# Theoretische Informatik

**Vorlesungsskript**

Mitschrift von Falk-Jonatan Strube

Vorlesung von Dr. Boris Hollas

13. April 2016

# Inhaltsverzeichnis

# Inhalte

Grundlage: Grundkurs Theoretische Informatik [**hollas2015grundkurs**]

- Formale Sprachen
  - Reguläre Sprachen
    - Endliche Automaten
    - Reguläre Ausdrücke
  - Nichtreguläre Sprachen
  - Kontextfreie Sprachen
    - Kellerautomaten
    - Grammatiken
- Berechenbarkeit
  - Halteproblem
- Komplexitätsklassen
  - $P$
  - $NP$
  - $NP$-vollständige Probleme

# 1 Automaten und Formale Sprachen

**Def.:** Ein Alphabet ist eine Menge $\Sigma \neq \emptyset$ (Symbole in $\Sigma$ – müssen nicht einzelne Buchstaben sein, auch Wörter usw. [bspw. „if“ oder „else“ im Alphabet der Programmiersprache C]).

**Def.:** Für $w_1, ..., w_n \in \Sigma$ ist $w = w_1...w_n$ ein Wort der Länge $n$.
$\Sigma^n$ beschreibt alle Worte mit der Länge genau $n$
Das Wort $\varepsilon$ ist das *leere Wort*.
Die Menge aller Wörter bezeichnen wir mit $\Sigma^*$ (einschließlich dem leeren Wort).

**Bsp.:** $\Sigma = \{a, b, c\} \quad \rightarrow \Sigma^* = \{\varepsilon, a, b, c, aa, ab, ac, aaa, ...\}$

**Def.:** Für Wörter $a, b \in \Sigma^*$ ist $ab$ die Konkatenation dieser Wörter.
Für ein Wort $w$ ist $w^n$ die $n$-fache Konkatenation von $w$, wobei $w^0 = \varepsilon$.

**Bemerkung:** Für alle $w \in \Sigma^*$ gilt $\varepsilon w = w = w\varepsilon$. $\varepsilon$ ist also das neutrale Element der Konkatenation.

**Def.:** Eine *formale Sprache* ist eine Teilmenge von $\sigma^*$.

**Def.:** Für Sprachen $A, B$ ist $AB = \{ab \mid a \in A,\ b \in B\}$ sowie $A^n = \prod_{i=1}^{n} A$, wobei $A^0 = \{\varepsilon\}$.

**Bemerkung:** $\emptyset, \varepsilon, \{\varepsilon\}$ sind unterschiedliche Dinge (leere Menge, leeres Wort, Menge mit leerem Wort).

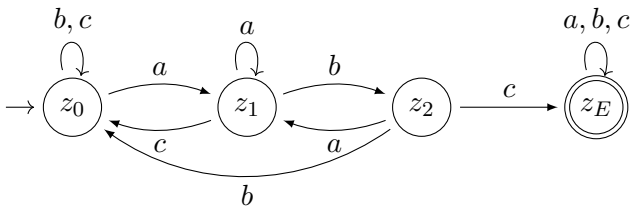
**Bemerkung:** $\Sigma^*$ lässt sich ebenfalls definieren durch $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$.

Ferner ist $\Sigma^+ = \Sigma^* - \{\varepsilon\}$.

## 1.1 Reguläre Sprachen

### 1.1.1 Deterministische endliche Automaten (DFA)

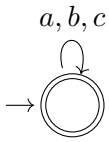
**Bsp.:**




(Pfeil zeigt auf Startzustand, Endzustand ist doppelt umrandet)

Dieser DFA akzeptiert alle Wörter über $\Sigma = \{a, b, c\}$, die $abc$ enthalten.
Deterministisch: Es gibt genau ein Folgezustand. Von jedem Knoten aus gibt es genau eine Kante für jedes Zeichen, nicht mehrere und nicht keine.

**Bsp.:**




Dieser DFA erkennt die Sprache $\{a, b, c\}^*$.

