Vorlesungsskript

Einführung in die Theoretische Informatik

Wintersemester 2016/17

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

21. Februar 2017

Inhaltsverzeichnis

Inhaltsverzeichnis

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* Σ kodiert.

Definition 1.

a) Ein ***Alphabet*** $\Sigma = \{a_1, \ldots, a_m\}$ *ist eine geordnete Menge von endlich vielen* ***Zeichen****.*

b) Eine Folge $x = x_1 \ldots x_n$ *von* n *Zeichen heißt* ***Wort*** *(der* ***Länge*** $|x| = n$*).*

c) Die Menge aller Wörter über Σ *ist*

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

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

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

e) Jede Teilmenge $L \subseteq \Sigma^*$ *heißt* ***Sprache*** *über dem Alphabet* Σ*.*

Beispiel 2. *Sei* Σ *ein Alphabet. Dann sind* $\varnothing, \Sigma^*, \Sigma$ *und* $\{\varepsilon\}$ *Sprachen über* Σ*. Die Sprache* $\varnothing$ *enthält keine Wörter und heißt* ***leere Sprache****. Die Sprache* Σ^* *enthält dagegen alle Wörter über* Σ*, während die Sprache* Σ *alle Wörter über* Σ *der Länge 1 enthält. Die Sprache*

$\{\varepsilon\}$ enthält nur das leere Wort, ist also einelementig. Einelementige Sprachen werden auch als ***Singletonsprachen*** *bezeichnet.*

Da Sprachen Mengen sind, können wir sie bzgl. Inklusion vergleichen. Zum Beispiel gilt

$$\varnothing \subseteq \{\varepsilon\} \subseteq \Sigma^*.$$

Wir können Sprachen auch vereinigen, schneiden und komplementieren. Seien A und B Sprachen über Σ. Dann ist

- $A \cap B = \{x \in \Sigma^* \mid x \in A, x \in B\}$ der ***Schnitt*** von A und B,
- $A \cup B = \{x \in \Sigma^* \mid x \in A \vee x \in B\}$ die ***Vereinigung*** von A und B, und
- $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$ das ***Komplement*** von A.

Neben den Mengenoperationen gibt es auch spezielle Sprachoperationen.

Definition 3.

- *Das* ***Produkt*** *(****Verkettung****,* ***Konkatenation****) der Sprachen A und B ist*

  $$AB = \{xy \mid x \in A, y \in B\}.$$

  Ist $A = \{x\}$ eine Singletonsprache, so schreiben wir für $\{x\}B$ auch einfach xB.
- *Die* ***n-fache Potenz A^n*** *einer Sprache A ist induktiv definiert durch*

  $$A^n = \begin{cases} \{\varepsilon\}, & n = 0, \\ A^{n-1}A, & n > 0. \end{cases}$$

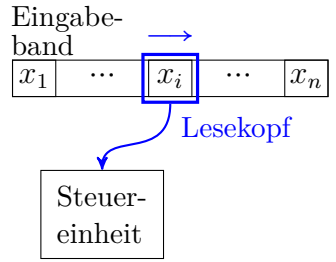
- *Die* ***Sternhülle A^**** *von A ist $A^* = \bigcup_{n \geq 0} A^n$.*
- *Die* ***Plushülle A^+*** *von A ist $A^+ = \bigcup_{n \geq 1} A^n = AA^*$.*

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

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

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

$$L(M) = \left\{ x_1 \dots x_n \in \Sigma^* \;\middle|\; \begin{array}{l} \text{es gibt } q_1, \dots, q_{n-1} \in Z, q_n \in E \text{ mit} \\ \delta(q_i, x_{i+1}) = q_{i+1} \text{ für } i = 0, \dots, n-1 \end{array} \right\}.$$

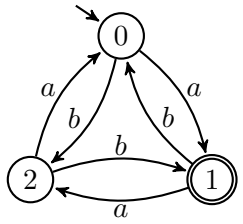
Eine Zustandsfolge $q_0, q_1, \ldots, q_n$ heißt ***Rechnung*** *von $M(x_1 \ldots x_n)$, falls $\delta(q_i, x_{i+1}) = q_{i+1}$ für $i = 0, \ldots, n-1$ gilt. Sie heißt* ***akzeptierend****, falls $q_n \in E$ ist, und andernfalls* verwerfend. *Eine von einem DFA akzeptierte Sprache wird als* ***regulär*** *bezeichnet. Die zugehörige Sprachklasse ist*

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

Beispiel 5. *Betrachte den DFA $M = (Z, \Sigma, \delta, 0, E)$ mit $Z = \{0,1,2\}$, $\Sigma = \{a,b\}$, $E = \{1\}$ und der Überführungsfunktion*

δ	0	1	2
a	1	2	0
b	2	0	1

Graphische Darstellung:



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

Bei Eingabe $w_1 = aba$ führt M die akzeptierende Rechnung $0,1,0,1$ durch, d.h. $w_1 \in L(M)$. Dagegen verwirft M das Wort $w_2 = abba$ (verwerfende Rechnung: $0,1,0,2,0$). ◁

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

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

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

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

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

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

schreiben.

Behauptung 6. *Der DFA M aus Beispiel 5 akzeptiert die Sprache*

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

wobei $\#_a(x)$ die Anzahl der Vorkommen des Zeichens a in x bezeichnet und $i \equiv_m j$ (in Worten: i ist kongruent zu j modulo m) bedeutet, dass $i - j$ durch m teilbar ist.

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 $x = x_1 \ldots x_{n+1}$ gegeben und sei $i = \hat{\delta}(0, x_1 \ldots x_n)$. Nach IV gilt dann

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

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

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

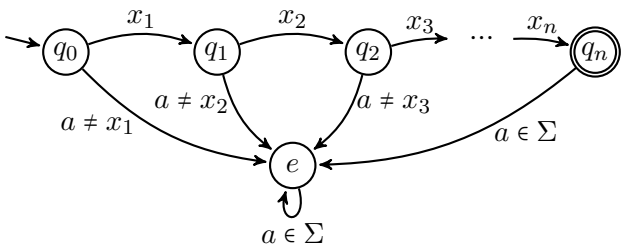
und somit

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

■

Beobachtung 7. *Alle Singletonsprachen sind regulär.*

Beweis. Für jedes Wort $x = x_1 \dots x_n$ existiert ein DFA M_x mit $L(M_x) = \{x\}$:



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 Überführungsfunktion

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

■

Als nächstes betrachten wir Abschlusseigenschaften der Sprachklasse REG.

Definition 8. *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 9. *Der Schnittoperator $\cap$ bildet zwei Sprachen L_1 und L_2 auf die Sprache $L_1 \cap L_2$ ab.* ◁

Definition 10. *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 11. *Der Abschluss der Singletonsprachen unter $\cap$ besteht aus allen Singletonsprachen und der leeren Sprache.*

Der Abschluss der Singletonsprachen unter $\cup$ besteht aus allen nichtleeren endlichen Sprachen. ◁

Definition 12. *Für eine Sprachklasse $\mathcal{C}$ bezeichne co-$\mathcal{C}$ die Klasse $\{\bar{L} \mid L \in \mathcal{C}\}$ aller Komplemente von Sprachen in $\mathcal{C}$.*

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

Beobachtung 13. *Mit $L_1, L_2 \in$ REG sind auch die Sprachen $\overline{L_1} = \Sigma^* \setminus L_1$, $L_1 \cap L_2$ und $L_1 \cup L_2$ regulär.*

Beweis. Sind $M_i = (Z_i, \Sigma, \delta_i, q_0, E_i)$, $i = 1, 2$, DFAs mit $L(M_i) = L_i$, so akzeptiert der DFA

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

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

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

mit

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

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

Aus Beobachtung 13 folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind. Da die in Beispiel 5 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 Produkt oder Sternhülle abgeschlossen ist. Im übernächsten Abschnitt werden wir sehen, dass die Klasse REG als der Abschluss der endlichen Sprachen unter Vereinigung, Produkt und Sternhülle charakterisierbar ist.

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

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

2.2 Nichtdeterministische endliche Automaten

Definition 14. *Ein **nichtdeterministischer endlicher Automat** (kurz: NFA; nondeterministic finite automaton) $N = (Z, \Sigma, \Delta, Q_0, E)$ ist ähnlich aufgebaut wie ein DFA, nur dass er mehrere Startzustände (zusammengefasst in der Menge $Q_0 \subseteq Z$) haben kann und seine Überführungsfunktion die Form*

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

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

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

*Eine Zustandsfolge $q_0, q_1, \dots, q_n$ heißt **Rechnung** von $N(x_1 \dots x_n)$, falls $q_{i+1} \in \Delta(q_i, x_{i+1})$ für $i = 0, \dots, n-1$ gilt.*

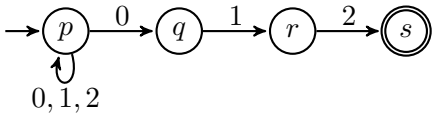
Ein NFA N kann bei einer Eingabe x also nicht nur eine, sondern mehrere verschiedene Rechnungen parallel ausführen. Ein Wort x gehört genau dann zu $L(N)$, wenn $N(x)$ mindestens eine akzeptierende Rechnung hat.

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

Beispiel 15. *Betrachte den NFA $N = (Z, \Sigma, \Delta, Q_0, E)$ mit Zustandsmenge $Z = \{p, q, r, s\}$, Eingabealphabet $\Sigma = \{0, 1, 2\}$, Start- und Endzustandsmenge $Q_0 = \{p\}$ und $E = \{s\}$ sowie der Überführungsfunktion*

Δ	p	q	r	s
0	$\{p, q\}$	$\varnothing$	$\varnothing$	$\varnothing$
1	$\{p\}$	$\{r\}$	$\varnothing$	$\varnothing$
2	$\{p\}$	$\varnothing$	$\{s\}$	$\varnothing$

Graphische Darstellung:



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

Beobachtung 16. *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.*

Beweis. Sei $L_i = L(N_i)$. Wir können $Z_1 \cap Z_2 = \varnothing$ 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

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

die Sprache $L_1 L_2$.

$L_1 L_2 \subseteq L(N)$*:* Seien $x = x_1 \cdots x_k \in L_1, y = y_1 \cdots y_l \in L_2$ und seien $q_0, \ldots, q_k$ und $p_0, \ldots, p_l$ akzeptierende Rechnungen von $N_1(x)$ und $N_2(y)$. Dann ist $q_0, \ldots, q_k, p_1, \ldots, p_l$ eine akz. Rechnung von $N(xy)$, da $q_0 \in Q_1$ und $p_l \in E_2$ ist, und

- im Fall $l \geq 1$ wegen $q_k \in E_1$, $p_0 \in Q_2$ und $p_1 \in \Delta_2(p_0, y_1)$ zudem $p_1 \in \Delta(q_k, y_1)$ und
- im Fall $l = 0$ wegen $q_k \in E_1$ und $p_l \in Q_2 \cap E_2$ zudem $q_k \in E$ ist.

$L(N) \subseteq L_1 L_2$*:* Sei $x = x_1 \cdots x_n \in L(N)$ und sei $q_0, \ldots, q_n$ eine akz. Rechnung von $N(x)$. Dann gilt $q_0 \in Q_1$, $q_n \in E$, $q_0, \ldots, q_i \in Z_1$ und $q_{i+1}, \ldots, q_n \in Z_2$ für ein $i \leq n$. Wir zeigen, dass ein $q \in Q_2$ existiert, so dass $q_0, \ldots, q_i$ eine akz. Rechnung von $N_1(x_1 \cdots x_i)$ und $q, q_{i+1}, \ldots, q_n$ eine akz. Rechnung von $N_2(x_{i+1} \cdots x_n)$ ist.

- Im Fall $i < n$ impliziert der Übergang $q_{i+1} \in \Delta(q_i, x_{i+1})$, dass $q_i \in E_1$ und $q_{i+1} \in \Delta_2(q, x_{i+1})$ für ein $q \in Q_2$ ist.
- Im Fall $i = n$ ist $q_n \in E_1$ und $Q_2 \cap E_2 \neq \varnothing$ (d.h. $\varepsilon \in L_2$).

Ganz ähnlich lässt sich zeigen, dass der NFA

$$N^* = (Z_1 \cup \{q_{neu}\}, \Sigma, \Delta_4, Q_1 \cup \{q_{neu}\}, E_1 \cup \{q_{neu}\})$$

mit

$$\Delta_4(p,a) = \begin{cases} \Delta_1(p,a), & p \in Z_1 \setminus E_1, \\ \Delta_1(p,a) \cup \bigcup_{q \in Q_1} \Delta_1(q,a), & p \in E_1, \\ \varnothing, & \text{sonst} \end{cases}$$

die Sprache L_1^* akzeptiert. ∎

Satz 17 (Rabin und Scott)**.**
$\mathsf{REG} = \{L(N) \mid N \textit{ ist ein NFA}\}$.

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

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

Die Menge $\delta(Q,a)$ enthält also alle Zustände, in die N gelangen kann, wenn N ausgehend von einem beliebigen Zustand $q \in Q$ das Zeichen a liest. Intuitiv bedeutet dies, dass der DFA M den NFA N simuliert, indem M in seinem aktuellen Zustand Q die Information speichert, in welchen Zuständen sich N momentan befinden könnte. Für die Erweiterung $\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ von δ (siehe Seite 3) können wir nun folgende Behauptung zeigen.

Behauptung. $\hat{\delta}(Q_0, x)$ enthält alle Zustände, die N ausgehend von einem Startzustand nach Lesen von 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 $x = x_1 \ldots x_n$ gegeben. Nach Induktionsvoraussetzung enthält

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

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.

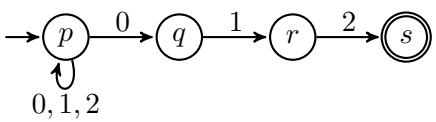
$$E' = \{Q \subseteq Z \mid Q \cap E \neq \varnothing\},$$

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

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

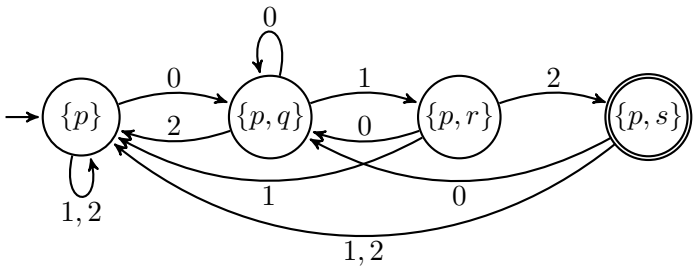
■

Beispiel 18. *Für den NFA $N = (Z, \Sigma, \Delta, Q_0, E)$ aus Beispiel 15*



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

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



◁

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

Korollar 19. *Die Klasse* REG *der regulären Sprachen ist unter folgenden Operationen abgeschlossen:*

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

2.3 Reguläre Ausdrücke

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

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

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

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

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

Definition 20. *Die Menge der* ***regulären Ausdrücke*** γ *(über einem Alphabet* Σ*) und die durch* γ *dargestellte Sprache* $L(\gamma)$ *sind induktiv wie folgt definiert. Die Symbole* $\varnothing$, ϵ *und* a $(a \in \Sigma)$ *sind reguläre Ausdrücke, die*

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

beschreiben. Sind α *und* β *reguläre Ausdrücke, die die Sprachen* $L(\alpha)$ *und* $L(\beta)$ *beschreiben, so sind auch* $\alpha\beta$, $(\alpha|\beta)$ *und* $(\alpha)^*$ *reguläre Ausdrücke, die die Sprachen*

- $L(\alpha\beta) = L(\alpha)L(\beta)$,
- $L(\alpha|\beta) = L(\alpha) \cup L(\beta)$ *und*
- $L((\alpha)^*) = L(\alpha)^*$

beschreiben.

Bemerkung 21.

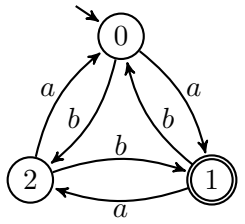
- *Um Klammern zu sparen, definieren wir folgende* ***Präzedenzordnung****: Der Sternoperator* * *bindet stärker als der Produktoperator und dieser wiederum stärker als der Vereinigungsoperator. Für* $((a|b(c)^*)|d)$ *können wir also kurz* $a|bc^*|d$ *schreiben.*
- *Da der reguläre Ausdruck* $\gamma\gamma^*$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

Beispiel 22. *Die regulären Ausdrücke* ϵ^*, $\varnothing^*$, $(0|1)^*00$ *und* $\epsilon 0|\varnothing 1^*$ *beschreiben folgende Sprachen:*

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

◁

Beispiel 23. *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 Sprache aller solchen Wörter* $w \in L_{0,0}$*, die den Zustand 0 nur zu Beginn und am Ende (aber nicht zwischendurch) besuchen. Dann setzt sich jedes* $x \in L_{0,0}$ *aus beliebig vielen Teilwörtern* $w_1, \ldots, w_k \in L_{0,0}^{\neq 0}$ *zusammen, d.h.* $L_{0,0} = (L_{0,0}^{\neq 0})^*$.

Jedes $w \neq \varepsilon$ *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*

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

beschreiben. Eine ähnliche Überlegung zeigt, dass sich die 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

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

◁

Satz 24. $\{L(\gamma) \mid \gamma$ *ist ein regulärer Ausdruck*$\} = \mathsf{REG}$.

Beweis. Die Inklusion von rechts nach links ist klar, da die Basisausdrücke $\varnothing$, ϵ und a, $a \in \Sigma^*$, nur reguläre Sprachen beschreiben und die Sprachklasse REG unter Produkt, Vereinigung und Sternhülle abgeschlossen ist (siehe Beobachtungen 13 und 16).

Für die Gegenrichtung konstruieren wir zu einem DFA M einen regulären Ausdruck γ mit $L(\gamma) = L(M)$. Sei also $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA, wobei wir annehmen können, dass $Z = \{1, \ldots, m\}$ und $q_0 = 1$ ist. Dann lässt sich $L(M)$ als Vereinigung

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

von Sprachen der Form

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

darstellen. Folglich reicht es zu zeigen, dass die Sprachen $L_{p,q}$ durch reguläre Ausdrücke beschreibbar sind. Hierzu betrachten wir die Sprachen

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

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

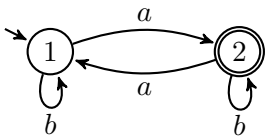
$$L_{p,q}^{r+1} = L_{p,q}^r \cup L_{p,r+1}^r (L_{r+1,r+1}^r)^* L_{r+1,q}^r$$

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:

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

■

Beispiel 25. *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 $\gamma^2_{1,2}$ für $L(M)$ zu erhalten, genügt es also, die regulären Ausdrücke $\gamma^0_{1,1}$, $\gamma^0_{1,2}$, $\gamma^0_{2,1}$, $\gamma^0_{2,2}$, $\gamma^1_{1,2}$ und $\gamma^1_{2,2}$ zu berechnen:

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

◁

Korollar 26. *Sei L eine Sprache. Dann sind folgende Aussagen äquivalent:*

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

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

2.4 Relationalstrukturen

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

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

Beispiel 27.

- (F, M) *mit* $F = \{f \mid f$ *ist Fluss in Europa*$\}$ *und*
$$M = \{(f, g) \in F \times F \mid f \text{ mündet in } g\}.$$
- (U, B) *mit* $U = \{x \mid x$ *ist Berliner*$\}$ *und*
$$B = \{(x, y) \in U \times U \mid x \text{ ist Bruder von } y\}.$$
- $(\mathsf{P}(M), \subseteq)$, *wobei* $\mathsf{P}(M)$ *die Potenzmenge einer beliebigen Menge M und $\subseteq$ die Inklusionsbeziehung auf den Teilmengen von M ist.*
- (A, Id_A), *wobei* $Id_A = \{(x, x) \mid x \in A\}$ *die* ***Identität auf A*** *ist.*
- $(\mathbb{R}, \leq)$.
- $(\mathbb{Z}, |)$, *wobei $|$ die "teilt"-Relation bezeichnet (d.h. $a|b$, falls ein $c \in \mathbb{Z}$ mit $b = ac$ existiert).* ◁

Da Relationen Mengen sind, sind auf ihnen die mengentheoretischen Operationen ***Schnitt***, ***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:

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

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 28. *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.* ◁

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

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

Eigenschaften von Relationen

Sei R eine Relation auf A. Dann heißt R

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

gilt.

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 29.

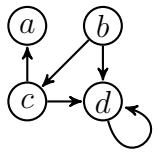
- *Die Relation "ist Schwester von" ist zwar in einer reinen Damengesellschaft symmetrisch, i.a. jedoch weder symmetrisch noch asymmetrisch noch antisymmetrisch.*

- *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* $(\mathsf{P}(M), \subseteq)$ *sind reflexiv, antisymmetrisch und transitiv und somit Ordnungen.*
- $(\mathbb{R}, \leq)$ *ist auch konnex und somit eine lineare Ordnung.*
- $(\mathsf{P}(M), \subseteq)$ *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 ***Knotenmenge*** $V = A$ und ***Kantenmenge*** $E = R$ veranschaulicht werden. Hierzu stellen wir jedes Element $x \in A$ als einen Knoten dar und verbinden jedes Knotenpaar $(x, y) \in R$ durch eine gerichtete Kante (Pfeil). Zwei durch eine Kante verbundene Knoten heißen ***benachbart*** oder ***adjazent***.

Beispiel 30. *Für die Relation* (A, R) *mit* $A = \{a, b, c, d\}$ *und* $R = \{(b, c), (b, d), (c, a), (c, d), (d, d)\}$ *erhalten wir folgende graphische Darstellung.*



◁

Der ***Ausgangsgrad*** eines Knotens $x \in V$ ist $\deg^+(x) = \|R[x]\|$, wobei $R[x] = \{y \in V \mid xRy\}$ die Menge der ***Nachfolger*** von x ist. Entsprechend ist $\deg^-(x) = \|\{y \in V \mid yRx\}\|$ 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 $d(x) = \deg^-(x) = \deg^+(x)$ 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***. Eine irreflexive und symmetrische Relation R wird meist als Menge der ungeordneten Paare $E = \big\{\{a, b\} \mid aRb\big\}$ notiert.

Darstellung durch Adjazenzmatrizen

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

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

darstellen. Beispielsweise hat die Relation

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

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

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

Darstellung durch Adjazenzlisten

Eine weitere Möglichkeit besteht darin, eine endliche Relation R in Form einer Tabelle darzustellen, die jedem Element $x \in A$ seine Nachfolger in Form einer Liste zuordnet. Für obige Relation R erhalten wir folgende Listen:

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

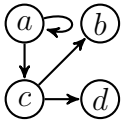
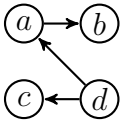
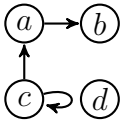
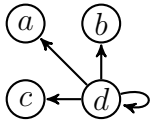
Sind $M_R = (r_{ij})$ und $M_S = (s_{ij})$ boolesche $n \times n$-Matrizen für R und S, so erhalten wir für $T = R \circ S$ die Matrix $M_T = (t_{ij})$ mit

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

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

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

◁

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

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 32. *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*

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

Satz 33. *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) $h_{äq}(R) = (R \cup R^T)^*$.

