# Theoretische Informatik

**Kurzskript zu den Vorlesungen von**

Prof. Van Bang Le Prof. Karsten Wolf

16. Februar 2018

# Inhaltsverzeichnis

# 0 Wichtige Begriffe aus der Berechenbarkeitstheorie

## Ist ein Problem algorithmisch lösbar?

**Definition 0.0.1** (Turingmaschine)**.** Eine deterministische Turingmaschine (DTM) (oder ein det. Turingprogramm) $M$ ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_a, z_e)$ mit:

- $Z$ ist eine endliche Menge von Zuständen
- $z_a$ ist der ausgezeichnete Anfangszustand
- $z_e$ ist der ausgezeichnete Endzustand
- $\Sigma$ ist eine endliche Menge, das Bandalphabet
  $\square$ ist ein ausgezeichnetes Symbol in $\Sigma$: es heißt Leersymbol (Blank) und zeigt an, dass die Bandzelle leer ist
- $Z \cap \Sigma = \emptyset$
- $\delta : Z \times \Sigma \to Z \times \Sigma \times \{+1, -1, 0\}$ ist eine nicht notwendig überall definierte Funktion, die Übergangsfunktion

**Definition 0.0.2** (Turing-berechenbar)**.** Eine (partielle) Funktion $f : \Sigma^* \to \Sigma^*$ heißt Turingprogramm-berechenbar, falls eine DTM $M$ existiert mit $f = f_M$.

**Definition 0.0.3** (Entscheidbarkeit)**.** Eine Menge $A \subseteq \mathbb{N}$ (bzw. $A \subseteq \mathbb{N}^k$ oder $A \subseteq \Sigma^*$) heißt entscheidbar, wenn die charakteristische Funktion $\chi_A : \mathbb{N} \to \{0, 1\}$ (bzw. $\chi_A : \mathbb{N}^k \to \{0, 1\}$ oder $\chi_A : \Sigma^* \to \{0, 1\}$) von $A$,

$$\chi_A(x) = \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{sonst, also falls } x \in A \end{cases}$$

berechenbar ist.

# 1 Komplexität

## Wie schwierig ist ein lösbares Problem?

## 1.1 Graphen

**Definition 1.1.1.** Ein (ungerichteter, einfacher) Graph $G$ ist ein Paar $G = (V, E)$ bestehend aus Knotenmenge $V$ und Kantenmenge $E \subseteq \binom{V}{2}$.
$\binom{V}{2}$ steht hierbei für die Menge aller 2-elementigen Teilmengen von $V$.

**Definition 1.1.2.** Sei $G = (V, E)$ ein Graph. Dann gilt:

1. Zwei Knoten $x, y \in V$ sind verbunden, wenn $\{x, y\} \in E$ ist.
2. Eine Menge $Q \subseteq V$ von Knoten ist eine Clique, wenn je zwei Knoten in $Q$ verbunden sind.
3. Eine Menge $U \subseteq V$ von Knoten ist eine unabhängige Menge (independent set), wenn je zwei Knoten in $U$ unverbunden sind.

## 1.2 Zeitaufwand und Komplexitätsklassen

**Definition 1.2.1** ($\mathcal{O}$-Notation)**.** Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ schreiben wir $f \in \mathcal{O}(g)$ oder auch $f = \mathcal{O}(g)$, falls es eine Konstante $c > 0$ gibt mit: Es existiert ein $n_0$, sodass $f(n) \leq c \cdot g(n)$ für alle $n \geq n_0$ gilt. Man sagt, dass $f$ asymptotisch höchstens so stark wächst wie $g$.
Formal: $f \in \mathcal{O}(g) \Leftrightarrow \exists c > 0 \, \exists n_0 \in \mathbb{N} \, \forall n \geq n_0 : f(n) \leq c \cdot g(n)$

**Definition 1.2.2** (Zeitaufwand von DTM-Programmen)**.** Sei $M$ eine DTM über dem Alphabet $\Sigma$. Der Zeitaufwand $t_M(w)$ von $M$ bei Eingabe $w \in \Sigma^*$ ist

$$t_M(w) = \begin{cases} \text{Zahl der Konfigurationsübergänge der Berechnung von } M \text{ bei Eingabe } w & \text{falls Berechnung abbricht} \\ \infty & \text{sonst} \end{cases}$$

Der Zeitaufwand $t_M(n)$ von $M$ bei Eingaben der Codierungslänge $n$ ist
$t_M(n) = \max\{t_M(w) | w \in \Sigma^n\}$

