Endzustand
- $f_2(y) = x$  in einem Nichtendzustand

hält. Da jede Eingabe  $x$  entweder in  $\text{img}(f_1) = A$  oder in  $\text{img}(f_2) = \bar{A}$  enthalten ist, hält  $M$  bei allen Eingaben. ■

## 5.1 Unentscheidbarkeit des Halteproblems

Eine für die Programmverifikation sehr wichtige Fragestellung ist, ob ein gegebenes Programm bei allen Eingaben nach endlich vielen Rechenschritten stoppt. In diesem Abschnitt werden wir zeigen, dass es zur Lösung dieses Problems keinen Algorithmus gibt, nicht einmal dann, wenn wir die Eingabe fixieren. Damit Turingmaschinen die Eingabe für Turingmaschinenprogramme bilden können, müssen wir eine geeignete Kodierung von Turingmaschinen vereinbaren (diese wird auch Gödelisierung genannt).

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

$$\# \text{bin}(i) \# \text{bin}(j) \# \text{bin}(i') \# \text{bin}(j') \# b_D \#$$

kodieren. Dabei ist  $\text{bin}(n)$  die Binärdarstellung von  $n$  und  $b_N = 0$ ,  $b_L = 1$ , sowie  $b_R = 10$ .  $M$  lässt sich nun als ein Wort über dem Alphabet  $\{0, 1, \#\}$  kodieren, indem wir die Anweisungen von  $M$  in kodierter Form auflisten. Kodieren wir die Zeichen  $0, 1, \#$  binär (z.B.  $0 \mapsto 00$ ,  $1 \mapsto 11$ ,  $\# \mapsto 10$ ), so gelangen wir zu einer Binärkodierung  $w_M$  von  $M$ .

Die Binärzahl  $w_M$  wird auch die **Gödel-Nummer** von  $M$  genannt (tatsächlich kodierte Kurt Gödel Turingmaschinen durch natürliche Zahlen und nicht durch Binärstrings). Die Maschine  $M_w$  ist durch die Angabe von  $w$  bis auf die Benennung ihrer Zustände und Arbeitszeichen eindeutig bestimmt. Ganz analog lassen sich auch DTMs mit einer beliebigen Anzahl von Bändern (sowie NTMs, Konfigurationen oder Rechnungen von TMs) kodieren.

Umgekehrt können wir jedem Binärstring  $w \in \{0, 1\}^*$  eine DTM  $M_w$  wie folgt zuordnen:

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

Hierbei ist  $M_0$  eine beliebige DTM.

**Definition 132.**

a) Das **Halteproblem** ist die Sprache

$$H = \left\{ w \# x \mid \begin{array}{l} w, x \in \{0, 1\}^* \text{ und die DTM} \\ M_w \text{ hält bei Eingabe } x \end{array} \right\}$$

b) Das **spezielle Halteproblem** ist

$$K = \left\{ w \in \{0, 1\}^* \mid \begin{array}{l} \text{die DTM } M_w \\ \text{hält bei Eingabe } w \end{array} \right\}$$

Der Werteverlauf der charakteristischen Funktion  $\chi_K$  von  $K$  stimmt also mit der Diagonalen der als Matrix dargestellten charakteristischen Funktion  $\chi_H$  von  $H$  überein.

**Satz 133.**  $K \in RE \setminus co-RE$ .

*Beweis.* Wir zeigen zuerst  $K \in RE$ . Sei  $w_0$  die Kodierung einer DTM,

$\chi_H$	$x_1$	$x_2$	$x_3$	$\dots$
$w_1$	1	1	0	$\dots$
$w_2$	0	0	1	$\dots$
$w_3$	1	1	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\chi_K$				
$w_1$	1			
$w_2$		0		
$w_3$			1	
$\vdots$				$\ddots$

die bei jeder Eingabe (sofort) hält und betrachte die Funktion

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

Da  $f$  berechenbar und  $img(f) = K$  ist, folgt  $K \in RE$ . Um zu zeigen, dass  $\bar{K}$  nicht semi-entscheidbar ist, führen wir die Annahme  $\bar{K} \in RE$  auf einen Widerspruch.

Wir erklären zuerst die Beweisidee. Da  $K \in RE$  ist, gibt es in der Matrixdarstellung von  $\chi_H$  eine Zeile (sprich DTM  $M$ ), die mit der Diagonalen der Matrix übereinstimmt. Beispielsweise können wir für  $M$  eine DTM wählen, die die partielle charakteristische Funktion  $\hat{\chi}_K$  von  $K$  berechnet. Wäre auch  $\bar{K} \in RE$ , so würde eine DTM  $\hat{M}$  existieren, so dass die zugehörige Zeile in der Matrix invers zur Zeile von  $M$  und damit zur Diagonalen ist. Beispielsweise können wir für  $\hat{M}$  eine DTM wählen, die die partielle charakteristische Funktion  $\hat{\chi}_{\bar{K}}$  von  $\bar{K}$  berechnet. Da in keiner Matrix eine Zeile existieren kann, die invers zur Diagonalen ist, führt dies auf den gewünschten Widerspruch.

Nehmen wir also an, die Sprache

$$\bar{K} = \{w \mid M_w(w) \text{ hält nicht}\} \quad (*)$$

wäre semi-entscheidbar. Dann existiert eine DTM  $\hat{M}$ , die die partielle charakteristische Funktion  $\hat{\chi}_{\bar{K}}$  von  $\bar{K}$  berechnet, d.h. es gilt

$$\hat{M}(w) \text{ hält} \Leftrightarrow w \in \bar{K} \quad (**)$$

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

$$\hat{w} \in \bar{K} \stackrel{(*)}{\Leftrightarrow} M_{\hat{w}}(\hat{w}) \text{ hält nicht} \stackrel{(**)}{\Leftrightarrow} \hat{w} \notin \bar{K} \quad \text{(Widerspruch!)} \quad \blacksquare$$

Die Methode in obigem Beweis wird als **Diagonalisierung** bezeichnet. Mit dieser Beweistechnik lässt sich auch eine entscheidbare Spra-

$\chi_H$	$x_1$	$x_2$	$x_3$	$\dots$
$w_1$	1	1	0	$\dots$
$w_2$	0	0	1	$\dots$
$w_3$	1	1	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

$\hat{w}$				
	0	1	0	$\dots$



che definieren, die sich von jeder kontextsensitiven Sprache unterscheidet (siehe Übungen).

**Korollar 134.**  $\text{REC} \not\subseteq \text{RE}$ .

*Beweis.* Klar da  $K \in \text{RE} \setminus \text{REC}$ . ■

**Definition 135.**

a) Eine Sprache  $A \subseteq \Sigma^*$  heißt auf  $B \subseteq \Gamma^*$  **reduzierbar** (kurz:  $A \leq B$ ), falls eine berechenbare Funktion  $f : \Sigma^* \rightarrow \Gamma^*$  ex., so dass gilt:

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

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

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

c) Eine Sprache  $A$  heißt **hart** für eine Sprachklasse  $\mathcal{C}$  (kurz: **C-hart** oder **C-schwer**), falls jede Sprache  $L \in \mathcal{C}$  auf  $A$  reduzierbar ist:

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

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

**Beispiel 136.** Es gilt  $K \leq H$  mittels  $f : w \mapsto w\#w$ , da für alle  $w \in \{0, 1\}^*$  gilt:

$$\begin{aligned} w \in K &\Leftrightarrow M_w \text{ ist eine DTM, die bei Eingabe } w \text{ hält} \\ &\Leftrightarrow w\#w \in H. \end{aligned}$$

Das Halteproblem  $H$  ist sogar RE-hart, da sich jede semi-entscheidbare Sprache  $A$  auf  $H$  reduzieren lässt. Die Reduktion  $A \leq H$  leistet beispielsweise die Funktion  $x \mapsto w\#x$ , wobei  $w$  die Kodierung einer DTM  $M_w$  ist, die die partielle charakteristische Funktion  $\hat{\chi}_A$  von  $A$  berechnet. ◁

**Satz 137.** Die Klasse  $\text{REC}$  ist unter  $\leq$  abgeschlossen.

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

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

Wegen

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

folgt

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

Also berechnet  $M'$  die Funktion  $\chi_A$ , d.h.  $A \in \text{REC}$ . ■

Der Abschluss von RE unter  $\leq$  folgt analog (siehe Übungen).

**Korollar 138.**

1.  $A \leq B \wedge A \notin \text{REC} \Rightarrow B \notin \text{REC}$ .
2.  $A \leq B \wedge A \notin \text{RE} \Rightarrow B \notin \text{RE}$ .

*Beweis.* Aus der Annahme, dass  $B$  entscheidbar (bzw. semi-entscheidbar) ist, folgt wegen  $A \leq B$ , dass dies auch auf  $A$  zutrifft (Widerspruch). ■

Wegen  $K \leq H$  überträgt sich die Unentscheidbarkeit von  $K$  auf  $H$ .

**Korollar 139.**  $H \notin \text{REC}$ .

**Definition 140.** Das **Halteproblem bei leerem Band** ist die Sprache

$$H_0 = \left\{ w \in \{0, 1\}^* \mid \begin{array}{l} \text{die DTM } M_w \text{ hält} \\ \text{bei Eingabe } \varepsilon \end{array} \right\}$$

$\chi_{H_0}$	$x_1 (= \varepsilon)$
$w_1$	1
$w_2$	1
$w_3$	0
$\vdots$	$\vdots$

**Satz 141.**  $H_0$  ist RE-vollständig.

*Beweis.* Da die Funktion  $w \mapsto w\#\varepsilon$  die Sprache  $H_0$  auf  $H$  reduziert und da  $H \in \text{RE}$  ist, folgt  $H_0 \in \text{RE}$ .

Für den Beweis, dass  $H_0$  RE-hart ist sei  $A \in \text{RE}$  beliebig und sei  $w$  die Kodierung einer DTM, die  $\hat{\chi}_A$  berechnet. Um  $A$  auf  $H_0$  zu reduzieren, transformieren wir  $x \in \{0,1\}^*$  in die Kodierung  $w_x$  einer DTM  $M$ , die zunächst ihre Eingabe durch  $x$  ersetzt und dann  $M_w(x)$  simuliert. Dann gilt

$$x \in A \Leftrightarrow w_x \in H_0$$

und somit  $A \leq H_0$  mittels der Reduktionsfunktion  $x \mapsto w_x$ . ■

Insbesondere folgt also  $K \leq H_0$ , d.h.  $H_0$  ist unentscheidbar.

## 5.2 Der Satz von Rice

**Frage.** Kann man einer beliebig vorgegebenen TM ansehen, ob die von ihr berechnete Funktion (bzw. die von ihr akzeptierte Sprache) eine gewisse Eigenschaft hat? Kann man beispielsweise entscheiden, ob eine gegebene DTM eine totale Funktion berechnet?

**Antwort.** Nur dann, wenn die fragliche Eigenschaft trivial ist (d.h. keine oder jede berechenbare Funktion hat diese Eigenschaft).

**Definition 142.** Zu einer Klasse  $\mathcal{F}$  von Funktionen definieren wir die Sprache

$$L_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \text{die DTM } M_w \text{ berechnet eine Funktion in } \mathcal{F}\}.$$

Die Eigenschaft  $\mathcal{F}$  heißt **trivial**, wenn  $L_{\mathcal{F}} = \emptyset$  oder  $L_{\mathcal{F}} = \{0,1\}^*$  ist.