**Def.:** Ein DFA ist ein Tupel $\mathcal{M} = (Z, \Sigma, \delta, z_0, E)$

- $Z$: Menge der Zustände
- $\Sigma$: Eingabealphabet
- $\delta$: Überführungsfunktion $Z \times \Sigma \to Z$. Dabei bedeutet $\delta(z, a) = z'$, dass $\mathcal{M}$ im Zustand $z$ für das Zeichen $a$ in den Zustand $z'$ wechselt.
- $z_0 \in Z$: Startzustand
- $E$: Menge der Endzustände

$\delta$:


**Def.:** Die erweiterte Überführungsfunktion $\hat{\delta} : Z \times \Sigma^* \to Z$ ist definiert durch

$$\hat{\delta}(z, w) = \begin{cases} z & \text{für } w = \varepsilon \\ \hat{\delta}(\delta(z, a), x) & \text{für } w = ax \text{ mit } a \in \Sigma, x \in \Sigma^* \end{cases}$$

Dazu vergleichbarer C-Code:

```c
int δ̂(int z, char* w){
  if ( strlen(w) == 0 )
    return z;
  else
    return δ̂(δ(z, w[0]), w[1]);
```

Veranschaulichung:
ABB4
Die erweiterte Überführungsfunktion bestimmt den Zustand nach dem vollständingen Lesen eines Wortes.

**Bsp.:**
ABB5

$$\hat{\delta}(z_0, aaba) = \hat{\delta}(\delta(z_0, a), aba) =$$
$$\hat{\delta}(z_0, aba) = \hat{\delta}(\delta(z_0, a), ba) =$$
$$\hat{\delta}(z_0, ba) = \hat{\delta}(\delta(z_0, b), a) =$$
$$\hat{\delta}(z_E, a) = \hat{\delta}(\delta(z_E, a), \varepsilon) =$$
$$\hat{\delta}(z_E, \varepsilon) = z_E$$

Die von $\mathcal{M}$ *akzeptierte Sprache* ist $L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E\}$

### 1.1.2 Nichtdeterministischer endliche Automaten (NFA)

ABB23
NFA, der alles akzeptiert, was $abc$ enthält:
ABB24
Beispiel: Wort $abaabcab$

**Beispiel:** NFA, der alle Wörter akzeptiert, die auf $001$ enden:
ABB25
Akzeptierte Worte unter anderem: $01011001$, $001001$
Ein Wort wird vom NFA akzeptiert, wenn es einen Weg, ausgehend von einem Startzustand, gibt, mit den ein End-Zustand erreicht wird.
Der NFA „weiß“ nicht, welcher Pfad zu durchlaufen ist; diesen muss der Benutzer ermitteln (wie bei einer Straßenkarte).
Ein NFA lässt sich formalisieren durch ein Tupel $\mathcal{M} = (Z, \Sigma, \delta, S, E)$

- $Z$: Zustände
- $\Sigma$: Eingabealphabet
- $\delta$: $Z \times \Sigma \to \mathcal{P}(Z)$ Überführungsfunktion (bildet ab in Potenzmenge von $Z$)
- $S$: Menge der Startzustände
- $E$: Menge der Endzustände

Dabei bedeutet $\delta(z, a) \ni z'$, dass der NEA im Zustand $z$ für die Eingabe $a$ die Möglichkeit besitzt, in den Zustand $z'$ zu wechseln.

### 1.1.3 Umwandlung eines NFA in einen DFA

Wir wollen den NFA
ABB 27
in einen DFA umwandeln. Der Startzustand des DFA besteht aus den Startzuständen des NFA:
ABB 28
Betrachten die Folgezustände für $a \in \Sigma$:
ABB 29
nächster Schritt:
ABB 30
nächster Schritt:
ABB 31
weitere Schritte:
$z_0, b : \{z_0\}$
$z_1, b : \{z_2\}$
ABB 32
$z_0, c : \{z_0\}$
$z_1, c : \{\}$
ABB 33
usw.:
ABB 34

Wenn ein Zustand des DFA einen Endzustand des NFA enthält, so ist es ein Endzustand.
Der auf diese Weise erhaltene DFA kann Zustände enthalten, die sich zu einem Zustand zusammen fassen lassen. Mit dem Algorithmus Minimalautomat lässt sich ein DFA konstruieren, der minimal bezüglich der Anzahl seiner Zustände ist. Der Minimalautomat ist eindeutig, d.h. Minimalautomaten unterscheiden sich höchstens in der Benennung der Zustände.