**Definition 1.2.3.** Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Mit DTIME($f$) bezeichnen wir die Menge aller Entscheidungsprobleme, die sich durch eine DTM $M$ mit Zeitaufwand $t_m = O(f)$ entscheiden lassen:
DTIME(f) $= \{L \subseteq \Sigma^* \mid L \text{ ist entscheidbar durch eine DTM } M \text{ mit } t_m = \mathcal{O}(f)\}$

**Definition 1.2.4** (Komplexitätsklasse $\mathsf{P}$)**.** $\mathsf{P} = \bigcup_{k=0}^{\infty} \mathsf{DTIME}(n^k)$ ist die Klasse aller in (deterministisch) **p**olynomiellem Zeitaufwand lösbaren (Entscheidungs-) Probleme. $\mathsf{P}$ ist also die Klasse von Problemen, die effizient gelöst werden können.

**Definition 1.2.5** (Komplexitätsklasse $\mathsf{EXPTIME}$)**.** $\mathsf{EXPTIME} = \bigcup_{k=0}^{\infty} \mathsf{DTIME}(2^{n^k})$ ist die Klasse aller in (deterministisch) exponentiellem Zeitaufwand lösbaren (Entscheidungs-) Probleme. (Es gilt: $\mathsf{P} \subseteq \mathsf{EXPTIME}$)

**Definition 1.2.6** (Nichtdeterministische Turingmaschine)**.** Eine nichtdeterministische Turingmaschine (NTM) (oder ein nichtdet. Turingprogramm) $M$ ist ein 5-Tupel $M = (Z, \Sigma, \delta, z_a, z_e)$ mit:

- $Z$,$z_a$,$z_e$,$\Sigma$ sind definiert wie bei einer deterministischen Turingmaschine
- $\delta \subseteq (Z \times \Sigma) \times (Z \times \Sigma \times \{+1, -1, 0\})$ ist eine Relation, die Übergangsrelation

In einem nichtdeterministischen Turingprogramm $\delta$ kann es zu einem Paar $(z, x) \in Z \times \Sigma$ mehr als einen Befehl mit der linken Seite $z, x$ geben.

**Definition 1.2.7** (Nichtdeterministische Berechnung)**.** Jedem Eingabewort $w \in \Sigma^*$ kann man einen „Berechnungsbaum “zuordnen, dessen maximale Pfade den möglichen Berechnungen entsprechen:

- Die Wurzel des Berechnungsbaums ist mit der Anfangskonfiguration $z_a w$ beschriftet
- Ist $v$ ein Knoten des Baums, der mit der Konfiguration $K = \alpha z x \beta$ markiert ist und ist $\delta(z, x) = \{(z_1, x_1, \lambda_1), \ldots, (z_r, x_r, \lambda_r)\}$, so hat $v$ genau $r$ Söhne, die jeweils mit den Nachfolgekonfigurationen $K_i$ von $K$ bezüglich $(z_i, x_i, \lambda_i)$ beschriftet sind, $i = 1, \ldots, r$.

**Definition 1.2.8** (Nichtdeterministische Entscheidbarkeit)**.** Eine NTM $M$ entscheidet die Menge $L \subseteq \Sigma^*$, falls der Berechnungsbaum jedes Eingabewortes $w \in \Sigma^*$ endlich ist und für $w \in L$ mindestens einen erfolgreichen Berechnungspfad enthält.

**Satz 1.2.1.** Sei $L \subseteq \Sigma^*$. Dann ist $L$ genau dann (deterministisch) entscheidbar, wenn $L$ nichtdeterministisch entscheidbar ist.

**Definition 1.2.9** (Zeitaufwand von NTM-Programmen)**.** Sei $M$ eine NTM über dem Alphabet $\Sigma$. Der Zeitaufwand $t_M(w)$ von $M$ bei Eingabe $w \in \Sigma^*$ ist

$$t_M(w) = \begin{cases} \text{Tiefe des Berechnungsbaumes von } M \text{ bei Eingabe } w & \text{falls Baum endlich} \\ \infty & \text{sonst} \end{cases}$$

Der Zeitaufwand $t_M(n)$ von $M$ bei Eingaben der Codierungslänge $n$ ist
$t_M(n) = \max\{t_M(w) | w \in \Sigma^n\}$

**Definition 1.2.10.** Es sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Mit $\mathsf{NTIME}(f)$ bezeichnen wir die Menge aller Entscheidungsprobleme, die sich durch eine NTM $M$ mit Zeitaufwand $t_m = O(f)$ entscheiden lassen:
$\mathsf{NTIME(f)} = \{L \subseteq \Sigma^* \mid L \text{ ist entscheidbar durch eine NTM } M \text{ mit } t_m = \mathcal{O}(f)\}$

**Definition 1.2.11** (Komplexitätsklasse NP)**.** $\mathsf{NP} = \bigcup_{k=0}^{\infty} \mathsf{NTIME}(n^k)$ ist die Klasse aller in **n**ichtdeterministisch **p**olynomiellem Zeitaufwand lösbaren (Entscheidungs-) Probleme.