Der Satz von Rice besagt, dass  $L_{\mathcal{F}}$  nur für triviale Eigenschaften entscheidbar ist:

$$L_{\mathcal{F}} \neq \emptyset \wedge L_{\mathcal{F}} \neq \{0,1\}^* \Rightarrow L_{\mathcal{F}} \notin \text{REC}.$$

**Satz 143** (Satz von Rice).

Für jede nicht triviale Eigenschaft  $\mathcal{F}$  ist  $L_{\mathcal{F}}$  unentscheidbar.

*Beweis.* Wir reduzieren  $H_0$  auf  $L_{\mathcal{F}}$  (oder auf  $\bar{L}_{\mathcal{F}}$ ). Die Idee besteht darin, für eine gegebene DTM  $M_w$  eine DTM  $M_{w'}$  zu konstruieren mit

$$w \in H_0 \Leftrightarrow M_{w'} \text{ berechnet (k)eine Funktion in } \mathcal{F}.$$

Hierzu lassen wir  $M_{w'}$  bei Eingabe  $x$  zunächst einmal die DTM  $M_w$  bei Eingabe  $\varepsilon$  simulieren. Falls  $w \notin H_0$  ist, berechnet  $M_{w'}$  also die überall undefinierte Funktion  $u$  mit  $u(x) = \uparrow$  für alle  $x \in \Sigma^*$ .

Für das Folgende nehmen wir o.B.d.A. an, dass  $u \notin \mathcal{F}$  ist. (Andernfalls können wir  $\mathcal{F}$  durch die komplementäre Eigenschaft  $\neg\mathcal{F}$  ersetzen. Wegen  $L_{\neg\mathcal{F}} = \bar{L}_{\mathcal{F}}$  ist  $L_{\mathcal{F}}$  genau dann unentscheidbar, wenn  $L_{\neg\mathcal{F}}$  unentscheidbar ist.)

Damit die Reduktion gelingt, müssen wir nur noch dafür sorgen, dass  $M_{w'}$  im Fall  $w \in H_0$  eine Funktion  $f \in \mathcal{F}$  berechnet.

Da  $\mathcal{F}$  nicht trivial ist, gibt es eine DTM  $M_f$ , die eine partielle Funktion  $f \in \mathcal{F}$  berechnet. Betrachte die Reduktionsfunktion  $h: \{0,1\}^* \rightarrow \{0,1\}^*$  mit

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

Dann ist  $h : w \mapsto w'$  eine totale berechenbare Funktion und es gilt

$$\begin{aligned} w \in H_0 &\Rightarrow M_w \text{ berechnet } f \Rightarrow w' \in L_{\mathcal{F}}, \\ w \notin H_0 &\Rightarrow M_w \text{ berechnet } u \Rightarrow w' \notin L_{\mathcal{F}}. \end{aligned}$$

Dies zeigt, dass  $h$  das Problem  $H_0$  auf  $L_{\mathcal{F}}$  reduziert, und da  $H_0$  unentscheidbar ist, muss auch  $L_{\mathcal{F}}$  unentscheidbar sein. ■

**Beispiel 144.** 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 Eigenschaft

$$\mathcal{F} = \{f \mid \{0\}^* \subseteq \text{dom}(f) \wedge f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

nicht trivial und  $L = L_{\mathcal{F}}$  ist.  $\mathcal{F}$  ist nicht trivial, da z.B. die berechenbare partielle Funktion

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

in  $\mathcal{F}$  und die konstante Funktion  $g(x) = 0$  nicht in  $\mathcal{F}$  enthalten ist. ◁

In den Übungen wird folgende Variante des Satzes von Rice für Spracheigenschaften bewiesen, wonach wir einer gegebenen TM nicht ansehen können, ob die von ihr akzeptierte Sprache eine gewisse Eigenschaft hat oder nicht.

**Satz 145.** Für eine beliebige Sprachklasse  $\mathcal{S}$  sei

$$L_{\mathcal{S}} = \{w \in \{0, 1\}^* \mid L(M_w) \in \mathcal{S}\}.$$

Dann ist  $L_{\mathcal{S}}$  unentscheidbar, außer wenn  $L_{\mathcal{S}} \in \{\emptyset, \{0, 1\}^*\}$  ist.

### 5.3 Das Postsche Korrespondenzproblem

**Definition 146.** Sei  $\Sigma$  ein beliebiges Alphabet mit  $\# \notin \Sigma$ . Das **Postsche Korrespondenzproblem** über  $\Sigma$  (kurz **PCP** $_{\Sigma}$ ) ist wie folgt definiert.

**Gegeben:**  $k$  Paare  $(x_1, y_1), \dots, (x_k, y_k)$  von Wörtern über  $\Sigma$ .

**Gefragt:** Gibt es eine Folge  $\alpha = (i_1, \dots, i_n)$ ,  $n \geq 1$ , von Indizes  $i_j \in \{1, \dots, k\}$  mit  $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$ ?

Das **modifizierte PCP** über  $\Sigma$  (kurz **MPCP** $_{\Sigma}$ ) fragt nach einer Lösung  $\alpha = (i_1, \dots, i_n)$  mit  $i_1 = 1$ .

Wir notieren eine PCP-Instanz meist in Form einer Matrix  $\begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$  und kodieren sie durch das Wort  $x_1 \# y_1 \# \dots \# x_k \# y_k$ .

**Beispiel 147.** Die Instanz  $I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & ab & caa \\ aca & bc & aa \end{pmatrix}$  besitzt wegen

$$\begin{aligned} x_1 x_3 x_2 x_3 &= acaabcaa \\ y_1 y_3 y_2 y_3 &= acaabcaa \end{aligned}$$

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

**Lemma 148.** Für jedes Alphabet  $\Sigma$  gilt  $\text{PCP}_{\Sigma} \leq \text{PCP}_{\{a,b\}}$ .

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

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

$f$  reduziert z.B. die  $\text{PCP}_{\{0,1,2\}}$ -Instanz  $I = \begin{pmatrix} 0 & 01 & 200 \\ 020 & 12 & 00 \end{pmatrix}$  auf die äquivalente  $\text{PCP}_{\{a,b\}}$ -Instanz  $f(I) = \begin{pmatrix} a & aab & abbaa \\ aabba & ababb & aa \end{pmatrix}$ .

Im Folgenden lassen wir im Fall  $\Sigma = \{a, b\}$  den Index weg und schreiben einfach PCP (bzw. MPCP).

**Satz 149.** MPCP  $\leq$  PCP.

*Beweis.* Wir zeigen MPCP  $\leq$  PCP $_{\Sigma}$  für  $\Sigma = \{a, b, \langle, |, \rangle\}$ . Für ein Wort  $w = w_1 \dots w_n$  sei

$$\frac{\overleftarrow{w} \quad \overleftarrow{w} \quad \overleftarrow{w} \quad \overleftarrow{w}}{\langle w_1 | \dots | w_n \quad \langle w_1 | \dots | w_n \quad | w_1 | \dots | w_n \quad w_1 | \dots | w_n \rangle}$$

Wir reduzieren MPCP mittels folgender Funktion  $f$  auf PCP $_{\Sigma}$ :

$$f : \left( \begin{array}{c} x_1 \dots x_k \\ y_1 \dots y_k \end{array} \right) \mapsto \left( \begin{array}{c} \overleftarrow{x_1} \quad \overleftarrow{x_1} \dots \overleftarrow{x_k} \\ \overleftarrow{y_1} \quad \overleftarrow{y_1} \dots \overleftarrow{y_k} \quad | \end{array} \right).$$

Beispielsweise ist

$$f \left( \begin{array}{c} aa \quad b \quad bab \quad bb \\ aab \quad bb \quad a \quad b \end{array} \right) = \left( \begin{array}{c} \langle a|a| \quad a|a| \quad b| \quad b|a|b| \quad b|b| \rangle \\ \langle a|a|b| \quad a|a|b| \quad |b|b| \quad |a| \quad |b| \rangle \end{array} \right).$$

Da jede MPCP-Lösung  $\alpha = (1, i_2, \dots, i_n)$  für  $I$  auf eine PCP-Lösung  $\alpha' = (1, i_2 + 1, \dots, i_n + 1, k + 2)$  für  $f(I)$  führt, folgt

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

Für die umgekehrte Implikation sei  $\alpha' = (i_1, \dots, i_n)$  eine PCP-Lösung für

$$f(I) = \left( \begin{array}{c} \overleftarrow{x_1} \quad \overleftarrow{x_1} \dots \overleftarrow{x_k} \\ \overleftarrow{y_1} \quad \overleftarrow{y_1} \dots \overleftarrow{y_k} \quad | \end{array} \right).$$

Dann muss  $i_1 = 1$  sein, da  $(\overleftarrow{x_1}, \overleftarrow{y_1})$  das einzige Paar in  $f(I)$  ist, bei dem beide Komponenten mit demselben Buchstaben anfangen. Zudem muss  $i_n = k + 2$  sein, da nur das Paar  $(\rangle, |)$  mit demselben Buchstaben aufhört. Wählen wir  $\alpha'$  von minimaler Länge, so ist  $i_j \in \{2, \dots, k + 1\}$  für  $j = 2, \dots, n - 1$ . Dann ist aber

$$\alpha = (i_1, i_2 - 1, \dots, i_{n-1} - 1)$$

eine MPCP-Lösung für  $I$ . ■

**Satz 150.** PCP ist RE-vollständig und damit unentscheidbar.

*Beweis.* Es ist leicht zu sehen, dass PCP  $\in$  RE ist. Um zu zeigen, dass PCP RE-hart ist, sei  $A$  eine beliebige Sprache in RE und sei  $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$  eine 1-DTM mit  $L(M) = A$ . Wir zeigen  $A \leq \text{MPCP}_{\Sigma'}$  für  $\Sigma' = \Gamma \cup Z \cup \{\langle, |, \rangle\}$ . Wegen MPCP $_{\Sigma'} \leq$  PCP folgt hieraus  $A \leq$  PCP.

*Idee:* Transformiere eine Eingabe  $w \in \Sigma^*$  in eine Instanz  $g(w) = \left( \begin{array}{c} x_1 \dots x_k \\ y_1 \dots y_k \end{array} \right)$ , so dass  $\alpha = (i_1, \dots, i_n)$  genau dann eine MPCP-Lösung für  $g(w)$  ist, wenn das zugehörige Lösungswort  $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$  eine akzeptierende Rechnung von  $M(w)$  kodiert.

