# Automaten, formale Sprachen und Entscheidbarkeit

**Zusammenfassung**

Fabian Damken

8. November 2023

TECHNISCHE UNIVERSITÄT DARMSTADT

# Inhaltsverzeichnis



## 0.1 Definitionen

### 0.1.1 Deterministischer Endlicher Automat (DFA)

**Formelle Definition** Ein deterministischer endlicher Automat $\mathcal{A}$ ist definiert durch das 5-Tupel $\mathcal{A} = (\Sigma, Q, q_0, \delta, A)$ mit

- dem endlichen Terminalalphabet $\Sigma$, $\Sigma \neq \emptyset$.
- der endlichen Zustandsmenge $Q$.
- dem Startzustand $q_0$.

- der Übergangsfunktion $\delta : Q \times \Sigma \to Q$ (Zustand, Gelesenes Terminalsymbol $\to$ Nächster Zustand).
- den akzeptierenden Zuständen $A \subseteq Q$.

**Darstellung** Ein (deterministischer) endlicher Automat ist dargestellt als ein Transitionssystem (gerichteter und gewichteter Graph) mit den Zuständen als Knoten, den Übergängen als Kanten (wobei $\delta(q, a) \coloneqq q'$ den Übergang darstellt von dem Zustand $q \in Q$ bei erhalt des Terminalsymbols $a \in \Sigma$ zu dem Zustand $q' \in Q$). Die Zustandsnamen werden die in die Kreise der Knoten geschrieben und die Terminalsymbole an die Kanten des Transitionssystems. Der Startzustand wird mit einem einfachen, quelllosen, Pfeil markiert. Die akzeptierenden Zustände werden doppel einkekreist.

**Eigenschaften**

Ein endlicher Automat ist deterministisch
$\iff \forall q \in Q : \forall a \in \Sigma : \exists^1 q' \in Q : \delta(q, a) = q'$
$\iff$ Jeder Zustand $q \in Q$ hat genau eine ausgehende Transition für jedes $a \in \Sigma$.

Jeder deterministische endliche Automat ist ein nichtdeterministischer endlicher Automat (DFA $\implies$ NFA).

**Verhalten** Der Automat startet bei dem Startzustand und ließt Schritt für Schritt ein Symbol des Eingabewortes ein. Mit jedem gelesenen Symbol wechelt der Zustand nach den definierten Übergängen. Sobald das Wort komplett eingelesen wurde (das heißt es sind keine weiteren Terminalsymbole mehr verfügbar), terminiert der Automat. Ist der Automat in einem akzeptierenden Zustand, so wird das Wort akzeptiert. Andernfalls wird das Wort nicht akzeptiert.

### 0.1.2 Nichtdeterministischer Endlicher Automat (NFA)

**Formelle Definition** Ein (nichtdeterministischer) endlicher Automat $\mathcal{A}$ ist definiert durch das 5-Tupel $\mathcal{A} = (\Sigma, Q, q_0, \Delta, A)$ mit

- dem endlichen Terminalalphabet $\Sigma$, $\Sigma \neq \emptyset$.
- der endlichen Zustandsmenge $Q$.
- dem Startzustand $q_0 \in Q$.
- der endlichen Übergangsrelation $\Delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ (Zustand, Gelesenes Terminalsymbol oder $\varepsilon$, Nächster Zustand).
- den akzeptierenden Zuständen $A \subseteq Q$.

**Darstellung** Die Darstellung eines nichtdeterministischen Automaten entspricht der Darstellung eines deterministischen Automaten (siehe 0.1.1).

**Eigenschaften** Ein nichtdeterministischer Automat hat keine speziellen Eigenschaften.

**Verhalten** Ein nichtdeterministischer Automat verhält sich wie ein deterministischer Automat (siehe 0.1.1) mit der Ausnahme, das er mehrere Zustände gleichzeitig annehmen kann (bei doppelten Transitionen). Kann eine Ausführung nicht weitergeführt werden (das heißt es existieren keine Transitionen mehr für das gelesene Terminalsymbol aus dem aktuellen Zustand), wird diese Ausführung verworfen/nicht akzeptiert. Ist keine weitere Ausführung mehr vorhanden, wird das Wort nicht akzeptiert. Bei einer $\varepsilon$-Transition wechselt der Zustand immer, beziehungsweise es wird ein neuer Ausführungszweig begonnen (der NFA befindet sich in zwei Zuständen gleichzeitig).