**Satz 1.2.2.** $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXPTIME}$

## 1.3 Polynomielle Reduktion und NP-Vollständigkeit

**Definition 1.3.1** (Polynomielle Reduktion)**.** Seien $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ zwei Entscheidungsprobleme. $L_1$ ist auf $L_2$ polynomiell reduzierbar, wenn es eine überall definierte, polynomiell berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $x \in \Sigma_1^*$ gilt:

$$x \in L_1 \Leftrightarrow f(x) \in L_2$$

Ist $L_1$ polynomiell reduzierbar auf $L_2$ via $f$, so schreiben wir: $L_1 \leq_p L_2$

**Definition 1.3.2** (NP-Vollständigkeit)**.** Es sei $L \subseteq \Sigma^*$. $L$ heißt NP-vollständig, falls gilt:

(i) $L \in \mathsf{NP}$

(ii) $\forall M \in \mathsf{NP} : M \leq_p L$

**Lemma 1.3.1.** Ist $L$ NP-vollständig, so gilt: Ł $\in \mathsf{P} \Leftrightarrow \mathsf{P} = \mathsf{NP}$

**Satz 1.3.1** (Satz von Cook und Levin)**.** SAT ist NP-vollständig.

**Satz 1.3.2.** Ist $A$ NP-vollständig, $A \leq_p B$ und $B \in \mathsf{NP}$, so ist $B$ NP-vollständig.

**Satz 1.3.3.** 3-SAT ist NP-vollständig.

**Satz 1.3.4.** CLIQUE ist NP-vollständig.

**Satz 1.3.5.** INDSET und VERTEX COVER sind NP-vollständig.

**Satz 1.3.6.** 3-Färbbarkeit ist NP-vollständig.

**Satz 1.3.7.** SUBSET-SUM ist NP-vollständig.

**Definition 1.3.3** (Komplexitätsklasse coK)**.** Sei K eine Komplexitätsklasse über dem Alphabet $\Sigma$. Dann heißt

$$\mathsf{coK} := \{L \subseteq \Sigma^* \mid \overline{L} \in K\}$$

die Klasse der Sprachen (Entscheidungsprobleme) $L$, deren Komplement $\overline{L}$ in $K$ liegt.

**Satz 1.3.8.** $\mathsf{coP} = \mathsf{P}$

## 1.4 Platzkomplexität

**Definition 1.4.1** (Speicherbedarf von DTM-Programmen)**.** Sei $M$ eine DTM über dem Alphabet $\Sigma$. Der Speicherbedarf $s_M(w)$ von $M$ bei Eingabe $w \in \Sigma^*$ ist

$$s_M(w) = \begin{cases} \text{Zahl der Bandzellen, die } M \text{ bei Bearbeitung} & \text{falls Berechnung abbricht} \\ \text{von Eingabe } w \text{ besucht} & \\ \infty & \text{sonst} \end{cases}$$

Der Speicherbedarf $s_M(n)$ von $M$ bei Eingaben der Codierungslänge $n$ ist
$s_M(n) = \max\{s_M(w) | w \in \Sigma^n\}$

**Definition 1.4.2.** Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion.

- $\mathsf{DSPACE}(f)$ ist die Menge aller Entscheidungsprobleme, die sich durch eine DTM $M$ mit Speicherbedarf $s_M = O(f)$ entscheiden lassen.
- $\mathsf{NSPACE}(f)$ ist die Menge aller Entscheidungsprobleme, die sich durch eine NTM $M$ mit Speicherbedarf $s_M = O(f)$ entscheiden lassen.

**Definition 1.4.3** (Platzkomplexitätsklassen PSPACE und NSPACE)**.**

$$\mathsf{PSPACE} := \bigcup_{k=0}^{\infty} \mathsf{DSPACE}(n^k) \quad \mathsf{NSPACE} := \bigcup_{k=0}^{\infty} \mathsf{NSPACE}(n^k)$$

**Satz 1.4.1** (Satz von Savitch)**.** $\mathsf{PSPACE} = \mathsf{NSPACE}$

**Satz 1.4.2.** $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$

## 1.5 Umgang mit schwierigen Problemen

### 1.5.1 Betrachtung von Spezialfällen

Manchmal sind nur spezielle Fälle eines schwierigen Problems praktisch relevant. Für diese kann es möglich sein, sie effizient zu lösen.