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

1.  $(\langle, \langle | z_0 w)$ , „Startregel“
2. für alle  $a \in \Gamma \cup \{|\}$ :  $(a, a)$ , „Kopierregeln“
3. für alle  $a, a', b \in \Gamma, z, z' \in Z$ : „Überführungsregeln“
  - $(za, z'a')$ , falls  $\delta(z, a) = (z', a', N)$ ,
  - $(za, a'z')$ , falls  $\delta(z, a) = (z', a', R)$ ,
  - $(bza, z'ba')$ , falls  $\delta(z, a) = (z', a', L)$ ,
  - $(|za, |z' \sqcup a')$ , falls  $\delta(z, a) = (z', a', L)$ ,
  - $(bz|, z'ba'|)$ , falls  $\delta(z, \sqcup) = (z', a', L)$ ,
  - $(z|, z'a'|)$ , falls  $\delta(z, \sqcup) = (z', a', N)$ ,
  - $(z|, a'z'|)$ , falls  $\delta(z, \sqcup) = (z', a', R)$ ,
4. für alle  $e \in E, a \in \Gamma$ :  $(ae, e)$ ,  $(ea, e)$ , „Löschregeln“
5. sowie für alle  $e \in E$ :  $(e|, \rangle)$ . „Abschlussregeln“

Ist nun

$$K_0 = z_0 w \vdash K_1 \vdash \dots \vdash K_t = uev$$

eine akzeptierende Rechnung von  $M(w)$  mit  $e \in E$ , so lässt sich aus dieser leicht eine MPCP-Lösung  $\alpha$  mit dem Lösungswort

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

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

Zudem ist leicht zu sehen, dass auch umgekehrt jedes Lösungswort  $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$  eine akzeptierende Rechnung von  $M$  bei Eingabe  $w$  beinhaltet. Somit gilt

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

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

Der Satz lässt sich alternativ auch unter Verwendung einer Typ-0 Grammatik  $G = (V, \Sigma, P, S)$  für  $A$  beweisen. Sei  $\Gamma = V \cup \Sigma \cup \{ \langle, |, \rangle \}$ . Dann können wir eine Eingabe  $w \in \Sigma^*$  in eine  $\text{MPCP}_{\Gamma}$ -Instanz  $f(w) = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$  transformieren, so dass  $\alpha = (i_1, \dots, i_n)$  genau dann eine Lösung für  $f(w)$  ist, wenn das zugehörige Lösungswort  $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$  eine Ableitung von  $w$  in  $G$  kodiert. Dies erreichen wir, indem wir  $f(w)$  aus folgenden Wortpaaren bilden:

1.  $(\langle, \langle | S)$  „Startregel“
2. für jede Regel  $l \rightarrow r$  in  $P$ :  $(l, r)$ , „Ableitungsregeln“
3. für alle  $a \in V \cup \Sigma \cup \{ | \}$ :  $(a, a)$ , „Kopierregeln“
4. sowie das Paar  $(w | \rangle, \rangle)$  „Abschlussregel“

Nun lässt sich leicht aus einer Ableitung  $S = \alpha_0 \Rightarrow \dots \Rightarrow \alpha_m = w$  von  $w$  in  $G$  eine MPCP-Lösung mit dem Lösungswort

$$\langle | \alpha_0 | \alpha_1 | \dots | \alpha_m | \rangle$$

angeben. Zudem lässt sich auch umgekehrt aus jeder MPCP-Lösung eine Ableitung von  $w$  in  $G$  gewinnen, womit

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

gezeigt ist. ■

**Beispiel 151.** Betrachte die 1-DTM  $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  mit  $Z = \{q_0, \dots, q_4\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{a, b, A, B, \sqcup\}$ ,  $E = \{q_4\}$  und den Anweisungen

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

Dann enthält die MPCP-Instanz  $g(ab)$  für  $u \in \Gamma$  die Wortpaare

Startregel	Kopierregeln	Löschregeln	Abschlussregel
$(\langle, \langle   q_0ab)$	$(u, u), ( ,  )$	$(q_4u, q_4), (uq_4, q_4)$	$(q_4   \rangle, \rangle)$

sowie folgende Überführungsregeln:

Anweisung	zugehörige Überführungsregeln
$q_0a \rightarrow q_1AR$ (1)	$(q_0a, Aq_1)$
$q_1a \rightarrow q_1aR$ (2)	$(q_1a, aq_1)$
$q_1B \rightarrow q_1BR$ (3)	$(q_1B, Bq_1)$
$q_1b \rightarrow q_2BL$ (4)	$(uq_1b, q_2uB), ( q_1b,  q_2\sqcup B)$
$q_2a \rightarrow q_2aL$ (5)	$(uq_2a, q_2ua), ( q_2a,  q_2\sqcup a)$
$q_2B \rightarrow q_2BL$ (6)	$(uq_2B, q_2uB), ( q_2B,  q_2\sqcup B)$
$q_2A \rightarrow q_0AR$ (7)	$(q_2A, Aq_0)$
$q_0B \rightarrow q_3BR$ (8)	$(q_0B, Bq_3)$
$q_3B \rightarrow q_3BR$ (9)	$(q_3B, Bq_3)$
$q_3\sqcup \rightarrow q_4\sqcup N$ (10)	$(q_3\sqcup, q_4\sqcup), (q_3 , q_4 )$

Der akzeptierenden Rechnung

$$q_0ab \stackrel{(1)}{\vdash} Aq_1b \stackrel{(4)}{\vdash} q_2AB \stackrel{(7)}{\vdash} Aq_0B \stackrel{(8)}{\vdash} ABq_3\sqcup \stackrel{(10)}{\vdash} ABq_4\sqcup$$

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

$$\begin{aligned} & \langle |q_0ab| Aq_1b| q_2AB| Aq_0B| ABq_3| ABq_4\sqcup| Aq_4\sqcup| q_4\sqcup| q_4| \rangle \\ & \langle |q_0ab| Aq_1b| q_2AB| Aq_0B| ABq_3| ABq_4\sqcup| Aq_4\sqcup| q_4\sqcup| q_4| \rangle \end{aligned}$$

Betrachte die Grammatik  $G = (\{S\}, \{a, b\}, \{S \rightarrow aSbS, \varepsilon\}, S)$  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 MPCP-Instanz  $f(aabb)$  enthält dann für  $u \in \{S, a, b, |\}$  die Paare

Startregel	Ableitungsregeln	Kopierregeln	Abschlussregel
$(\langle, \langle  S)$	$(S, aSbS), (S, \varepsilon)$	$(u, u)$	$(aabb ), \rangle)$

Obiger Ableitung entspricht dann folgendes Lösungswort für  $f(aabb)$ :

$$\begin{aligned} & \langle |S| aSbS| aaSbSbS| aaSbbS| aabbS| aabb| \rangle \\ & \langle |S| aSbS| aaSbSbS| aaSbbS| aabbS| aabb| \rangle \end{aligned}$$

Das kürzeste MPCP-Lösungswort für  $f(aabb)$  ist

$$\begin{aligned} & \langle |S| aSbS| aaSbSb| aabb| \rangle \\ & \langle |S| aSbS| aaSbSb| aabb| \rangle \end{aligned}$$

Dieses entspricht der „parallelisierten“ Ableitung

$$\underline{S} \Rightarrow a\underline{S}b\underline{S} \Rightarrow^2 aa\underline{S}b\underline{S}b \Rightarrow^2 aabb$$

◁

## 5.4 Weitere Unentscheidbarkeitsresultate

In diesem Abschnitt leiten wir aus der Unentscheidbarkeit des Post-schen Korrespondenzproblems eine Reihe von weiteren Unentscheidbarkeitsresultaten her. Wir zeigen zuerst, dass das Schnittproblem für kontextfreie Grammatiken unentscheidbar ist.

### Schnittproblem für kontextfreie Grammatiken

**Gegeben:** Zwei kontextfreie Grammatiken  $G_1$  und  $G_2$ .

**Gefragt:** Ist  $L(G_1) \cap L(G_2) \neq \emptyset$ ?

**Satz 152.** Das Schnittproblem für kontextfreie Grammatiken ist RE-vollständig.

*Beweis.* Es ist leicht zu sehen, dass das Problem semi-entscheidbar ist. Um PCP auf dieses Problem zu reduzieren, betrachten wir für eine Folge  $s = (x_1, \dots, x_k)$  von Strings  $x_i \in \{a, b\}^*$  die Sprache

$$L_s = \{i_n \dots i_1 x_{i_1} \dots x_{i_n} \mid n \geq 1, 1 \leq i_1, \dots, i_n \leq k\}.$$

$L_s$  wird von der Grammatik  $G_s = (\{S\}, \{a, b, 1, \dots, k\}, P_s, S)$  erzeugt mit

$$P_s: S \rightarrow 1Sx_1, \dots, kSx_k, 1x_1, \dots, kx_k$$

Zu einer PCP-Instanz  $I = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$  bilden wir das Paar  $(G_s, G_t)$ , wobei  $s = (x_1, \dots, x_k)$  und  $t = (y_1, \dots, y_k)$  ist. Dann ist  $L(G_s) \cap L(G_t)$  die Sprache

$$\{i_n \dots i_1 x_{i_1} \dots x_{i_n} \mid 1 \leq n, x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}\}.$$

Folglich ist  $\alpha = (i_1, \dots, i_n)$  genau dann eine Lösung für  $I$ , wenn  $i_n \dots i_1 x_{i_1} \dots x_{i_n} \in L(G_s) \cap L(G_t)$  ist. Also vermittelt  $f: I \mapsto (G_s, G_t)$  eine Reduktion von PCP auf das Schnittproblem für CFL. ■

**Beispiel 153.** Die PCP-Instanz

$$I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & aab & abbaa \\ aabba & ababb & aa \end{pmatrix}$$

wird auf das Grammatikpaar  $(G_s, G_t)$  mit folgenden Regeln reduziert:

$$\begin{aligned} P_s: S &\rightarrow 1Sa, 2Saab, 3Sabbaa, 1a, 2aab, 3abbaa, \\ P_t: S &\rightarrow 1Saabba, 2Sababb, 3Saa, 1aabba, 2ababb, 3aa. \end{aligned}$$

Der PCP-Lösung  $\alpha = (1, 3, 2, 3)$  entspricht dann das Wort

$$\begin{aligned} 3231x_1x_3x_2x_3 &= 3231aabbbaaababbaa \\ &= 3231aabbbaaababbaa = 3231y_1y_3y_2y_3 \end{aligned}$$

in  $L(G_s) \cap L(G_t)$ . ◀

Des weiteren erhalten wir die Unentscheidbarkeit des Schnitt- und des Inklusionsproblems für DPDAs.

### Inklusionsproblem für DPDAs

**Gegeben:** Zwei DPDAs  $M_1$  und  $M_2$ .

**Gefragt:** Ist  $L(M_1) \subseteq L(M_2)$ ?

### Korollar 154.

- (i) Das Schnittproblem für DPDAs ist RE-vollständig.
- (ii) Das Inklusionsproblem für DPDAs ist co-RE-vollständig.

Korollar 154 wird in den Übungen bewiesen. Die Idee dabei ist, die kontextfreie Grammatik  $G_s$  im Beweis von Satz 152 in DPDAs  $M_s$  und  $\bar{M}_s$  zu überführen mit  $L(M_s) = L(G_s)$  und  $L(\bar{M}_s) = \bar{L}(G_s)$ .

Schließlich ergeben sich für CFL noch folgende Unentscheidbarkeitsresultate.