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^nb, d.h. es gibt Elemente $x_0, \ldots, x_n \in A$ mit $x_0 = a$, $x_n = b$ und

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

In der Graphentheorie nennt man $x_0, \ldots, x_n$ einen ***Weg*** der Länge n von a nach b. Ein Digraph G heißt ***zusammenhängend***, wenn es für je zwei Knoten a und b einen Weg von a nach b oder einen Weg von b nach a gibt. G heißt ***stark zusammenhängend***, wenn es von jedem Knoten a einen Weg zu jedem Knoten b in G gibt.

2.4.1 Ordnungs- und Äquivalenzrelationen

Wir betrachten zunächst 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 34.

- *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*

$$[r] = \{a \in \mathbb{Z} \mid a \equiv_m r\},\ r = 0, 1, \ldots, m-1.$$

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

Die (bzgl. Inklusion) 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. $[x]_{Id_A} = \{x\}$ für alle $x \in A$, und die Allrelation erzeugt nur eine Äquivalenzklasse, nämlich $[x]_{A \times A} = A$ für jedes $x \in A$. Die Identität Id_A hat nur ein Repräsentantensystem, nämlich A. Dagegen kann jede Singletonmenge $\{x\}$ mit $x \in A$ als Repräsentantensystem für die Allrelation $A \times A$ fungieren.

Definition 35. *Eine Familie $\{B_i \mid i \in I\}$ von nichtleeren Teilmengen $B_i \subseteq A$ heißt* ***Partition*** *der Menge A, falls gilt:*

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

Wie der nächste Satz zeigt, bilden die Äquivalenzklassen einer Äquivalenzrelation E eine Partition $\{[x] \mid x \in A\}$ von A. 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.

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.

Satz 36. *Sei E eine Relation auf A. Dann sind folgende Aussagen äquivalent.*

(i) *E ist eine Äquivalenzrelation auf A.*

(ii) *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. Wir zeigen, dass dann $\{E[x] \mid x \in A\}$ eine Partition von A mit der gewünschten Zusatzeigenschaft bildet:

Da E reflexiv ist, gilt xEx und somit $x \in E[x]$, d.h. $A = \bigcup_{x \in A} E[x]$.

Ist $E[x] \cap E[y] \neq \varnothing$ und $u \in E[x] \cap E[y]$, so folgt $E[x] = E[y]$:

$$z \in E[x] \Leftrightarrow xEz \overset{xEu}{\Leftrightarrow} uEz \overset{yEu}{\Leftrightarrow} yEz \Leftrightarrow z \in E[y]$$

Zudem gilt

$$\begin{aligned} \exists z \in A : x, y \in E[z] &\Leftrightarrow \exists z : z \in E[x] \cap E[y] \\ &\Leftrightarrow E[x] = E[y] \overset{y \in E[y]}{\Leftrightarrow} xEy \end{aligned}$$

$(ii) \Rightarrow (i)$ Existiert umgekehrt 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,
- symmetrisch, da aus $x, y \in B_i$ auch $y, x \in B_i$ folgt, und
- transitiv, da aus $x, y \in B_i$ und $y, z \in B_j$ wegen $y \in B_i \cap B_j$ die Gleichheit $B_i = B_j$ und somit $x, z \in B_i$ folgt.

■

Als nächstes betrachten wir Ordnungsrelationen, die durch die drei Eigenschaften reflexiv, antisymmetrisch und transitiv definiert sind.

Beispiel 37.

- $(\mathsf{P}(M), \subseteq)$, $(\mathbb{Z}, \leq)$, $(\mathbb{R}, \leq)$ *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 $\mathbb{N}$.* ◁

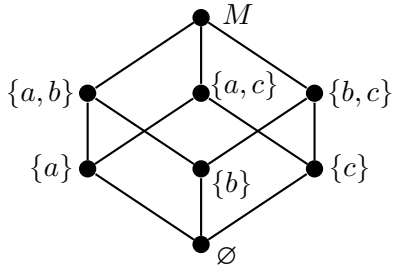
Ordnungen lassen sich sehr anschaulich durch Hasse-Diagramme darstellen. Sei $\leq$ eine Ordnung auf A und sei $<$ die Relation $\leq \cap \overline{Id}_A$. Um die Ordnung $\leq$ in einem ***Hasse-Diagramm*** darzustellen, wird nur der Graph der Relation

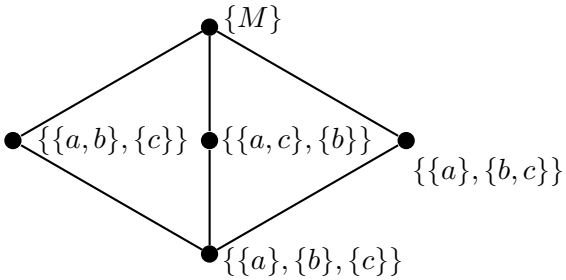
$$\lessdot = < \setminus <^2, \text{ d.h. } x \lessdot y \Leftrightarrow x < y \wedge \neg \exists z : x < z < y$$

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

Beispiel 38.

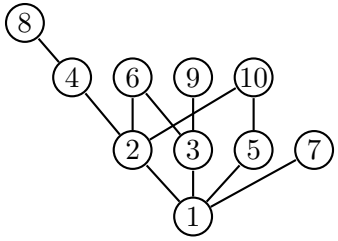
Das Hasse-Diagramm rechts zeigt die Inklusionsrelation auf der Potenzmenge $\mathcal{P}(M)$ von $M = \{a, b, c\}$.





Das Hasse-Diagramm links zeigt die feiner-Relation auf der Menge aller Partitionen von $M = \{a, b, c\}$.

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



◁

Definition 39. *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 40. *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 41. *Sei $\leq$ eine Ordnung auf A und sei $B \subseteq A$.*

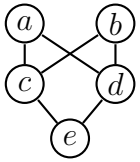
- *Jedes Element $u \in A$ mit $u \leq b$ für alle $b \in B$ heißt **untere** und jedes $o \in A$ mit $b \leq o$ für alle $b \in B$ heißt **obere Schranke** von B.*
- *B heißt **nach oben beschränkt**, wenn B eine obere Schranke hat, und **nach unten beschränkt**, wenn B eine untere Schranke hat.*
- *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 ($s = \sup B$):*

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

Beispiel 42. *Betrachte nebenstehende Ordnung. Die folgende Tabelle zeigt für verschiedene Teilmengen $B \subseteq \{a, b, c, d, e\}$ alle minimalen und maximalen Elemente, alle unteren und oberen Schranken sowie Minimum, Maximum, 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 43.

- *Es kann nicht mehr als ein Supremum und ein Infimum geben.*
- *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*

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

  weder ein Supremum noch ein Infimum.
- *Dagegen hat in $(\mathbb{R},\leq)$ jede beschränkte Teilmenge ein Supremum und ein Infimum (aber eventuell kein Maximum oder Minimum).*

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*

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

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

Bemerkung 45.

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

Definition 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 $f^{-1} : N(f) \to A$ 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

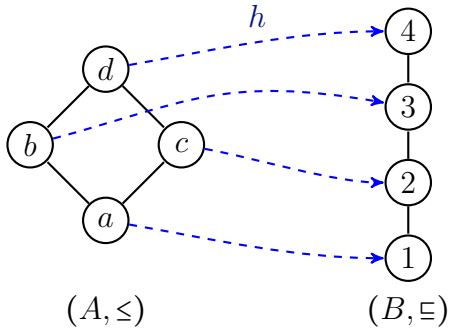
Definition 47. *Seien (A_1,R_1) und (A_2,R_2) Relationalstrukturen.*

- *Eine Abbildung $h : A_1 \to A_2$ heißt* **Homomorphismus**, *falls für alle $a, b \in A_1$ gilt:*

$$aR_1b \Rightarrow h(a)R_2h(b).$$

- *Sind (A_1, R_1) und (A_2, R_2) Ordnungen, so spricht man von* **Ordnungshomomorphismen** *oder einfach von* **monotonen** *Abbildungen.*
- *Injektive Ordnungshomomorphismen werden auch* **streng monotone** *Abbildungen genannt.*

Beispiel 48. *Folgende Abbildung $h : A_1 \to A_2$ ist ein bijektiver Ordnungshomomorphismus.*



Obwohl h ein bijektiver Homomorphismus ist, ist die Umkehrung h^{-1} kein Homomorphismus, da h^{-1} nicht monoton ist. Es gilt nämlich

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

◁

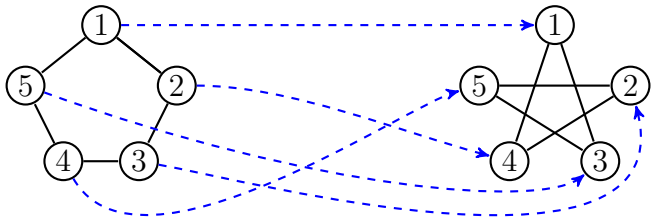
Definition 49. *Ein bijektiver Homomorphismus $h : A_1 \to A_2$, bei dem auch h^{-1} ein Homomorphismus ist, d.h. es gilt*

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

heißt **Isomorphismus**. *In diesem Fall heißen die Strukturen (A_1, R_1) und (A_2, R_2)* **isomorph** *(kurz: $(A_1, R_1) \cong (A_2, R_2)$).*

Beispiel 50.

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



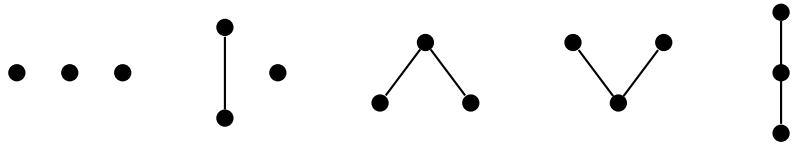
$G = (V, E)$

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

$G' = (V, E')$

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

- *Die Abbildung $h : \mathbb{R} \to \mathbb{R}^+$ mit $h(x) = e^x$ 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).*

◁

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: $q \sim_M p$), wenn M von q und von p ausgehend jeweils dieselben Wörter akzeptiert. Bezeichnen wir den DFA $(Z, \Sigma, \delta, q, E)$ mit M_q, so sind q und p genau dann verschmelzbar, wenn $L(M_q) = L(M_p)$ ist. Offensichtlich ist $\sim_M$ eine Äquivalenzrelation.

Fassen wir alle mit einem Zustand z verschmelzbaren Zustände in dem neuen Zustand

$$[z]_{\sim_M} = \{z' \in Z \mid L(M_{z'}) = L(M_z)\}$$

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

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

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

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

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

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

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

$$\begin{aligned}
L(M_q) = L(M_p) &\Rightarrow \forall x \in \Sigma^* : x \in L(M_q) \leftrightarrow x \in L(M_p) \\
&\Rightarrow \forall x \in \Sigma^* : ax \in L(M_q) \leftrightarrow ax \in L(M_p) \\
&\Rightarrow \forall x \in \Sigma^* : x \in L(M_{\delta(q,a)}) \leftrightarrow x \in L(M_{\delta(p,a)}) \\
&\Rightarrow L(M_{\delta(q,a)}) = L(M_{\delta(p,a)}).
\end{aligned}$$

Als nächstes zeigen wir, dass $L(M') = L(M)$ ist. Sei $x = x_1 \dots x_n$ eine Eingabe und seien

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

die von M bei Eingabe x durchlaufenen Zustände. Wegen

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

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 52).

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 $k = \|\tilde{Z}\| = \|\{L(M_z) \mid z \in Z\}\|$ 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 $\{L_x \mid x \in \Sigma^*\} \subseteq \{L(M_z) \mid z \in Z\}$, da $L_x = L(M_{\hat{\delta}(q_0,x)})$ ist. Die umgekehrte Inklusion gilt ebenfalls, da nach Voraussetzung jeder Zustand $q \in Z$ über ein $x \in \Sigma^*$ erreichbar ist. Also hängt $k = \|\{L(M_z) \mid z \in Z\}\| = \|\{L_x \mid x \in \Sigma^*\}\|$ 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

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

auf Σ^* definierte Äquivalenzrelation $\sim_L$ endlichen Index hat. Die Relation $\sim_L$ wird als *Nerode-Relation* von L bezeichnet.

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 $A \triangle B = (A \setminus B) \cup (B \setminus A)$ die *symmetrische Differenz* von zwei Mengen A und B. Dann ist die Inäquivalenz $p \not\sim_M q$ zweier Zustände p und q gleichbedeutend mit $L(M_p) \triangle L(M_q) \neq \varnothing$. Wir nennen ein Wort $x \in L(M_p) \triangle L(M_q)$ einen *Unterscheider* zwischen p und q. Für $i \geq 0$ sei D_i die Menge aller Paare $\{p, q\}$, die einen Unterscheider x der Länge $|x| \leq i$ haben. Dann enthält $D = \bigcup_{i \geq 0} D_i$ alle inäquivalenten Paare $\{p, q\}$, d.h. es gilt $p \sim_M q \Leftrightarrow \{p, q\} \notin D$.

Beobachtung 52.

- *Das leere Wort ε unterscheidet Endzustände und Nichtendzustände, d.h.*

$$D_0 = \left\{\{p, q\} \subseteq Z \,\middle|\, p \in E, q \notin E\right\}.$$

- *Zudem haben p und q genau dann einen Unterscheider ax der Länge $i + 1$, wenn x die beiden Zustände $\delta(p, a)$ und $\delta(q, a)$ unterscheidet, d.h.*

$$D_{i+1} = D_i \cup \left\{\{p, q\} \subseteq Z \,\middle|\, \exists a \in \Sigma : \{\delta(p, a), \delta(q, a)\} \in D_i\right\}.$$

Da es nur endlich viele Zustandspaare gibt, muss es ein k geben mit $D = D_k$. Offensichtlich gilt

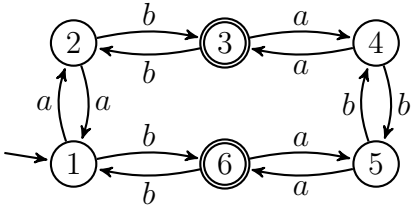
$$D = D_k \Leftrightarrow D_{k+1} = D_k.$$

Der folgende Algorithmus berechnet für einen beliebigen DFA M den zugehörigen Minimal-DFA M'.

Algorithmus `min-DFA`(M)

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

Beispiel 53. *Betrachte den DFA M:*



Dann enthält D_0 die Paare

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

Die Paare in D_0 sind in der folgenden Matrix durch den Unterscheider ε markiert.

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

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 die zusätzlichen Paare $\{1,4\}$, $\{1,5\}$, $\{2,4\}$, $\{2,5\}$ (in obiger Matrix durch den Unterscheider a markiert). Da nun jedoch keines der verbliebenen Paare $\{1,2\}$, $\{3,6\}$, $\{4,5\}$ wegen

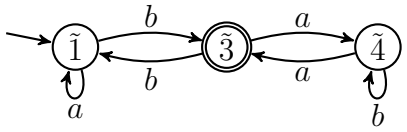
$\{p,q\}$	$\{1,2\}$	$\{3,6\}$	$\{4,5\}$
$\{\delta(p,a),\delta(q,a)\}$	$\{1,2\}$	$\{4,5\}$	$\{3,6\}$
$\{\delta(p,b),\delta(q,b)\}$	$\{3,6\}$	$\{1,2\}$	$\{4,5\}$

zu D_2 hinzugefügt werden kann, gilt $D_2 = D_1$ und somit $D = D_1$.

Aus den unmarkierten Paaren $\{1,2\}$, $\{3,6\}$ und $\{4,5\}$ erhalten wir die Äquivalenzklassen

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

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

$$\begin{aligned}\widetilde{\hat\delta(q_0,x)} = \widetilde{\hat\delta(q_0,y)} &\Leftrightarrow \hat\delta(q_0,x) \sim_M \hat\delta(q_0,y)\\ &\Leftrightarrow L(M_{\hat\delta(q_0,x)}) = L(M_{\hat\delta(q_0,y)}) \Leftrightarrow L_x = L_y\end{aligned}$$

zwei Eingaben x und y den DFA M' genau dann in denselben Zustand $\widetilde{\hat\delta(q_0,x)} = \widetilde{\hat\delta(q_0,y)}$ überführen, wenn $L_x = L_y$ ist, können wir den von M' bei Eingabe x erreichten Zustand $\widetilde{\hat\delta(q_0,x)}$ 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

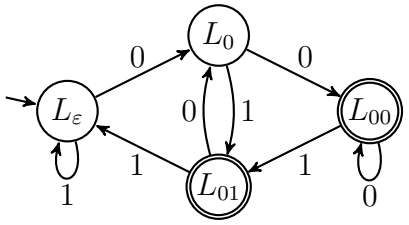
$$\begin{aligned} Z_L &= \{L_x \mid x \in \Sigma^*\},\\ E_L &= \{L_x \mid x \in L\} \text{ und}\\ \delta_L(L_x,a) &= L_{xa}.\end{aligned}$$

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

Beispiel 54. *Für $L = \{x_1 \dots x_n \in \{0,1\}^* \mid n \geq 2$ 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 55 (Myhill und Nerode)**.**

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

Beispiel 56. *Sei $L = \{a^i b^i \mid i \geq 0\}$. Wegen $b^i \in L_{a^i} \triangle L_{a^j}$ für $i \neq j$ hat $\sim_L$ unendlichen Index, d.h. L ist nicht regulär.* ◁

Die Zustände von M_L können anstelle von L_x auch mit den Äquivalenzklassen $\tilde{x}$ von $\sim_L$ benannt werden. Der resultierende Minimal-DFA $M_{\sim_L} = (Z, \Sigma, \delta, \tilde{\varepsilon}, E)$ mit

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

wird auch als ***Äquivalenzklassenautomat*** bezeichnet.

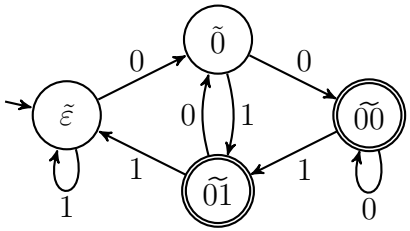
Die Konstruktion von $M_{\sim_L}$ ist meist einfacher als die von M_L, da die Bestimmung der Sprachen L_x entfällt. Um die Überführungsfunktion von $M_{\sim_L}$ aufzustellen, reicht es, ausgehend von $r_1 = \varepsilon$ eine Folge $r_1, \ldots, r_k$ von paarweise bzgl. $\sim_L$ inäquivalenten Wörtern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

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

1. *Wir beginnen mit $r_1 = \varepsilon$.*
2. *Da $r_1 0 = 0 \not\sim_L \varepsilon$ ist, erhalten wir $r_2 = 0$ und setzen $\delta(\tilde{\varepsilon}, 0) = \tilde{0}$.*
3. *Da $r_1 1 = 1 \sim_L \varepsilon$ ist, setzen wir $\delta(\tilde{\varepsilon}, 1) = \tilde{\varepsilon}$.*
4. *Da $r_2 0 = 00 \not\sim_L r_i$ für $i = 1, 2$ ist, erhalten wir $r_3 = 00$ und setzen $\delta(\tilde{0}, 0) = \widetilde{00}$.*
5. *Da $r_2 1 = 01 \not\sim_L r_i$ für $i = 1, 2, 3$ ist, erhalten wir $r_4 = 01$ und setzen $\delta(\tilde{0}, 1) = \widetilde{01}$.*
6. *Da zudem $r_3 0 = 000 \sim_L 00$, $r_3 1 = 001 \sim_L 01$, $r_4 0 = 010 \sim_L 0$ und $r_4 1 = 011 \sim_L \varepsilon$ gilt, setzen wir $\delta(\widetilde{00}, 0) = \widetilde{00}$, $\delta(\widetilde{00}, 1) = \widetilde{01}$, $\delta(\widetilde{01}, 0) = \tilde{0}$ und $\delta(\widetilde{01}, 1) = \tilde{\varepsilon}$.*

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

r	ε	0	00	01
$\widetilde{r0}$	$\tilde{0}$	$\widetilde{00}$	$\widetilde{00}$	$\tilde{0}$
$\widetilde{r1}$	$\tilde{\varepsilon}$	$\widetilde{01}$	$\widetilde{01}$	$\tilde{\varepsilon}$



◁

Wir fassen nochmals die wichtigsten Ergebnisse zusammen.

Korollar 58. *Für jede Sprache L sind folgende Aussagen äquivalent:*

- *L ist regulär (d.h. es gibt einen DFA M mit $L = L(M)$),*
- *es gibt einen NFA N mit $L = L(N)$,*
- *es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$,*
- *die Nerode-Relation $\sim_L$ von 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 59 (Pumping-Lemma für reguläre Sprachen)**.**
Zu jeder regulären Sprache L gibt es eine Zahl $l \geq 0$, 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 $l \geq 0$ mit diesen Eigenschaften existiert, wird das kleinste solche l die ***Pumping-Zahl*** *von L genannt.*

Beispiel 60. *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 61. *Eine endliche Sprache L hat die Pumping-Zahl*

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

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 59 aufgeführten Eigenschaften verletzt ist.

Beispiel 62. *Die Sprache*

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

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

Beispiel 63. *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 64. *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 65. *Mit Hilfe des Pumping-Lemmas kann nicht für jede Sprache $L \notin \mathsf{REG}$ gezeigt werden, dass L nicht regulär ist, da seine Umkehrung falsch ist. So hat beispielsweise die Sprache*

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

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

2.7 Grammatiken

Eine beliebte Methode, Sprachen zu beschreiben, sind Grammatiken. Implizit haben wir diese Methode bei der Definition der regulären Ausdrücke bereits benutzt.

Beispiel 66. *Die Sprache RA aller regulären Ausdrücke über einem Alphabet $\Sigma = \{a_1, \ldots, a_k\}$ lässt sich aus dem Symbol R durch wiederholte Anwendung folgender Regeln erzeugen:*

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

◁

Definition 67. *Eine **Grammatik** ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei*

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

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

Definition 68. *Seien $\alpha, \beta \in (V \cup \Sigma)^*$.*

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

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

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

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

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

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

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

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

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

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

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

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

Beispiel 69. *Wir betrachten nochmals die Grammatik $G = (\{R\}, \Sigma \cup \{\varnothing, \epsilon, (,), ^*, |\}, P, R)$, die die Menge der regulären Ausdrücke über dem Alphabet Σ erzeugt, wobei P die oben angegebenen Regeln enthält. Ist $\Sigma = \{0, 1\}$, so lässt sich der reguläre Ausdruck $(01)^*(\epsilon|\varnothing)$ beispielsweise wie folgt ableiten:*

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

Man unterscheidet vier verschiedene Typen von Grammatiken.

Definition 70. *Sei $G = (V, \Sigma, P, S)$ eine Grammatik.*

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

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

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

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

(*context free languages*) und

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

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

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

Die Sprachklassen

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

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

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

Lemma 71. *Zu jeder regulären Grammatik $G = (V, \Sigma, P, S)$ gibt es eine äquivalente reguläre Grammatik G', die keine Produktionen der Form $A \to a$ hat.*

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

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

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

Satz 72. $\mathsf{REG} = \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}$.