**Beispiel 1.5.1.**
*SUBSET SUM ist für „super-wachsende“ Eingaben polynomiell lösbar.*

### 1.5.2 Annäherungsverfahren

Man betrachtet die Optimierungsversion des zugehörigen Entscheidungsproblems und probiert die Lösung zu approximieren.

**Definition 1.5.1.** Ein $c$-Approximationsalgorithmus $A$ (mit Güte $c > 1$) ist ein Algorithmus mit:

- Die Laufzeit von $A$ ist polynomiell zur Eingabelänge.

- Die Ausgabe $v$ von $A$ zur Eingabe $w$ ist eine zulässige Lösung.
- $c = \begin{cases} \frac{v}{v^*} & \text{bei Minimierungsproblemen} \\ \frac{v^*}{v} & \text{bei Maximierungsproblemen} \end{cases}$, wobei $v^*$ eine optimale Lösung ist.

### 1.5.3 Parametrisierte Algorithmen

Komplexe Parameter des Problems werden von der Eingabe getrennt.

### 1.5.4 Randomisierte Algorithmen

Ein randomisierter Algorithmus hat in seinem Ablauf Zugriff auf eine Quelle von Zufallszahlen.

# 2 Formale Sprachen

## Wie können Sprachen beschrieben werden?

### 2.1 Grundbegriffe

**Definition 2.1.1** (Alphabete, Wörter, Sprachen)**.**

- Ein Alphabet $\Sigma$ ist eine nichtleere, endliche Menge.
- Die Elemente von $\Sigma$ heißen Zeichen (Symbole, Buchstaben) des Alphabets.
- Ein Wort $w$ über dem Alphabet $\Sigma$ ist eine endliche Folge von Zeichen aus $\Sigma$.
- Die Länge $|w|$ des Wortes $w$ ist die Anzahl der Zeichen in $w$.
- Das leere Wort $\varepsilon$ bezeichnet das Wort der Länge 0.
- $\Sigma^*$ ist die Menge aller Wörter über $\Sigma$.
- Eine formale Sprache über dem Alphabet $\Sigma$ ist eine Teilmenge von $\Sigma^*$.

**Definition 2.1.2** (Grammatik)**.** Eine Grammatik ist ein 4-Tupel $G = (N, \Sigma, R, S)$ mit

- einem endlichem Alphabet $N$ von **N**ichtterminalen,
- einem endlichen Alphabet $\Sigma$ von Terminalen, $N \cap \Sigma = \varnothing$,
- einer endlichen Menge $R \subseteq (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$ von (Produktions-) **R**egeln,
- und einem **S**tartsymbol $S \in N$.

**Definition 2.1.3.** Die von einer Grammatik $G = (N, \Sigma, R, S)$ erzeugte Sprache ist $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

**Definition 2.1.4.** Zwei Grammatiken $G_1 = (N_1, \Sigma_1, R_1, S_1)$ und $G_2 = (N_2, \Sigma_2, R_2, S_2)$ heißen äquivalent, wenn $L(G_1) = L(G_2)$ gilt.

**Satz 2.1.1.** $L \subseteq \Sigma^*$ ist aufzählbar $\Leftrightarrow$ Es gibt Grammatik $G = (N, \Sigma, R, S)$ mit $L = L(G)$.

## 2.2 Chomsky Hierarchie

**Definition 2.2.1** (Typ 0: Unbeschränkte Grammatiken)**.**

- Eine Grammatik $G = (N, \Sigma, R, S)$ heißt unbeschränkt.
- Eine Sprache $L$ ist aufzählbar, falls es eine unbeschränkte Grammatik $G$ gibt mit $L(G) = L$.

**Definition 2.2.2** (Typ 1: Kontextsensitive Grammatiken)**.**

- Eine Grammatik $G = (N, \Sigma, R, S)$ heißt kontextsensitiv, wenn für jede Regel $u \to v$ in $R$ gilt: $|u| \leq |v|$, mit der Ausnahme $S \to \varepsilon$, falls das Startsymbol $S$ auf keiner rechten Seite einer Regel vorkommt.
- Eine Sprache $L$ heißt kontextsensitiv, falls es eine kontextsensitive Grammatik $G$ gibt mit $L(G) = L$.

**Satz 2.2.1.** Kontextsensitive Sprachen sind entscheidbar.

**Definition 2.2.3** (Typ 2: Kontextfreie Grammatiken)**.**

- Eine Grammatik $G = (N, \Sigma, R, S)$ heißt kontextfrei, wenn für jede Regel $u \to v$ in $R$ gilt: $|u| \in N$.
- Eine Sprache $L$ heißt kontextfrei, falls es eine kontextfreie Grammatik $G$ gibt mit $L(G) = L$.