**Korollar 155.** Für kontextfreie Grammatiken sind folgende Fragestellungen unentscheidbar:

- (i) Ist  $L(G) = \Sigma^*$ ? (Ausschöpfungsproblem)
- (ii) Ist  $L(G_1) = L(G_2)$ ? (Äquivalenzproblem)
- (iii) Ist  $G$  mehrdeutig? (Mehrdeutigkeitsproblem)

*Beweis.*

- (i) Wir reduzieren das Komplement von PCP auf das Ausschöpfungsproblem für CFL. Es gilt

$$I \notin \text{PCP} \Leftrightarrow L_s \cap L_t = \emptyset \Leftrightarrow \bar{L}_s \cup \bar{L}_t = \Sigma^*,$$

wobei  $L_s$  und  $L_t$  die im Beweis von Satz 152 definierten Sprachen sind. Diese sind sogar in DCFL und in den Übungen wird gezeigt, dass sich DLBAs (und damit auch kontextfreie Grammatiken) für  $\bar{L}_s$  und  $\bar{L}_t$  aus  $I$  berechnen lassen. Daher vermittelt die Funktion  $f: I \mapsto G$ , wobei  $G$  eine kontextfreie Grammatik mit

$$L(G) = \bar{L}_s \cup \bar{L}_t$$

ist, die gewünschte Reduktion.

- (ii) Wir zeigen, dass das Äquivalenzproblem für CFL ebenfalls co-RE-vollständig ist, indem wir das Ausschöpfungsproblem für CFL darauf reduzieren. Dies leistet beispielsweise die Funktion

$$f: G \mapsto (G, G_{all}),$$

wobei  $G_{all}$  eine kontextfreie Grammatik mit  $L(G_{all}) = \Sigma^*$  ist.

- (iii) Schließlich zeigen wir, dass das Mehrdeutigkeitsproblem RE-vollständig ist, indem wir PCP darauf reduzieren. Betrachte die Funktion  $f: \binom{x_1 \dots x_k}{y_1 \dots y_k} \mapsto G$  mit

$$G = (\{S, A, B\}, \{a, b, 1, \dots, k\}, P_1 \cup P_2 \cup \{S \rightarrow A, S \rightarrow B\}, S)$$

und den Regeln

$$P_1: A \rightarrow 1Ax_1, \dots, kAx_k, 1x_1, \dots, kx_k,$$

$$P_2: B \rightarrow 1By_1, \dots, kB y_k, 1y_1, \dots, ky_k.$$

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

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

Wie wir im Beweis der Unentscheidbarkeit des Schnittproblems für CFL gesehen haben, ist dies genau dann der Fall, wenn die PCP-Instanz  $I = \begin{pmatrix} x_1 \dots x_k \\ y_1 \dots y_k \end{pmatrix}$  eine PCP-Lösung hat. ■

Als weitere Folgerung erhalten wir die Unentscheidbarkeit des Leerheitsproblems für DLBAs.

### Leerheitsproblem für DLBAs

**Gegeben:** Ein DLBA  $M$ .  
**Gefragt:** Ist  $L(M) = \emptyset$ ?

**Satz 156.** Das Leerheitsproblem für DLBAs ist co-RE-vollständig.

*Beweis.* Wir reduzieren das Ausschöpfungsproblem für CFL auf das Leerheitsproblem für DLBAs. Eine kontextfreie Grammatik  $G$  lässt sich wie folgt in einen DLBA  $M$  mit  $L(M) = \overline{L(G)}$  überführen (siehe Übungen):

Bestimme zunächst einen DLBA  $M$  mit  $L(M) = L(G)$ .

Konstruiere daraus einen DLBA  $\overline{M}$  mit  $L(\overline{M}) = \overline{L(M)}$ .

Dann gilt  $L(G) = \Sigma^* \Leftrightarrow L(\overline{M}) = \emptyset$ , d.h. die Funktion  $f : G \mapsto \overline{M}$  berechnet die gewünschte Reduktion. ■

Dagegen ist es nicht schwer, für eine kontextfreie Grammatik  $G$  zu entscheiden, ob mindestens ein Wort in  $G$  ableitbar ist. Ebenso ist es möglich, für eine kontextsensitive Grammatik  $G$  und ein Wort  $x$  zu entscheiden, ob  $x$  in  $G$  ableitbar ist.

### Satz 157.

- (i) Das Leerheitsproblem für kfr. Grammatiken ist entscheidbar.
- (ii) Das Wortproblem für kontextsensitive Grammatiken ist entscheidbar.

## 5.5 Die arithmetische Hierarchie

Mit Hilfe der arithmetischen Hierarchie lassen sich unentscheidbare Probleme entsprechend dem Grad ihrer Unentscheidbarkeit einordnen.

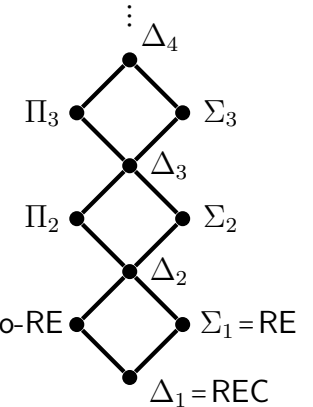
### Definition 158.

a) Sei  $A \subseteq \Sigma^*$  eine Sprache und  $\mathcal{K}$  eine Sprachklasse. Dann ist

$$\exists A = \{x \in \Sigma^* \mid \exists y \in \{0, 1\}^* : x\#y \in A\}$$

und

$$\exists \mathcal{K} = \mathcal{K} \cup \{\exists A \mid A \in \mathcal{K}\}. \quad \Pi_1 = \text{co-RE} \quad \Sigma_1 = \text{RE}$$



b) Weiter sei  $\Sigma_0 = \text{REC}$ , sowie  $\Pi_i = \text{co-}\Sigma_i$ ,  $\Delta_i = \Sigma_i \cap \Pi_i$  und  $\Sigma_{i+1} = \exists \Pi_i$  für  $i \geq 0$ .

c) Die Sprachklassen  $\Sigma_i, \Pi_i, \Delta_i$ ,  $i \geq 0$ , bilden die Stufen der **arithmetischen Hierarchie**.

### Proposition 159.

- (i)  $\Sigma_1 = \text{RE}$ ,  $\Pi_1 = \text{co-RE}$ .
- (ii)  $\Delta_0 = \Sigma_0 = \Pi_0 = \Delta_1 = \text{REC}$ .
- (iii)  $\Sigma_i \cup \Pi_i \subseteq \Delta_{i+1}$  für  $i \geq 0$ .

*Beweis.*

(i) Wir zeigen zuerst die Inklusion  $\Sigma_1 \subseteq \text{RE}$ . Sei  $A \in \Sigma_1$  und sei  $A = \exists B$  für eine Sprache  $B \in \text{REC}$ . Dann wird  $A$  von einer DTM akzeptiert, die bei Eingabe  $x$  systematisch nach einem String  $y \in \{0, 1\}^*$  mit  $x\#y \in B$  sucht. Folglich ist  $A \in \text{RE}$ .

Für die umgekehrte Inklusion sei  $A \subseteq \Sigma^*$  eine Sprache in RE und sei  $A = L(M)$  für eine DTM  $M$ . Dann ist  $A = \exists B$  für die Sprache  $B = \{x\#y \mid y \text{ kodiert eine akz. Rechnung von } M(x)\}$ . Da



$B$  entscheidbar ist, folgt  $A \in \exists\text{REC}$ . Die Gleichheit  $\Pi_1 = \text{co-RE}$  folgt wegen  $\Pi_1 = \text{co-}\Sigma_1$  direkt aus  $\Sigma_1 = \text{RE}$ .

(ii) Die Gleichheiten  $\Delta_0 = \Sigma_0 = \Pi_0 = \Delta_1 = \text{REC}$  ergeben sich unmittelbar aus  $\text{REC} = \text{co-REC}$  und  $\text{REC} = \text{RE} \cap \text{co-RE}$ .

(iii) Die Inklusion  $\Sigma_i \subseteq \Pi_{i+1}$  ist klar, da  $\Sigma_i \subseteq \exists\Sigma_i = \Pi_{i+1}$  ist.

Die Inklusionen  $\Sigma_i \subseteq \Sigma_{i+1}$  lassen sich induktiv wie folgt zeigen.

$i = 0$ : Es gilt  $\Sigma_0 = \text{REC} \subseteq \text{RE} = \Sigma_1$ .

$i \rightsquigarrow i + 1$ : Nach IV gilt  $\Sigma_{i-1} \subseteq \Sigma_i$ . Folglich ist  $\Pi_{i-1} \subseteq \Pi_i$ , was wegen der Monotonie des  $\exists$ -Operators (d.h.  $\mathcal{K} \subseteq \mathcal{K}' \Rightarrow \exists\mathcal{K} \subseteq \exists\mathcal{K}'$ ) wiederum  $\Sigma_i = \exists\Pi_{i-1} \subseteq \exists\Pi_i = \Sigma_{i+1}$  impliziert.

Durch Übergang zu den **co**-Klassen erhalten wir auch die Inklusionen  $\Pi_i \subseteq \Pi_{i+1}$  und  $\Pi_i \subseteq \Sigma_{i+1}$ . ■

Mittels Diagonalisierung kann man zeigen, dass die Klassen  $\Sigma_i$  und  $\Pi_i$  für  $i \geq 1$  verschieden sind. Dies impliziert, dass die Inklusionen  $\Sigma_i \subseteq \Delta_{i+1} \subseteq \Sigma_{i+1}$  und  $\Pi_i \subseteq \Delta_{i+1} \subseteq \Pi_{i+1}$  für alle  $i \geq 1$  echt sind.

Die betrachteten Entscheidungsprobleme für die verschiedenen Stufen der Chomsky-Hierarchie lassen sich nun wie folgt in die arithmetische Hierarchie einordnen.

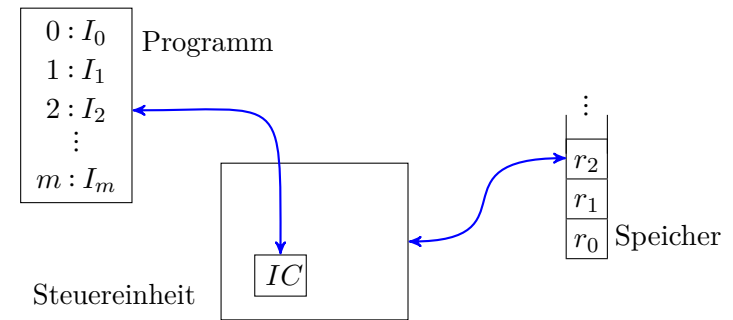
	Wort- problem $x \in L?$	Leerheits- problem $L = \emptyset?$	Aus- schöpfung $L = \Sigma^*?$	Äquivalenz- problem $L_1 = L_2?$	Inklusions- problem $L_1 \subseteq L_2$	Schnitt- problem $L_1 \cap L_2 \neq \emptyset?$
REG	REC	REC	REC	REC	REC	REC
DCFL	REC	REC	REC	REC <sup>a</sup>	co-RE	RE
CFL	REC	REC	co-RE	co-RE	co-RE	RE
D CSL	REC	co-RE	co-RE	co-RE	co-RE	RE
CSL	REC	co-RE	co-RE	co-RE	co-RE	RE
RE	RE	co-RE	$\Pi_2$	$\Pi_2$	$\Pi_2$	RE

<sup>a</sup>Bewiesen in 1997 von Géraud Sénizergues (Univ. Bordeaux).

Tatsächlich sind alle betrachteten Entscheidungsprobleme sogar vollständig für die angegebenen Sprachklassen.