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

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

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

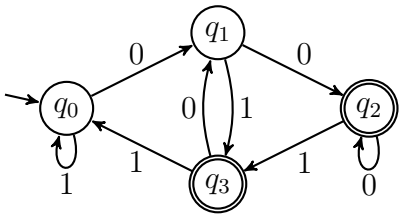
$$\begin{aligned} x \in L(M) &\Leftrightarrow \exists\, q_1, \dots, q_{n-1} \in Z\ \exists\, q_n \in E: \\ &\quad \delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \dots, n \\ &\Leftrightarrow \exists\, q_1, \dots, q_n \in V: \\ &\quad q_{i-1} \to_G x_i q_i \text{ für } i = 1, \dots, n \text{ und } q_n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

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

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

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

Beispiel 73. *Der DFA*



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

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

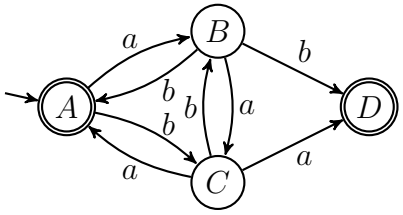
Umgekehrt führt die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ *mit*

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

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

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

auf den NFA



◁

3 Kontextfreie Sprachen

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

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

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

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

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

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

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

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

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

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

■

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

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

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

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

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

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

◁

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

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

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

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

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

Als nächstes zeigen wir folgende Abschlusseigenschaften der kontextfreien Sprachen.

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

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

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

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

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

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

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

erzeugt. ∎

Offen bleibt zunächst, ob die kontextfreien Sprachen auch unter Schnitt und Komplement abgeschlossen sind. Um dies zu verneinen, müssen wir für bestimmte Sprachen nachweisen, dass sie nicht kontextfrei sind. Dies gelingt mit einem Pumping-Lemma für kontextfreie Sprachen.

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. $uv^i wx^i y \in L$ für alle $i \geq 0$.

Für den Beweis benötigen wir Grammatiken in Chomsky-Normalform, die wir im nächsten Abschnitt behandeln werden.

Beispiel 78. *Betrachte die Sprache* $L = \{a^n b^n | n \geq 0\}$. *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* $y = b^{n-1}$. ◁

Beispiel 79. *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.* ◁

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

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

Definition 81. *Regeln der Form $A \to B$ heißen **Variablenumbenennungen**.*

Satz 82. *Zu jeder kontextfreien Grammatik G ex. eine kontextfreie Grammatik G' ohne Variablenumbenennungen mit $L(G') = L(G)$.*

Beweis. Zuerst entfernen wir sukzessive alle Zyklen

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

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

Nun entfernen wir sukzessive die restlichen Variablenumbenennungen, indem wir

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

■

Beispiel 83. *Ausgehend von den Produktionen*

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

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

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

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

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

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

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

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

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

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

Satz 84. *Zu jeder kontextfreien Sprache $L \in \mathsf{CFL}$ gibt es eine CNF-Grammatik G' mit $L(G') = L \setminus \{\varepsilon\}$.*

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

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

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

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

Beispiel 85. *In der Produktionenmenge*

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

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

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

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

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

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

a) Eine Ableitung

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

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

*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 87. *Wir betrachten die Grammatik $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$. Offenbar hat das Wort aabb in G acht verschiedene Ableitungen, die sich allerdings nur in der Reihenfolge der Regelan-*

wendungen unterscheiden:

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

Darunter sind genau eine Links- und genau eine Rechtsableitung:

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

und

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

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

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

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

◁

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

Definition 88. *Sei $G = (V, E)$ ein Digraph.*

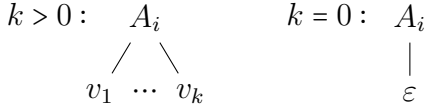
- *Ein **v_0-v_k-Weg** in G ist eine Folge von Knoten $v_0, \ldots, v_k$ mit $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, k-1$. Seine **Länge** ist k.*
- *Ein Weg heißt **einfach** oder **Pfad**, falls alle seine Knoten paarweise verschieden sind.*
- *Ein u-v-Weg der Länge ≥ 1 mit $u = v$ heißt **Zyklus**.*
- *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 89. *Wir ordnen einer Ableitung*

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

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

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

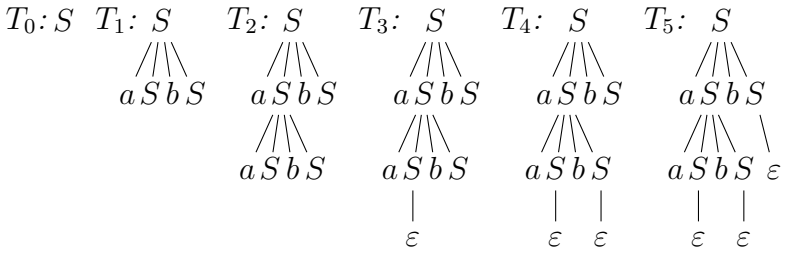


- *Hierbei stellen wir uns die Kanten von oben nach unten gerichtet und die Kinder $v_1 \ldots v_k$ von links nach rechts geordnet vor.*

Beispiel 90. *Betrachte die Grammatik* $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ *und die Ableitung*

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

Die zugehörigen Syntaxbäume sind dann



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

Bemerkung 91.

- *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 Binärbäume sind.

Definition 92. *Die* ***Tiefe*** *eines Baumes mit Wurzel* w *ist die maximale Pfadlänge von* w *zu einem Blatt.*

Lemma 93. *Ein Binärbaum* B *der Tiefe* k *hat höchstens* 2^k *Blätter.*

Beweis. Wir führen den Beweis durch Induktion über k.

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

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

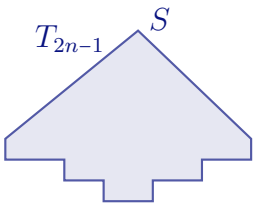
Korollar 94. *Ein Binärbaum* B *mit mehr als* 2^{k-1} *Blättern hat mindestens Tiefe* k*.*

Beweis. Würde B mehr als 2^{k-1} Blätter und eine Tiefe $\leq k-1$ besitzen, so würde dies im Widerspruch zu Lemma 93 stehen. ■

Satz 95 (Pumping-Lemma für kontextfreie Sprachen)**.**
Zu jeder kontextfreien Sprache L *gibt es eine Zahl* l*, so dass sich alle Wörter* $z \in L$ *mit* $|z| \geq l$ *in* $z = uvwxy$ *zerlegen lassen mit*

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

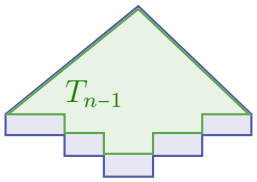
Beweis. Sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik für $L \setminus \{\varepsilon\}$. Dann gibt es in G für jedes Wort $z = z_1 \dots z_n \in L$ mit $n \geq 1$, eine Ableitung

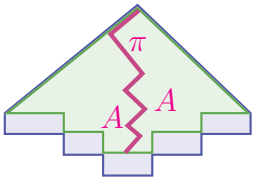


$$S = \alpha_0 \Rightarrow \alpha_1 \cdots \Rightarrow \alpha_m = z.$$

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

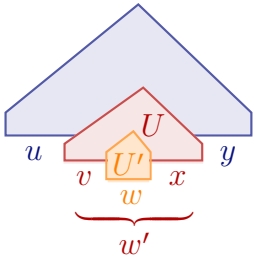
dass zuerst alle Regeln der Form $A \to BC$ und danach die Regeln der Form $A \to a$ zur Anwendung kommen. Dann besteht die Satzform α_{n-1} aus n Variablen und der Syntaxbaum T_{n-1} hat ebenfalls n Blätter. Setzen wir $l = 2^k$, wobei $k = \|V\|$ ist, so hat T_{n-1} im Fall $n \geq l$ mindestens



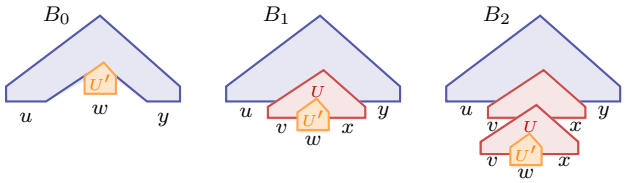


$l = 2^k > 2^{k-1}$ Blätter und daher mindestens die Tiefe k. Sei π ein von der Wurzel ausgehender Pfad maximaler Länge in T_{n-1}. Dann hat π die Länge $\geq k$ und unter den letzten $k+1$ Knoten von π müssen zwei mit derselben Variablen A markiert sein.

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



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



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

Satz 96. *Die Klasse* CFL *ist nicht abgeschlossen unter Schnitt und Komplement.*

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 kontextfrei. Nicht jedoch $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$. Also ist CFL nicht unter Schnitt 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 97. *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 $l = 1, \ldots, n$ und $k = 1, \ldots, n - l + 1$ die Menge

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

aller Variablen, aus denen das an Position k beginnende Teilwort $x_k \ldots x_{k+l-1}$ von x der Länge l ableitbar ist. Dann gilt offensichtlich

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

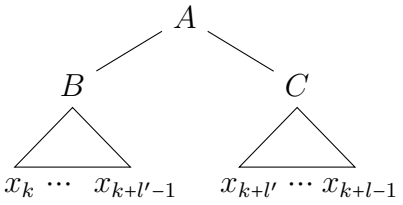
Für $l = 1$ ist

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

und für $l = 2, \ldots, n$ ist

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

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



■

Algorithmus CYK(G, x)

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

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

Beispiel 98. *Betrachte die CNF-Grammatik mit den Produktionen*

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

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

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

Wegen $S \notin V_{3,1}(abb)$ ist $x \notin L(G)$.

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

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

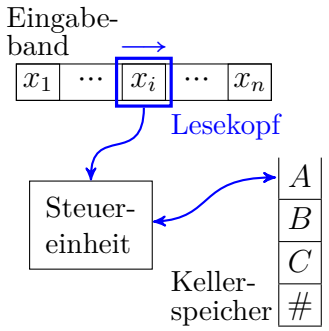
◁

3.4 Kellerautomaten

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 ε-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 99. *Ein* ***Kellerautomat*** *(kurz: PDA; pushdown automaton) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ beschrieben, wobei*

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

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

- 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** $quA \to pB_1 \dots B_k$ aus. Da im Fall $u = \varepsilon$ kein Eingabezeichen gelesen wird, spricht man auch von einem **spontanen** Übergang (oder **ε-Übergang**). Eine **Konfiguration** wird durch ein Tripel

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

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 $quA_1 \to pB_1 \dots B_k$ (mit $u \in \{\varepsilon, x_i\}$) überführt 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 $K_0 \vdash K_1 \vdash K_2 \cdots$. K_0 heißt **Startkonfiguration**

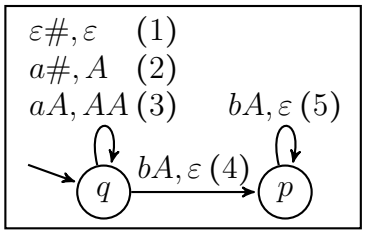
von M bei Eingabe x. Die reflexive, transitive Hülle von $\vdash$ bezeichnen wir wie üblich mit $\vdash^*$. Die von M **akzeptierte** oder **erkannte Sprache** ist

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

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

Beispiel 100. *Sei $M = (Z, \Sigma, \Gamma, \delta, q, \#)$ ein PDA mit $Z = \{q, p\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$ und den Anweisungen*

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



Dann akzeptiert M die Eingabe aabb:

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

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

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

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

$$\underset{(4)}{\vdash} (p, b^{n-1}, A^{n-1}) \overset{n-1}{\underset{(5)}{\vdash}} (p, \varepsilon, \varepsilon).$$

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

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

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. ◁

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

Satz 101. $\mathsf{CFL} = \{L(M) \mid M \text{ ist ein PDA}\}$.

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 $A \to_G \alpha$ in P die Anweisung $q\varepsilon A \to q\alpha$ und für jedes $a \in \Sigma$ die Anweisung $qaa \to q\varepsilon$ zu δ 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_{pAq}, $A \in \Gamma$, $p, q \in Z$, so dass folgende Äquivalenz gilt:

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

Ein Wort x soll also genau dann in G aus X_{pAq} ableitbar sein, wenn M ausgehend vom Zustand p bei Lesen von x in den Zustand q gelangen kann und dabei das Zeichen A aus dem Keller entfernt. Um dies zu erreichen, fügen wir für jede Anweisung $puA \to p_1 A_1 \dots A_k$, $k \geq 0$, die folgenden $\|Z\|^k$ Regeln zu P hinzu:

Für jede Zustandsfolge $p_2, \dots, p_{k+1}$: $X_{pAp_{k+1}} \to u X_{p_1 A_1 p_2} \dots X_{p_k A_k p_{k+1}}$.

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

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

und P enthält neben den Regeln $S \to X_{q_0 \# q}$, $q \in Z$, für jede Anweisung $puA \to p_1 A_1 \dots A_k$, $k \geq 0$, von M und jede Zustandsfolge $p_2, \dots, p_{k+1}$ die Regel $X_{pAp_{k+1}} \to u X_{p_1 A_1 p_2} \dots X_{p_k A_k p_{k+1}}$.

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

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

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

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

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

$m \rightsquigarrow m+1$: Wir zeigen zuerst die Implikation von links nach rechts. Sei x aus X_{pAq} in $m+1$ Schritten ableitbar und sei α die im ersten Schritt abgeleitete Satzform:

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

Wegen $X_{pAq} \to_G \alpha$ gibt es eine Anweisung $puA \to p_1 A_1 \dots A_k$, $k \geq 0$, und Zustände $p_2, \dots, p_{k+1} \in Z$ mit

$$\alpha = u\, X_{p_1 A_1 p_2} \dots X_{p_k A_k p_{k+1}},$$

wobei $p_{k+1} = q$ ist. Wegen $\alpha \Rightarrow^m x$ ex. eine Zerlegung $x = u u_1 \dots u_k$ und Zahlen $m_i \geq 1$ mit $m_1 + \dots + m_k = m$ und

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

Nach IV gibt es somit Rechnungen

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

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

$$\begin{aligned} (p, u u_1 \dots u_k, A) &\vdash (p_1, u_1 \dots u_k, A_1 \dots A_k) \\ &\vdash^{m_1} (p_2, u_2 \dots u_k, A_2 \dots A_k) \\ &\ \ \vdots \\ &\vdash^{m_{k-1}} (p_k, u_k, A_k) \\ &\vdash^{m_k} (p_{k+1}, \varepsilon, \varepsilon). \end{aligned}$$

Sei nun umgekehrt eine Rechnung der Länge $m+1$ gegeben und sei $puA \to p_1 A_1 \dots A_k$ die erste in dieser Rechnung zur Ausführung kommende Anweisung (d.h. $x = ux'$):

$$(p, x, A) \vdash (p_1, x', A_1 \dots A_k) \vdash^m (q, \varepsilon, \varepsilon)$$

Zudem sei p_i für $i = 2, \dots, k+1$ der erste Zustand in dieser Rechnung, in den M mit dem Kellerinhalt $A_i \dots A_k$ gelangt (d.h. $p_{k+1} = q$). Dann enthält P die Regel $X_{pAp_{k+1}} \to uX_{p_1A_1p_2} \dots X_{p_kA_kp_{k+1}}$. Weiter sei u_i für $i = 1, \dots, k$ das Teilwort von x', das M zwischen den Besuchen von p_i und p_{i+1} liest.
Dann gibt es Zahlen $m_i \geq 1$ mit $m_1 + \dots + m_k = m$ und

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

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

$$X_{p_iA_ip_{i+1}} \Rightarrow^{m_i} u_i,\ i = 1, \dots, k,$$

die wir zu der gesuchten Ableitung zusammensetzen können:

$$\begin{aligned} X_{pAp_{k+1}} &\Rightarrow && uX_{p_1A_1p_2} \dots X_{p_{k-1}A_{k-1}p_k}X_{p_kA_kp_{k+1}} \\ &\Rightarrow^{m_1} && uu_1 \dots X_{p_{k-1}A_{k-1}p_k}X_{p_kA_kp_{k+1}} \\ &\ \ \vdots \\ &\Rightarrow^{m_{k-1}} && uu_1 \dots u_{k-1}X_{p_kA_kp_{k+1}} \\ &\Rightarrow^{m_k} && uu_1 \dots u_k = x. \end{aligned}$$

■

Beispiel 102. *Sei $G = (\{S\}, \{a, b\}, P, S)$ mit*

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

Der zugehörige PDA besitzt dann die Anweisungen

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

Der Linksableitung

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

in G entspricht beispielsweise die akzeptierende Rechnung

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

von M und umgekehrt. ◁

Beispiel 103. *Sei M der PDA $(\{p, q\}, \{a, b\}, \{A, \#\}, \delta, p, \#)$ mit*

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

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

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

Die Regelmenge P enthält neben den beiden Startregeln

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

die folgenden Produktionen:

Anweisung $puA \to p_1A_1 \dots A_k$		k	$p_2, \dots, p_{k+1}$	*zugehörige Regel* $X_{pAp_{k+1}} \to uX_{p_1A_1p_2} \dots X_{p_{k-1}A_kp_{k+1}}$	
$p\varepsilon\# \to q\varepsilon$	(1)	0	-	$X_{p\#q} \to \varepsilon$	$(1')$
$pa\# \to pA$	(2)	1	p	$X_{p\#p} \to aX_{pAp}$	$(2')$
			q	$X_{p\#q} \to aX_{pAq}$	$(2'')$
$paA \to pAA$	(3)	2	p, p	$X_{pAp} \to aX_{pAp}X_{pAp}$	$(3')$
			p, q	$X_{pAq} \to aX_{pAp}X_{pAq}$	$(3'')$
			q, p	$X_{pAp} \to aX_{pAq}X_{qAp}$	$(3''')$
			q, q	$X_{pAq} \to aX_{pAq}X_{qAq}$	$(3'''')$
$pbA \to q\varepsilon$	(4)	0	-	$X_{pAq} \to b$	$(4')$
$qbA \to q\varepsilon$	(5)	0	-	$X_{qAq} \to b$	$(5')$

Der akzeptierenden Rechnung

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

von M entspricht dann die Ableitung

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

in G und umgekehrt. ◁

3.5 Deterministisch kontextfreie Sprachen

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

Definition 104. *Ein Kellerautomat heißt* ***deterministisch****, falls $\vdash$ eine rechtseindeutige Relation ist:*

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

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

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

Hat nämlich eine Konfiguration $K = (q, x_i \dots x_n, A_1 \dots A_l)$ zwei Folgekonfigurationen $K_1 \neq K_2$, so gibt es 3 Fälle:

- In K_1 und K_2 wurde x_i gelesen. Dann ist $\|\delta(q, x_i, A_1)\| \geq 2$.
- Weder in K_1 noch in K_2 wurde x_i gelesen. In diesem Fall ist $\|\delta(q, \varepsilon, A_1)\| \geq 2$.
- In K_1 wurde x_i gelesen, aber nicht in K_2. In diesem Fall ist $\|\delta(q, x_i, A_1)\| \geq 1$ und $\|\delta(q, \varepsilon, A_1)\| \geq 1$.

D.h. in allen 3 Fällen ist $(*)$ verletzt. Gilt umgekehrt $(*)$ nicht, so lassen sich für $K = (q, a, A)$ leicht zwei Folgekonfigurationen angeben.

Beispiel 105. *Der PDA $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#)$ mit der Überführungsfunktion*

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

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

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

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

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

erfüllt. ◁

Verlangen wir von einem deterministischen Kellerautomaten, dass er seine Eingabe durch Leeren des Kellers akzeptiert, so können nicht alle regulären Sprachen von deterministischen Kellerautomaten erkannt werden. Um beispielsweise die Sprache $L = \{a, aa\}$ zu erkennen, muss der Keller von M nach Lesen von a geleert werden. Daher ist es M nicht mehr möglich, die Eingabe aa zu akzeptieren. 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 lösen, indem wir deterministischen Kellerautomaten erlauben, ihre Eingabe durch Erreichen eines Endzustands zu akzeptieren.

Definition 106.

- *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*

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

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

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

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

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

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

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

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

Satz 107. *Jede Sprache $L \in \mathsf{DCFL}$ wird von einem DPDA M' erkannt, der alle Eingaben zu Ende liest.*

Beweis. Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA mit $L(M) = L$. Falls M eine Eingabe $x = x_1 \dots x_n$ 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 $\delta(q, x_i, A) = \delta(q, \varepsilon, A) = \varnothing$ 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, ε, A) eine unendliche Folge von ε-Anweisungen ausführt.

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

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

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

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

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

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

Umweg über den neuen Endzustand t vor.

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

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

Schließlich übernehmen wir von M die folgenden Anweisungen:

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

Zusammenfassend transformieren wir M in den DPDA

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

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

Beispiel 108. *Wenden wir diese Konstruktion auf den DPDA*

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

mit der Überführungsfunktion

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

an, so erhalten wir den DPDA

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

mit folgender Überführungsfunktion δ':

δ'	$q_0,\#$	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	$q_2,\square$
ε	–	–	–	–	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	$t,\square$
ε	$q_0\#\square$	–	–	–	–	–	–	–	$r\#$	–	–	–
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 109. *Die Klasse* DCFL *ist unter Komplement abgeschlossen, d.h. es gilt* DCFL = co-DCFL.

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

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

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

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

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

$$\begin{aligned}(q,1)\varepsilon A &\to (p,i)\gamma\\ (q,2)\varepsilon A &\to (p,2)\gamma\end{aligned} \quad \text{mit} \quad i = \begin{cases}1, & p\in E,\\ 2, & p\notin E,\end{cases}$$

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

$$\begin{aligned}(q,1)aA &\to (p,i)\gamma\\ (q,2)\varepsilon A &\to (q,3)A\\ (q,3)aA &\to (p,i)\gamma\end{aligned} \quad \text{mit} \quad i = \begin{cases}1, & p\in E,\\ 2, & p\notin E\end{cases}$$

hinzufügen. ■

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

Beispiel 110. *Angenommen, ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ führt bei der Eingabe $x = a$ folgende Rechnung aus:*

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

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

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

ausführen. Da $(q_1,2), (q_2,1) \notin Z\times\{3\}$ 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

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

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 111. *Die Klasse* DCFL *ist nicht abgeschlossen unter Schnitt, 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 Schnitt 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:

$$L_3 = \{a^i b^j c^k \mid i \neq j \wedge i, j, k \geq 1\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k \wedge i, j, k \geq 1\}.$$

Ihre Vereinigung $L_3 \cup L_4 = \{a^i b^j c^k \mid (i \neq j \vee j \neq k) \wedge i, j, k \geq 1\}$ 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

$$(\overline{L_3 \cup L_4}) \cap L(a^+b^+c^+) = \{a^n b^n c^n \mid n \geq 1\}$$

(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, ε, γ) die Konfiguration,

die M nach Lesen der Eingabe 0 erreicht. Dann erkennt der DPDA $M' = (Z \cup \{s\}, \Sigma, \Gamma, \delta', s, \#, E)$ die Sprache L, wobei δ' wie folgt definiert ist:

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

Es ist leicht zu sehen, dass die beiden Sprachen $\{\varepsilon, 0\}$ 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

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.