**Definition 2.2.4** (Typ 3: Reguläre Grammatiken)**.**

- Eine Grammatik $G = (N, \Sigma, R, S)$ heißt regulär (oder rechtslinear), wenn für jede Regel $u \to v$ in $R$ gilt: $|u| \in N$ und $v \in \{\varepsilon\} \cup \Sigma \cup \Sigma N$.
- Eine Sprache $L$ heißt regulär, falls es eine reguläre Grammatik $G$ gibt mit $L(G)$.

**Satz 2.2.2** (Chomsky-Hierarchie)**.** Typ 3 $\subset$ Typ 2 $\subset$ Typ 1 $\subset$ Typ 0

## 2.3 Endliche Automaten

**Definition 2.3.1** (Endliche Automaten)**.** Ein nichtdeterministischer endlicher Automat (NEA) ist ein 5-Tupel $A = (Z, \Sigma, \delta, z_0, F)$ mit:

- $Z$ endliche Zustandsmenge
- $z_0 \in Z$ Anfangszustand
- $F \in Z$ Menge der akzeptierenden Zustände (Endzustände)
- $\Sigma$ endlichens Eingabealphabet

- $\delta : Z \times \Sigma \to 2^Z$ Überführungsrelation

Ist die Überführungsrelation eine Funktion, also $\delta : Z \times \Sigma \to Z$, so ist der endliche Automat deterministisch (DEA).

**Definition 2.3.2** (Sprache eines NEA)**.** Sei $A = (Z, \Sigma, \delta, z_0, F)$ ein NEA.

- Ein Lauf von $A$, gesteuert durch Eingabefolge $w = x_0 x_1 \ldots x_n \in \Sigma^*$ ist eine Folge von Zuständen $z_0, z_1, \ldots, z_{n+1}$ mit $z_{i+1} \in \delta(z_i, x_i)$, $0 \leq i \leq n$.

  Ist $w = \varepsilon$, so besteht der Lauf nur aus dem Anfangszustand $z_0$.

- Die von $A$ akzeptierte Sprache ist

$$\begin{aligned} L(A) = \{w \in \Sigma^* \mid & \text{ es gibt einen Lauf von } A \text{, gesteuert durch } w, \\ & \text{der zu einem Endzustand führt}\} \end{aligned}$$

**Definition 2.3.3** (Sprache eines DEA)**.** Sei $A = (Z, \Sigma, \delta, z_0, F)$ ein DEA.

- Ein Lauf von $A$, gesteuert durch Eingabefolge $w = x_0 x_1 \ldots x_n \in \Sigma^*$ ist die Folge von Zuständen $z_0, z_1, \ldots, z_{n+1}$ mit $z_{i+1} = \delta(z_i, x_i)$, $0 \leq i \leq n$.

  Ist $w = \varepsilon$, so besteht der Lauf nur aus dem Anfangszustand $z_0$.

- Die von $A$ akzeptierte Sprache ist

$$\begin{aligned} L(A) = \{w \in \Sigma^* \mid & \text{ der Lauf von } A \text{, gesteuert durch } w, \\ & \text{führt zu einem Endzustand}\} \end{aligned}$$

**Definition 2.3.4.** Zwei endliche Automaten sind äquivalent, wenn sie die gleiche Sprache akzeptieren.

**Satz 2.3.1.** Zu jedem NEA gibt es einen äquivalenten DEA.
Ist $A = (Z, \Sigma, \delta, z_0, F)$ ein NEA, dann ist der Potenzmengenautomat $A' = (2^Z, \Sigma, \delta', \{z_0\}, F')$ mit:

▷ $F' = \{M \subseteq Z \mid M \cap F \neq \varnothing\}$

▷ $\delta'(M, x) = \bigcup\limits_{z \in M} \delta(z, x)$

ein DEA mit $L(A) = L(A')$.

**Satz 2.3.2.** Zu jedem DEA $A$ gibt es eine reguläre Grammatik $G$ mit $L(G) = L(A)$.
Ist $A = (Z, \Sigma, \delta, z_0, F)$ ein DEA, dann ist $G = (Z, \Sigma, R, z_0)$ mit

▷ ist $z_0 \in F$, so ist $z_0 \to \varepsilon \in R$

▷ ist $\delta(z, x) = z'$, so ist $z \to xz' \in R$

  ist $z' \in F$, so ist außerdem $z' \to \varepsilon \in R$

eine reguläre Grammatik mit $L(G) = L(A)$.