## 5.6 GOTO-Berechenbarkeit

In diesem Abschnitt führen wir das Rechenmodell der Registermaschine (*random access machine, RAM*) ein und zeigen, dass es die gleiche Rechenstärke wie das Modell der Turingmaschine besitzt. Das Modell der RAM ist an die realen Rechenmaschinen angelehnt, die über einen Prozessor mit einem vorgegebenen Befehlssatz verfügen.



Die Registermaschine

- führt ein Programm  $P = I_0, \dots, I_m$  aus, das aus einer endlichen Folge von Befehlen (*instructions*)  $I_i$  besteht,
- hat einen Befehlszähler (*instruction counter*)  $IC$ , der die Nummer des nächsten Befehls angibt (zu Beginn ist  $IC = 0$ ),
- verfügt über einen frei adressierbaren Speicher (*random access memory*) mit unendlich vielen Speicherzellen (Registern)  $r_i$ , die beliebig große natürliche Zahlen aufnehmen können.

Auf Registermaschinen lassen sich **GOTO-Programme** ausführen, die über folgenden Befehlssatz verfügen ( $i, j, c \in \mathbb{N}$ ):

Befehl	Semantik
$r_i := r_j + c$	setzt Register $r_i$ auf den Wert $r_j + c$
$r_i := r_j \div c$	setzt Register $r_i$ auf $\max(0, r_j - c)$
<b>GOTO</b> $j$	setzt den Befehlszähler $IC$ auf $j$
<b>IF</b> $r_i = c$ <b>THEN GOTO</b> $j$	setzt $IC$ auf $j$ , falls $r_i = c$ ist
<b>HALT</b>	beendet die Programmausführung

Falls nichts anderes angegeben ist, wird zudem  $IC$  auf den Wert  $IC+1$  gesetzt.

Eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$  heißt **GOTO-berechenbar**, falls es ein GOTO-Programm  $P = (I_0, \dots, I_m)$  mit folgender Eigenschaft gibt:

Wird  $P$  auf einer RAM mit den Werten  $r_i = n_i$  für  $i = 1, \dots, k$ , sowie  $IC = 0$  und  $r_i = 0$  für  $i = 0, k+1, k+2, \dots$  gestartet, so gilt:

- $P$  hält genau dann, wenn  $(n_1, \dots, n_k) \in \text{dom}(f)$  ist, und
- falls  $P$  hält, hat  $r_0$  nach Beendigung von  $P$  den Wert  $f(n_1, \dots, n_k)$ .

**Beispiel 160.** Folgendes GOTO-Programm berechnet die Funktion  $f(x, y) = xy$ :

```

0  IF  $r_1 = 0$  THEN GOTO 4
1   $r_1 := r_1 \div 1$ 
2   $r_0 := r_0 + r_2$ 
3  GOTO 0
4  HALT
    
```

Dabei ist der Befehl  $r_0 := r_0 + r_2$  in Zeile 2 zwar unzulässig. Wir können ihn jedoch durch den Befehl **GOTO** 5 ersetzen und folgende Befehle hinzufügen.

```

5   $r_3 := r_2$ 
6  IF  $r_3 = 0$  THEN GOTO 3
    
```

```

7   $r_3 := r_3 \div 1$ 
8   $r_0 := r_0 + 1$ 
9  GOTO 6
    
```

◁

Um mit einem GOTO-Programm auch Wortfunktionen berechnen zu können, müssen wir Wörter numerisch repräsentieren.

Sei  $\Sigma = \{a_0, \dots, a_{m-1}\}$  ein Alphabet. Dann können wir jedes Wort  $x = a_{i_1} \dots a_{i_n} \in \Sigma^*$  durch eine natürliche Zahl  $\text{num}_\Sigma(w)$  kodieren:

$$\text{num}_\Sigma(x) = \sum_{j=0}^{n-1} m^j + \sum_{j=1}^n i_j m^{n-j} = \frac{m^n - 1}{m - 1} + (i_1 \dots i_n)_m.$$

Da die Abbildung  $\text{num}_\Sigma : \Sigma^* \rightarrow \mathbb{N}$  bijektiv ist, können wir umgekehrt jede natürliche Zahl  $n$  durch das Wort  $\text{str}_\Sigma(n) = \text{num}_\Sigma^{-1}(n)$  kodieren.

Für das Alphabet  $\Sigma = \{a, b, c\}$  erhalten wir beispielsweise folgende Kodierung:

$w$	$\varepsilon$	$a$	$b$	$c$	$aa$	$ab$	$ac$	$ba$	$bb$	$bc$	$ca$	$cb$	$cc$	$aaa$	...
$\text{num}_\Sigma(w)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...

Ist  $\Sigma = \{0, 1\}$ , so lassen wir den Index weg und schreiben einfach  $\text{num}$  und  $\text{str}$  anstelle von  $\text{num}_\Sigma$  und  $\text{str}_\Sigma$ . Zudem erweitern wir die Kodierungsfunktion  $\text{str} : \mathbb{N} \rightarrow \{0, 1\}$  zu einer Kodierungsfunktion  $\text{str}_k : \mathbb{N}^k \rightarrow \{0, 1, \#\}$  wie folgt:

$$\text{str}_k(n_1, \dots, n_k) = \text{str}(n_1)\# \dots \#\text{str}(n_k).$$

Nun können wir eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$  durch folgende partielle Wortfunktion  $\hat{f} : \{0, 1, \#\}^* \rightarrow \{0, 1\}^* \cup \{\uparrow\}$  repräsentieren:

$$\hat{f}(w) = \begin{cases} \text{str}(n), & w = \text{str}_k(n_1, \dots, n_k) \text{ und } f(n_1, \dots, n_k) = n \in \mathbb{N}, \\ \uparrow, & \text{sonst.} \end{cases}$$

Es ist klar, dass  $f$  durch  $\hat{f}$  eindeutig bestimmt ist. Wir nennen  $f$  die *numerische Repräsentation* von  $\hat{f}$ .

**Satz 161.** Sei  $f$  die numerische Repräsentation einer partiellen Funktion  $\hat{f}$ . Dann ist  $f$  genau dann GOTO-berechenbar, wenn  $\hat{f}$  berechenbar ist.

*Beweis.* Wir zeigen zuerst die Simulation eines GOTO-Programms durch eine DTM. Sei  $\hat{f}$  eine partielle Funktion, deren numerische Repräsentation  $f$  von einem GOTO-Programm  $P$  auf einer RAM  $R$  berechnet wird. Dann existiert eine Zahl  $k'$ , so dass  $P$  nur Register  $r_i$  mit  $i \leq k'$  benutzt. Daher lässt sich eine Konfiguration von  $R$  durch Angabe der Inhalte des Befehlszählers  $IC$  und der Register  $r_0, \dots, r_{k'}$  beschreiben. Wir konstruieren eine  $(k' + 2)$ -DTM  $M$ , die

- den Inhalt von  $IC$  in ihrem Zustand,
- die Registerwerte  $r_1, \dots, r_{k'}$  auf den Bändern  $1, \dots, k'$  und
- den Wert von  $r_0$  auf dem Ausgabeband  $k' + 2$  speichert.

Ein Registerwert  $r_i$  wird hierbei in der Form  $str(r_i)$  gespeichert. Band  $k' + 1$  wird zur Ausführung von Hilfsberechnungen benutzt.

Die Aufgabe von  $M$  ist es, bei Eingabe  $w = str_k(n_1, \dots, n_k)$  das Wort  $str(f(n_1, \dots, n_k))$  auszugeben, wenn  $(n_1, \dots, n_k) \in dom(f)$  ist, und andernfalls nicht zu halten.

Zuerst kopiert  $M$  die Teilwörter  $str(n_i)$  für  $i = 2, \dots, k$  auf das  $i$ -te Band und löscht auf dem 1. Band alle Eingabezeichen bis auf  $str(n_1)$ . Da das leere Wort den Wert  $num(\varepsilon) = 0$  kodiert, sind nun auf den Bändern  $1, \dots, k'$  und auf Band  $k' + 2$  die der Startkonfiguration von  $R$  bei Eingabe  $(n_1, \dots, n_k)$  entsprechenden Registerinhalte gespeichert. Danach führt  $M$  das Programm  $P$  Befehl für Befehl aus.

Es ist leicht zu sehen, dass sich jeder Befehl  $I$  in  $P$  durch eine Folge von Anweisungen realisieren lässt, die die auf den Bändern gespeicherten Registerinhalte bzw. den im Zustand von  $M$  gespeicherten Wert von  $IC$  entsprechend modifizieren. Sobald  $P$  stoppt, hält auch  $M$  und gibt das Wort  $str(r_0) = str(f(n_1, \dots, n_k))$  aus.

Nun betrachten wir die Simulation einer DTM durch ein GOTO-Programm. Sei  $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$  eine partielle Funktion und sei

$M = (Z, \Sigma, \Gamma, \delta, q_0, E)$  eine DTM mit Eingabealphabet  $\Sigma = \{0, 1, \#\}$ , die die zugehörige Wortfunktion  $\hat{f} : \Sigma^* \rightarrow \{0, 1\}^* \cup \{\uparrow\}$  berechnet.  $M$  gibt also bei Eingabe  $str_k(n_1, \dots, n_k)$  das Wort  $str(f(n_1, \dots, n_k))$  aus, falls  $f(n_1, \dots, n_k)$  definiert ist, und hält andernfalls nicht.

Wir konstruieren ein GOTO-Programm  $P$ , das bei Eingabe  $(n_1, \dots, n_k)$  die DTM  $M$  bei Eingabe  $w = str_k(n_1, \dots, n_k)$  simuliert und im Fall, dass  $M(w)$  hält, den Wert  $num(M(w)) = f(n_1, \dots, n_k)$  berechnet. Wir können annehmen, dass  $M$  eine 1-DTM ist. Sei  $Z = \{q_0, \dots, q_r\}$  und  $\Gamma = \{a_0, \dots, a_{m-1}\}$ , wobei wir annehmen, dass  $a_0 = \sqcup$ ,  $a_1 = 0$ ,  $a_2 = 1$  und  $a_3 = \#$  ist.

Eine Konfiguration  $K = uq_iv$  von  $M$  wird wie folgt in den Registern  $r_0, r_1, r_2$  gespeichert. Sei  $u = a_{i_1} \dots a_{i_n}$  und  $v = a_{j_1} \dots a_{j_{n'}}$ .

- $r_0 = (i_1 \dots i_n)_m$ ,
- $r_1 = i$ ,
- $r_2 = (j_{n'} \dots j_1)_m$ .

$P$  besteht aus 3 Programmteilen  $P = P_1, P_2, P_3$ .  $P_1$  berechnet in Register  $r_2$  die Zahl  $(j_n \dots j_1)_m$ , wobei  $(j_1, \dots, j_n)$  die Indexfolge der Zeichen von  $str_k(n_1, \dots, n_k) = a_{j_1} \dots a_{j_n}$  ist. Die übrigen Register setzt  $P_1$  auf den Wert 0.  $P_1$  stellt also die Startkonfiguration  $K_w = q_0w$  von  $M$  bei Eingabe  $w = str_k(n_1, \dots, n_k)$  in den Registern  $r_0, r_1, r_2$  her.