### 0.1.3 Grammatik

**Formelle Definition** Eine Grammatik $\mathcal{G}$ ist definiert durch das 4-Tupel $\mathcal{G} = (\Sigma, V, P, X_0)$ mit

- dem endlichen Terminalalphabet $\Sigma$, $\Sigma \neq \emptyset$.
- der endlichen Variablenmenge $V$, $V \cap \Sigma = \emptyset$.
- der endlichen Menge von Produktionsregeln $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ (Quellsymbol, Zielsymbole oder $\varepsilon$).
- der Startvariable $X_0 \in V$.

**Darstellung** Eine Grammatik wird formell durch das oben beschriebene 4-Tupel definiert. Die Produktionsregeln werden als Tabellenartige Liste definiert, wobei die Quellsymbole (Variablensymbole/Terminalsymbol) links stehen und durch einen Pfeil ($\rightarrow$) von den Zielsymbolen (Variablensymbol/Terminalsymbol) getrennt werden. Dabei kann ein Quellsymbol zu mehreren Zielsymbolen werden, diese werden durch einen vertikalen Strich (|) getrennt. Ein $\varepsilon$ als Zielsymbol entfernt das Quellsymbol.

**Eigenschaften** Eine Grammatik kann nur zur Konstruktion von Wörtern, aber nicht zur Entscheidung von Wörtern (ob diese in der Sprache liegen) genutzt werden.

**Verhalten** Bei der Konstruktion eines Wortes unter Nutzung einer Grammatik werden die Variablen so lange ersetzt, bis keine Variablen mehr vorhanden sind. das erhaltene Wort liegt in der von der Grammatik erzeugten Sprache.

### 0.1.4 Kellerautomat

**Formelle Definition** Ein (nichtdeterministischer) Kellerautomat $\mathcal{P}$ ist definiert durch das 7-Tupel $\mathcal{P} = (\Sigma, Q, q_0, \Delta, A, \Gamma, \#)$ mit

- dem endlichen Terminalalphabet $\Sigma$, $\Sigma \neq \emptyset$.
- der endlichen Zustandsmenge $Q$.
- dem Startzustand $q_0 \in Q$.
- der endlichen Übergangsrelatation $\Delta \subseteq Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^* \times Q$ (Zustand, Gelesenes Kellersymbol, Gelesenes Terminalsymbol oder $\varepsilon$, Zu schreibende Kellersymbole oder $\varepsilon$, Nächster Zustand).
- den akzeptierenden Zuständen $A \subseteq Q$.

- dem endlichen Kelleralphabet $\Gamma$.
- dem Startkellersymbol $\# \in \Gamma$.

Ein $\varepsilon$ in der Übergangsrelation stellt da, das kein Symbol gelesen werden muss (das heißt die Transition geschieht immer) oder dass kein Symbol in den Kellerspeicher geschrieben wird.

**Darstellung** Eine Kellerautomat wird dargestellt als ein nichtdeterministischer Automat und einem so genannten Kellerspeicher (Stack).

**Verhalten** Ein Kellerautomat verhält sich wie ein nichtdeterministischer Automat mit dem Zusatz eines Kellerspeichers. Mit jedem Einlesen eines Terminalsymbols beziehungsweise jedem Auführen eines Übergangs wird ein Kellersymbol aus dem Kellerspeicher gelesen und gelöscht. Dieses wird zusammen mit dem gelesenen Terminalsymbol als Bedingung genutzt welcher Übergang ausgeführt wird. Um ein oder mehrere Kellersymbole in den Kellerspeicher zu schreiben, werden diese in dem Übergang angegeben. Die angegebenen Kellersymbole ersetzen das gelesene Kellersymbol. Es kann unter anderem $\varepsilon$ angegeben werden, wodurch nichts in den Kellerspeicher geschrieben wird. Es können mehrere Kellersymbole in den Kellerspeicher geschrieben, wobei diese von rechts in den Speicher geschrieben werden (das heißt das nächste Symbol welches gelesen wird ist das rechteste Symbol in der Auflistung). Hierbei werden die Kellersymbole einfach hintereinander geschrieben.