**Satz 2.3.3.** Zu jeder regulären Grammatik $G$ gibt es einen NEA $A$ mit $L(A) = L(G)$.

## 2.4 Reguläre Ausdrücke

**Definition 2.4.1** (Syntax regulärer Ausdrücke)**.** Sei $\Sigma$ ein Alphabet.

1. $\varnothing$ und $\varepsilon$ sind reguläre Ausdrücke.
2. Für jedes $a \in \Sigma$ ist $a$ ein regulärer Ausdruck.
3. Sind $x$ und $y$ reguläre Ausdrücke, so sind auch
   $(x) + (y)$, $(x)(y)$ und $(x)^*$ reguläre Ausdrücke.
4. Weitere reguläre Ausdrücke gibt es nicht.

**Definition 2.4.2** (Semantik regulärer Ausdrücke)**.** Sei $x$ ein regulärer Ausdruck über $\Sigma$. Die von $x$ beschriebene Sprache $L(x)$ ist:

1. $L(\varnothing) = \varnothing$ und $L(\varepsilon) = \{\varepsilon\}$.
2. Für jedes $a \in \Sigma$ ist $L(a) = \{a\}$.
3. $L(x + y) = L(x) \cup L(y)$, $L(xy) = L(x)L(y)$ und $L(x^*) = L(x)^*$.

**Definition 2.4.3.** Zwei reguläre Ausdrücke $x, y$ über $\Sigma$ heißen äquivalent, wenn $L(x) = L(y)$ gilt, dass heißt wenn sie die gleiche Sprache beschreiben. Schreibweise: $x \equiv y$.

**Satz 2.4.1.** Zu jedem regulären Ausdruck $x$ gibt es eine reguläre Grammatik $G$ mit $L(G) = L(x)$.

**Satz 2.4.2.** Zu jedem DEA $A$ gibt es einen regulären Ausdruck $r$ mit $L(r) = L(A)$.

**Satz 2.4.3.** $L \subseteq \Sigma^*$ ist regulär genau dann, wenn $L$ aus den Sprachen $\varnothing$, $\{\varepsilon\}$ und $\{a\}$, $a \in \Sigma$, durch Vereinigung $\cup$, Konkatenation (Produkt) und Iteration $^*$ konstruiert werden kann.

## 2.5 Reguläre Sprachen

**Definition 2.5.1.** Ein vollständig definierter DEA $A$ ist minimal, falls für alle zu $A$ äquivalenten vollständig definierten DEA $A'$ gilt: $A'$ enthält mindestens soviele Zustände wie $A$.

**Definition 2.5.2** (Nerode-Relation)**.** Sei $L \subseteq \Sigma^*$. Auf der Wortmenge $\Sigma^*$ ist die Relation $\sim_L$ wie folgt definiert:

$$v \sim_L w \iff (\forall u \in \Sigma^* : vu \in L \Leftrightarrow wu \in L)$$

Die Relation $\sim_L$ ist eine Äquivalenzrelation auf $\Sigma^*$. Die Anzahl der Äquivalenzklassen heißt der Myhill-Nerode-Index von $\sim_L$. Schreibweise: $\textsc{Index}(\sim_L)$

**Satz 2.5.1** (Myhill & Nerode)**.** $L$ ist regulär $\iff$ $\textsc{Index}(\sim_L)$ ist endlich.

**Algorithmus 2.5.1** (Minimierung endlicher Automaten)**.**
*Sei auf der Zustandsmenge $Z$ eine Äquivalenzrelation wie folgt definiert:*

$$z \equiv z' \iff (\forall u \in \Sigma^* : \delta^*(z,u) \in F \Leftrightarrow \delta^*(z',u) \in F)$$

*Dann gilt $z \not\equiv z'$ genau dann, wenn:*

▷ $z \in F$, $z' \notin F$ *oder umgekehrt, oder*

▷ *es gibt ein $a \in \Sigma$ mit $\delta(z,a) \not\equiv \delta(z',a)$.*

*Sei $A = (Z, \Sigma, \delta, z_0, F)$ ein DEA ohne überflüssige Zustände.*

(1) *Markiere alle Paare $\{z, z'\}$ mit: $z \in F$, $z' \notin F$ oder umgekehrt*

(2) *Für unmarkierte Paare $\{z, z'\}$ prüfe, ob es ein $a \in \Sigma$ gibt mit markiertem Paar $\{\delta(z,a,), \delta(z',a)\}$.*
*Wenn ja, dann markiere auch $\{z, z'\}$.*