4 Kontextsensitive Sprachen

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

4.1 Kontextsensitive Grammatiken

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

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

Beispiel 112. *Betrachte die kontextsensitive Grammatik* $G = (V, \Sigma, P, S)$ *mit* $V = \{S, B\}$, $\Sigma = \{a, b, c\}$ *und den Regeln*

$$P\colon S \to aSBc, abc\ (1,2) \quad cB \to Bc\ (3) \quad bB \to bb\ (4)$$

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

$$\underline{S} \underset{(1)}{\Rightarrow} a\underline{S}Bc \underset{(2)}{\Rightarrow} aabc\underline{B}c \underset{(3)}{\Rightarrow} aab\underline{B}cc \underset{(4)}{\Rightarrow} aabbcc$$

Allgemein gilt für alle $n \geq 1$*:*

$$S \underset{(1)}{\Rightarrow^{n-1}} a^{n-1}S(Bc)^{n-1} \underset{(2)}{\Rightarrow} a^n bc(Bc)^{n-1} \underset{(3)}{\Rightarrow^{\binom{n}{2}}} a^n bB^{n-1}c^n \underset{(4)}{\Rightarrow^{n-1}} a^n b^n c^n$$

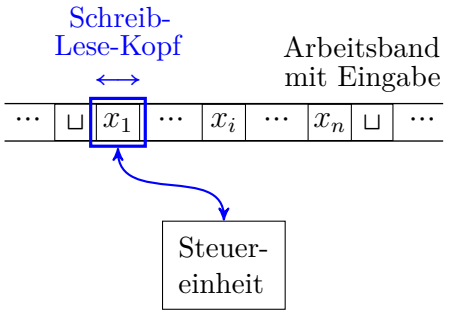
Also gilt $a^n b^n c^n \in L(G)$ *für alle* $n \geq 1$*. Umgekehrt folgt durch Induktion über die Ableitungslänge* m*, dass jede Satzform* u *mit* $S \Rightarrow^m \alpha$ *die folgenden Bedingungen erfüllt:*

- $\#_a(\alpha) = \#_b(\alpha) + \#_B(\alpha) = \#_c(\alpha)$,
- *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.* ◁

4.2 Turingmaschinen

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



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

Definition 113. *Sei* $k \geq 1$*.*

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

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

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

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

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 114. *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 **Folgekonfiguration** von $K = (p, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$ (kurz $K \vdash K'$), falls eine Anweisung*

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

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

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

Man beachte, dass sich die Länge der Bandinschrift $u_i a_i v_i$ beim Übergang von K zu K' 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 $K_0, K_1, K_2 \ldots$ mit $K_0 = K_x$ und $K_0 \vdash K_1 \vdash K_2 \cdots$.*

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

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

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

Beispiel 115. *Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{A, B, \sqcup\}$, $E = \{q_4\}$, wobei δ folgende Anweisungen enthält:*

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

Dann führt M bei Eingabe aabb folgende Rechnung aus:

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

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

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

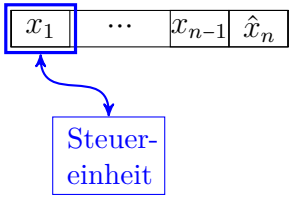
die nicht weiter fortsetzbar ist. Da M deterministisch ist, kann $M(abb)$ auch nicht durch eine andere Rechnung den Endzustand q_4 erreichen. D.h. abb gehört nicht zu $L(M)$. Tatsächlich lässt sich durch Betrachtung der übrigen Fälle ($x = a^n b^m$, $n > m$, $x = a^n b^m a^k$, $m, k \geq 1$, etc.) zeigen,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

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

4.3 Linear beschränkte Automaten

Eine 1-NTM M, die bei keiner Eingabe $x \neq \varepsilon$, deren letztes Zeichen markiert ist, den Bereich der Eingabe verlässt, wird als LBA (linear

beschränkter Automat) bezeichnet. Ein LBA darf also bei Eingaben der Länge $n > 0$ während der Rechnung nur die n mit der Eingabe beschrifteten Bandfelder besuchen und überschreiben. Tatsächlich lässt sich zeigen, dass jede k-NTM, die bei Eingaben der Länge n höchstens linear viele (also $cn+c$ für eine Konstante c) Bandfelder besucht, von einem LBA simuliert werden kann.



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

Definition 116.

a) Für ein Alphabet Σ 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 bei jeder Eingabe $\hat{x}$ der Länge $n \geq 1$ höchstens n Bandfelder besucht:*

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

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

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

*d) Ein deterministischer LBA wird auch als **DLBA** bezeichnet.*

*e) Die Klasse der **deterministisch kontextsensitiven** Sprachen ist definiert als*

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

Beispiel 117. *Es ist nicht schwer, die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ aus Beispiel 115 mit der Überführungsfunktion*

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

in einen DLBA M' für die Sprache $\{a^nb^n \mid n \geq 1\}$ umzuwandeln. Ersetze hierzu

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

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

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

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

$$\begin{aligned} q_0aab\hat{b} &\underset{(1)}{\vdash} Aq_1ab\hat{b} \underset{(2)}{\vdash} Aaq_1b\hat{b} \underset{(4)}{\vdash} Aq_2aB\hat{b} \underset{(5)}{\vdash} q_2AaB\hat{b} \\ &\underset{(7)}{\vdash} Aq_0aB\hat{b} \underset{(1)}{\vdash} AAq_1B\hat{b} \underset{(3)}{\vdash} AABq_1\hat{b} \underset{(4a)}{\vdash} AAq_2B\hat{B} \\ &\underset{(6)}{\vdash} Aq_2AB\hat{B} \underset{(7)}{\vdash} AAq_0B\hat{B} \underset{(8)}{\vdash} AABq_3\hat{B} \underset{(10')}{\vdash} AABq_4\hat{B} \end{aligned}$$

akzeptiert und das Wort ab wird durch die Rechnung

$$q_0a\hat{b} \underset{(1)}{\vdash} Aq_1\hat{b} \underset{(4a)}{\vdash} q_2A\hat{B} \underset{(7)}{\vdash} Aq_0\hat{B} \underset{(8a)}{\vdash} Aq_4\hat{B}$$

akzeptiert. ◁

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. $B \in \mathsf{DCSL} \setminus \mathsf{CFL}$. Die Inklusion von CFL in DCSL wird in den Übungen gezeigt.

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

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

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 $\hat{x} = x_1 \dots x_{n-1}\hat{x}_n$ mit $n > 0$:

1. Markiere das Eingabezeichen x_1 mittels $\tilde{x}_1$ (bzw. $\hat{x}_1$ mittels $\tilde{\hat{x}}_1$)
2. Wähle (nichtdeterministisch) eine Regel $\alpha \to \beta$ aus P
3. Wähle ein beliebiges Vorkommen von β auf dem Band (falls β nicht vorkommt, halte ohne zu akzeptieren)
4. Ersetze die ersten $|\alpha|$ Zeichen von β durch α
5. Falls das erste (oder letzte) Zeichen von β markiert war, markiere auch das erste (letzte) Zeichen von α
6. Verschiebe die Zeichen rechts von β um $|\beta| - |\alpha|$ Positionen nach links und überschreibe die frei werdenden Felder mit Blanks
7. Enthält das Band nur noch das (doppelt markierte) Startsymbol gefolgt von Blanks, 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). 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

$$V = \{S, A\} \cup (Z\Gamma \cup \Gamma) \times \Sigma = \{S, A, (qc, a), (c, a) \mid q \in Z, c \in \Gamma, a \in \Sigma\},$$

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

P:	$S \to A(\hat{a}, a),\ (q_0\hat{a}, a)$		(S)	„Startregeln“
	$A \to A(a, a),\ (q_0 a, a)$		(A)	„A-Regeln“
	$(c, a) \to a$		(F)	„Finale Regeln“
	$(qc, a) \to a,$	falls $q \in E$	(E)	„E-Regeln“
	$(qc, a) \to (q'c', a),$	falls $qc \to_M q'c'N$	(N)	„N-Regeln“
	$(qc, a)(d, b) \to (c', a)(q'd, b),$	falls $qc \to_M q'c'R$	(R)	„R-Regeln“
	$(d, a)(qc, b) \to (q'd, a)(c', b),$	falls $qc \to_M q'c'L$	(L)	„L-Regeln“

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:

$$q_0 x_1 \dots x_{n-1}\hat{x}_n \vdash^m a_1 \dots a_{i-1} q a_i \dots a_n \iff (q_0 x_1, x_1) \dots (\hat{x}_n, x_n) \underset{(N,R,L)}{\Rightarrow}^m (a_1, x_1) \dots (q a_i, x_i) \dots (a_n, x_n)$$

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

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

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 119. *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*

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

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

$$\begin{array}{ll}
S \to A(\hat{a}, a),\ A(\hat{b}, b),\ (q_0\hat{a}, a),\ (q_0\hat{b}, b) & (S_1\text{-}S_4) \\
A \to A(a, a),\ A(b, b),\ (q_0a, a),\ (q_0b, b) & (A_1\text{-}A_4)
\end{array}$$

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

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

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

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

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

$$\begin{array}{ll}
(d, a)(q_1b, a) \to (q_2d, a)(B, a), & (d, b)(q_1b, a) \to (q_2d, b)(B, a) \\
(d, a)(q_1b, b) \to (q_2d, a)(B, b), & (d, b)(q_1b, b) \to (q_2d, b)(B, b)
\end{array}$$

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

$$\begin{array}{ll}
(q_0a, a)(d, a) \to (A, a)(q_1d, a), & (q_0a, a)(d, b) \to (A, a)(q_1d, b) \\
(q_0a, b)(d, a) \to (A, b)(q_1d, a), & (q_0a, b)(d, b) \to (A, b)(q_1d, b)
\end{array}$$

◁

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

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

5 Entscheidbare und semi-entscheidbare Sprachen

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

Definition 120.

a) *Eine NTM M **hält** bei Eingabe x, falls alle Rechnungen von $M(x)$ eine endliche Länge haben. Falls $M(x)$ nicht hält, schreiben wir auch kurz $M(x) = \uparrow$.*

b) *Eine NTM M **entscheidet** eine Eingabe x, falls $M(x)$ hält oder eine Konfiguration mit einem Endzustand erreichen kann.*

c) *Eine Sprache $L \subseteq \Sigma^*$ heißt **entscheidbar**, falls eine DTM M mit $L(M) = L$ existiert, die jede Eingabe $x \in \Sigma^*$ entscheidet.*

d) *Jede von einer DTM M erkannte Sprache heißt **semi-entscheidbar**.*

Bemerkung 121.

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

Wir wenden uns nun der Berechnung von Funktionen zu.

Definition 122. *Eine k-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ **berechnet** eine Funktion $f : \Sigma^* \to \Gamma^*$, falls M bei jeder Eingabe $x \in \Sigma^*$ in einer Konfiguration*

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

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

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

Definition 123.

a) *Eine **partielle Funktion** hat die Form $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$.*

b) *Für $f(x) = \uparrow$ sagen wir auch $f(x)$ ist **undefiniert**.*

c) *Der **Definitionsbereich** (engl. domain) von f ist*

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

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

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

e) *f heißt **total**, falls $dom(f) = \Sigma^*$ ist.*

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

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:

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

Dann gilt $\mathsf{FREC} \subsetneq \mathsf{FREC}_p$ und

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

Wir wissen bereits, dass die Inklusionen $\mathsf{REG} \subsetneq \mathsf{DCFL} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DCSL}$ echt sind. In diesem Abschnitt werden wir die Echtheit der Inklusion $\mathsf{REC} \subsetneq \mathsf{RE}$ zeigen. Dass CSL eine echte Teilklasse von REC ist, wird in den Übungen gezeigt.

Beispiel 124. *Bezeichne x^+ den **lexikografischen Nachfolger** von $x \in \Sigma^*$. Für $\Sigma = \{0,1\}$ ergeben sich beispielsweise folgende Werte:*

x	ε	0	1	00	01	10	11	000	...
x^+	0	1	00	01	10	11	000	001	...

Betrachte die auf Σ^* definierten partiellen Funktionen f_1, f_2, f_3, f_4 mit

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

Da diese vier partiellen Funktionen alle berechenbar sind, gehören die totalen Funktionen f_1, f_2, f_3 zu FREC, während f_4 zu FREC_p gehört.◁

Wie der nächste Satz zeigt, lässt sich jedes Entscheidungsproblem auf ein funktionales Problem zurückführen.

Definition 125. *Für eine Sprache $A \subseteq \Sigma^*$ sind die **charakteristische Funktion χ_A** und die **partielle charakteristische Funktion $\hat{\chi}_A$** wie folgt definiert:*

$$\chi_A(x) = \begin{cases}1, & x \in A\\ 0, & x \notin A\end{cases} \qquad \text{und} \qquad \hat{\chi}_A(x) = \begin{cases}1, & x \in A\\ \uparrow, & x \notin A\end{cases}$$

Satz 126.

(i) Eine Sprache $A \subseteq \Sigma^*$ ist genau dann entscheidbar, wenn ihre charakteristische Funktion χ_A berechenbar ist.

(ii) Eine Sprache $A \subseteq \Sigma^*$ ist genau dann semi-entscheidbar, wenn ihre partielle charakteristische Funktion $\hat{\chi}_A$ berechenbar ist.

Beweis. Teil (i) wird in den Übungen bewiesen. Teil (ii) folgt direkt aus Satz 127. ■

Satz 127. *Folgende Eigenschaften sind äquivalent:*

1. *A ist semi-entscheidbar (d.h. A wird von einer DTM akzeptiert),*
2. *$\hat{\chi}_A$ ist berechenbar,*
3. *$A = dom(f)$ für eine berechenbare partielle Funktion f (d.h. es gibt eine DTM M mit $A = \{x \in \Sigma^* \mid M(x)\downarrow\}$).*

Beweis.

1) ⇒ 2)**:** Sei M eine DTM mit $L(M) = A$. Dann lässt sich M so modifizieren, dass sie eine 1 ausgibt, sobald sie einen Endzustand erreicht, und in eine Endlosschleife geht, sobald sie eine Konfiguration ohne Folgekonfiguration erreicht.

2) ⇒ 3)**:** Der Definitionsbereich von $\hat{\chi}_A$ ist die Sprache A. Folglich hat die partielle Funktion $f = \hat{\chi}_A$ die gewünschte Eigenschaft.

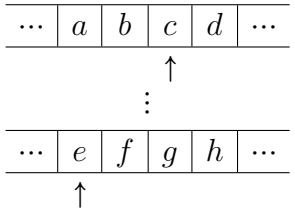
3) ⇒ 1)**:** Sei M eine DTM, die eine partielle Funktion f mit $dom(f) = A$ berechnet. Dann lässt sich M so modifizieren, dass sie genau dann in einen Endzustand übergeht, wenn M eine Konfiguration ohne Folgekonfiguration erreicht. ■

Definition 128. *Eine Sprache $A \subseteq \Sigma^*$ heißt* ***rekursiv aufzählbar****, falls A entweder leer oder das Bild $img(f)$ einer berechenbaren Funktion $f : \Gamma^* \to \Sigma^*$ für ein beliebiges Alphabet Γ ist.*

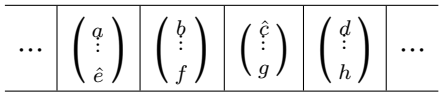
Satz 129. *Folgende Eigenschaften sind äquivalent:*

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

Beweis. 1) ⇒ 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

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

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

> Zuerst geht M' solange nach rechts, bis sie alle mit ˆ markierten Zeichen (z.B. $\hat{a}_1, \dots, \hat{a}_k$) 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) ⇒ 3) ⇒ 4)**:** Diese beiden Implikationen lassen sich ganz ähnlich wie die Charakterisierung der Typ-1 Sprachen durch LBAs zeigen (siehe Übungen).

4) ⇒ 5)**:** Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM, die eine Sprache $A \neq \varnothing$ 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

$$code(K) = \#q\#u_1\#a_1\#v_1\# \dots \#u_k\#a_k\#v_k\#$$

über dem Alphabet $\tilde{\Gamma} = Z \cup \Gamma \cup \{\#\}$ und eine Rechnung $K_0 \vdash \cdots \vdash K_t$ durch $code(K_0) \dots code(K_t)$, so lassen sich die Wörter von A durch folgende Funktion $f : \tilde{\Gamma}^* \to \Sigma^*$ aufzählen (dabei ist x_0 ein beliebiges Wort in A):

$$f(x) = \begin{cases} y, & x \text{ kodiert eine Rechnung } K_0 \vdash \cdots \vdash K_t \text{ von } M \\ & \text{mit } K_0 = K_y \text{ und } K_t \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k \\ x_0, & \text{sonst.} \end{cases}$$

Da f berechenbar ist, ist $A = img(f)$ rekursiv aufzählbar.

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

> M' berechnet bei Eingabe x auf dem 2. Band der Reihe nach für alle Wörter $y \in \Gamma^*$ den Wert $f(y)$ durch Simulation von $M(y)$ und akzeptiert, sobald sie ein y mit $f(y) = x$ findet. ■

Satz 130. *Folgende Eigenschaften sind äquivalent:*

1. *A ist entscheidbar (d.h. A wird von einer DTM akzeptiert, die alle Eingaben entscheidet),*
2. *die charakteristische Funktion χ_A von A ist berechenbar,*
3. *A wird von einer 1-DTM akzeptiert, die bei allen Eingaben hält,*
4. *A wird von einer NTM akzeptiert, die bei allen Eingaben hält,*
5. *A und $\bar{A}$ sind vom Typ 0.*

Die Äquivalenz der ersten vier Bedingungen wird in den Übungen gezeigt. Hier zeigen wir nur die Äquivalenz dieser vier Bedingungen zur fünften.

Satz 131. *A ist genau dann entscheidbar, wenn A und $\bar{A}$ semi-entscheidbar sind, d.h.* $\mathsf{REC} = \mathsf{RE} \cap \mathsf{co\text{-}RE}$.

Beweis. Sei A entscheidbar. Dann ist χ_A und somit auch $\chi_{\bar{A}}$ berechenbar. Dies bedeutet jedoch, dass auch $\bar{A}$ entscheidbar ist. Also sind sowohl A als auch $\bar{A}$ semi-entscheidbar. Für die Rückrichtung seien $f_1, f_2 : \Gamma^* \to \Sigma^*$ Turing-berechenbare Funktionen mit $img(f_1) = A$ und $img(f_2) = \bar{A}$. Wir betrachten folgende k-DTM M, die bei Eingabe x für jedes $y \in \Gamma^*$ die beiden Werte $f_1(y)$ und $f_2(y)$ bestimmt und im Fall

- $f_1(y) = x$ in einem Endzustand
- $f_2(y) = x$ in einem Nichtendzustand

hält. Da jede Eingabe x entweder in $img(f_1) = A$ oder in $img(f_2) = \bar{A}$ 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 wir einer Turingmaschine eine andere Turingmaschine als Eingabe vorlegen können, müssen wir eine geeignete Kodierung von Turingmaschinen vereinbaren (diese wird auch Gödelisierung genannt).

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

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

kodieren. Dabei ist $bin(n)$ die Binärdarstellung von n und $b_N = 0$, $b_L = 1$, sowie $b_R = 10$. M lässt sich nun als ein Wort über dem Alphabet $\{0, 1, \#\}$ kodieren, indem wir die Anweisungen von M in kodierter Form auflisten. Kodieren wir die Zeichen $0, 1, \#$ binär (z.B. $0 \mapsto 00$, $1 \mapsto 11$, $\# \mapsto 10$), so gelangen wir zu einer Binärkodierung w_M von M. Die Binärzahl w_M wird auch die ***Gödel-Nummer*** von M genannt (tatsächlich kodierte Kurt Gödel Turingmaschinen durch natürliche Zahlen und nicht durch Binärstrings). Die Maschine M_w ist durch die Angabe von w bis auf die Benennung ihrer Zustände und Arbeitszeichen eindeutig bestimmt. Ganz analog lassen sich auch DTMs mit

einer beliebigen Anzahl von Bändern (sowie NTMs, Konfigurationen oder Rechnungen von TMs) kodieren.

Umgekehrt können wir jedem Binärstring $w \in \{0,1\}^*$ eine DTM M_w wie folgt zuordnen:

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

Hierbei ist M_0 eine beliebige DTM.

Definition 132.

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

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

b) Das ***spezielle Halteproblem*** *ist*

$$\mathrm{K} = \left\{ w \in \{0,1\}^* \,\middle|\, \begin{array}{l} \text{die DTM } M_w \\ \text{hält bei Eingabe } w \end{array} \right\}$$

χ_H	x_1	x_2	x_3	$\cdots$
w_1	1	1	0	$\cdots$
w_2	0	0	1	$\cdots$
w_3	1	1	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

χ_K				
w_1	1			
w_2		0		
w_3			1	
$\vdots$				$\ddots$

Der Werteverlauf der charakteristischen Funktion χ_K von K stimmt also mit der Diagonalen der als Matrix dargestellten charakteristischen Funktion χ_H von H überein.

Satz 133. $\mathrm{K} \in \mathsf{RE} \setminus \mathsf{co\text{-}RE}$.

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

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

Da f berechenbar und $img(f) = \mathrm{K}$ ist, folgt $\mathrm{K} \in \mathsf{RE}$. Um zu zeigen, dass $\bar{K}$ nicht semi-entscheidbar ist, führen wir die Annahme $\bar{K} \in \mathsf{RE}$ auf einen Widerspruch.

Wir erklären zuerst die Beweisidee. Für eine DTM M sei $dom(M) = \{x \in \Sigma^* \mid M(x)\downarrow\}$ der Definitionsbereich der von M berechneten Funktion. Wir wissen bereits, dass $\mathsf{RE} = \{dom(M) \mid M \text{ ist eine DTM}\}$ ist (siehe Satz 127). Sei $A = (a_{w,x})_{w,x\in\{0,1\}^*}$ die durch $a_{w,x} = \chi_H(w\#x)$ definierte Binärmatrix. Wegen

$$a_{w,x} = \chi_H(w\#x) = \chi_{dom(M_w)}(x)$$

repräsentiert Zeile w von A die Sprache $dom(M_w)$, d.h. die Zeilen von A repräsentieren genau die semi-entscheidbaren Binärsprachen. Wegen $a_{w,w} = \chi_H(w\#w) = \chi_K(w)$ repräsentiert die Diagonale von A die Sprache K. Da aber die komplementierte Diagonale von A mit keiner Zeile von A übereinstimmen kann, folgt $\bar{K} \notin \mathsf{RE}$ und somit $K \notin \mathsf{REC}$.

Nehmen wir also an, die Sprache

$$\bar{K} = \{w \mid M_w(w) \text{ hält nicht}\}$$

wäre semi-entscheidbar. Dann existiert eine DTM $M_{w'}$ mit $dom(M_{w'}) = \bar{K}$, d.h. es gilt

$$M_{w'}(w) \text{ hält} \Leftrightarrow w \in \bar{K} \qquad (*)$$

χ_H	x_1	x_2	x_3	$\cdots$
w_1	**1**	1	0	$\cdots$
w_2	0	**0**	1	$\cdots$
w_3	1	1	**1**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

w'	0	1	0	$\cdots$

Für die Kodierung w' von $M_{w'}$ folgt dann aber

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

■

Die Methode in obigem Beweis wird als ***Diagonalisierung*** bezeichnet. Mit dieser Beweistechnik lässt sich auch eine entscheidbare Sprache definieren, die sich von jeder kontextsensitiven Sprache unterscheidet (siehe Übungen).