Der Kellerautomat ist akzeptiert ein Wort genau dann, wenn der Automat in einem akzeptierenden Zustand steht und der Kellerspeicher leer ist. Ist der Kellerspeicher leer, das Wort allerdings noch nicht abgearbeitet, terminiert der Kellerautomat und akzeptiert das Wort nicht.

### 0.1.5 Turingmaschine

**Formelle Definition** Eine (deteministische) Turingmaschine $\mathcal{M}$ ist definiert durch das 6-Tupel $\mathcal{M} = (\Sigma, Q, q_0, \delta, q^+, q^-)$ mit

- dem endlichen Terminalalphabet $\Sigma$, $\Sigma \neq \emptyset$.
- der endlichen Zustandsmenge $Q$.
- dem Startzustand $q_0 \in Q$.
- der endlichen Übergangsfunktion $\delta : Q \times (\Sigma \cup \{\square\} \cup \Gamma) \to ((\Sigma \cup \{\square\}) \times \{<, \circ, >\}) \times Q$
- dem akzeptierenden Endzustand $q^+$.
- dem verwerfenden Endzustand $q^-$.

und dem Hilfsalphabet $\Gamma$, $\Gamma \cap \Sigma = \emptyset$.

**Darstellung** Eine Turingmaschine wird dargestellt als ein nichtdeterministischer Automat und einem unendlichen Speicherband.

**Verhalten** Eine Turingmaschine verhält sich wie ein nichtdeterministischer Automat mit dem Zusatz eines unendlichen Speicherbandes welches von einem Schreib- und Lesekopf beschrieben und gelesen werden kann. Mit jeder Transition beziehungsweise jedem Ausführen eines Übergangs wird das Terminalsymbol welches sich aktuell unter dem Kopf befindet gelesen und gelöscht und auf Basis von diesem und dem aktuellen Zustand entschieden, welcher Übergang ausgeführt wird. Das Wort befindet sich auf dem Band, eine Speicherzelle rechts von der initialen Kopfposition. Eine leere Speicherzelle wird formal als Kasten (□) dargestellt, dieses kann auch gelesen werden. Während dem Ausführen eines Übergangs wird ein neues Symbol (aus dem Terminalalphabet, dem Hilfsalphabet oder □) auf das Band geschrieben und der Kopf bewegt. Die möglichen Bewegungen sind links ($<$), rechts ($>$) und keine Bewegung ($\circ$), wobei sich der Kopf jeweils um eine Speicherzelle verschiebt. Ferner wechelt der Zustand in den angegebenen Zustand.

Eine Turingmaschine akzeptiert ein Wort genau dann, wenn der Automat in den akzeptierenden Endzustand wechselt und akzeptiert das Wort nicht, wenn der Automat in den verwerfenden Endzustand wechselt. Solange der Automat nicht einen einen dieser beiden Zustände wechselt, läuft er unendlich weiter.

## 0.2 Endliche Automaten / Reguläre Sprachen

### 0.2.1 Satz von Kleene

Die Menge der von endlichen Automaten erkannten Sprachen ist identisch zu der Menge der von regulären Ausdrücken erkannten Sprachen.

Hierzu werden die Formeln in **??** rekursiv angewendet um den regulären Ausdruck zu produzieren. Hierfür müssen alle Zustände durchgehend von $1$ bis $n$ (Anzahl der Zustände) nummeriert sein. $L^k l, m$ entspricht dem regulären Ausdruck, um von dem Zustand $l$ in den Zustand $m$ zu Wechseln unter Nutzung der Zustände $\{q_i : 1 \leq i \leq k\}$ (das heißt es dürfen nur Zustände in dieser Menge während des Laufes genutzt werden). Durch rekursives Anwenden des Satzes von Kleene vereinfachen sich alle regulären Ausdrücke zu regulären Ausdrücken der Form $L^0_{l,m}$, was den Rekursionsanker/den Induktionsanfang darstellt.