Anschließend führt  $P_2$  eine schrittweise Simulation von  $M$  aus. Hierzu überführt  $P_2$  solange die in  $r_0, r_1, r_2$  gespeicherte Konfiguration von  $M$  in die zugehörige Nachfolgekonfiguration, bis  $M$  hält. Das Programmstück  $P_2$  hat die Form

```

 $M_2$   $r_3 := r_2 \text{ MOD } m$ 
IF  $r_1 = 0 \wedge r_3 = 0$  THEN GOTO  $M_{0,0}$ 
 $\vdots$ 
IF  $r_1 = r \wedge r_3 = m - 1$  THEN GOTO  $M_{r,m-1}$ 

```

Die Befehle ab Position  $M_{i,j}$  hängen von  $\delta(q_i, a_j)$  ab. Wir betrachten exemplarisch den Fall  $\delta(q_i, a_j) = \{(q_{i'}, a_{j'}, L)\}$ :

$M_{i,j}$   $r_1 := i'$

```

 $r_2 := r_2 \text{ DIV } m$ 
 $r_2 := r_2 m + j'$ 
 $r_2 := r_2 m + (r_0 \text{ MOD } m)$ 
 $r_0 := r_0 \text{ DIV } m$ 
GOTO  $M_2$ 

```

Die hierbei benutzten Makrobefehle  $r_3 := r_2 \text{ MOD } m$ ,  $r_2 := r_2 \text{ DIV } m$  etc. können leicht durch GOTO-Programmstücke ersetzt werden (siehe Übungen).

Im Fall  $\delta(q_i, a_j) = \emptyset$  erfolgt ein Sprung an den Beginn von  $P_3$ .  $P_3$  transformiert den Inhalt  $r_0 = (j_1 \dots j_n)_m$  von Register  $r_0$  in die Zahl  $\text{num}(a_{j_1} \dots a_{j_n})$  und hält. ■

## 5.7 WHILE- und LOOP-Berechenbarkeit

Die Syntax von **WHILE-Programmen** ist induktiv wie folgt definiert ( $i, j, c \in \mathbb{N}$ ):

- Jede Wertzuweisung der Form  $x_i := x_j + c$  oder  $x_i := x_j \div c$  ist ein WHILE-Programm.
- Falls  $P$  und  $Q$  WHILE-Programme sind, so auch
  - $P; Q$  und
  - **IF**  $x_i = c$  **THEN**  $P$  **ELSE**  $Q$  **END**
  - **WHILE**  $x_i \neq c$  **DO**  $P$  **END**

Die Syntax von **LOOP-Programmen** ist genauso definiert, nur dass Schleifen der Form **LOOP**  $x_i$  **DO**  $P$  **END** an die Stelle von WHILE-Schleifen treten. Die Semantik von WHILE-Programmen ist selbsterklärend. Eine LOOP-Schleife **LOOP**  $x_i$  **DO**  $P$  **END** wird sooft ausgeführt, wie der Wert von  $x_i$  zu Beginn der Schleife angibt.

Eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$  heißt **WHILE-berechenbar**, falls es ein WHILE-Programm  $P$  mit folgender Eigenschaft gibt: Wird  $P$  mit den Werten  $x_i = n_i$  für  $i = 1, \dots, k$  gestartet,

so gilt:

- $P$  hält genau dann, wenn  $(n_1, \dots, n_k) \in \text{dom}(f)$  ist, und
- falls  $P$  hält, hat  $x_0$  nach Beendigung von  $P$  den Wert  $f(n_1, \dots, n_k)$ .

Die **LOOP-Berechenbarkeit** von  $f$  ist entsprechend definiert.

**Beispiel 162.** Die Funktion  $f(x_1, x_2) = x_1 x_2$  wird von dem WHILE-Programm

```

WHILE  $x_1 \neq 0$  DO
   $x_0 := x_0 + x_2;$ 
   $x_1 := x_1 \div 1$ 
END

```

sowie von folgendem LOOP-Programm berechnet:

```

LOOP  $x_1$  DO
   $x_0 := x_0 + x_2$ 
END

```

◁

Als nächstes beweisen wir die Äquivalenz von WHILE- und GOTO-Berechenbarkeit.

**Satz 163.** Eine partielle Funktion  $f : \mathbb{N}^k \rightarrow \mathbb{N} \cup \{\uparrow\}$  ist genau dann GOTO-berechenbar, wenn sie WHILE-berechenbar ist.

*Beweis.* Sei  $P$  ein WHILE-Programm, das  $f$  berechnet. Wir übersetzen  $P$  wie folgt in ein äquivalentes GOTO-Programm  $P'$ .  $P'$  speichert den Variablenwert  $x_i$  im Register  $r_i$ . Damit lassen sich alle Wertzuweisungen von  $P$  direkt in entsprechende Befehle von  $P'$  transformieren. Eine Schleife der Form **WHILE**  $x_i \neq c$  **DO**  $Q$  **END** simulieren wir durch folgendes GOTO-Programmstück:

```

 $M_1$  IF  $r_i = c$  THEN GOTO  $M_2$ 
       $Q'$ 

```

**GOTO**  $M_1$   
 $M_2$   
 $\vdots$

Ähnlich lässt sich die Verzweigung **IF**  $x_i = c$  **THEN**  $Q_1$  **ELSE**  $Q_2$  **END** in ein GOTO-Programmstück transformieren. Zudem fügen wir ans Ende von  $P'$  den HALT-Befehl an.

Für die umgekehrte Implikation sei nun  $P = (I_0, \dots, I_m)$  ein GOTO-Programm, das  $f$  berechnet, und sei  $r_z$ ,  $z > k$ , ein Register, das in  $P$  nicht benutzt wird. Dann können wir  $P$  wie folgt in ein äquivalentes WHILE-Programm  $P'$  übersetzen:

```

 $x_z := 0;$ 
WHILE  $x_z \neq m + 1$  DO
  IF  $x_z = 0$  THEN  $P'_0$  END;
   $\vdots$ 
  IF  $x_z = m$  THEN  $P'_m$  END
END

```

Dabei ist  $P'_j$  abhängig vom Befehl  $I_j$  folgendes WHILE-Programm:

- $r_i := r_k + c$ ;  $x_i := x_k + c$ ;  $x_z := x_z + 1$ ,
- $r_i := r_k \div c$ ;  $x_i := x_k \div c$ ;  $x_z := x_z + 1$ ,
- **GOTO**  $k$ :  $x_z := k$ ,
- **IF**  $r_i = c$  **THEN GOTO**  $k$ :  
     **IF**  $x_i = c$  **THEN**  $x_z := k$  **ELSE**  $x_z := x_z + 1$  **END**
- **HALT**:  $x_z := m + 1$ .

Man beachte, dass  $P'$  nur eine WHILE-Schleife enthält. ■

Es ist leicht zu sehen, dass sich jedes LOOP-Programm durch ein WHILE-Programm simulieren lässt. Offensichtlich können LOOP-Programme nur totale Funktionen berechnen. Daher kann nicht jedes WHILE-Programm durch ein LOOP-Programm simuliert werden. Mittels Diagonalisierung lässt sich eine totale WHILE-berechenbare

Funktion  $f$  angeben, die nicht LOOP-berechenbar ist. Ein bekanntes Beispiel einer totalen WHILE-berechenbaren Funktion, die nicht LOOP-berechenbar ist, ist die **Ackermannfunktion**  $a(x, y)$ , die induktiv wie folgt definiert ist:

$$a(x, y) = \begin{cases} y + 1, & x = 0, \\ a(x - 1, 1), & x \geq 1, y = 0, \\ a(x - 1, a(x, y - 1)), & x, y \geq 1. \end{cases}$$

## 6 Komplexitätsklassen

### 6.1 Zeitkomplexität

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

**Definition 164.**

a) Die **Laufzeit** einer NTM  $M$  bei Eingabe  $x$  ist definiert als

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

wobei  $\sup \mathbb{N} = \infty$  ist.

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

$$time_M(x) \leq t(|x|).$$

Die Zeitschranke  $t(n)$  beschränkt also die Laufzeit bei allen Eingaben der Länge  $n$  (worst-case Komplexität).

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

**Definition 165.**

a) Die in deterministischer Zeit  $t(n)$  entscheidbaren Sprachen bilden die Sprachklasse

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

b) Die in nichtdeterministischer Zeit  $t(n)$  entscheidbaren Sprachen bilden die Sprachklasse

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

c) Die in deterministischer Zeit  $t(n)$  berechenbaren Funktionen bilden die Funktionenklasse

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

Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

$$\mathbf{LINTIME} = \bigcup_{c \geq 1} \mathbf{DTIME}(cn + c) \quad \text{„Linearzeit“}$$

$$\mathbf{P} = \bigcup_{c \geq 1} \mathbf{DTIME}(n^c + c) \quad \text{„Polynomialzeit“}$$

$$\mathbf{E} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{cn+c}) \quad \text{„Lineare Exponentialzeit“}$$

$$\mathbf{EXP} = \bigcup_{c \geq 1} \mathbf{DTIME}(2^{n^c+c}) \quad \text{„Exponentialzeit“}$$

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

Für eine Funktionenklasse  $\mathcal{F}$  sei  $\mathbf{DTIME}(\mathcal{F}) = \bigcup_{t \in \mathcal{F}} \mathbf{DTIME}(t(n))$  (die Klassen  $\mathbf{NTIME}(\mathcal{F})$  und  $\mathbf{FTIME}(\mathcal{F})$  seien analog definiert).

### Asymptotische Laufzeit und Landau-Notation

**Definition 166.** Seien  $f$  und  $g$  Funktionen von  $\mathbb{N}$  nach  $\mathbb{R}^+$ . Wir schreiben  $f(n) = O(g(n))$ , falls es Zahlen  $n_0$  und  $c$  gibt mit

$$\forall n \geq n_0 : f(n) \leq c \cdot g(n).$$

Die Bedeutung der Aussage  $f(n) = O(g(n))$  ist, dass  $f$  „nicht wesentlich schneller“ als  $g$  wächst. Formal bezeichnet der Term  $O(g(n))$  die

Klasse aller Funktionen  $f$ , die obige Bedingung erfüllen. Die Gleichung  $f(n) = O(g(n))$  drückt also in Wahrheit eine Element-Beziehung  $f \in O(g(n))$  aus.  $O$ -Terme können auch auf der linken Seite vorkommen. In diesem Fall wird eine Inklusionsbeziehung ausgedrückt. So steht  $n^2 + O(n) = O(n^2)$  für die Aussage  $\{n^2 + f \mid f \in O(n)\} \subseteq O(n^2)$ .

### Beispiel 167.

- $7 \log(n) + n^3 = O(n^3)$  ist *richtig*.
- $7 \log(n)n^3 = O(n^3)$  ist *falsch*.
- $2^{n+O(1)} = O(2^n)$  ist *richtig*.
- $2^{O(n)} = O(2^n)$  ist *falsch* (siehe Übungen). ◁

Unter Benutzung der  $O$ -Notation lassen sich die wichtigsten Komplexitätsklassen wie folgt definieren:  $L = \text{DSPACE}(O(\log n))$ ,  $\text{LINTIME} = \text{DTIME}(O(n))$ ,  $P = \text{DTIME}(n^{O(1)})$ ,  $E = \text{DTIME}(2^{O(n)})$  und  $\text{EXP} = \text{DTIME}(2^{n^{O(1)}})$  etc.