Korollar 134. $\mathsf{REC} \subsetneq \mathsf{RE}$.

Beweis. Klar da $\mathrm{K} \in \mathsf{RE} \setminus \mathsf{REC}$. ■

Definition 135.

*a) Eine Sprache $A \subseteq \Sigma^*$ heißt auf $B \subseteq \Gamma^*$ **reduzierbar** (kurz: $A \leq B$), falls eine berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ ex., so dass gilt:*

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

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

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

Beispiel 136. *Es gilt* $\mathrm{K} \leq \mathrm{H}$ *mittels* $f : w \mapsto w\#w$, *da für alle* $w \in \{0,1\}^*$ *gilt:*

$$\begin{aligned} w \in K &\Leftrightarrow M_w \text{ ist eine DTM, die bei Eingabe } w \text{ hält} \\ &\Leftrightarrow w\#w \in H. \end{aligned}$$

◁

Satz 137. *Die Klasse* REC *ist unter* $\leq$ *abgeschlossen.*

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

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

Wegen

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

folgt

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

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

Der Abschluss von RE unter $\leq$ folgt analog (siehe Übungen).

Korollar 138.

1. $A \leq B \wedge A \notin \mathsf{REC} \Rightarrow B \notin \mathsf{REC}$,
2. $A \leq B \wedge A \notin \mathsf{RE} \Rightarrow B \notin \mathsf{RE}$.

Beweis. Aus der Annahme, dass B entscheidbar (bzw. semi-entscheidbar) ist, folgt wegen $A \leq B$, dass dies auch auf A zutrifft (Widerspruch). ■

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

Korollar 139. $\mathrm{H} \notin \mathsf{REC}$.

Definition 140.

*a) Eine Sprache B heißt **hart** für eine Sprachklasse $\mathcal{C}$ (kurz: **$\mathcal{C}$-hart** oder **$\mathcal{C}$-schwer**), falls jede Sprache $A \in \mathcal{C}$ auf B reduzierbar ist:*

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

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

Beispiel 141. *Das Halteproblem H ist* RE*-hart, da sich jede semi-entscheidbare Sprache A auf H reduzieren lässt. Die Reduktion $A \leq H$ leistet beispielsweise die Funktion $x \mapsto w\#bin(x)$, wobei w die Kodierung einer DTM M_w mit $dom(M_w) = \{bin(x) \mid x \in A\}$ ist.* ◁

Definition 142. *Das **Halteproblem bei leerem Band** ist die Sprache*

$$\mathrm{H}_0 = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l} \text{die DTM } M_w \text{ hält} \\ \text{bei Eingabe } \varepsilon \end{array} \right\}$$

χ_{H_0}	x_1 $(= \varepsilon)$
w_1	1
w_2	1
w_3	0
$\vdots$	$\vdots$

Satz 143. H_0 *ist* RE*-vollständig.*

Beweis. Da die Funktion $w \mapsto w\#\varepsilon$ die Sprache H_0 auf H reduziert und da $H \in \mathsf{RE}$ ist, folgt $H_0 \in \mathsf{RE}$.

Für den Beweis, dass H_0 RE-hart ist, sei $A \in \mathsf{RE}$ beliebig und sei M_A eine DTM, die $\hat{\chi}_A$ berechnet. Um A auf H_0 zu reduzieren, transformieren wir ein Wort x in die Kodierung w_x einer DTM, die zunächst ihre Eingabe durch x ersetzt und dann $M_A(x)$ simuliert. Dann gilt

$$x \in A \Leftrightarrow w_x \in H_0$$

und somit $A \leq H_0$ mittels der Reduktionsfunktion $x \mapsto w_x$. ∎

Insbesondere folgt also $K \leq \mathrm{H}_0$, d.h. H_0 ist unentscheidbar.

5.2 Der Satz von Rice

Frage. *Kann man einer beliebig vorgegebenen TM ansehen, ob die von ihr berechnete partielle 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 partielle Funktion hat diese Eigenschaft).*

Definition 144. *Zu einer Klasse $\mathcal{F}$ von partiellen Funktionen definieren wir die Sprache*

$$L_{\mathcal{F}} = \left\{ w \in \{0,1\}^* \;\middle|\; \text{die DTM } M_w \text{ ber. eine partielle Fkt. in } \mathcal{F} \right\}.$$

Die Eigenschaft $\mathcal{F}$ heißt ***trivial****, wenn $L_{\mathcal{F}} = \varnothing$ oder $L_{\mathcal{F}} = \{0,1\}^*$ ist.*

Der Satz von Rice besagt, dass $L_{\mathcal{F}}$ nur für triviale Eigenschaften entscheidbar ist:

$$L_{\mathcal{F}} \neq \varnothing \wedge L_{\mathcal{F}} \neq \{0,1\}^* \Rightarrow L_{\mathcal{F}} \notin \mathsf{REC}.$$

Satz 145 (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 $\overline{L}_{\mathcal{F}}$). Die Idee besteht darin, für eine gegebene DTM M_w eine DTM $M_{w'}$ zu konstruieren mit

$$w \in \mathrm{H}_0 \Leftrightarrow M_{w'} \text{ berechnet (k)eine partielle Funktion in } \mathcal{F}.$$

Hierzu lassen wir $M_{w'}$ bei Eingabe x zunächst einmal die DTM M_w bei Eingabe ε simulieren. Falls $w \notin \mathrm{H}_0$ ist, berechnet $M_{w'}$ also die überall undefinierte Funktion u mit $u(x) = \uparrow$ für alle $x \in \Sigma^*$.

Für das Folgende nehmen wir 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 $L_{\neg\mathcal{F}} = \overline{L}_{\mathcal{F}}$ 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 partielle Funktion $f \in \mathcal{F}$ berechnet.

Da $\mathcal{F}$ nicht trivial ist, gibt es eine DTM M_f, die eine partielle Funktion $f \in \mathcal{F}$ berechnet. Betrachte die Reduktionsfunktion $h : \{0,1\}^* \to \{0,1\}^*$ mit

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

Dann ist $h : w \mapsto w'$ eine totale berechenbare Funktion und es gilt

$$w \in H_0 \Rightarrow M_{w'} \text{ berechnet } f \Rightarrow w' \in L_{\mathcal{F}},$$
$$w \notin H_0 \Rightarrow M_{w'} \text{ berechnet } u \Rightarrow w' \notin L_{\mathcal{F}}.$$

Dies zeigt, dass h das Problem H_0 auf $L_{\mathcal{F}}$ reduziert, und da H_0 unentscheidbar ist, muss auch $L_{\mathcal{F}}$ unentscheidbar sein. ∎

Beispiel 146. *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

$$\mathcal{F} = \{f \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\}$$

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 $g(x) = 0$ auf $\{0\}^*$ nicht in $\mathcal{F}$ enthalten ist.

Dagegen ist die Eigenschaft $\mathcal{F}' = \{\chi_H\}$ trivial, da $L_{\mathcal{F}'} = \varnothing$ 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 147. *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 $L_{\mathcal{S}} \in \{\varnothing, \{0,1\}^*\}$ ist.

5.3 Entscheidungsprobleme für Sprachklassen

Neben dem Wortproblem sind für eine Sprachklasse $\mathcal{C}$ auch folgende Entscheidungsprobleme interessant:

Das Schnittproblem

Gegeben: Zwei Sprachen L_1 und L_2 aus $\mathcal{C}$.
Gefragt: Ist $L_1 \cap L_2 = \varnothing$?

Das Äquivalenzproblem

Gegeben: Zwei Sprachen L_1 und L_2 aus $\mathcal{C}$.
Gefragt: Gilt $L_1 = L_2$?

Das Leerheitsproblem

Gegeben: Eine Sprache L aus $\mathcal{C}$.
Gefragt: Ist $L = \varnothing$?

Hierbei repräsentieren wir Sprachen in $\mathcal{C} = \mathsf{REG}, \mathsf{CFL}, \mathsf{CSL}, \mathsf{RE}$ durch entsprechende Grammatiken und Sprachen in $\mathcal{C} = \mathsf{DCFL}, \mathsf{DCSL}$ durch entsprechende Akzeptoren. Um die Unentscheidbarkeit einiger dieser Probleme zu beweisen, betrachten wir zunächst das Postsche Korrespondenzproblem.

Definition 148. *Sei Σ ein beliebiges Alphabet mit $\# \notin \Sigma$. Das **Postsche Korrespondenzproblem** über Σ (kurz **$\mathbf{PCP}_\Sigma$**) ist wie folgt definiert.*

Gegeben: *k Paare $(x_1, y_1), \ldots, (x_k, y_k)$ von Wörtern über Σ.*

Gefragt: *Gibt es eine Folge $\alpha = (i_1, \ldots, i_n)$, $n \geq 1$, von Indizes $i_j \in \{1, \ldots, k\}$ mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?*

*Das **modifizierte PCP über Σ** (kurz **$\mathbf{MPCP}_\Sigma$**) fragt nach einer Lösung $\alpha = (i_1, \ldots, i_n)$ mit $i_1 = 1$.*

Wir notieren eine PCP-Instanz meist in Form einer Matrix $\binom{x_1 \ldots x_k}{y_1 \ldots y_k}$ und kodieren sie durch das Wort $x_1\#y_1\#\ldots\#x_k\#y_k$.

Beispiel 149. *Die Instanz $I = \binom{x_1\ x_2\ x_3}{y_1\ y_2\ y_3} = \binom{a\ \ ab\ caa}{aca\ bc\ aa}$ besitzt wegen*

$$x_1x_3x_2x_3 = acaaabcaa$$
$$y_1y_3y_2y_3 = acaaabcaa$$

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

Lemma 150. *Für jedes Alphabet Σ gilt* $\mathrm{PCP}_\Sigma \leq \mathrm{PCP}_{\{a,b\}}$.

Beweis. Sei $\Sigma = \{a_1, \ldots, a_m\}$. Für ein Zeichen $a_i \in \Sigma$ sei $c(a_i) = 01^{i-1}$ und für ein Wort $w = w_1 \ldots w_n \in \Sigma^*$ mit $w_i \in \Sigma$ sei $c(w) = c(w_1) \ldots c(w_n)$. Dann folgt $\mathrm{PCP}_\Sigma \leq \mathrm{PCP}_{\{a,b\}}$ mittels der Reduktionsfunktion

$$f : \begin{pmatrix} x_1 \ldots x_k \\ y_1 \ldots y_k \end{pmatrix} \mapsto \begin{pmatrix} c(x_1) \ldots c(x_k) \\ c(y_1) \ldots c(y_k) \end{pmatrix}.$$

∎

f reduziert z.B. die $\mathrm{PCP}_{\{0,1,2\}}$-Instanz $I = \left(\begin{smallmatrix} 0 & 01 & 200 \\ 020 & 12 & 00 \end{smallmatrix}\right)$ auf die äquivalente $\mathrm{PCP}_{\{a,b\}}$-Instanz $f(I) = \left(\begin{smallmatrix} a & aab & abbaa \\ aabba & abab & aa \end{smallmatrix}\right)$.
Im Folgenden lassen wir im Fall $\Sigma = \{a, b\}$ den Index weg und schreiben einfach PCP (bzw. MPCP).

Satz 151. $\mathrm{MPCP} \leq \mathrm{PCP}$.

Beweis. Wir zeigen $\mathrm{MPCP} \leq \mathrm{PCP}_\Sigma$ für $\Sigma = \{a, b, \langle, |, \rangle\}$. Wegen $\mathrm{PCP}_\Sigma \leq \mathrm{PCP}$ folgt hieraus $\mathrm{MPCP} \leq \mathrm{PCP}$. Für ein Wort $w = w_1 \ldots w_n$ sei

$\overleftarrow{w}$	$\overleftarrow{w}$	$\overleftarrow{w}$	$\overrightarrow{w}$
$\langle w_1 \vert \ldots \vert w_n \vert$	$\langle w_1 \vert \ldots \vert w_n$	$\vert w_1 \vert \ldots \vert w_n$	$w_1 \vert \ldots \vert w_n \vert$

Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ:

$$f : \begin{pmatrix} x_1 \ldots x_k \\ y_1 \ldots y_k \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \cdots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \cdots & \overleftarrow{y_k} & \vert\rangle \end{pmatrix}.$$

Dabei nehmen wir an, dass $(x_i, y_i) \neq (\varepsilon, \varepsilon)$ ist, da wir diese Paare im Fall $i > 1$ einfach weglassen und im Fall $i = 1$ $f(I) = \left(\begin{smallmatrix} a \\ a \end{smallmatrix}\right)$ setzen können. Folglich enthält auch $f(I)$ nicht das Paar $(\varepsilon, \varepsilon)$. Beispielsweise ist

$$f\begin{pmatrix} aa & b & bab & bb \\ aab & bb & a & b \end{pmatrix} = \begin{pmatrix} \langle a\vert a\vert & a\vert a\vert & b\vert & b\vert a\vert b\vert & b\vert b\vert & \rangle \\ \langle a\vert a\vert b & \vert a\vert a\vert b & \vert b\vert b & \vert a & \vert b & \vert\rangle \end{pmatrix}.$$

Da jede MPCP-Lösung $\alpha = (1, i_2, \ldots, i_n)$ für I auf eine PCP-Lösung $\alpha' = (1, i_2 + 1, \ldots, i_n + 1, k + 2)$ für $f(I)$ führt, folgt

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

Für die umgekehrte Implikation sei $\alpha' = (i_1, \ldots, i_n)$ eine PCP-Lösung für

$$f(I) = \begin{pmatrix} \overleftarrow{x_1} & \overleftarrow{x_1} & \cdots & \overleftarrow{x_k} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \cdots & \overleftarrow{y_k} & \vert\rangle \end{pmatrix}.$$

Dann muss $i_1 = 1$ und $i_n = k + 2$ sein, da das Lösungswort mit $\langle$ beginnen und mit $\rangle$ enden muss (und $f(I)$ nicht das Paar $(\varepsilon, \varepsilon)$ enthält). Wählen wir α' von minimaler Länge, so ist $i_j \in \{2, \ldots, k + 1\}$ für $j = 2, \ldots, n - 1$. Dann ist aber

$$\alpha = (i_1, i_2 - 1, \ldots, i_{n-1} - 1)$$

eine MPCP-Lösung für I. ∎

Satz 152. PCP *ist* RE*-vollständig und damit unentscheidbar.*

Beweis. Es ist leicht zu sehen, dass $\mathrm{PCP} \in \mathsf{RE}$ ist. Um zu zeigen, dass PCP RE-hart ist, sei A eine beliebige Sprache in RE und sei $G = (V, \Sigma, P, S)$ eine Typ-0 Grammatik für A. Sei $\Gamma = V \cup \Sigma \cup \{\langle, |, \rangle\}$. Dann können wir eine Eingabe $w \in \Sigma^*$ in eine MPCP_Γ-Instanz $f(w) = \left(\begin{smallmatrix} x_1 \ldots x_k \\ y_1 \ldots y_k \end{smallmatrix}\right)$ transformieren, so dass $\alpha = (i_1, \ldots, i_n)$ genau dann eine Lösung für $f(w)$ ist, wenn das zugehörige *Lösungswort* $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$ eine Ableitung von w in G kodiert. Dies erreichen wir, indem wir $f(w)$ aus folgenden Wortpaaren bilden:

1. $(\langle, \langle \mid S)$, „Startpaar“
2. für jede Regel $l \to r$ in P: (l, r), „Ableitungspaare“
3. für alle $a \in V \cup \Sigma \cup \{|\}$: (a, a), „Kopierpaare“
4. sowie das Paar $(w \mid \rangle, \rangle)$ „Abschlusspaar“

Nun lässt sich leicht aus einer Ableitung $S = \alpha_0 \Rightarrow \cdots \Rightarrow \alpha_m = w$ von w in G eine MPCP-Lösung mit dem Lösungswort

$$\langle |\alpha_0|\alpha_1|\ldots|\alpha_m|\rangle$$

angeben. Zudem lässt sich auch umgekehrt aus jeder MPCP-Lösung eine Ableitung von w in G gewinnen, womit

$$w \in L(M) \Leftrightarrow f(w) \in \mathrm{MPCP}_\Gamma$$

gezeigt ist. ■

Beispiel 153. *Betrachte die Grammatik* $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ *und die Ableitung*

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

Die MPCP-Instanz $f(aabb)$ *enthält dann die acht Wortpaare*

$$f(aabb) = \left(\begin{array}{cccccc|c} \langle & S & S & S & a & b & aabb|\rangle \\ \langle|S & aSbS & \varepsilon & S & a & b & |\rangle \end{array} \right).$$

Obiger Ableitung entspricht dann folgendes Lösungswort für $f(aabb)$*:*

$$\langle|S|aSbS|aaSbSbS|aaSbbS|aabbS|aabb|\rangle$$
$$\langle|S|aSbS|aaSbSbS|aaSbbS|aabbS|aabb|\rangle$$

Das kürzeste MPCP-Lösungswort für $f(aabb)$ *ist*

$$\langle|S|aSbS|aaSbSb|aabb|\rangle$$
$$\langle|S|aSbS|aaSbSb|aabb|\rangle$$

Dieses entspricht der „parallelisierten“ Ableitung

$$\underline{S} \Rightarrow a\underline{S}b\underline{S} \Rightarrow^2 aa\underline{S}b\underline{S}b \Rightarrow^2 aabb$$

◁

Wir benutzen nun die Unentscheidbarkeit des Postschen Korrespondenzproblems, um eine Reihe von weiteren Unentscheidbarkeitsresultaten zu erhalten. 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 $L(G_1) \cap L(G_2) \neq \varnothing$?

Satz 154. *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 $s = (x_1, \ldots, x_k)$ von Strings $x_i \in \{a, b\}^*$ die Sprache

$$L_s = \{i_n \ldots i_1 \# x_{i_1} \cdots x_{i_n} \mid 1 \leq n, 1 \leq i_1, \ldots, i_n \leq k\}.$$

L_s wird von der Grammatik $G_s = (\{A\}, \{1, \ldots, k, \#, a, b\}, P_s, A)$ erzeugt mit

$$P_s\colon A \to 1Ax_1, \ldots, kAx_k, 1\#x_1, \ldots, k\#x_k$$

Zu einer PCP-Instanz $I = \binom{x_1 \ldots x_k}{y_1 \ldots y_k}$ bilden wir das Paar (G_s, G_t), wobei $s = (x_1, \ldots, x_k)$ und $t = (y_1, \ldots, y_k)$ ist. Dann ist $L(G_s) \cap L(G_t)$ die Sprache

$$\{i_n \ldots i_1 \# x_{i_1} \ldots x_{i_n} \mid 1 \leq n,\ x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}\}.$$

Folglich ist $\alpha = (i_1, \ldots, i_n)$ genau dann eine Lösung für I, wenn $i_n \ldots i_1 x_{i_1} \ldots x_{i_n} \in L(G_s) \cap L(G_t)$ ist. Also vermittelt $f : I \mapsto (G_s, G_t)$ eine Reduktion von PCP auf das Schnittproblem für CFL. ■

Beispiel 155. *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:*

$$\begin{aligned} P_s\colon A &\to 1Aa,\ 2Aaab,\ 3Aabbaa, \\ &\quad 1\#a,\ 2\#aab,\ 3\#abbaa, \\ P_t\colon A &\to 1Aaabba,\ 2Aababb,\ 3Aaa, \\ &\quad 1\#aabba,\ 2\#ababb,\ 3\#aa. \end{aligned}$$

Der PCP-Lösung $\alpha = (1,3,2,3)$ *entspricht dann das Wort*

$$\begin{aligned} 3231\#x_1x_3x_2x_3 &= 3231\#aabbaaaababbaa \\ &= 3231\#aabbaaaababbaa = 3231\#y_1y_3y_2y_3 \end{aligned}$$

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 156.

(i) Das Schnittproblem für DPDAs ist RE*-vollständig.*

(ii) Das Inklusionsproblem für DPDAs ist co-RE*-vollständig.*

Korollar 156 wird in den Übungen bewiesen. Die Idee dabei ist, die kontextfreie Grammatik G_s im Beweis von Satz 154 in DPDAs M_s und $\overline{M}_s$ zu überführen mit $L(M_s) = L(G_s)$ und $L(\overline{M}_s) = \overline{L(G_s)}$.

Schließlich ergeben sich für CFL noch folgende Unentscheidbarkeitsresultate.

Korollar 157. *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

$$I \notin \mathrm{PCP} \Leftrightarrow L_s \cap L_t = \varnothing \Leftrightarrow \overline{L}_s \cup \overline{L}_t = \Sigma^*,$$

wobei L_s und L_t die im Beweis von Satz 154 definierten Sprachen sind. Diese sind sogar in DCFL und in den Übungen wird gezeigt, dass sich DPDAs (und damit auch kontextfreie Grammatiken) für $\overline{L}_s$ und $\overline{L}_t$ aus I berechnen lassen. Daher vermittelt die Funktion $f : I \mapsto G$, wobei G eine kontextfreie Grammatik mit

$$L(G) = \overline{L}_s \cup \overline{L}_t$$

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 $f : \binom{x_1 \ldots x_k}{y_1 \ldots y_k} \mapsto G$ mit

$$G = (\{S, A, B\}, \{1, \ldots, k, \#, a, b\}, P \cup \{S \to A, S \to B\}, S)$$

und den Regeln

$$P\colon A \to 1Ax_1, \ldots, kAx_k, 1\#x_1, \ldots, k\#x_k,$$
$$B \to 1By_1, \ldots, kBy_k, 1\#y_1, \ldots, k\#y_k.$$

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

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

Wie wir im Beweis der Unentscheidbarkeit des Schnittproblems für CFL gesehen haben, ist dies genau dann der Fall, wenn die PCP-Instanz $I = \binom{x_1 \ldots x_k}{y_1 \ldots y_k}$ eine PCP-Lösung hat.

■

Als weitere Folgerung erhalten wir die Unentscheidbarkeit des Leerheitsproblems für DLBAs.

Leerheitsproblem für DLBAs

Gegeben: Ein DLBA M.
Gefragt: Ist $L(M) = \varnothing$?

Satz 158. *Das Leerheitsproblem für DLBAs ist* co-RE*-vollständig.*

Beweis. Wir reduzieren $\overline{\text{PCP}}$ auf das Leerheitsproblem für DLBAs. Hierzu transformieren wir eine PCP-Instanz $I = \binom{x_1 \ldots x_k}{y_1 \ldots y_k}$ in einen DLBA M für die Sprache $L_s \cap L_t$, wobei $s = (x_1, \ldots, x_k)$ und $t = (y_1, \ldots, y_k)$ ist (siehe Übungen). Dann gilt $I \notin \text{PCP} \Leftrightarrow L(M) = \varnothing$.

■

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 159.

(i) Das Leerheitsproblem für kfr. Grammatiken ist entscheidbar.

(ii) Das Wortproblem für kontextsensitive Grammatiken ist entscheidbar.