(3) *Wiederhole Schritt (2) bis kein neues markiertes Paar mehr entsteht.*

*Dann gilt $z \not\equiv z' \Leftrightarrow \{z, z'\}$ ist markiert. Den gesuchten Minimalautomat $A' = (Z', \Sigma, \delta', z_0', F')$ erhält man aus $A$, indem man die $\equiv$-äquivalenten Zustände zusammen zieht.*

▷ $Z' = \{[z] : z \in Z\}$, $z_0' = [z_0]$, $F' = \{[z] : z \in F\}$

▷ $\delta'([z], a) = [\delta(z,a)]$ *für alle* $[z] \in Z'$ *und* $a \in \Sigma$

**Lemma 2.5.1** (Pumping-Lemma für reguläre Sprachen)**.** Sei $L$ regulär. Dann existiert eine Zahl $n \in \mathbb{N}$, so dass jedes Wort $w \in L$ mit $|w| \geq n$ eine Zerlegung $w = xyz$ mit folgenden Eigenschaften besitzt:

(1) $|y| \geq 1$

(2) $|xy| \leq n$

(3) $xy^kz \in L$ für alle $k = 0, 1, 2, \ldots$

**Satz 2.5.2** (Abschlusseigenschaften)**.** Die Klasse aller regulären Sprachen über demselben Alphabet ist abgeschlossen unter Vereinigung, Produkt, Stern, Komplement und Schnitt: Sind $L$ und $M$ regulär, so auch $L \cup M$, $LM$, $L^*$, $\overline{L}$ $(= \Sigma^* \setminus L)$ und $L \cap M$ regulär.

**Beispiel 2.5.1** (Wichtige Entscheidungsprobleme regulärer Sprachen)**.**
***Das Wortproblem***
*Gegeben:* DEA $A = (Z, \Sigma, \delta, z_0, F)$ *und* $w \in \Sigma^*$.
*Frage:* *Ist* $w \in L(A)$*?*

***Äquivalenztest***
*Gegeben:* *DEA's* $A = (Z, \Sigma, \delta, z_0, F)$ *und* $A' = (Z', \Sigma, \delta', z_0', F')$.
*Frage:* *Ist* $L(A) = L(A')$*?*
***Leerheitstest***
*Gegeben:* *DEA* $A = (Z, \Sigma, \delta, z_0, F)$.
*Frage:* *Ist* $L(A) = \varnothing$*?*
***Endlichkeitstest***
*Gegeben:* *DEA* $A = (Z, \Sigma, \delta, z_0, F)$.
*Frage:* *Ist* $L(A)$ *endlich?*

## 2.6 Kontextfreie Sprachen

**Definition 2.6.1.** Eine kontextfreie Grammatik ist in Chomsky-Normalform (CNF), wenn alle Regeln von der Form $X \to YZ$ oder $X \to a$ für $X, Y, Z \in N$ und $a \in \Sigma$ sind.

**Lemma 2.6.1** (Pumping-Lemma für kontextfreie Sprachen)**.** Sei $L$ kontextfrei. Dann existiert eine Zahl $n \in \mathbb{N}$, so dass jedes Wort $w \in L$ mit $|w| \geq n$ eine Zerlegung $w = uvxyz$ mit folgenden Eigenschaften besitzt:

(1) $|vy| \geq 1$

(2) $|vxy| \leq n$

(3) $uv^kxy^kz \in L$ für alle $k = 0, 1, 2, \ldots$

**Satz 2.6.1** (Abschlusseigenschaften)**.** Die Klasse aller kontextfreien Sprachen über demselben Alphabet ist abgeschlossen unter Vereinigung, Produkt und Iteration: Sind $L$ und $M$ kontextfrei, so auch $L \cup M$, $LM$ und $L^*$ kontextfrei.

Die Klasse aller kontextfreien Sprachen ist jedoch nicht abgeschlossen unter Schnitt und Komplement.

**Definition 2.6.2** (Kellerautomat)**.** Ein nichtdeterministischer Kellerautomat (NKA) ist ein 7-Tupel $K = (Z, \Sigma, \Gamma, \delta, z_0, \$, F)$ mit:

- $Z$ endliche Zustandsmenge
- $z_0 \in Z$ Anfangszustand
- $F \subseteq Z$ Menge der akzeptierenden Zustände (Endzustände)
- $\Sigma$ endliches Eingabealphabet
- $\Gamma$ endliches Kelleralphabet
- $\$ \in \Gamma$ das Kellerbodensymbol
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to 2^{(Z \cup \Gamma^*)}$ Überführungsrelation

**Satz 2.6.2** (Sprache von NKA's)**.**

1) Jede von einem NKA akzeptierte Sprache ist kontextfrei.