## 6.2 Das P-NP-Problem

Wie wir im letzten Kapitel gesehen haben (siehe Satz 130), sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden. Die Frage, wieviel Zeit eine DTM zur Simulation einer NTM benötigt, ist eines der wichtigsten offenen Probleme der Informatik. Wegen  $\text{NTIME}(t) \subseteq \text{DTIME}(2^{O(t)})$  erhöht sich die Laufzeit im schlimmsten Fall exponentiell. Insbesondere die Klasse NP enthält viele für die Praxis überaus wichtige Probleme, für die kein Polynomialzeitalgorithmus bekannt ist. Da jedoch nur Probleme in P als effizient lösbar gelten, hat das so genannte **P-NP-Problem**, also die Frage, ob alle NP-Probleme effizient lösbar sind, eine immense praktische Bedeutung.

**Definition 168.** Seien  $A \subseteq \Sigma^*$  und  $B \subseteq \Gamma^*$  Sprachen.

- a)  $A$  ist auf  $B$  **in Polynomialzeit reduzierbar** ( $A \leq^p B$ ), falls eine Funktion  $f: \Sigma^* \rightarrow \Gamma^*$  in FP existiert mit

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

- b)  $A$  heißt  **$\leq^p$ -hart** für eine Sprachklasse  $\mathcal{C}$  (kurz: **C-hart** oder **C-schwer**), falls gilt:

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

- c) Eine C-harte Sprache, die zu  $\mathcal{C}$  gehört, heißt **C-vollständig**. Die Klasse aller NP-vollständigen Sprachen bezeichnen wir mit **NPC**.

Aus  $A \leq^p B$  folgt offenbar  $A \leq B$  und wie die Relation  $\leq$  ist auch  $\leq^p$  reflexiv und transitiv (s. Übungen). In diesem Kapitel verlangen wir also von einer  $\mathcal{C}$ -vollständigen Sprache  $A$ , dass jede Sprache  $L \in \mathcal{C}$  auf  $A$  in Polynomialzeit reduzierbar ist. Es ist leicht zu sehen, dass alle im letzten Kapitel als RE-vollständig nachgewiesenen Sprachen (wie z.B. K, H,  $H_0$ , PCP etc.) sogar  $\leq^p$ -vollständig für RE sind.

**Satz 169.** Die Klassen P und NP sind unter  $\leq^p$  abgeschlossen.

*Beweis.* Sei  $B \in P$  und gelte  $A \leq^p B$  mittels einer Funktion  $f \in \text{FP}$ . 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

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

ist  $M'$  polynomiell zeitbeschränkt und somit  $A$  in P. Den Abschluss von NP unter  $\leq^p$  zeigt man vollkommen analog. ■

**Satz 170.**

- (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  $B \in \text{NP}$   $\Rightarrow B \in \text{NPC}$ .
- (iii)  $\text{NPC} \cap \text{P} \neq \emptyset \Rightarrow \text{P} = \text{NP}$ .

*Beweis.*

- (i) Sei  $L \in \text{NP}$  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  $A \in \text{P}$  eine NP-vollständige Sprache und sei  $L \in \text{NP}$  beliebig. Dann folgt  $L \leq^p A$  und da P unter  $\leq^p$  abgeschlossen ist, folgt  $L \in \text{P}$ . ■

Eine einfache Möglichkeit, ein vollständiges Problem für NP und andere Komplexitätsklassen zu erhalten, besteht darin, als Eingabe beliebige TMs  $M$  zuzulassen, und zu fragen, ob  $M$  ein gegebenes Wort in einer vorgegebenen Zeit- oder Platzschränke akzeptiert.

Um die Komplexität einer solchen Sprache nach oben abschätzen zu können, müssen wir eine Schranke für die Laufzeit einer universellen TM herleiten.

**Lemma 171.** *Es gibt eine NTM  $U$  für die Sprache*

$$\left\{ w\#x \left| \begin{array}{l} w \in \{0,1\}^*, x \in \{0,1,\#\}^* \text{ und } M_w \\ \text{ist eine NTM, die } x \text{ akzeptiert} \end{array} \right. \right\}$$

*mit der Laufzeit*

$$\text{time}_U(w\#x) = \mathcal{O}(|w| \cdot \text{time}_{M_w}(x)^2).$$

*Beweis.* Betrachte die 3-NTM  $U$  die bei Eingabe  $w\#x$  zunächst den Binärstring  $w$  auf das 2. Band verschiebt und anschließend die Binärkodierung des Startzustands von  $M_w$  auf das 3. Band schreibt. Sodann

ersetzt  $U$  jedes Zeichen  $a_i$  des auf dem Eingabeband verbliebenen Strings  $x$  durch  $\text{bin}(i)$  und trennt diese durch  $\#$ . Hierbei benutzt  $U$  Binärzahlen einer festen Länge, die ausreicht, alle Arbeitszeichen von  $M_w$  darstellen zu können. Schließlich ergänzt  $U$  das 1. Band um  $k-1$  Bandinschriften (getrennt durch  $\#\#$ ), die jeweils nur das Zeichen  $\sqcup$  enthalten. Die aktuellen Kopfpositionen speichert  $U$  durch Markierung des vorangehenden  $\#$ -Zeichens.

Nun simuliert  $U$  jeden einzelnen Rechenschritt von  $M_w$  in jeweils  $\mathcal{O}(|w| \cdot \text{time}_{M_w}(x))$  Schritten, was auf die gewünschte Gesamtlaufzeit führt. ■

**Satz 172.** *Folgende Sprache ist NP-vollständig:*

$$L = \left\{ w\#x\#0^m \left| \begin{array}{l} w, x \in \{0,1\}^* \text{ und } M_w \text{ ist eine NTM,} \\ \text{die } x \text{ in } \leq m \text{ Schritten akzeptiert} \end{array} \right. \right\}$$

*Beweis.* Wir modifizieren die NTM  $U$  im Beweis von Lemma 171 wie folgt zu einer NTM  $U'$  mit  $L(U') = L$ .  $U'$  simuliert bei Eingabe  $w\#x\#0^m$  die NTM  $M_w(x)$  und zählt dabei die simulierten Schritte.  $U'$  verwirft, falls der Zähler den Wert  $m$  überschreitet und akzeptiert nur dann, wenn  $M_w(x)$  nach höchstens  $m$  Schritten akzeptiert. Da  $U'$  hierzu  $\mathcal{O}(|w|(|x|+m)^2)$  Zeit benötigt, ist entscheidet  $U'$  die Sprache  $L$  in Polynomialzeit.

Sei nun  $A$  eine beliebige NP-Sprache. Dann ist  $A$  in Polynomialzeit auf eine Sprache  $B \subseteq \{0,1\}^*$  in NP reduzierbar (siehe Übungen). Sei  $M_w$  eine durch ein Polynom  $p$  zeitbeschränkte NTM für  $B$ . Dann reduziert folgende FP-Funktion  $f$  die Sprache  $B$  auf  $L$ :

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

■



### 6.3 Platzkomplexität

Als nächstes definieren wir den Platzverbrauch von NTMs. Intuitiv ist dies die Anzahl aller während einer Rechnung benutzten Bandfelder. Wollen wir auch sublinearen Platz sinnvoll definieren, so dürfen wir hierbei das Eingabeband offensichtlich nicht berücksichtigen. Um sicherzustellen, dass eine NTM  $M$  das Eingabeband nicht als Speicher benutzt, verlangen wir, dass  $M$  die Felder auf dem Eingabeband nicht verändert und sich nicht mehr als ein Feld von der Eingabe entfernt.

**Definition 173.** Eine NTM  $M$  heißt **Offline-NTM** (oder NTM mit **Eingabeband**), falls für jede von  $M$  bei Eingabe  $x$  erreichbare Konfiguration

$$K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$$

gilt, dass  $u_1 a_1 v_1$  ein Teilwort von  $\sqcup x \sqcup$  ist.

**Definition 174.** Der **Platzverbrauch** einer Offline-NTM  $M$  bei Eingabe  $x$  ist definiert als

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

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

$$space_M(x) \leq s(|x|).$$

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

**Definition 175.** Die auf deterministischem Platz  $s(n)$  entscheidbaren Sprachen bilden die Klasse

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

Die auf nichtdeterministischem Platz  $s(n)$  entscheidbaren Sprachen bilden die Klasse

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

Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$$\mathbf{L} = \mathbf{DSPACE}(O(\log n)) \quad \text{„Logarithmischer Platz“}$$

$$\mathbf{Linspace} = \mathbf{DSPACE}(O(n)) \quad \text{„Linearer Platz“}$$

$$\mathbf{PSPACE} = \mathbf{DSPACE}(n^{O(1)}) \quad \text{„Polynomieller Platz“}$$

Die nichtdeterministischen Klassen **NL**, **NLinspace** und **NPSPACE** sind analog definiert.

Dass nichtdeterministische Berechnungen nicht sehr viel Platz gegenüber deterministischen Berechnungen einsparen können, wurde von Savitch 1970 bewiesen (siehe Vorlesung Komplexitätstheorie).

**Satz 176.** Jede  $s(n)$ -platzbeschränkte NTM kann von einer  $s^2(n)$ -platzbeschränkten DTM simuliert werden (ohne Beweis).

Als Konsequenz hiervon fallen die Klassen **NPSPACE** und **PSPACE** zusammen. Zwischen den verschiedenen Zeit- und Platzklassen bestehen die folgenden elementaren Inklusionsbeziehungen. Für jede Funktion  $s(n) \geq \log n$  gilt

$$\mathbf{DSPACE}(s) \subseteq \mathbf{NSPACE}(s) \subseteq \mathbf{DTIME}(2^{O(s)})$$

und für jede Funktion  $t(n) \geq n + 2$  gilt

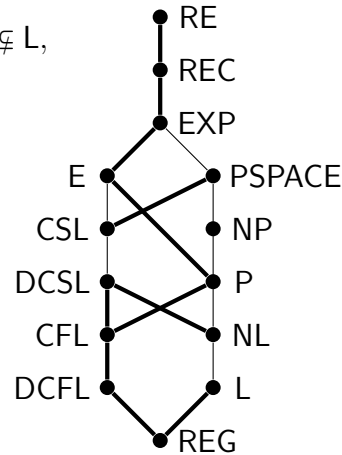
$$\mathbf{DTIME}(t) \subseteq \mathbf{NTIME}(t) \subseteq \mathbf{DSPACE}(t).$$

Eine unmittelbare Konsequenz hiervon sind folgende Inklusionen:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP} \subseteq \mathbf{EXSPACE}.$$

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

$$\begin{aligned}
 \text{REG} &= \text{DSPACE}(O(1)) = \text{NSPACE}(O(1)) \not\subseteq L, \\
 \text{DCFL} &\not\subseteq \text{LINTIME} \cap \text{CFL} \not\subseteq \text{LINTIME} \not\subseteq P, \\
 \text{CFL} &\not\subseteq \text{NLINTIME} \cap \text{DTIME}(n^3) \not\subseteq P, \\
 \text{DCSL} &= \text{Linspace} \subseteq \text{CSL}, \\
 \text{CSL} &= \text{NLinspace} \subseteq \text{PSPACE} \cap E, \\
 \text{REC} &= \bigcup \text{DSPACE}(f(n)) \\
 &= \bigcup \text{NSPACE}(f(n)) \\
 &= \bigcup \text{DTIME}(f(n)) \\
 &= \bigcup \text{NTIME}(f(n)),
 \end{aligned}$$



wobei  $f$  alle berechenbaren (oder äquivalent: alle) Funktionen  $f : \mathbb{N} \rightarrow \mathbb{N}$  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  $\text{DCSL} \subseteq P$  gilt. Man kann jedoch zeigen, dass  $\text{CSL} \neq P \neq \text{DCSL}$  ist. Ähnlich verhält es sich mit den Klassen E und PSPACE: Man kann zwar zeigen, dass sie verschieden sind, aber ob eine in der anderen enthalten ist, ist nicht bekannt.