### 1.1.4 Reguläre Ausdrücke

**Def.:** Sei $\Sigma$ ein Alphabet. Ein *regulärer Ausdruck* $E$ sowie die durch $E$ *erzeugte Sprache* $L(E)$ sind induktiv definiert:

- $\emptyset$ ist ein regulärer Ausdruck und $L(\emptyset) = \emptyset$.
  Bsp.:
  ABB35
- Für $a \in \Sigma \cup \{\varepsilon\}$ ist $a$ ein regulärer Ausdruck und $L(a) = \{a\}$.
- Für reguläre Ausdrücke $E_1, E_2$ sind $(E_1|E_2),\ (E_1E_2),\ (E_1^*)$ reguläre Ausdrücke (hier: $|$ =„oder“) und $L(E_1|E_2) = L(E_1) \cup L(E_2),\ L(E_1E_2) = L(E_1)L(E_2),\ L(E_1^*) = L(E_1)^*$ die davon erzeugten Sprachen:

| Ausdruck | Sprache | |
|---|---|---|
| $E_1\|E_2$ | $L(E_1\|E_2)$ | $= L(E_1) \cup L(E_2)$ |
| $E_1E_2$ | $L(E_1E_2)$ | $= L(E_1)L(E_2)$ |
| $E_1^*$ | $L(E_1^*)$ | $= L(E_1)^*$ |

  Hinweis: $E^+ = EE^*$, $E? = \varepsilon|E$

  Wenn $E_1, E_2$ regulär, dann auch $(E_1|E_2),\ (E_1E_2),\ (E_1^*)$ regulär

**Bsp.:**

- $L((0|1)^*) = (L(0|1))^* = (L(0) \cup L(1))^* = (\{0\} \cup \{1\})^* = \{0,1\}^*$
- Regulärer Ausdruck über $\Sigma = \{a,b,c\}$, der die gleiche Sprache erzeugt wie der DFA aus dem letzten Automaten-Beispiel:
  $L((a|b|c)^*abc(a|b|c)^*) = \{a,b,c\}^*\{abc\}\{a,b,c\}^*$

**Satz:** Reguläre Ausdrücke erzeugen genau die regulären Sprachen.

**Skizze:** Umwandlung eines regulären Ausdrucks in einen endlichen Automaten.

- $\emptyset$: ABB 40
- $a \in \Sigma$: ABB 41 1.
  $\varepsilon$: ABB 41 2.
- Seien $E_1, E_2$ reguläre Ausdrücke und $\mathcal{M}_1, \mathcal{M}_2$ DFAs mit $L(E_1) = L(\mathcal{M}_1), L(E_2) = L(\mathcal{M}_2)$.
  - $E_1|E_2$: $\mathcal{M}_1, \mathcal{M}_2$ sind zusammen ein NFA, der $L(\mathcal{M}_1) \vee L(\mathcal{M}_2)$ erkennt.
    Bsp.: $E_1 = a, E_2 = b$
    ABB 42

- $E_1E_2$: $\mathcal{M}_1, \mathcal{M}_2$ müssen hintereinander geschaltet werden, wobei ggf. neue Kanten eingefügt werden müssen. Dazu betrachtet man die Kante nach der neuen Verbindung und erzeugt dem entsprechend die Übergangskanten.
  ABB 43
- $E_1^*$: Es müssen Kanten zurück zum Startzustand eingefügt werden
  ABB 44

Der Beweis für die umgekehrte Richtung (DFA $\rightarrow$ reg. Ausdruck) ist schwierig.

**Bsp.:**

- $E = 0(0|1)^*$
  ABB 45
- ABB 46
  Beobachtungen:
  - um zum Endzustand zu kommen, braucht man eine $1$.
  - vor der $1$ kann $\varepsilon$ stehen, oder beliebig viele $0$en der $1$en.

  $\Rightarrow E = (0|1)^*1$

### 1.1.5 Das Pumping-Lemma

Wenn ein DFA ein Wort akzeptiert, das mindestens so lang ist wie die Anzahl seiner Zustände, dann muss er einen Zustand zweimal durchlaufen (Schubfachprinzip). Daraus folgt, dass der DFA dabei eine Schleife durchläuft.

**Bsp.:**
ABB 47
Für $x = abcdecfg$ durchläuft der Automat eine Schleife: $x = ab\boxed{cde}cfg$. Daher akzeptiert der DFA auch alle Wärter $ab(cde)^kcfg$ für $k \geq 0$.

**Satz:** (Pumping Lemma)
Für jede reguläre Sprache $L$ gibt es ein $n > 0$ ($n$: Anzahl Zustände des Minimalautomaten), so dass es für alle Wörter $x \in L$ mit $|x| \geq n$ eine Zerlegung $x = uvw$ gibt (in vorherigem Bsp.: $u = ab,\ v = cde\ w = cfg$), so dass gilt:

1.) $|v| \geq 1$

2.) $|uv| \leq n$ ($u, w$ können auch $\varepsilon$ sein)

3.) $uv^kw \in L$ für alle $k \geq 0$.