$$L^0_{l,m} = \begin{cases} \{a \in \Sigma : \delta(l,a) = m\} & \text{für } l \neq m \\ \{\varepsilon\} \cup \{a \in \Sigma : \delta(l,a) = l\} & \text{für } l = m \end{cases}$$

$$L^k_{l,m} = L^{k-1}_{l,m} \cup L^{k-1}_{l,k} \cdot (L^{k-1}_{k,k})^* \cdot L^{k-1}_{k,m}$$

Abbildung 1: Satz von Kleene

### 0.2.2 Pumping Lemma für reguläre Sprachen

Für jede reguläre Sprache $L \subseteq \Sigma^*$ existiert ein $n \in \mathbb{N}$, sodass für jedes Wort $x \in L$ mit $|x| \geq n$ eine Zerlegung $x = u \cdot v \cdot w$ mit $v \neq \varepsilon$ existiert, sodass für alle $m \in \mathbb{N}$ gilt, dass auch $u \cdot v^m \cdot w \in L$ gilt. Dabei kann $|u \cdot v| \leq n$ gelten.

### 0.2.3 Satz von Myhill-Nerode

Für eine Sprache $L \subseteq \Sigma$ gilt: $L$ regulär $\iff \sim_L$ hat endlichen Index

### 0.2.4 Produktkonstruktion

Seien $\mathcal{A}_1 = (\Sigma, Q_1, s_1, \delta_1, A_1)$ und $\mathcal{A}_2 = (\Sigma, Q_2, s_2, \delta_2, A_2)$ zwei DFAs über dem Alphabet $\Sigma$.

Mit Hilfe der Produktkonstruktion ist es nun möglich, die Vereinigungsmenge ($\mathcal{A}_1 \cup \mathcal{A}_2$), die Schnittmenge ($\mathcal{A}_1 \cap \mathcal{A}_2$) und die Differenzmenge ($\mathcal{A}_1 \setminus \mathcal{A}_2$) zu bilden.

Für den durch Produktkonstruktion konstruierten DFA $\mathcal{A} = (\Sigma, Q, s, \delta, A)$ gilt:

$$
\begin{aligned}
Q &= Q_1 \times Q_2 \\
s &= (s_1, s_2) \\
\delta &: Q \times \Sigma \to Q : ((q_1, q_2), a) \mapsto (\delta_1(q_1, a), \delta_2(q_2, a)) \\
A &= \begin{cases} \{(q_1, q_2) \in Q : q_1 \in A_1 \vee q_2 \in A_2\} & \text{für } A_1 \cup A_2 \\ \{(q_1, q_2) \in Q : q_1 \in A_1 \wedge q_2 \in A_2\} & \text{für } A_1 \cap A_2 \\ \{(q_1, q_2) \in Q : q_1 \in A_1 \wedge q_2 \notin A_2\} & \text{für } A_1 \setminus A_2 \end{cases}
\end{aligned}
$$

### 0.2.5 Potenzmengenkonstruktion

Sei $\mathcal{A} = (\Sigma, Q, s, \Delta, A)$ ein NFA über dem Alphabet $\Sigma$.

Mit Hilfe der Potenzmengenkonstruktion ist es nun möglich, den NFA $\mathcal{A}$ in einen DFA zu überführen.

Für den durch Potenzmengenkonstruktion konstruierten DFA $\mathcal{A}^{\text{det}} = (\Sigma, \hat{Q}, \hat{s}, \hat{\delta}, \hat{A})$ gilt:

$$
\begin{aligned}
\hat{Q} &= \mathcal{P}(Q) \\
\hat{s} &= \{s\} \\
\hat{\delta}(S, a) &\coloneqq \{q' \in Q : (q, a, q') \in \Delta \text{ für mindestens ein } q \in S\} \\
\hat{A} &= \{S \subseteq Q : S \cap A \neq \emptyset\}
\end{aligned}
$$

**Beispiel** Gegeben ist der NFA $\mathcal{A} = (\Sigma, Q, s, \Delta, A)$ mit $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2, q_3\}$, $s = q_0$, $A = \{q_3\}$ und den Übergängen in der Tabelle **??**.

| $\delta$ | a | b |
|---|---|---|
| $q_0$ | $q_0$, $q_2$ | $q_1$ |
| $q_1$ | $q_3$ | - |
| $q_2$ | $q_3$ | $q_2$ |
| $q_3$ | - | - |

Abbildung 2: NFA: Übergängen

Um diesen aus diesem NFA nun einen DFA $\mathcal{A}^{\text{det}}$ zu konstruieren, wird die Tabelle **??** Zeile für Zeile ausgefüllt.