Beweis. Siehe Übungen. ■

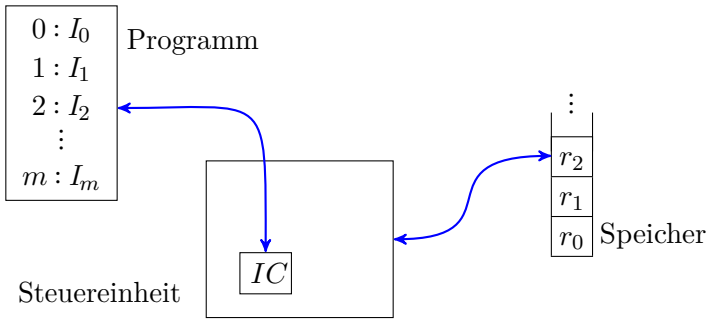
Die folgende Tabelle gibt an, welche Probleme für Sprachen in den verschiedenen Stufen der Chomsky-Hierarchie entscheidbar sind.

	Wort-problem $x \in L$?	Leerheits-problem $L = \varnothing$?	Aus-schöpfung $L = \Sigma^*$?	Äquivalenz-problem $L_1 = L_2$?	Inklusions-problem $L_1 \subseteq L_2$	Schnitt-problem $L_1 \cap L_2 \neq \varnothing$?
REG	ja	ja	ja	ja	ja	ja
DCFL	ja	ja	ja	ja[a]	nein	nein
CFL	ja	ja	nein	nein	nein	nein
DCSL	ja	nein	nein	nein	nein	nein
CSL	ja	nein	nein	nein	nein	nein
RE	nein	nein	nein	nein	nein	nein

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

5.4 GOTO-, WHILE- und LOOP-Programme

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 $P = (I_0, \ldots, I_m)$ 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 $r_j + c$
$r_i := r_j \dot{-} c$	setzt Register r_i auf $\max(0, r_j - c)$
GOTO j	setzt den Befehlszähler IC auf j
IF $r_i = c$ **THEN GOTO** j	setzt IC auf j, falls $r_i = c$ ist
HALT	beendet die Programmausführung

Falls nichts anderes angegeben ist, wird zudem IC auf den Wert $IC+1$ gesetzt.

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt ***GOTO-berechenbar***, falls es ein GOTO-Programm $P = (I_0, \ldots, I_m)$ mit folgender Eigenschaft gibt:

Wird P auf einer RAM mit den Werten $r_i = n_i$ für $i = 1, \ldots, k$, sowie $IC = 0$ und $r_i = 0$ für $i = 0, k+1, k+2, \ldots$ gestartet, so gilt:

- P hält genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
- falls P hält, hat r_0 nach Beendigung von P den Wert $f(n_1, \ldots, n_k)$.

Beispiel 160. *Folgendes GOTO-Programm berechnet die Funktion* $f(x, y) = xy$*:*

```
0  IF r1 = 0 THEN GOTO 4
1  r1 := r1 ∸ 1
2  r0 := r0 + r2
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  r3 := r2
6  IF r3 = 0 THEN GOTO 3
7  r3 := r3 ∸ 1
8  r0 := r0 + 1
9  GOTO 6
```

◁

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 $x_i := x_j \dot{-} c$ 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 so oft ausgeführt, wie der Wert von x_i zu Beginn der Schleife angibt.

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt ***WHILE-berechenbar***, falls es ein WHILE-Programm P mit folgender Eigenschaft gibt: Wird P mit den Werten $x_i = n_i$ für $i = 1, \ldots, k$ gestartet, so gilt:

- P hält genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
- falls P hält, hat x_0 nach Beendigung von P den Wert $f(n_1, \ldots, n_k)$.

Die ***LOOP-Berechenbarkeit*** von f ist entsprechend definiert.

Beispiel 161. *Die Funktion $f(x_1, x_2) = x_1 x_2$ wird von dem WHILE-Programm*

```
WHILE x1 ≠ 0 DO
  x3 := x2;
  WHILE x3 ≠ 0 DO
    x0 := x0 + 1;
    x3 := x3 ∸ 1
  END
  x1 := x1 ∸ 1
END
```

sowie von folgendem LOOP-Programm berechnet:

```
LOOP x1 DO
  LOOP x2 DO
    x0 := x0 + 1
  END
END
```

◁

Um mit einem GOTO-Programm auch Wortfunktionen berechnen zu können, müssen wir Wörter numerisch repräsentieren.

Sei $\Sigma = \{a_0, \ldots, a_{m-1}\}$ ein Alphabet. Dann können wir jedes Wort $x = a_{i_1} \ldots a_{i_n} \in \Sigma^*$ durch eine natürliche Zahl $num_\Sigma(x)$ kodieren:

$$num_\Sigma(x) = \sum_{j=0}^{n-1} m^j + \sum_{j=1}^{n} i_j m^{n-j} = \begin{cases} n, & m = 1, \\ \frac{m^n - 1}{m-1} + (i_1 \ldots i_n)_m, & m > 1. \end{cases}$$

Da die Abbildung $num_\Sigma : \Sigma^* \to \mathbb{N}$ bijektiv ist, können wir umgekehrt jede natürliche Zahl n durch das Wort $str_\Sigma(n) = num_\Sigma^{-1}(n)$ kodieren.

Für das Alphabet $\Sigma = \{a, b, c\}$ erhalten wir beispielsweise folgende Kodierung:

w	ε	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc	aaa	…
$num_\Sigma(w)$	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 num und str anstelle von num_Σ und str_Σ:

n	0	1	2	3	4	5	6	7	8	9	10	11	…
$str(n)$	ε	0	1	00	01	10	11	000	001	010	011	100	…

Zudem erweitern wir die Kodierungsfunktion $str : \mathbb{N} \to \{0, 1\}$ zu einer Kodierungsfunktion $str_k : \mathbb{N}^k \to \{0, 1, \#\}$ wie folgt:

$$str_k(n_1, \ldots, n_k) = str(n_1) \# \ldots \# str(n_k).$$

Nun können wir eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ durch folgende partielle Wortfunktion $\hat{f} : \{0, 1, \#\}^* \to \{0, 1, \#\}^* \cup \{\uparrow\}$ repräsentieren:

$$\hat{f}(w) = \begin{cases} str(n), & w = str_k(n_1, \ldots, n_k) \text{ und } f(n_1, \ldots, n_k) = n \in \mathbb{N}, \\ \uparrow, & \text{sonst.} \end{cases}$$

Wir nennen $\hat{f}$ die *String-Repräsentation* von f und f die *numerische Repräsentation* von $\hat{f}$.

Beispiel 162. *Die Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ mit $f(n_1, n_2) = n_1 n_2$ wird durch folgende Wortfunktion repräsentiert:*

$$\hat{f}(w) = \begin{cases} str(n_1 n_2), & w = str_2(n_1, n_2), \\ \uparrow, & sonst. \end{cases}$$

Folgende Tabelle enthält die Werte $\hat{f}(w)$ für einige Wörter $w \in \{0, 1, \#\}^*$. Falls w nicht genau ein $\#$-Zeichen enthält, gilt $\hat{f}(w) = \uparrow$.

w	ε	0	1	#	00	01	0#	10	11	1#	#0	#1	##
(n_1, n_2)	-	-	-	$(0,0)$	-	-	$(1,0)$	-	-	$(2,0)$	$(0,1)$	$(0,2)$	-
$n_1 n_2$	-	-	-	0	-	-	0	-	-	0	0	0	-
$\hat{f}(w)$	$\uparrow$	$\uparrow$	$\uparrow$	ε	$\uparrow$	$\uparrow$	ε	$\uparrow$	$\uparrow$	ε	ε	ε	$\uparrow$

000	001	00#	010	011	01#	0#0	0#1	0##	100	101	10#	⋯
-	-	$(3,0)$	-	-	$(4,0)$	$(1,1)$	$(1,2)$	-	-	-	$(5,0)$	⋯
-	-	0	-	-	0	1	2	-	-	-	0	⋯
$\uparrow$	$\uparrow$	ε	$\uparrow$	$\uparrow$	ε	0	1	$\uparrow$	$\uparrow$	$\uparrow$	ε	⋯

◁

Satz 163. *Die String-Repräsentation $\hat{f}$ einer partiellen Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ ist genau dann Turing-berechenbar, wenn f GOTO-berechenbar ist.*

Beweis. Wir zeigen zuerst die Simulation eines GOTO-Programms durch eine DTM. Sei P ein GOTO-Programm, das eine Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ auf einer RAM R berechnet. Dann existiert eine Zahl m, so dass P nur Register r_i mit $i \leq m$ benutzt. Daher lässt sich eine Konfiguration von R durch Angabe der Inhalte des Befehlszählers IC und der Register $r_0, \ldots, r_m$ beschreiben. Wir konstruieren eine $(m+2)$-DTM M, die

- den Inhalt von IC in ihrem Zustand,
- die Registerwerte $r_1, \ldots, r_m$ auf den Bändern $1, \ldots, m$ und
- den Wert von r_0 auf dem Ausgabeband $m+2$ speichert.

Ein Registerwert r_i wird hierbei in der Form $str(r_i)$ gespeichert. Band $m+1$ wird zur Ausführung von Hilfsberechnungen benutzt.

Die Aufgabe von M ist es, bei Eingabe $w = str_k(n_1, \ldots, n_k)$ das Wort $str(f(n_1, \ldots, n_k))$ auszugeben, wenn $w = str_k(n_1, \ldots, n_k)$ für ein Tupel $(n_1, \ldots, n_k) \in dom(f)$ ist, und andernfalls nicht zu halten.

Zuerst überprüft M, ob in w das $\#$-Zeichen genau $(k-1)$-mal vorkommt. Wenn nicht, geht M in eine Unendlichschleife. Andernfalls kopiert M die Teilwörter $str(n_i)$ für $i = 2, \ldots, k$ 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 $1, \ldots, m$ und auf Band $m+2$ die der Startkonfiguration von R bei Eingabe $(n_1, \ldots, n_k)$ 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 auf Band $m+2$ befindliche Wort $str(r_0) = str(f(n_1, \ldots, n_k)) = \hat{f}(w)$ aus.

Nun betrachten wir die Simulation einer DTM durch ein GOTO-Programm. Sei $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ eine partielle Funktion und sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine DTM mit Eingabealphabet $\Sigma = \{0, 1, \#\}$, die die zugehörige Wortfunktion $\hat{f} : \Sigma^* \to \Sigma^* \cup \{\uparrow\}$ berechnet. M gibt also bei Eingabe w das Wort $str(f(n_1, \ldots, n_k))$ aus, falls w die Form $w = str_k(n_1, \ldots, n_k)$ hat und $f(n_1, \ldots, n_k)$ definiert ist, und hält andernfalls nicht.

Wir konstruieren ein GOTO-Programm P, das bei Eingabe $(n_1, \ldots, n_k)$ die DTM M bei Eingabe $w = str_k(n_1, \ldots, n_k)$ simuliert und im Fall, dass $M(w)$ hält, den Wert $num(M(w)) = f(n_1, \ldots, n_k)$ berechnet. Wir können annehmen, dass M eine 1-DTM ist. Sei

$Z = \{q_0, \ldots, q_r\}$ und sei $\Gamma = \{a_0, \ldots, a_{m-1}\}$, wobei wir annehmen, dass $a_0 = \sqcup$, $a_1 = 0$, $a_2 = 1$ und $a_3 = \#$ ist.

Eine Konfiguration $K = uq_iv$ von M mit $u = a_{i_1} \ldots a_{i_s}$ und $v = a_{j_1} \ldots a_{j_t}$ wird wie folgt in den Registern r_0, r_1, r_2 gespeichert:

- $r_0 = (i_1 \ldots i_n)_m$,
- $r_1 = i$,
- $r_2 = (j_{n'} \ldots j_1)_m$.

P besteht aus 3 Programmteilen $P = P_1, P_2, P_3$. P_1 berechnet in Register r_2 die Zahl $(j_n \ldots j_1)_m$, wobei $(j_1, \ldots, j_n)$ die Indexfolge der Zeichen von $str_k(n_1, \ldots, n_k) = a_{j_1} \ldots a_{j_n}$ 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 $w = str_k(n_1, \ldots, n_k)$ 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 $r_3 \coloneqq r_2$ MOD m
IF $r_1 = 0 \wedge r_3 = 0$ **THEN GOTO** $M_{0,0}$
⋮
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. Im Fall $\delta(q_i, a_j) = \varnothing$ erfolgt ein Sprung an den Beginn von P_3. Dagegen werden im Fall $\delta(q_i, a_j) = \{(q_{i'}, a_{j'}, L)\}$ folgende Befehle ausgeführt:

$M_{i,j}$ $r_1 \coloneqq i'$
$r_2 \coloneqq r_2$ DIV m
$r_2 \coloneqq r_2 m + j'$
$r_2 \coloneqq r_2 m + (r_0$ MOD $m)$
$r_0 \coloneqq r_0$ DIV m
GOTO M_2

Die übrigen Fälle sind ähnlich. Die hierbei benutzten Makrobefehle

$r_3 \coloneqq r_2$ MOD m, $r_2 \coloneqq r_2$ DIV m etc. können leicht durch entsprechende GOTO-Programmstücke ersetzt werden (siehe Übungen).
Schließlich transformiert P_3 noch den Inhalt $(j_1 \ldots j_n)_m$ von Register r_0 in die Zahl $num(a_{j_1} \ldots a_{j_n})$ und hält. ■

Als nächstes beweisen wir die Äquivalenz von WHILE- und GOTO-Berechenbarkeit.

Satz 164. *Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ 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 ⋮

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 $P = (I_0, \ldots, I_m)$ 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:

$x_z \coloneqq 0$;
WHILE $x_z \neq m + 1$ **DO**
IF $x_z = 0$ **THEN** P'_0 **END**;
⋮

IF $x_z = m$ **THEN** P'_m **END**
END

Dabei ist P'_ℓ abhängig vom Befehl I_ℓ folgendes WHILE-Programm:

I_ℓ	P'_ℓ
$r_i := r_j \dot{-} c$	$x_i := x_j \dot{-} c;\ x_z := x_z + 1$
GOTO j	$x_z := j$
IF $r_i = c$ **THEN GOTO** j	**IF** $x_i = c$ **THEN** $x_z := j$ **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*** $a(x,y) : \mathbb{N}^2 \to \mathbb{N}$, 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 Komplexität von algorithmischen Problemen

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 165.

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

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

Die Zeitschranke $t(n)$ beschränkt also die Laufzeit bei allen Eingaben der Länge n (worst-case Komplexität).

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

Definition 166.

a) Die in deterministischer Zeit $t(n)$ entscheidbaren Sprachen bilden die Sprachklasse

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

b) Die in nichtdeterministischer Zeit $t(n)$ entscheidbaren Sprachen bilden die Sprachklasse

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

c) Die in deterministischer Zeit $t(n)$ berechenbaren Funktionen bilden die Funktionenklasse

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

Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

$$\mathsf{LINTIME} = \bigcup_{c\geq 1} \mathsf{DTIME}(cn+c) \qquad \text{„Linearzeit“}$$

$$\mathsf{P} = \bigcup_{c\geq 1} \mathsf{DTIME}(n^c+c) \qquad \text{„Polynomialzeit“}$$

$$\mathsf{E} = \bigcup_{c\geq 1} \mathsf{DTIME}(2^{cn+c}) \qquad \text{„Lineare Exponentialzeit“}$$

$$\mathsf{EXP} = \bigcup_{c\geq 1} \mathsf{DTIME}(2^{n^c+c}) \qquad \text{„Exponentialzeit“}$$

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

Für eine Funktionenklasse $\mathcal{F}$ sei $\mathsf{DTIME}(\mathcal{F}) = \bigcup_{t\in\mathcal{F}} \mathsf{DTIME}(t(n))$ (die Klassen $\mathsf{NTIME}(\mathcal{F})$ und $\mathsf{FTIME}(\mathcal{F})$ seien analog definiert).

Asymptotische Laufzeit und Landau-Notation

Definition 167. *Seien f und g Funktionen von $\mathbb{N}$ nach $\mathbb{R}^+$. Wir schreiben $f(n) = \mathcal{O}(g(n))$, 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 $f(n) = \mathcal{O}(g(n))$ ist, dass f „nicht wesentlich schneller“ als g wächst. Formal bezeichnet der Term $\mathcal{O}(g(n))$ die Klasse aller Funktionen f, die obige Bedingung erfüllen. Die Gleichung $f(n) = \mathcal{O}(g(n))$ drückt also in Wahrheit eine Element-Beziehung $f \in \mathcal{O}(g(n))$ aus. O-Terme können auch auf der linken Seite vorkommen. In diesem Fall wird eine Inklusionsbeziehung ausgedrückt. So steht $n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$ für die Aussage $\{n^2 + f \mid f \in \mathcal{O}(n)\} \subseteq \mathcal{O}(n^2)$.

Beispiel 168.

- $7\log(n) + n^3 = \mathcal{O}(n^3)$ *ist richtig.*
- $7\log(n)n^3 = \mathcal{O}(n^3)$ *ist falsch.*
- $2^{n+\mathcal{O}(1)} = \mathcal{O}(2^n)$ *ist richtig.*
- $2^{\mathcal{O}(n)} = \mathcal{O}(2^n)$ *ist falsch (siehe Übungen).* ◁

Unter Benutzung der O-Notation lassen sich die wichtigsten Komplexitätsklassen wie folgt definieren: $\mathsf{L} = \mathsf{DSPACE}(O(\log n))$, $\mathsf{LINTIME} = \mathsf{DTIME}(O(n))$, $\mathsf{P} = \mathsf{DTIME}(n^{\mathcal{O}(1)})$, $\mathsf{E} = \mathsf{DTIME}(2^{\mathcal{O}(n)})$ und $\mathsf{EXP} = \mathsf{DTIME}(2^{n^{\mathcal{O}(1)}})$ etc.

6.2 NP-Vollständigkeit

Wie wir im letzten Kapitel gesehen haben (siehe Satz 129), sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden. Die Frage, wieviel Zeit eine DTM zur Simulation einer NTM benötigt, ist eines der wichtigsten offenen Probleme der Informatik. Wegen $\mathsf{NTIME}(t) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(t)})$ 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 169. *Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen.*

a) *A ist auf B* ***in Polynomialzeit reduzierbar*** *(**$A \leq^p B$**), falls eine Funktion $f : \Sigma^* \to \Gamma^*$ in* FP *existiert mit*

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

b) *A heißt* ***$\leq^p$-hart*** *für eine Sprachklasse $\mathcal{C}$ (kurz:* ***$\mathcal{C}$-hart*** *oder* ***$\mathcal{C}$-schwer****), falls gilt:*

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

c) *Eine $\mathcal{C}$-harte Sprache, die zu $\mathcal{C}$ gehört, heißt* ***$\mathcal{C}$-vollständig****. Die Klasse aller* NP*-vollständigen Sprachen bezeichnen wir mit* $\mathbf{NPC}$.

Aus $A \leq^p B$ folgt offenbar $A \leq B$ und wie die Relation $\leq$ ist auch $\leq^p$ reflexiv und transitiv (s. Übungen). In diesem Kapitel verlangen wir also von einer $\mathcal{C}$-vollständigen Sprache A, dass jede Sprache $L \in \mathcal{C}$ auf A in Polynomialzeit reduzierbar ist. Es ist leicht zu sehen, dass alle im letzten Kapitel als RE-vollständig nachgewiesenen Sprachen (wie z.B. $\mathrm{K}, \mathrm{H}, \mathrm{H}_0, \mathrm{PCP}$ etc.) sogar $\leq^p$-vollständig für RE sind.

Satz 170. *Die Klassen* P *und* NP *sind unter $\leq^p$ abgeschlossen.*

Beweis. Sei $B \in \mathsf{P}$ und gelte $A \leq^p B$ mittels einer Funktion $f \in \mathsf{FP}$. Seien M und T DTMs mit $L(M) = B$ und $T(x) = f(x)$. Weiter seien p und q polynomielle Zeitschranken für M und T. Betrachte die DTM M', die bei Eingabe x zuerst T simuliert, um $f(x)$ zu berechnen, und danach M bei Eingabe $f(x)$ simuliert. Dann gilt

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow f(x) \in L(M) \Leftrightarrow x \in L(M').$$

Also ist $L(M') = A$ und wegen

$$time_{M'}(x) \leq time_T(x) + time_M(f(x)) \leq q(|x|) + p(q(|x|))$$

ist M' polynomiell zeitbeschränkt und somit A in P. Den Abschluss von NP unter $\leq^p$ zeigt man vollkommen analog. ∎

Satz 171.

(i) $A \leq^p B$ *und* A *ist* NP*-schwer* $\Rightarrow$ B *ist* NP*-schwer.*

(ii) $A \leq^p B$*,* A *ist* NP*-schwer und* $B \in \mathsf{NP}$ $\Rightarrow$ $B \in \mathsf{NPC}$.

(iii) $\mathsf{NPC} \cap \mathsf{P} \neq \varnothing$ $\Rightarrow$ $\mathsf{P} = \mathsf{NP}$.

Beweis.

(i) Sei $L \in \mathsf{NP}$ beliebig. Da A NP-schwer ist, folgt $L \leq^p A$. Da zudem $A \leq^p B$ gilt und $\leq^p$ transitiv ist, folgt $L \leq^p B$.

(ii) Klar, da mit (i) folgt, dass B NP-schwer und B nach Voraussetzung in NP ist.

(iii) Sei $A \in \mathsf{P}$ eine NP-vollständige Sprache und sei $L \in \mathsf{NP}$ beliebig. Dann folgt $L \leq^p A$ und da P unter $\leq^p$ abgeschlossen ist, folgt $L \in \mathsf{P}$. ∎

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 172. *Eine* NTM M *heißt* ***Offline*-NTM** *(oder* NTM *mit* ***Eingabeband****), falls für jede von M bei Eingabe x erreichbare Konfiguration*

$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$$

gilt, dass $u_1 a_1 v_1$ ein Teilwort von $\sqcup x \sqcup$ ist.

Definition 173. *Der* ***Platzverbrauch*** *einer Offline-*NTM M *bei Eingabe x ist definiert als*

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

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

$$space_M(x) \leq s(|x|)\ und\ time_M(x) < \infty.$$

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

Definition 174. *Die auf deterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse*

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

Die auf nichtdeterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse

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

Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$\mathbf{L}$	$= \mathsf{DSPACE}(\mathcal{O}(\log n))$	„Logarithmischer Platz“
$\mathbf{LINSPACE}$	$= \mathsf{DSPACE}(\mathcal{O}(n))$	„Linearer Platz“
$\mathbf{PSPACE}$	$= \mathsf{DSPACE}(n^{\mathcal{O}(1)})$	„Polynomieller Platz“

Die nichtdeterministischen Klassen **NL**, **NLINSPACE** und **NPSPACE** sind analog definiert.

Zwischen den verschiedenen Zeit- und Platzklassen bestehen die folgenden elementaren Inklusionsbeziehungen. Für jede Funktion $t(n) \geq n+2$ gilt

$$\mathsf{DTIME}(t) \subseteq \mathsf{NTIME}(t) \subseteq \mathsf{DSPACE}(t)$$

und für jede Funktion $s(n) \geq \log n$ gilt

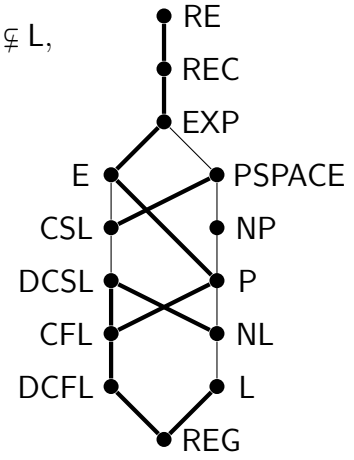
$$\mathsf{DSPACE}(s) \subseteq \mathsf{NSPACE}(s) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(s)}) \text{ und}$$
$$\mathsf{NSPACE}(s) \subseteq \mathsf{DSPACE}(s^2).$$

Als unmittelbare Konsequenz hiervon erhalten wir

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}.$$

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

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



wobei f alle berechenbaren (oder äquivalent: alle) Funktionen $f : \mathbb{N} \to \mathbb{N}$ durchläuft.