2) Zu jeder kontextfreien Grammatik $G$ existiert ein NKA $K$ mit $L(G) = L_\varepsilon(K)$.

# 3 Semantik von Programmiersprachen

## Welche (rechnerische) Bedeutung hat ein Programm?

## 3.1 Operationelle Semantik

**Wie entsteht der Effekt eines Programms?**
Die Bedeutung eines Programms wird angegeben als Sequenz von Schritten, die Zustandstransformationen eines abstrakten Maschinenmodells beschreiben. Über Axiome und Schlussregeln kann so das Verhalten des Programms aus dem Verhalten seiner Komponenten hergeleitet werden. Formale operationelle Semantik ermöglicht die Argumentation über Terminierung, semantische Äquivalenz, Determiniertheit und Korrektheit einer Übersetzung.
**Anwendung:** Vergleich von Systemen, Model Checking

### 3.1.1 Natural Semantics (Big Step)

**Jedes Programm ist eine Zustandstransformation.**
$\langle C, s\rangle \to s'$ Gestartet in Zustand $s$ terminiert Programm $C$ und führt zu Zustand $s'$
Geeignet für Blockstrukturen.

### 3.1.2 Structural Operational Semantics (Small Step)

**Jedes Programm ist eine Folge elementarer Zustandstransformation.**

| | |
|---|---|
| $\langle C, s\rangle \to s'$ | Gestartet in Zustand $s$ terminiert Programm $C$ und führt zu Zustand $s'$ |
| $\langle C, s\rangle \to \langle C', s'\rangle$ | Die Ausführung von Programm $C$ im Zustand $s$ kann auf die Ausführung von Programm $C'$ im Zustand $s'$ reduziert werden |

Geignet für Nichtdeterminismus und Parallelität.

## 3.2 Denotationelle Semantik

**Was ist der Effekt eines Programms?**
Die Bedeutung eines Programms wird angegeben durch mathematische Formalismen wie Funktionen, Relationen und Gleichungen.
**Anwendung:** Statische Analyse (erhalten von Informationen über die Semantik eines Programms ohne dessen Ausführung)

### 3.2.1 Direct-Style-Semantik

- kompositional: Semantik eines Konstrukts allein auf der Basis seiner Teilkonstrukte definiert
- Kleinste Fixpunkte werden zur Definition der Semantik von Schleifen verwendet
- Fixpunkttheorie liefert Existenz solcher Fixpunkte

## 3.3 Axiomatische Semantik

**Was lässt sich über den Effekt eines Programms aussagen?**
Die Bedeutung eines Programms wird angegeben durch Logische Aussagen und Beweiskalküle.
**Anwendung:** Programmverifikation

### 3.3.1 Hoare-Kalkül

**Definition 3.3.1** (Hoare-Tripel)**.** $\{P\}\,S\,\{S\}$ heißt Hoare-Tripel, wobei $P$ die Vorbedingungen, $S$ ein Programmsegment und $Q$ die Nachbedingungen bezeichnet.

- Partielle Korrektheit
  Wenn $S$ terminiert und vor Abarbeitung von $S$ die Aussage $P$ gilt, so gilt nach der Abarbeitung von $S$ die Aussage $Q$.
- Totale Korrektheit
  Wenn vor Abarbeitung von $S$ die Aussage $P$ gilt, dann terminiert $S$ und nach der Abarbeitung von $S$ gilt die Aussage $Q$.

# 4 Formale Systeme

## Wie kann man Systeme modellieren?

## 4.1 Prozessalgebra

## 4.2 Petri-Netze

**Definition 4.2.1** (Petri-Netz)**.** Ein Petri-Netz ist ein 5-Tupel $N = (P, T, F, W, m_0)$ mit:

- $P$ ist eine endliche Menge von Stellen (*Places*)
- $T$ ist eine endliche Menge von Transitionen
- $P \cap T = \varnothing$
- $P \cup T$ ist die Menge der Knoten
- $F \subseteq (P \times T) \cup (T \times P)$ ist eine endliche Menge von Bögen (*flow relations*)
- $W : F \to \mathbb{N} \setminus \{0\}$ sind die Bogengewichte (*Weights*)
- $m_0 : P \to \mathbb{N}$ ist die Anfangsmarkierung

**Definition 4.2.2** (Bereiche von Knoten)**.**

(a) Der Vorbereich eines Knotens $x$ ist $\bullet x = \{y \mid (y, x) \in F\}$.

(b) Der Nachbereich eines Knotens $x$ ist $x\bullet = \{y \mid (x, y) \in F\}$.