Somit ergibt sich $\mathcal{A}^{\text{det}} = (\Sigma, \hat{Q}, \hat{s}, \hat{\Delta}, \hat{A})$ mit $\hat{Q} = \{\{0\}, \{0, 2\}, \{1\}, \emptyset, \{3\}, \{0, 2, 3\}, \{1, 2\}, \{2\}\}$, $\hat{s} = \{0\}$, $A = \{\{3\}, \{0, 2, 3\}\}$ und den Übergängen in der Tabelle **??**.

### 0.2.6 DFA Minimalisierung

Sei $\mathcal{A} = (\Sigma, Q, q_0, \Delta, A)$ ein DFA über dem Alphabet $\Sigma$.

Das Ziel der Minimalisierung ist es nun, die Zustandsmenge $Q$ zu minimieren.

Um einen DFA zu minimieren, sind die folgenden Schritte nötig:

| $\delta$ | a | b |
|---|---|---|
| $\{0\}$ | $\{0,2\}$ | $\{1\}$ |
| $\{0,2\}$ | $\{0,2,3\}$ | $\{1,2\}$ |
| $\{1\}$ | $\{3\}$ | $\emptyset$ |
| $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\{3\}$ | $\emptyset$ | $\emptyset$ |
| $\{0,2,3\}$ | $\{0,2,3\}$ | $\{1,2\}$ |
| $\{1,2\}$ | $\{3\}$ | $\{2\}$ |
| $\{2\}$ | $\{3\}$ | $\{2\}$ |

Abbildung 3: DFA: Übergänge

1. Den akzeptierenden Zuständen wird eine Äquivalenzklasse zugeordnet und den nicht-akzeptierenden Zuständen wird eine andere Äquivalenzklasse zugeordnet.
2. Für jeden Zustand wird aufgeschrieben, in welche Äquivalenzklasse der Automat übergeht, wenn ein bestimmtes Terminalsymbol gelesen wird. Dies wird für alle Terminalsymbole des Alphabets $\Sigma$ durchgeführt.
3. Den Zuständen werden neue Äquivalenzklassen zugeordnet wobei bereits getrennte Zustände getrennt bleiben und alle Zustände, die in unterschiedliche Äquivalenzklassen übergehen in neue Äquivalenzklassen geschrieben werden.
4. Schritt 2 und Schritt 3 werden so lange wiederholt, bis sich keine neuen Äquivalenzklassen mehr ergeben.

### 0.2.7 Konkatenation

Seien $\mathcal{A}_1 = (\Sigma, Q_1, s_1, \Delta_1, A_1)$ und $\mathcal{A}_2 = (\Sigma, Q_2, s_2, \Delta_2, A_2)$ zwei NFAs über dem Alphabet $\Sigma$.

Durch die Konkatenation kann nun die Sprache $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ gebildet und als NFA dargestellt werden.

Vorerst müssen die Zustände so umbenannt werden, dass $Q_1 \cap Q_2 \neq \emptyset$ gilt. Dannach ist der NFA $\mathcal{A} = (\Sigma, Q, s, \Delta, A)$ wie folgt definiert:

$$
\begin{aligned}
Q &= Q_1 \cup Q_2 \\
s &= s_1 \\
\Delta &= \Delta_1 \cup \Delta_2 \cup \{(a_1, \varepsilon, s_2) : a_1 \in A_1\} \\
A &= A_2
\end{aligned}
$$

## 0.3 Grammatiken

### 0.3.1 Pumping Lemma für kontextfreie Sprachen

Für jede kontextfreie Sprache $L \subseteq \Sigma^*$ existiert ein $n \in \mathbb{N}$, sodass für jedes Wort $x \in L$ mit $|x| \geq n$ eine Zerlegung $x = y \cdot u \cdot v \cdot w \cdot z$ mit $u \cdot w \neq \varepsilon$ existiert, sodass für alle $m \in \mathbb{N}$ gilt, dass auch $y \cdot u^m \cdot v \cdot w^m \cdot z \in L$ gilt. Dabei kann $|u \cdot v \cdot w| \leq n$ gelten.