Die Klasse L ist nicht in CFL enthalten, da beispielsweise die Sprache $\{a^n b^n c^n \mid n \geq 0\}$ in logarithmischem Platz (und linearer Zeit) entscheidbar ist. Ob P in CSL enthalten ist, ist nicht bekannt. Auch nicht ob

DCSL ⊆ P gilt. Man kann jedoch zeigen, dass CSL ≠ P ≠ 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.

6.4 Aussagenlogische Erfüllbarkeitsprobleme

Definition 175.

a) Die Menge der ***booleschen*** *(oder* ***aussagenlogischen****)* ***Formeln*** *über den Variablen $x_1, \dots, x_n$ ist induktiv wie folgt definiert:*

- *Die Konstanten 0 und 1 sind boolesche Formeln.*
- *Jede Variable x_i ist eine boolesche Formel.*
- *Mit G und H sind auch die* ***Negation $\neg G$*** *von G und die* ***Konjunktion $(G \wedge H)$*** *von G und H boolesche Formeln.*

b) Eine ***Belegung*** *von $x_1, \dots, x_n$ 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	0	1	x_i	$\neg G$	$(G \wedge H)$
$F(a)$	0	1	a_i	$1 - G(a)$	$G(a)H(a)$

c) Durch eine boolesche Formel F wird also eine n-stellige ***boolesche Funktion*** *$F : \{0,1\}^n \to \{0,1\}$ definiert, die wir ebenfalls mit F bezeichnen.*

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

e) Gilt sogar für alle Belegungen a, dass $F(a) = 1$ ist, so heißt F ***Tautologie****.*

Beispiel 176 (Erfüllbarkeitstest mittels Wahrheitswerttabelle)**.** *Da die Formel $F = ((x_1 \vee x_2) \to (\neg x_2 \wedge x_3))$ für die Belegungen $a \in \{000, 001, 101\}$ den Wert $F(a) = 1$ annimmt, ist sie erfüllbar:*

a	$(x_1 \vee x_2)$	$(\neg x_2 \wedge x_3)$	$((x_1 \vee x_2) \to (\neg x_2 \wedge x_3))$
000	0	0	1
001	0	1	1
010	1	0	0
011	1	0	0
100	1	0	0
101	1	1	1
110	1	0	0
111	1	0	0

◁

Notation. *Wir verwenden die* ***Disjunktion $(G \vee H)$*** *und die* ***Implikation $(G \to H)$*** *als Abkürzungen für die Formeln $\neg(\neg G \wedge \neg H)$ bzw. $(\neg G \vee H)$. Zudem verwenden wir die* ***Äquivalenz $(G \leftrightarrow H)$*** *als Abkürzung für $(G \to H) \wedge (H \to G)$.*

Um Klammern zu sparen, vereinbaren wir folgende Präzedenzregeln:

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

Beispiel 177. *Die Formel*

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

nimmt unter einer Belegung $a = a_1 \dots a_n$ 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 $x_1, \ldots, x_n$.
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 178 (Cook, Karp, Levin)**.** SAT *ist* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass SAT $\in$ NP ist, da eine NTM zunächst eine Belegung a für eine gegebene booleschen Formel F nichtdeterministisch raten und dann in Polynomialzeit testen kann, ob $F(a) = 1$ ist (*guess and verify* Strategie).

Es bleibt zu zeigen, dass SAT NP-hart ist. Sei L eine beliebige NP-Sprache und sei $M = (Z, \Sigma, \Gamma, \delta, q_0)$ eine durch ein Polynom p zeitbeschränkte k-NTM mit $L(M) = L$. Da sich eine $t(n)$-zeitbeschränkte k-NTM in Zeit $t^2(n)$ durch eine 1-NTM simulieren lässt, können wir $k = 1$ annehmen. Unsere Aufgabe besteht nun darin, in Polynomialzeit zu einer gegebenen Eingabe $w = w_1 \ldots w_n$ eine Formel F_w zu konstruieren, die genau dann erfüllbar ist, wenn $w \in L$ ist,

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

Wir können o.B.d.A. annehmen, dass $Z = \{q_0, \ldots, q_m\}$, $E = \{q_m\}$ und $\Gamma = \{a_1, \ldots, a_l\}$ ist. Zudem können wir annehmen, dass δ für jedes Zeichen $a \in \Gamma$ die Anweisung $q_m a \to q_m a N$ enthält.

Die Idee besteht nun darin, die Formel F_w so zu konstruieren, dass sie unter einer Belegung a genau dann wahr wird, wenn a eine akzeptierende Rechnung von $M(w)$ beschreibt. Hierzu bilden wir F_w über den Variablen

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

die für folgende Aussagen stehen:

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

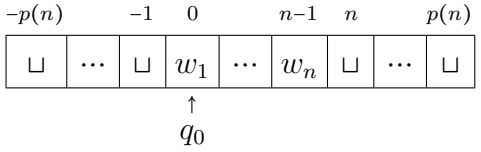
Konkret sei nun $F_w = R \wedge S \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$. Dabei stellt die Formel $R = \bigwedge_{t=0}^{p(n)} R_t$ (Randbedingungen) sicher, dass wir jeder erfüllenden Belegung eindeutig eine Folge von Konfigurationen $K_0, \ldots, K_{p(n)}$ zuordnen können:

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

Die Teilformel R_t sorgt also dafür, dass zum Zeitpunkt t

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

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



vorliegt:

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

Die Formel $\ddot{U}_1$ sorgt dafür, dass der Inhalt von nicht besuchten Feldern beim Übergang von K_t zu K_{t+1} unverändert bleibt:

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

Die Formel $\ddot{U}_2$ achtet darauf, dass sich bei jedem Übergang der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in δ verändern:

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

wobei

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

ist. Schließlich überprüft E, ob M zur Zeit $p(n)$ den Endzustand q_m erreicht hat:

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

Da der Aufbau der Formel $f(w) = F_w$ einem einfachen Bildungsgesetz folgt und ihre Länge polynomiell in n ist, folgt $f \in \mathsf{FP}$. Es ist klar, dass F_w im Fall $w \in L(M)$ erfüllbar ist, indem wir die Variablen von F_w gemäß einer akz. Rechnung von $M(w)$ belegen. Umgekehrt führt eine Belegung a mit $F_w(a) = 1$ wegen $R(a) = 1$ eindeutig auf eine Konfigurationenfolge $K_0, \ldots, K_{p(n)}$, so dass gilt:

- K_0 ist Startkonfiguration von $M(w)$ (wegen $S(a) = 1$),
- $K_i \vdash K_{i+1}$ für $i = 0, \ldots, p(n)-1$ (wegen $\ddot{U}_1(a) = \ddot{U}_2(a) = 1$),
- M nimmt spätestensin der Konfiguration $K_{p(n)}$ den Endzustand q_m an (wegen $E(a) = 1$).

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

Korollar 179. $\textsc{Sat} \in \mathsf{P} \Leftrightarrow \mathsf{P} = \mathsf{NP}$.

Gelingt es also, einen Polynomialzeit-Algorithmus für SAT zu finden, so lässt sich daraus leicht ein effizienter Algorithmus für jedes NP-Problem ableiten. Als nächstes betrachten wir das Erfüllbarkeitsproblem für boolesche Schaltkreise.

Definition 180.

a) Ein ***boolescher Schaltkreis*** *über den Variablen $x_1, \ldots, x_n$ ist eine Folge $S = (g_1, \ldots, g_m)$ von* ***Gattern***

$$g_l \in \{0, 1, x_1, \ldots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\}$$

mit $1 \le j, k < l$.

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

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

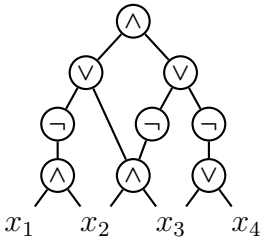
c) S berechnet die boolesche Funktion $S(a) = g_m(a)$.

d) S heißt ***erfüllbar****, wenn eine Eingabe $a \in \{0,1\}^n$ mit $S(a) = 1$ existiert.*

Beispiel 181. *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 182.

- *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 genau den booleschen Schaltkreisen $S = (g_1, \ldots, g_m)$, bei denen jedes Gatter g_i, $1 \leq i \leq m-1$, Fanout 1 hat.*

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 Sat $\leq^p$ CirSat.

Korollar 183. CirSat *ist* NP*-vollständig.*

Bemerkung 184. *Da* Sat NP*-vollständig ist, ist* CirSat *in Polynomialzeit auf* Sat *reduzierbar. Dies bedeutet, dass sich jeder Schaltkreis S in Polynomialzeit in eine* erfüllbarkeitsäquivalente *Formel F_S überführen lässt. F_S und S müssen aber nicht logisch äquivalent sein.*

CirSat ist sogar auf eine ganz spezielle Sat-Variante reduzierbar.

Definition 185.

*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 $C = \{l_1, \ldots, l_k\}$ ihrer Literale und KNF-Formeln als Menge $F = \{C_1, \ldots, C_m\}$ 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?

Folgende Variante von 3-Sat ist für den Nachweis weiterer NP-Vollständigkeitsresultate sehr nützlich.

Not-All-Equal-SAT (NaeSat)**:**

Gegeben: Eine Formel F in 3-KNF.
Gefragt: Hat F eine (erfüllende) Belegung, unter der in keiner Klausel alle Literale denselben Wahrheitswert haben?

Beispiel 186. *Die 3-KNF Formel $F = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$ ist alternativ durch folgende Klauselmenge darstellbar:*

$$F = \{\{x_1, \bar{x}_2\}, \{\bar{x}_1, x_3\}, \{x_2, \bar{x}_3, x_4\}\}$$

Offenbar ist $F(1111) = 1$, d.h. $F \in$ 3-SAT. Da unter dieser Belegung in jeder Klausel von F nicht nur mindestens ein Literal wahr, sondern auch mindestens ein Literal falsch wird, ist F auch in NAESAT. $\triangleleft$

Satz 187. 3-SAT *ist* NP*-vollständig.*

Beweis. Es ist nicht schwer zu sehen, dass 3-SAT in NP entscheidbar ist. Wir zeigen, dass 3-SAT NP-hart ist, indem wir CIRSAT auf 3-SAT reduzieren. Hierzu transformieren wir einen Schaltkreis $S = (g_1, \ldots, g_m)$ mit n Eingängen in eine 3-KNF Formel F_S über den Variablen $x_1, \ldots, x_n, y_1, \ldots, y_m$. F_S enthält neben der Klausel $\{y_m\}$ für jedes Gatter g_i die Klauseln von F_i, wobei F_i eine Formel in 3-KNF ist, die zu folgender Formel G_i äquivalent ist.

Gatter g_i	G_i	Klauseln von F_i
0	$y_i \leftrightarrow 0$	$\{\bar{y}_i\}$
1	$y_i \leftrightarrow 1$	$\{y_i\}$
x_j	$y_i \leftrightarrow x_j$	$\{\bar{y}_i, x_j\}, \{\bar{x}_j, y_i\}$
$(\neg, j)$	$y_i \leftrightarrow \bar{y}_j$	$\{\bar{y}_i, \bar{y}_j\}, \{y_j, y_i\}$
$(\wedge, j, k)$	$y_i \leftrightarrow y_j \wedge y_k$	$\{\bar{y}_i, y_j\}, \{\bar{y}_i, y_k\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$y_i \leftrightarrow y_j \vee y_k$	$\{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}, \{\bar{y}_i, y_j, y_k\}$

Nun ist leicht zu sehen, dass für alle $a \in \{0,1\}^n$ folgende Äquivalenz gilt:

$$S(a) = 1 \Leftrightarrow \exists b \in \{0,1\}^m : F_S(ab) = 1.$$

Ist nämlich $a \in \{0,1\}^n$ eine Eingabe mit $S(a) = 1$. Dann erhalten wir mit

$$b_l = g_l(a) \text{ für } l = 1, \ldots, m$$

eine erfüllende Belegung $ab_1 \ldots b_m$ für F_S. Ist umgekehrt $ab_1 \ldots b_m$ eine erfüllende Belegung für F_S, so folgt durch Induktion über $i = 1, \ldots, m$, dass

$$g_i(a) = b_i$$

ist. Insbesondere muss also $g_m(a) = b_m$ gelten, und da $\{y_m\}$ eine Klausel in F_S ist, folgt $S(a) = g_m(a) = b_m = 1$. Damit haben wir gezeigt, dass der Schaltkreis S und die 3-KNF-Formel F_S erfüllbarkeitsäquivalent sind, d.h.

$$S \in \text{CIRSAT} \Leftrightarrow F_S \in \text{3-SAT}.$$

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

6.5 Graphprobleme

Definition 188. *Ein* ***(ungerichteter) Graph*** *ist ein Paar $G = (V, E)$, wobei*

V - eine endliche Menge von ***Knoten/Ecken*** *und*

E - die Menge der ***Kanten*** *ist.*

Hierbei gilt

$$E \subseteq \binom{V}{2} = \big\{\{u, v\} \subseteq V \mid u \neq v\big\}.$$

a) Die ***Knotenzahl*** *von G ist $n(G) = \|V\|$.*

b) Die ***Kantenzahl*** *von G ist $m(G) = \|E\|$.*

c) Die ***Nachbarschaft*** *von $v \in V$ ist*

$$N_G(v) = \{u \in V \mid \{u, v\} \in E\}$$

und die Nachbarschaft von $U \subseteq V$ ist $N_G(U) = \bigcup_{u \in U} N_G(u)$.

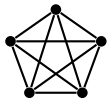
d) Der ***Grad*** *von v ist $\deg_G(v) = \|N_G(v)\|$.*

e) Der ***Minimalgrad*** *von G ist $\delta(G) = \min_{v \in V} \deg_G(v)$ und der* ***Maximalgrad*** *von G ist $\Delta(G) = \max_{v \in V} \deg_G(v)$.*

Falls G aus dem Kontext ersichtlich ist, schreiben wir auch einfach n, m, $N(v)$, $N(U)$, $\deg(v)$, δ usw.

Beispiel 189.

- *Der* ***vollständige Graph*** (V, E) *auf n Knoten, d.h. $\|V\| = n$ und $E = \binom{V}{2}$, wird mit K_n und der* ***leere Graph*** $(V, \varnothing)$ *auf n Knoten wird mit E_n bezeichnet.*

K_1: K_2: K_3:  K_4:  K_5: 

- *Der* ***vollständige bipartite Graph*** (A, B, E) *auf $a + b$ Knoten, d.h. $A \cap B = \varnothing$, $\|A\| = a$, $\|B\| = b$ und $E = \{\{u, v\} \mid u \in A, v \in B\}$ wird mit $K_{a,b}$ bezeichnet.*

$K_{1,1}$:  $K_{1,2}$:  $K_{2,2}$:  $K_{2,3}$:  $K_{3,3}$: 

- *Der* ***Pfadgraph*** *(oder* ***lineare Graph****) mit n Knoten wird mit P_m bezeichnet.*

P_1:  P_2:  P_3:  P_4: 

- *Der* ***Kreisgraph*** *(kurz* ***Kreis****) mit n Knoten wird mit C_n bezeichnet.*

C_3:  C_4:  C_5:  C_6: 

6.5.1 Cliquen, Stabilität und Knotenüberdeckungen

Definition 190. *Sei $G = (V, E)$ ein Graph.*

a) Ein Graph $G' = (V', E')$ heißt ***Sub-/Teil-/Untergraph*** *von G, falls $V' \subseteq V$ und $E' \subseteq E$ ist. Ein Subgraph $G' = (V', E')$ heißt (durch V')* ***induziert****, falls $E' = E \cap \binom{V'}{2}$ ist. Hierfür schreiben wir auch $H = G[V']$.*

b) Ein ***Weg*** *ist eine Folge von (nicht notwendig verschiedenen) Knoten $v_0, \ldots, v_\ell$ mit $\{v_i, v_{i+1}\} \in E$ für $i = 0, \ldots, \ell - 1$. Sind alle Knoten auf dem Weg paarweise verschieden, so heißt der Weg* ***einfach*** *oder* ***Pfad****. Die* ***Länge*** *des Weges ist die Anzahl der Kanten, also ℓ. Im Fall $\ell = 0$ heißt der Weg* ***trivial****. Ein Weg $v_0, \ldots, v_\ell$ heißt auch v_0-v_ℓ-Weg.*

c) Ein ***Zyklus*** *ist ein u-v-Weg der Länge $\ell \geq 2$ mit $u = v$.*

d) Ein ***Kreis*** *ist ein Zyklus $v_0, v_1 \ldots, v_{\ell-1}, v_0$ der Länge $\ell \geq 3$, für den $v_0, v_1, \ldots, v_{\ell-1}$ paarweise verschieden sind.*

e) G heißt ***zusammenhängend****, wenn es von jedem Knoten u in G zu jedem Knoten v in G einen Weg gibt.*

f) Eine Knotenmenge $U \subseteq V$ heißt ***stabil*** *oder* ***unabhängig****, wenn keine Kante in G beide Endpunkte in U hat, d.h. es gilt $E \cap \binom{U}{2} = \varnothing$. Die* ***Stabilitätszahl*** *ist*

$$\alpha(G) = \max\{\|U\| \mid U \text{ ist stabile Menge in } G\}.$$

g) Zwei Kanten $e, e' \in E$ heißen ***unabhängig****, falls $e \cap e' = \varnothing$ ist. Eine Kantenmenge $M \subseteq E$ heißt* Matching *in G, falls alle Kanten in M paarweise unabhängig sind. Die* ***Matchingzahl*** *ist*

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

h) Eine Knotenmenge $U \subseteq V$ heißt ***Clique****, wenn E alle Kanten enthält, die beide Endpunkte in U haben, d.h. es gilt $\binom{U}{2} \subseteq E$. Die* ***Cliquenzahl*** *ist*

$$\omega(G) = \max\{\|U\| \mid U \text{ ist Clique in } G\}.$$

i) Eine Knotenmenge $U \subseteq V$ heißt ***Knotenüberdeckung*** *(engl. vertex cover), wenn jede Kante $e \in E$ mindestens einen Endpunkt in U hat, d.h. es gilt $e \cap U \neq \varnothing$ für alle Kanten $e \in E$. Die* ***Überdeckungszahl*** *ist*

$$\beta(G) = \min\{\|U\| \mid U \text{ ist eine Knotenüberdeckung in } G\}.$$

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

CLIQUE: Hat G eine Clique der Größe k?

MATCHING: Hat G ein Matching der Größe k?

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

VERTEX COVER (VC): Hat G eine Knotenüberdeckung der Größe k?

Satz 191.

- CLIQUE, IS *und* VC *sind* NP*-vollständig.*
- MATCHING *ist in* P *entscheidbar (ohne Beweis).*

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

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

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

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

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

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

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

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

auf VC reduzieren. ∎

6.5.2 Euler- und Hamiltonkreise

Definition 192. *Sei $G = (V, E)$ ein Graph und sei $s = (v_0, v_1, \ldots, v_l)$ eine Folge von Knoten mit $\{v_i, v_{i+1}\} \in E$ für $i = 0, \ldots, l-1$.*

*a) s heißt **Eulerlinie** (auch **Eulerzug** oder **Eulerweg**) in G, falls s jede Kante in E genau einmal durchläuft, d.h. es gilt $\left\{\{v_i, v_{i+1}\} \mid i = 0, \ldots, l-1\right\} = E$ und $l = \|E\|$.*

*b) Gilt zudem $v_l = v_0$, so heißt s **Eulerkreis** (auch **Eulerzyklus** oder **Eulertour**).*

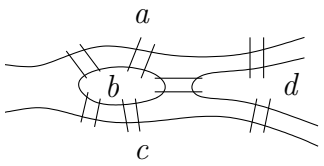
*c) s heißt **Hamiltonpfad** in G, falls s jeden Knoten in V genau einmal durchläuft, d.h. es gilt $\{v_0, \ldots, v_l\} = V$ und $l = \|V\| - 1$.*

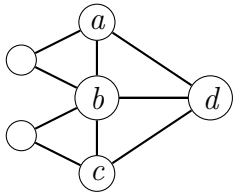
*d) Ist zudem $\{v_0, v_l\} \in E$, d.h. $s' = (v_0, v_1, \ldots, v_l, v_0)$ ist ein Kreis, so heißt s' **Hamiltonkreis**.*

Die angegebenen Definitionen lassen sich unmittelbar auf Digraphen übertragen, indem wir jede darin vorkommende ungerichtete Kante $\{u, v\}$ durch die gerichtete Kante (u, v) ersetzen.

Beispiel 193 (Das Königsberger Brückenproblem)**.**

Gibt es einen Spaziergang über alle 7 Brücken, bei dem keine Brücke mehrmals überquert wird und der zum Ausgangspunkt zurückführt?

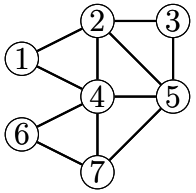


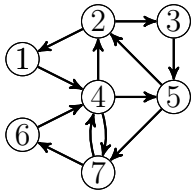


Diese Frage wurde von Euler (1707 – 1783) durch Betrachtung des nebenstehenden Graphen beantwortet. Dieser Graph hat offenbar genau dann einen Eulerkreis, wenn die Antwort „ja“ ist. (Wir werden gleich sehen, dass die Antwort „nein“ ist.) ◁

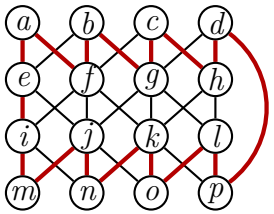
Beispiel 194.

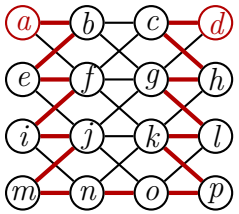
Der nebenstehende Graph besitzt die Eulerlinie $(4,1,2,3,5,7,6,4,5,2,4,7)$*, aber keinen Eulerkreis.*

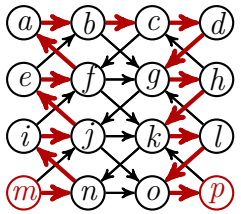




Der nebenstehende Digraph besitzt den Eulerkreis $s = (1,4,5,2,3,5,7,4,7,6,4,2,1)$*.*
Es folgen ein Hamiltonkreis in einem Graphen sowie ein a-d-Hamiltonpfad in einem Graphen und ein m-p-Hamiltonpfad in einem Digraphen:







◁

Wir betrachten für einen gegebenen Graphen (bzw. Digraphen) G und zwei Knoten s und t folgende Entscheidungsprobleme:

Das Eulerlinienproblem (EulerPath **bzw.** DiEulerPath)
Hat G eine Eulerlinie von s nach t?

Das Hamiltonpfadproblem (HamPath **bzw.** DiHamPath)
Hat G einen Hamiltonpfad von s nach t?