## 7 NP-vollständige Probleme

### 7.1 Aussagenlogische Erfüllbarkeitsprobleme

**Definition 177.**

- Die Menge der **booleschen** (oder **aussagenlogischen**) **Formeln** über den Variablen  $x_1, \dots, x_n$  ist induktiv wie folgt definiert:
  - Jede Variable  $x_i$  ist eine boolesche Formel.
  - Mit  $G$  und  $H$  sind auch die **Negation**  $\neg G$  von  $G$  und die **Konjunktion**  $(G \wedge H)$  von  $G$  und  $H$  boolesche Formeln.
- Eine **Belegung** von  $x_1, \dots, x_n$  ist ein Wort  $a = a_1 \dots a_n \in \{0, 1\}^n$ . Der **Wert**  $F(a)$  von  $F$  unter  $a$  ist induktiv über den Aufbau von  $F$  definiert:

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

- Durch eine boolesche Formel  $F$  wird also eine  $n$ -stellige **boolesche Funktion**  $F : \{0, 1\}^n \rightarrow \{0, 1\}$  definiert, die wir ebenfalls mit  $F$  bezeichnen.
- $F$  heißt **erfüllbar**, falls es eine Belegung  $a$  mit  $F(a) = 1$  gibt.
- Gilt dagegen für alle Belegungen  $a$ , dass  $F(a) = 1$  ist, so heißt  $F$  **Tautologie**.

**Beispiel 178** (Erfüllbarkeitstest mittels Wahrheitstabelle).  
Da die Formel  $F = ((x_1 \vee x_2) \rightarrow (\neg x_2 \wedge x_3))$  für die Belegungen

$a \in \{000, 001, 101\}$  den Wert  $F(a) = 1$  annimmt, ist sie erfüllbar:

$a$	$(x_1 \vee x_2)$	$(\neg x_2 \wedge x_3)$	$((x_1 \vee x_2) \rightarrow (\neg x_2 \wedge x_3))$
000	0	0	1
001	0	1	1
010	1	0	0
011	1	0	0
100	1	0	0
101	1	1	1
110	1	0	0
111	1	0	0

◁

**Notation.** Wir verwenden die **Disjunktion**  $(G \vee H)$  und die **Implikation**  $(G \rightarrow H)$  als Abkürzungen für die Formeln  $\neg(\neg G \wedge \neg H)$  bzw.  $(\neg G \vee H)$ .

Um Klammern zu sparen, vereinbaren wir folgende Präzedenzregeln:

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

**Beispiel 179.** Die Formel

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

nimmt unter einer Belegung  $a = a_1 \dots a_n$  genau dann den Wert 1 an, wenn  $\sum_{i=1}^n a_i = 1$  ist. D.h. es gilt genau dann  $G(a) = 1$ , wenn genau eine Variable  $x_i$  mit dem Wert  $a_i = 1$  belegt ist. Diese Formel wird im Beweis des nächsten Satzes benötigt. ◁

Bei vielen praktischen Anwendungen ist es erforderlich, eine erfüllende Belegung für eine vorliegende boolesche Formel zu finden (sofern es eine gibt). Die Bestimmung der Komplexität des Erfüllbarkeitsproblems

(engl. *satisfiability*) für boolesche Formeln hat also große praktische Bedeutung.

**Aussagenlogisches Erfüllbarkeitsproblem (SAT):**

**Gegeben:** Eine boolesche Formel  $F$  in den Variablen  $x_1, \dots, x_n$ .

**Gefragt:** Ist  $F$  erfüllbar?

Dabei kodieren wir boolesche Formeln  $F$  durch Binärstrings  $w_F$  und ordnen umgekehrt jedem Binärstring  $w$  eine Formel  $F_w$  zu. Um die Notation zu vereinfachen, werden wir jedoch meist  $F$  anstelle von  $w_F$  schreiben.

**Satz 180** (Cook, Karp, Levin). SAT ist NP-vollständig.

*Beweis.* Es ist leicht zu sehen, dass  $\text{SAT} \in \text{NP}$  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  $w = w_1 \dots w_n$  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  $Z = \{q_0, \dots, q_m\}$ ,  $E = \{q_m\}$  und  $\Gamma = \{a_1, \dots, a_l\}$  ist. Zudem können wir annehmen, dass  $\delta$  für jedes Zeichen  $a \in \Gamma$  die Anweisung  $q_m a \rightarrow q_m a N$  enthält.

Die Idee besteht nun darin, die Formel  $F_w$  so zu konstruieren, dass sie unter einer Belegung  $a$  genau dann wahr wird, wenn  $a$  eine akzeptierende Rechnung von  $M(w)$  beschreibt. Hierzu bilden wir  $F_w$  über

den Variablen

- $x_{t,q}$ , für  $0 \leq t \leq p(n), q \in Z$ ,
- $y_{t,i}$ , für  $0 \leq t \leq p(n), -p(n) \leq i \leq p(n)$ ,
- $z_{t,i,a}$ , für  $0 \leq t \leq p(n), -p(n) \leq i \leq p(n), a \in \Gamma$ ,

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.

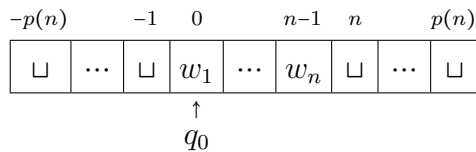
Konkret sei nun  $F_w = R \wedge S \wedge \dot{U}_1 \wedge \dot{U}_2 \wedge E$ . Dabei stellt die Formel  $R = \bigwedge_{t=0}^{p(n)} R_t$  (Randbedingungen) sicher, dass wir jeder erfüllenden Belegung eindeutig eine Folge von Konfigurationen  $K_0, \dots, K_{p(n)}$  zuordnen können:

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

Die Teilformel  $R_t$  sorgt also dafür, dass zum Zeitpunkt  $t$

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

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



vorliegt:

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

Die Formel  $\dot{U}_1$  sorgt dafür, dass der Inhalt von nicht besuchten Feldern beim Übergang von  $K_t$  zu  $K_{t+1}$  unverändert bleibt:

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

Die Formel  $\dot{U}_2$  achtet darauf, dass sich bei jedem Übergang der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in  $\delta$  verändern:

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

wobei

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

ist. Schließlich überprüft  $E$ , ob  $M$  zur Zeit  $p(n)$  den Endzustand  $q_m$  erreicht hat:

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

Da der Aufbau der Formel  $f(w) = F_w$  einem einfachen Bildungsgesetz folgt und ihre Länge polynomiell in  $n$  ist, folgt  $f \in \text{FP}$ . 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  $K_0, \dots, K_{p(n)}$ , so dass gilt:

- $K_0$  ist Startkonfiguration von  $M(w)$  (wegen  $S(a) = 1$ ),

- $K_i \vdash K_{i+1}$  für  $i = 0, \dots, p(n) - 1$  (wegen  $\dot{U}_1(a) = \dot{U}_2(a) = 1$ ),
- $M$  nimmt spätestens in der Konfiguration  $K_{p(n)}$  den Endzustand  $q_m$  an (wegen  $E(a) = 1$ ).

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

**Korollar 181.**  $\text{SAT} \in \text{P} \Leftrightarrow \text{P} = \text{NP}$ .

Gelingt es also, einen Polynomialzeit-Algorithmus für SAT zu finden, so lässt sich daraus leicht ein effizienter Algorithmus für jedes NP-Problem ableiten. Als nächstes betrachten wir das Erfüllbarkeitsproblem für boolesche Schaltkreise.

**Definition 182.**

- a) Ein **boolescher Schaltkreis** über den Variablen  $x_1, \dots, x_n$  ist eine Folge  $s = (g_1, \dots, g_m)$  von **Gattern**

$$g_l \in \{0, 1, x_1, \dots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$$

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

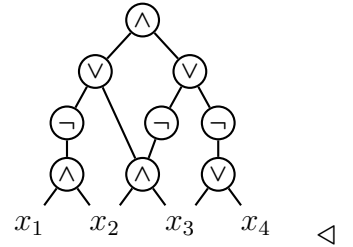
- b) Die am Gatter  $g_l$  berechnete  $n$ -stellige boolesche Funktion ist induktiv wie folgt definiert:

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

- 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 183.** Der Schaltkreis

$$s = (x_1, x_2, x_3, x_4, (\wedge, 1, 2), (\wedge, 2, 3), (\vee, 3, 4), (\neg, 5), (\neg, 6), (\neg, 7), (\vee, 6, 8), (\vee, 9, 10), (\wedge, 11, 12))$$



ist nebenstehend graphisch dargestellt.

**Bemerkung 184.**

- Die Anzahl der Eingänge eines Gatters  $g$  wird als **Fanin** von  $g$  bezeichnet, die Anzahl der Ausgänge von  $g$  (d.h. die Anzahl der Gatter, die  $g$  als Eingabe benutzen) als **Fanout**.
- Boolesche Formeln entsprechen also booleschen Schaltkreisen mit Fanout  $\leq 1$  und umgekehrt.

**Erfüllbarkeitsproblem für boolesche Schaltkreise (CIRSAT):**

**Gegeben:** Ein boolescher Schaltkreis  $s$ .

**Gefragt:** Ist  $s$  erfüllbar?

Da eine boolesche Formel  $F$  leicht in einen äquivalenten Schaltkreis  $s$  mit  $s(a) = F(a)$  für alle Belegungen  $a$  transformiert werden kann, folgt  $\text{SAT} \leq^p \text{CIRSAT}$ .

**Korollar 185.** CIRSAT ist NP-vollständig.

**Bemerkung 186.** Da SAT NP-vollständig ist, ist CIRSAT in Polynomialzeit auf SAT reduzierbar. Dies bedeutet 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 187.**

- 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. Hierbei ist auch  $k = 0$  zulässig, d.h. die **leere Klausel** repräsentiert die Konstante 0 und wird üblicherweise mit  $\square$  bezeichnet.
- c) Eine Formel  $F$  ist in **konjunktiver Normalform** (kurz **KNF**), falls  $F$  eine Konjunktion

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

von  $m \geq 0$  Klauseln ist. Im Fall  $m = 0$  repräsentiert  $F$  die Konstante 1.

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

**Notation.** Klauseln werden oft als Menge  $C = \{l_1, \dots, l_k\}$  ihrer Literale und KNF-Formeln als Menge  $F = \{C_1, \dots, C_m\}$  ihrer Klauseln dargestellt. Enthält  $F$  die leere Klausel, so ist  $F$  unerfüllbar, wogegen die leere KNF-Formel immer wahr (also eine Tautologie) ist.

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

**Gegeben:** Eine boolesche Formel  $F$  in  $k$ -KNF.

**Gefragt:** Ist  $F$  erfüllbar?