### 0.3.2 Chomsky-Normalform

Eine Grammatik in der Chomsky-Normalform enthält nur Produktionsregeln zu zwei Variablen ($A \to BC$) oder einem Terminalsymbol ($A \to a$). Außerdem kann die Startvariable zu $\varepsilon$ werden, solange diese nicht mehr referenziert wird.

Um eine beliebige Grammatik in die Chomsky-Normalform umzuwandeln, werden die folgenden Schritte nacheinander ausgeführt:

1. Ersetze alle Terminalsymbole durch eindeutige Variablen, welche nur das Terminalsymbol referezieren.
2. Ersetze alle Variablenpaare in n-Tupeln (n > 2) von Variablen durch eindeutige Variablen.
3. Wiederhole Schritt 2 so lange, bis nur noch Produktionsregeln zu zwei oder einer Variablen oder einem Terminalsymbol existieren.
4. Ersetze alle einzelnen Variablen durch die jeweiligen Produktionsregeln.
5. Wiederhole Schritt 4 so lange, bis die Grammatik in der Chomsky-Normalform vorliegt.

### 0.3.3 CYK Algorithmus

Der CYK Algorithmus wird genutzt um zu prüfen, ob ein Wort von einer Grammatik produziert wird oder nicht. Hierzu muss die Grammatik in der Chomsky-Normalform vorliegen!

Für ein Wort $w \in \Sigma^*$ welches zu prüfen ist wird eine Tabelle angelegt mit $|w|$ Spalten und $|w| + 1$ Zeilen. In der ersten Zeile wird das Wort notiert. Nun wird in der zweiten Zeile notiert, von welcher Variablen das darüber stehende Terminalsymbol produziert wird. Dies wird in einer Menge notiert (beispielsweise $\{S\}$) In der dritten Zeile wird notiert, aus Welchen Variablen zwibuchstabige Wörter unter der Nutzung der darüber liegenden Variablen produziert werden können. Dies wird sukzessive für die nächsten Zeilen fortgeführt. Ist es nicht möglich das Wort zu produzieren, so wird dies mit der leeren Menge ($\emptyset$) notiert. Steht in der untersten und damit letzten Zelle die Startvariable, so ist das Wort mit der Grammatik produzierbar.

**Beispiel** Gegeben ist die Grammatik $\mathcal{G} = (\Sigma, V, P, X_0)$ mit $\Sigma = \{x, m, p\}$, $V = \{S, T, M, N, P\}$, $X_0 = S$ und den folgenden Produktionsregeln $P$:

$$\begin{aligned} S &\to x|ST|MS \\ T &\to SP \\ M &\to NS \\ N &\to m \\ P &\to p \end{aligned}$$

In der Tabelle **??** ist zu sehen, wie schrittweise geprüft wird, ob $xmxxp$ von der Grammatik $\mathcal{G}$ produziert wird. Da $S = X_0$ gilt, wird $xmxxp$ von der Grammatik $\mathcal{G}$ produziert.

## 0.4 Entscheidbarkeit und Aufzählbarkeit

#### Definition

- Eine Turingmaschine $\mathcal{M}$ akzeptiert/erkennt ein Wort $w \in \Sigma^*$ gdw. die Berechnung von $\mathcal{M}$ akzeptierend ist (das heißt die Berechnung in $q^+$ terminiert).

| $x$ | $m$ | $x$ | $x$ | $p$ |
|---|---|---|---|---|
| $\{S\}$ | $\{N\}$ | $\{S\}$ | $\{S\}$ | $\{P\}$ |
| $\emptyset$ | $\{M\}$ | $\emptyset$ | $\{T\}$ | - |
| $\emptyset$ | $\{S\}$ | $\{S\}$ | - | - |
| $\emptyset$ | $\{T, M\}$ | - | - | - |
| $\{S\}$ | - | - | - | - |

Abbildung 4: CYK: Beispiel

- Die von einer Turingmaschine $\mathcal{M}$ akzeptierte/erkannt Sprache ist $L(\mathcal{M}) = \{w \in \Sigma^* : \mathcal{M} \text{ akzeptiert } w\}$.
- Eine Turingmaschine $\mathcal{M}$ entscheidet eine Sprache (löst das Wortproblemder Sprache $L$) $L \subseteq \Sigma^*$ gdw. jede Eingabe von $w \in \Sigma^*$ terminiert und $L = L(\mathcal{M})$ gilt (das heißt jede Eingabe von $w \in L$ terminiert in $q^+$ und jede Eingabe von $w \in \Sigma^* \setminus L$ terminiert in $q^-$).