Zudem betrachten wir für einen gegebenen Graphen (bzw. Digraphen) G die folgenden Probleme:

Das Eulerkreisproblem (EulerCycle **bzw.** DiEulerCycle)
Hat G einen Eulerkreis?

Das Hamiltonkreisproblem (HamCycle **bzw.** DiHamCycle)
Hat G einen Hamiltonkreis?

Satz 195 (Euler, 1736)**.** *Sei G ein zusammenhängender Graph.*

(i) G besitzt genau dann einen Eulerkreis, wenn alle seine Knoten geraden Grad haben.

(ii) G besitzt im Fall $s \neq t$ genau dann eine Eulerlinie von s nach t, wenn s und t ungeraden Grad und alle übrigen Knoten geraden Grad haben.

Beweis.

(i) Falls G einen Eulerkreis s besitzt, existiert zu jeder Kante, auf der s einen Knoten erreicht, eine weitere Kante, auf der s den Knoten wieder verlässt. Daher hat jeder Knoten geraden Grad.

Ist umgekehrt G zusammenhängend und hat jeder Knoten geraden Grad, so können wir wie folgt einen Eulerkreis s konstruieren:

Berechnung eines Eulerkreises in $G = (V, E)$

1 Wähle $u \in V$ beliebig und initialisiere s zu $s = (u)$
2 Wähle einen beliebigen Knoten u auf dem Weg s, der mit einer unmarkierten Kante verbunden ist.

3 Folge ausgehend von u den unmarkierten Kanten auf einem beliebigen Weg z solange wie möglich und markiere dabei jede durchlaufene Kante. (Da von jedem erreichten Knoten $v \neq u$ ungerade viele markierte Kanten ausgehen, muss der Weg z zum Ausgangspunkt u zurückführen.)
4 Füge den Zyklus z an der Stelle u in s ein.
5 Wenn noch nicht alle Kanten markiert sind, gehe zu 2.
6 **Output**: s

(ii) Da G im Fall $s \neq t$ genau dann eine Eulerlinie von s nach t hat, wenn der Graph $G' = (V \cup \{u_{neu}\}, E \cup \{\{t, u_{neu}\}, \{u_{neu}, s\}\})$ einen Eulerkreis hat, folgt dies aus Teil (i) des Satzes.

■

Ganz ähnlich lässt sich ein entsprechender Satz für Digraphen beweisen.

Satz 196 (Euler, 1736)**.** *Sei $G = (V, E)$ ein stark zusammenhängender Digraph.*

(i) G besitzt genau dann einen Eulerkreis, wenn für jeden Knoten u in V der Ein- und Ausgangsgrad übereinstimmen.

(ii) G besitzt genau dann eine Eulerlinie von s nach t, wenn für jeden Knoten $u \in V \setminus \{s, t\}$ der Ein- und Ausgangsgrad übereinstimmen und $\deg^+(s) - \deg^-(s) = \deg^-(t) - \deg^+(t) = 1$ ist.

Korollar 197. *Die Probleme* EulerPath*,* EulerCycle*,* DiEulerPath *und* DiEulerCycle *sind alle in* P *entscheidbar.*

Beim Problem des Handlungsreisenden sind die Entfernungen d_{ij} zwischen n Städten $i, j \in \{1, \ldots, n\}$ gegeben. Gesucht ist eine Rundreise $(i_1, \ldots, i_n)$ mit minimaler Länge $d_{i_1,i_2} + \cdots + d_{i_{n-1},i_n} + d_{i_n,i_1}$, die jede Stadt genau einmal besucht. Die Entscheidungsvariante dieses Optimierungsproblems ist wie folgt definiert.

Problem des Handlungsreisenden (TSP; *traveling-salesman-problem*)

Gegeben: Eine $n \times n$ Matrix $D = (d_{i,j}) \in \mathbb{N}^{n \times n}$ und eine Zahl k.
Gefragt: Existiert eine Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, so dass die Rundreise $(\pi(1), \ldots, \pi(n))$ die Länge $\leq k$ hat?

Wir zeigen nun, dass die Probleme DiHamPath, HamPath, DiHamCycle, HamCycle und TSP alle NP-vollständig sind. Es ist leicht zu sehen, dass diese Probleme in NP entscheidbar sind. Zum Nachweis der NP-Härte zeigen wir folgende Reduktionen:

$$\text{3-Sat} \leq^p \text{DiHamPath} \leq^p \begin{matrix}\text{HamPath},\\ \text{DiHamCycle}\end{matrix} \leq^p \text{HamCycle} \leq^p \text{TSP}.$$

Wir reduzieren zuerst HamCycle auf TSP.

Satz 198. HamCycle $\leq^p$ TSP.

Beweis. Sei ein Graph $G = (V, E)$ gegeben. Wir können annehmen, dass $V = \{1, \ldots, n\}$ ist. Dann lässt sich G in Polynomialzeit auf die TSP Instanz (D, n) mit $D = (d_{i,j})$ und

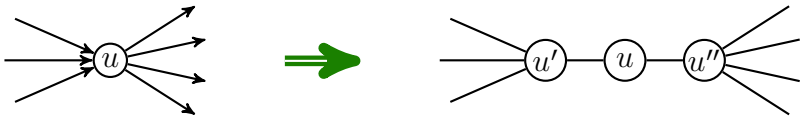
$$d_{i,j} = \begin{cases} 1, & \text{falls } \{i, j\} \in E, \\ 2, & \text{sonst,} \end{cases}$$

transformieren. Diese Reduktion ist korrekt, da G genau dann einen Hamiltonkreis hat, wenn es in dem Distanzgraphen D eine Rundreise $(\pi(1), \ldots, \pi(n))$ der Länge $L(\pi) \leq n$ gibt. ■

Als nächstes reduzieren wir DiHamCycle auf HamCycle.

Satz 199. DiHamCycle $\leq^p$ HamCycle.

Beweis. Wir transformieren wir einen Digraphen G auf einen Graphen G', indem wir lokal für jeden Knoten $u \in V$ die folgende Ersetzung durchführen:



Dann ist klar, dass die Funktion $G \mapsto G'$ in FP berechenbar ist, und G genau dann einen Hamiltonkreis enthält, wenn dies auf G' zutrifft. Ähnlich lässt sich auch DiHamPath auf HamPath reduzieren. ■

Satz 200. DiHamPath $\leq^p$ DiHamCycle.

Beweis. Um DiHamPath auf DiHamCycle zu reduzieren, transformieren wir einen gegebenen Digraphen $G = (V, E)$ mit zwei ausgezeichneten Knoten $s, t \in V$ in den Digraphen $G' = (V', E')$ mit

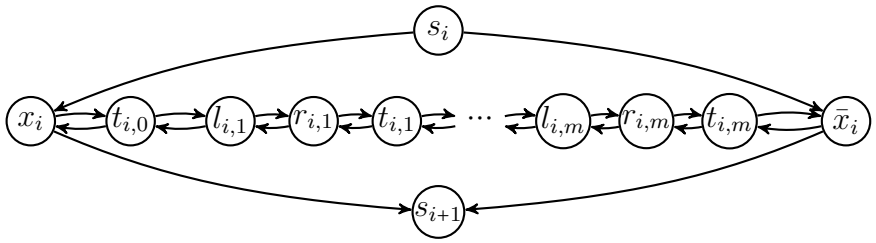
$$V' = V \cup \{u_{\text{neu}}\} \text{ und}$$
$$E' = E \cup \{(t, u_{\text{neu}}), (u_{\text{neu}}, s)\}.$$

Offenbar ist G' in Polynomialzeit aus G berechenbar und besitzt genau dann einen Hamiltonkreis, wenn G einen s-t-Hamiltonpfad besitzt. Ähnlich lässt sich auch HamPath auf HamCycle reduzieren. ■

Satz 201. 3-Sat $\leq^p$ DiHamPath.

Beweis. Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$. Wir transformieren F in Polynomialzeit in einen Digraphen $G_F = (V, E)$ mit zwei ausgezeichneten Knoten s und t, der genau dann einen hamiltonschen s-t-Pfad besitzt, wenn F erfüllbar ist.

Jede Klausel C_j repräsentieren wir durch einen Knoten c_j und jede Variable x_i repräsentieren wir durch folgenden Graphen X_i.



Die Knotenmenge V von G_F ist also

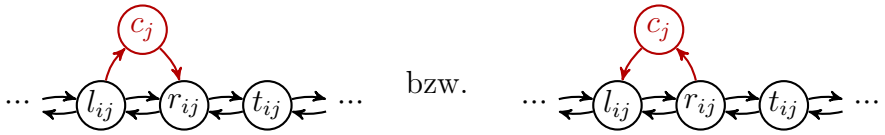
$$V = \{c_1, \ldots, c_m\} \cup \{s_1, \ldots, s_{n+1}\} \cup \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$$
$$\cup \bigcup_{i=1}^{n} \{t_{i,0}, l_{i,1}, r_{i,1}, t_{i,1}, \ldots, l_{i,m}, r_{i,m}, t_{i,m}\}.$$

Dabei haben die Graphen X_{i-1} und X_i den Knoten s_i gemeinsam. Als Startknoten wählen wir $s = s_1$ und als Zielknoten $t = s_{n+1}$.

Ein Pfad von s nach t kann ausgehend von jedem Knoten s_i ($i = 1, \ldots, n$) entweder zuerst den Knoten x_i oder zuerst den Knoten $\bar{x}_i$ besuchen. Daher können wir jedem s-t-Pfad P eine Belegung $b_P = b_1 \ldots b_n$ zuordnen mit $b_i = 1$ gdw. P den Knoten x_i vor dem Knoten $\bar{x}_i$ besucht.

Die Klauselknoten c_j verbinden wir mit den Teilgraphen X_i so, dass ein s-t-Pfad P genau dann einen „Abstecher“ nach c_j machen kann, wenn die Belegung b_P die Klausel C_j erfüllt.

Hierzu fügen wir zu E für jedes Literal $l \in C_j$ im Fall $l = x_i$ die beiden Kanten (l_{ij}, c_j) und (c_j, r_{ij}), und im Fall $l = \bar{x}_i$ die Kanten (r_{ij}, c_j) und (c_j, l_{ij}) hinzu:

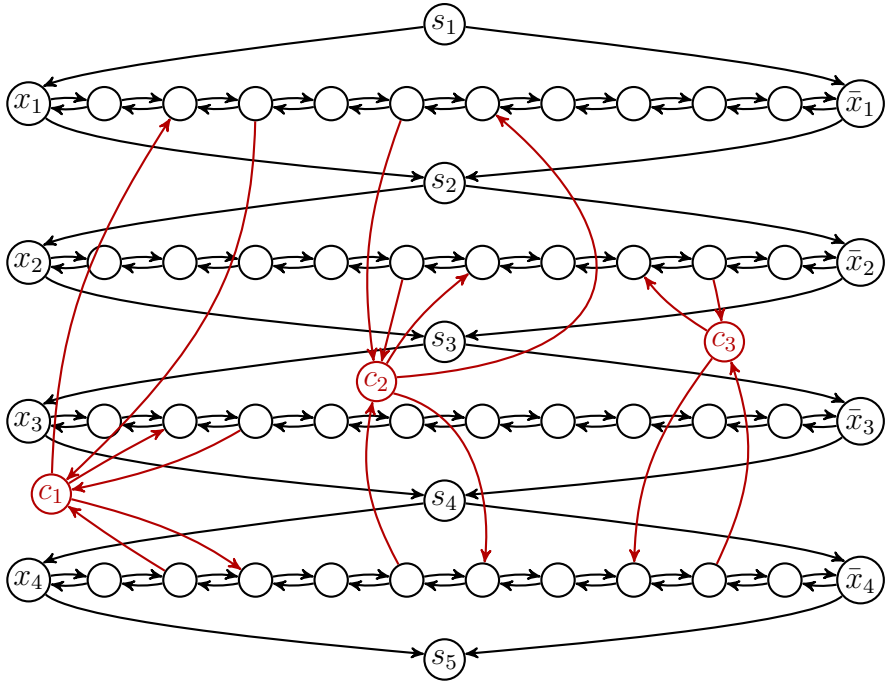


Man beachte, dass einem s-t-Pfad P genau dann ein Abstecher über diese Kanten zu c_j möglich ist, wenn die Belegung b_P das Literal l wahr macht.

Nun ist klar, dass die Reduktionsfunktion $F \mapsto (G_F, s, t)$ in Polynomialzeit berechenbar ist. Es bleibt also zu zeigen, dass F genau dann erfüllbar ist, wenn in G_F ein Hamiltonpfad von $s = s_1$ nach $t = s_{n+1}$ existiert.

Falls $F(b) = 1$ ist, so lässt sich der zu b gehörige s-t-Pfad P wie folgt zu einem Hamiltonpfad erweitern. Wir wählen in jeder Klausel C_j ein wahres Literal $l = x_i$ bzw. $l = \bar{x}_i$ und bauen in den Pfad P einen Abstecher vom Knotenpaar l_{ij}, r_{ij} zum Klauselknoten c_j ein.

Ist umgekehrt P ein s-t-Hamiltonpfad in G_F, so müssen der Vorgänger- und Nachfolgerknoten jedes Klauselknotens c_j ein Paar l_{ij}, r_{ij} bilden, da P andernfalls nicht beide Pufferknoten $t_{i,j-1}$ und $t_{i,j}$ besuchen kann. Da aber P alle Klauselknoten besucht und ausgehend von dem Paar l_{ij}, r_{ij} nur dann ein Abstecher zu c_j möglich ist, wenn die Belegung b_P die Klausel C_j erfüllt, folgt $F(b_P) = 1$. ■



Beispiel 202. *Die* 3-SAT*-Instanz*

$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$$

lässt sich auf den links abgebildeten Digraphen G mit Startknoten $s = s_1$ und Zielknoten $t = s_5$ reduzieren. Der erfüllenden Belegung $b = 0110$ entspricht beispielsweise der Hamiltonpfad, der von s_1 über $\bar{x}_1$, c_1, x_1, s_2, x_2, c_2, $\bar{x}_2$, s_3, x_3, $\bar{x}_3$, s_4, $\bar{x}_4$, c_3 und x_4 nach s_5 geht. ◁

6.6 Das Rucksack-Problem

Wie schwierig ist es, einen Rucksack der Größe w mit einer Auswahl aus k Gegenständen der Größe $u_1, \ldots, u_k$ möglichst voll zu packen? Dieses Optimierungsproblem lässt sich leicht auf folgendes Entscheidungsproblem reduzieren.

Rucksack-Problem (Rucksack):

Gegeben: Eine Folge $(u_1, \ldots, u_k, v, w)$ von natürlichen Zahlen.
Gefragt: Ex. eine Auswahl $S \subseteq \{1, \ldots, k\}$ mit $v \leq \sum_{i \in S} u_i \leq w$?

Beim SubsetSum-Problem möchte man dagegen nur wissen, ob der Rucksack randvoll gepackt werden kann.

SubsetSum:

Gegeben: Eine Folge $(u_1, \ldots, u_k, w)$ von natürlichen Zahlen.
Gefragt: Ex. eine Auswahl $S \subseteq \{1, \ldots, k\}$ mit $\sum_{i \in S} u_i = w$?

Satz 203. RUCKSACK *und* SUBSETSUM *sind* NP*-vollständig.*

Beweis. Es ist leicht zu sehen, dass beide Probleme in NP enthalten sind. Zum Nachweis der NP-Härte zeigen wir die folgenden Reduktionen:

$$\text{3-SAT} \leq^p \text{SUBSETSUM} \leq^p \text{RUCKSACK}.$$

Da SubsetSum einen Spezialfall des Rucksack-Problems darstellt, lässt es sich leicht darauf reduzieren:

$$(u_1, \ldots, u_k, w) \mapsto (u_1, \ldots, u_k, w, w).$$

Es bleibt also 3-Sat $\leq^p$ SubsetSum zu zeigen. Sei $F = \{C_1, \ldots, C_m\}$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$. Betrachte die Reduktionsfunktion

$$f : F \mapsto (u_1, \ldots, u_n, u'_1, \ldots, u'_n, v_1, \ldots, v_m, v'_1, \ldots, v'_m, w),$$

wobei u_i und u'_i die Dezimalzahlen

$$u_i = b_{i1} \ldots b_{im} 0^{i-1} 1 0^{n-i} \text{ und } u'_i = b'_{i1} \ldots b'_{im} 0^{i-1} 1 0^{n-i}$$

mit

$$b_{ij} = \begin{cases} 1, & x_i \in C_j, \\ 0, & \text{sonst}, \end{cases} \quad \text{und} \quad b'_{ij} = \begin{cases} 1, & \bar{x}_i \in C_j, \\ 0, & \text{sonst}, \end{cases}$$

und $v_j = v'_j = 0^{j-1} 1 0^{m-j-1} 0^n$ sind. Die Zielsumme w setzen wir auf den Wert $w = \underbrace{3 \ldots 3}_{m\text{-mal}} \underbrace{1 \ldots 1}_{n\text{-mal}}$.

Sei nun $a = a_1 \ldots a_n$ eine erfüllende Belegung für F. Da a in jeder Klausel mindestens ein und höchstens drei Literale wahr macht, hat die Zahl

$$\sum_{a_i=1} u_i + \sum_{a_i=0} u'_i$$

eine Dezimaldarstellung der Form $b_1 \ldots b_m 1 \ldots 1$ mit $1 \leq b_j \leq 3$ für $j = 1, \ldots, m$. Durch Addition von

$$\sum_{b_j \leq 2} v_j + \sum_{b_j=1} v'_j$$

erhalten wir den gewünschten Wert w. Dies zeigt, dass $F \in$ 3-Sat die Zugehörigkeit von $f(F) \in$ SubsetSum impliziert. Für die umgekehrte Implikation sei $S = P \cup N \cup I \cup J$ eine Auswahlmenge für die SubsetSum-Instanz $f(F)$ mit

$$\sum_{i \in P} u_i + \sum_{i \in N} u'_i + \sum_{j \in I} v_j + \sum_{j \in J} v'_j = \underbrace{3 \ldots 3}_{m\text{-mal}} \underbrace{1 \ldots 1}_{n\text{-mal}}.$$

Da die Teilsumme $\sum_{j \in I} v_j + \sum_{j \in J} v'_j$ die Form $c_1 \ldots c_m 0 \ldots 0$ mit $c_j \leq 2$ hat, muss die Teilsumme $\sum_{i \in P} u_i + \sum_{i \in N} u'_i$ die Form $b_1 \ldots b_m 1 \ldots 1$ mit $b_j \geq 1$ haben. Da keine Überträge auftreten, muss also $P = \{1, \ldots, n\} - N$ gelten und jede Klausel C_j mindestens ein Literal aus der Menge $\{x_i \mid i \in P\} \cup \{\bar{x}_i \mid i \in N\}$ enthalten. Folglich wird F von folgender Belegung $a = a_1 \ldots a_n$ erfüllt:

$$a_i = \begin{cases} 1, & i \in P, \\ 0, & i \in N. \end{cases}$$

Damit haben wir die Korrektheit von f gezeigt. Da f zudem in Polynomialzeit berechenbar ist, folgt 3-Sat $\leq^p$ SubsetSum. ■

Beispiel 204. *Betrachte die 3-KNF Formel*

$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4).$$

Die zu F gehörige SubsetSum*-Instanz $f(F)$ ist*

$$(u_1, u_2, u_3, u_4, u'_1, u'_2, u'_3, u'_4, v_1, v_2, v_3, v'_1, v'_2, v'_3, w)$$

mit

$$\begin{array}{lll} u_1 = 010\,1000, & u'_1 = 100\,1000, & v_1 = v'_1 = 100\,0000, \\ u_2 = 010\,0100, & u'_2 = 001\,0100, & v_2 = v'_2 = 010\,0000, \\ u_3 = 000\,0010, & u'_3 = 100\,0010, & v_3 = v'_3 = 001\,0000, \\ u_4 = 110\,0001, & u'_4 = 001\,0001, & \end{array}$$

sowie $w = 333\,1111$. Der erfüllenden Belegung $a = 0100$ entspricht dann die Auswahl $(u'_1, u_2, u'_3, u'_4, v_1, v_2, v'_2, v_3, v'_3)$. ◁

6.7 Ganzzahlige lineare Programmierung

In bestimmten Anwendungen tritt das Problem auf, einen Lösungsvektor mit ganzzahligen Koeffizienten für ein System linearer Ungleichungen zu finden.

Ganzzahlige Programmierung (IP; *integer programming*)

Gegeben: Eine ganzzahlige $m \times n$ Matrix $A = (a_{ij}) \in \mathbb{Z}^{m \times n}$ und ein ganzzahliger Vektor $\boldsymbol{b} \in \mathbb{Z}^m$.

Gefragt: Existiert ein ganzzahliger Vektor $\boldsymbol{x} \in \mathbb{Z}^n$ mit $A\boldsymbol{x} \geq \boldsymbol{b}$ wobei $\geq$ komponentenweise zu verstehen ist.

Satz 205. IP *ist* NP*-hart.*

Beweis. Wir reduzieren 3-SAT auf IP. Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$. Wir transformieren F in ein Ungleichungssystem $A\boldsymbol{x} \geq \boldsymbol{b}$ für den Lösungsvektor $\boldsymbol{x} = (x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$, das

- für $i = 1, \ldots, n$ die vier Ungleichungen

$$x_i + \bar{x}_i \geq 1, \quad -x_i - \bar{x}_i \geq -1, \quad x_i \geq 0, \quad \bar{x}_i \geq 0 \qquad (*)$$

- und für jede Klausel $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ folgende Ungleichung enthält:

$$l_{j1} + \cdots + l_{jk_j} \geq 1. \qquad (**)$$

Die Ungleichungen $(*)$ sind für ganzzahlige $x_i, \bar{x}_i$ genau dann er- füllt, wenn x_i den Wert 0 und $\bar{x}_i$ den Wert 1 hat oder umgekehrt. Die Klauselungleichungen $(**)$ stellen sicher, dass mindestens ein Literal in jeder Klausel C_j wahr wird. Nun ist leicht zu sehen, dass jede Lösung $\boldsymbol{x}$ von $A\boldsymbol{x} \geq \boldsymbol{b}$ einer erfüllenden Belegung von F entspricht und umgekehrt. ∎

Bemerkung 206.

- *Es ist nicht leicht zu sehen, dass IP in* NP *entscheidbar ist.*
- *Ein nichtdeterministischer Algorithmus kann zwar eine Lösung raten, aber a priori ist nicht klar, ob eine Lösung $\boldsymbol{x}$ ex., deren Binärkodierung polynomiell in der Länge der Eingabe $(A, \boldsymbol{b})$ ist.*
- *Mit Methoden der linearen Algebra lässt sich jedoch zeigen, dass jede lösbare IP-Instanz $(A, \boldsymbol{b})$ auch eine Lösung $\boldsymbol{x}$ hat, deren Kodierung polynomiell in der Länge von $(A, \boldsymbol{b})$ ist.*
- *Wenn wir nicht verlangen, dass die Lösung $\boldsymbol{x}$ der IP-Instanz ganzzahlig ist, dann spricht man von einem* ***linearen Programm****.*
- *Für* **LP** *(Lineare Programmierung) gibt es Polynomialzeitalgorithmen (von Khachiyan 1979 und von Karmarkar 1984).*