Ohne Einschränkung ist $n$ die Anzahl Zustände des Minimalautomaten.
$\Rightarrow \forall$ regulären Sprachen $L \quad \exists\ n > 0 \quad \forall\ x \in L,\ |x| \geq n \quad \exists\ u, v, w$ mit $x = uvw$ und $|v| \geq 1, |uv| \geq n \quad \forall\ k \geq 0\ uv^kw \in L$.
Das Pumping-Lemma lässt sich nutzen, um zu zeigen, dass eine Sprache nicht regulär ist.


Mitschrift von
Falk-Jonatan Strube

**Bsp.:** Wir zeigen, dass $L = \{a^n b^n | n \in \mathbb{N}\}$ nicht regulär ist.
Problemstellung: Der Automat kann sich das $n$ nicht „merken“, um nach $n$ $a$s wieder $n$ $b$s zu erzeugen.
Beweis (Widerspruch):

- Angenommen, $L$ sei regulär.
- Nach Pumping-Lemma gibt es dann ein $n > 0$, so dass sich alle $x \in L$ mit $|x| \geq n$ gemäß Pumping-Lemma zerlegen lassen.
- Sei $x = a^n b^n$.
- Angenommen $v$ enthalte ein $b$, dann wäre $|uv| > n$.
  Aus $|uv| \leq n$ folgt aber, dass $v$ kein $b$ enthält. aus $|v| \geq 1$ folgt, dass $v$ mindestens ein $a$ enthält.
  ABB 48
- Das Wort $uw$ enthält daher weniger $a$s als $b$s und kann somit nicht in $L$ enthalten sein (denn $w$ enthält $b^n$, da $v$ mindestens ein $a$ enthält, ist durch $uw$ mindestens ein $a$ „verloren gegangen“: $uw = a^{n-|v|}b^n$) und ist deshalb nicht in $L$ enthälten, Widerspruch ↯ #

Vorgehen:

- ist regulär
- Def. Pumping Lemma
- $x$ finden (gilt für alle $x$, also ein günstiges $x$ aussuchen, mit dem sich Beweis führen lässt)
- durch 1.) und/oder 2.) einschränken
- durch 3.) zum Widerspruch führen

**Bsp.:** Wir zeigen, dass $L = \{zz | z \in \{a, b\}^*\}$ nicht regulär ist.
Intuitiver Hinweis: Kann nicht regulär sein, da sich der Automat nicht merken kann, wie viele $a$s und $b$s im ersten $z$ gelesen wurden, um dann das gleiche im zweiten $z$ zu fabrizieren.
Beweis:

- Angenommen, $L$ ist regulär.
- Dann gibt es ein $n > 0$, so dass sich alle $x \in L$ mit $|x| \geq n$ zerlegen lassen gemäß Pumping-Lemma.
- Sei $x = a^n b a^n b$.
- Wegen $|uv| \leq n$ und $|v| \geq 1$ besteht $v$ aus mindestens einem $a$.
  ABB 61
- Dann enthält $uw = a^{n-|v|}ba^n b$ (für $k = 0$) weniger $a$s in der vorderen Hälfte als in der hinteren Hälfte. Da sich $uw$ deshalb nicht in die Form $zz$ mit $z \in \{a, b\}^*$ brigen lässt, ist $uw \notin L$, Widerspruch!

**Satz:** Seien $L$ regulär und $n$ die Anzahl Zustände des Minimalautomaten zu $L$. Dann gilt $|L| = \infty$ genau dann, wenn es ein $x \in L$ gibt mit $n \leq |x| < 2n$.
Beweis:
($\Leftarrow$):
Gemäß Pumping Lemma gibt es eine Zerlegung $x = uvw$ mit $|v| \geq 1$ und $uv^k w \in L$ für alle $k \in \mathbb{N}_0$ ($\mathbb{N}$ ist unendlich).
Daraus folg $|L| = \infty$.
($\Rightarrow$):
Da es nur endlich viele Wörter $x$ mit $|x| < n$ gibt, gibt es ein $x \in L$ mit $|x| \geq n$.
Sei daher $x \in L$ mit $|x| \geq n$ und $|x|$ minimal.
Gemäß PL lässt sich $x$ zerlegen in $x = uvw$.
Da $uw \in L$ und $|x|$ minimal ist, gilt $|uw| < n$.
Wegen $|x| \geq \underbrace{|uv|}_{<n \text{ gemäß PL}} + \underbrace{|uw|}_{<n \text{ Satz zuvor}} < n + n = 2n$ folgt die Behauptung $n \leq |x| \leq 2n$.
Regulärer Ausdruck: Generator
Automat: Validator

## 1.2 Kontextfreie Sprachen

### 1.2.1 Kellerautomaten (PDA)

Ein Kellerautomat (Pushdown Automaton, PDA) besitzt gegenüber einem NFA zwei zusätzliche Eigenschaften:

- Es gibt $\varepsilon$-Übergänge.
- Er besitzt einen Stack, auf dem Zeichen abgelegt oder von dem Zeichen gelesen werden können.