Eine Sprache $L \subseteq \Sigma^*$ heißt

- (rekursiv) aufzählbar/semi-entscheidbar $\iff$ $L$ wird von einer Turingmaschine akzeptiert
- entscheidbar (rekursiv) $\iff$ $L$ wird von einer Turingmaschine entschieden $\iff$ $L$ aufzählbar $\wedge$ $\Sigma^* \setminus L$ aufzählbar

## 0.5 Wortproblem

Mit dem Wortproblem wird bezeichnet, ob eine Sprache entscheidbar ist oder nicht (siehe Entscheidbarkeit, 0.4).

## 0.6 Chomsky-Hierarchie

Die Chomsky-Hierarchie ist eine Hierarchie von Klassen von Sprachen die bestimmten Regeln und Einschränkungen der Produktionsregeln unterliegen.

### 0.6.1 Typ 0

**Typnummer** 0

**Grammatik** Beliebige formale Grammatik

**Regeln** -

**Sprachen** rekursiv aufzählbar

**Entscheidbarkeit** -

**Automaten** Turingmaschine

**Abgeschlossenheit** Konkatenation ($\circ$), Schnitt ($\cap$), Vereinigung ($\cup$), Sternchen ($^*$)

**Komplexitätsklasse** -

### 0.6.2 Typ 1

**Typnummer** 1

**Grammatik** Kontextsensitive Grammatik

**Regeln** Nur Produktionen der Form $\alpha A\beta \to \alpha\gamma\beta$ ($A \in V$, $\alpha, \beta \in (\Sigma \cup V)^*$, $\gamma \in (\Sigma \cup V)^+$) sind erlaubt. Maximal eine Produktion der Form $S \to \varepsilon$ ist erlaubt, wenn Produktion der Form $\alpha \to \beta S\gamma$ existiert.

**Sprachen** kontextsensitiv

**Entscheidbarkeit** Wortproblem

**Automaten** linear eingeschränkte Turingmaschinen

**Abgeschlossenheit** Komplement ($^\complement$), Konkatenation ($\circ$), Schnitt ($\cap$), Vereinigung ($\cup$), Sternchen ($^*$)

**Komplexitätsklasse** $2^{O(n)}$

### 0.6.3 Typ 2

**Typnummer** 2

**Grammatik** Kontextfreie Grammatik

**Regeln** Nur Produktionen der Form $A \to \gamma$ ($A \in V$, $\gamma \in (\Sigma \cup V)^*$) sind erlaubt.

**Sprachen** kontextfrei

**Entscheidbarkeit** Wortproblem

**Automaten** Kellerautomat

**Abgeschlossenheit** Konkatenation ($\circ$), Vereinigung ($\cup$), Sternchen ($^*$)

**Komplexitätsklasse** $O(n^3)$

### 0.6.4 Typ 2

**Typnummer** 3

**Grammatik** Reguläre Grammatik

**Regeln** Nur rechts- oder linkslineare Produktionen: $A \to aB$ oder $A \to Ba$ ($A, B \in V$, $a \in \Sigma$). Zusätzlich können Produktionen der folgenden Formen existieren: $A \to a$, $A \to \varepsilon$ ($A \in V$, $a \in \Sigma$)

**Sprachen** regulär

**Entscheidbarkeit** Wortproblem

**Entscheidbarkeit** Endlicher Automat

**Abgeschlossenheit** Komplement ($^\complement$), Konkatenation ($\circ$), Schnitt ($\cap$), Verinigung ($\cup$), Sternchen ($^*$)

**Komplexitätsklasse** $O(n)$

## 0.7 Abkürzungen

**DFA** Deterministischer, endlicher, formaler Automat

**DTM** Deterministische Turingmaschine

**NFA** Nichtdeterministischer, endlicher, formaler Automat

**PDA** Pushdown-Automat (Kellerautomat)