Zur graphischen Darstellung von PDAs verwenden wir eine erweiterte Automatennotation:
ABB62
Unten auf dem Stack liegt das Symbol #. Dies ist das einzige Symbol, das sich zu Beginn einer Rechnung auf dem Stack befindet.

**Bsp.:** PDA, der $\{a^n b^n | n \in \mathbb{N}\}$ akzeptiert.
ABB63
Wir erlauben nun, dass der PDA in einem Schritt auch mehrere Zeichen auf den Stack schreibt. Dazu erweitern wir die graphische Notation wie folgt:
ABB 68

**Def.:** Ein PDA ist ein Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$

- $Z$: Zustände
- $\Sigma$: Eingabealphabet
- $\Gamma$: Stackalphabet
- $\delta$: $Z \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Z \times \Gamma_\varepsilon)$, wobei $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$, $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$
- $z_0 \in Z$: Startzustand
- $\# \in \Gamma$: Unterstes Stackzeichen

- $E \in Z$: Endzustände

ABB 69
$a \in \Sigma \cup \{\varepsilon\}$
$\gamma \in \Gamma \cup \{\varepsilon\}$
$\gamma' \in \Gamma \cup \{\varepsilon\}$

**Def.:** Die von einem PDA $M$ akzeptierte Sprache $L(M)$ ist die Menge aller $x \in \Sigma^*$, für die gilt: Der PDA $M$ kann, ausgehend vom Startzustand und dem initialen Stackzustand $\#$, durch das Lesen des Wortes $x$ einen Endzustand erreichen.

### 1.2.2 Kontextfreie Grammatiken

Eine kontextfreie Grammatik beschreibt, wie durch das Ersetzen von variablen Wörter der Sprache erzeugt werden können. Jede Ersetzungsregel hat die Form „linke Seite $\rightarrow$ rechte Seite“ (linke Seite der Regel kann ersetzt werden durch die rechte Seite), wobei „linke Seite“ eine Variable ist.

Beginnend mit dem Startsymbol werden solange Ersetzungsregeln angewendet, bis alle Variablen durch Terminalsymbole (Elemente aus $\Sigma$) ersetzt wurden.

**Bsp.:**

- Satz $\rightarrow$ NP VP*
- NP $\rightarrow$ Artikel Nomen
- Artikel $\rightarrow$ die
- Nomen $\rightarrow$ Katze
- Nomen $\rightarrow$ Maus
- VP $\rightarrow$ Verb NP
- Verb $\rightarrow$ jagt

Satz $\Rightarrow$ NP VP $\Rightarrow$ Artikel Nomen VP $\Rightarrow$ Artikel Nomen Verb NP $\Rightarrow$ Artikel Nomen Verb Artikel Nomen $\Rightarrow \ldots \Rightarrow$ die Katze jagt die Maus
Syntax dazu:
ABB 71

**Def.:** Eine kontextfreie Grammatik ist ein Tupel $\sigma = (V, \Sigma, P, S)$

- $V$: Endliche Menge der Variablen oder Nonterminalzeichen
- $\Sigma$: Alphabet oder Terminalzeichen $V \cap \Sigma = \emptyset$
- $P$: Regeln oder Produktionen der Form $u \rightarrow v$ mit $u \in V$ und $v \in (V \cup \Sigma)^*$
- $S \in V$

Für $x, y \in (V \cup \Sigma)^*$ schreiben wir $x \Rightarrow y$, wenn sich durch das Ersetzen einer Variablen in $x$ die Satzform $y$ erzeugen lässt.

*Nominalphase, Verbalphase

**Bsp.:** die Nomen Verb $\Rightarrow$ die Katze Verb

Die reflexive und transitive Hülle der Relation $\Rightarrow$ bezeichnen wir mit $\Rightarrow^*$. Umgangssprachlich: durch $\Rightarrow^*$ werden nicht alle $\Rightarrow$-Umformungen dargestellt, sondern teils übersprungen.

**Bsp.:**
Satz $\Rightarrow^*$ die Katze VP,
Satz $\Rightarrow^*$ die Katze jagt die Maus.

**Def.:** Die von einer Grammatik erzeugte Sprache ist $L(G) = \{w \in \Sigma^* | S \Rightarrow^* w\}$.