# Beispiele

Typ 3: $L = \{a^n \mid n \in \mathbb{N}\}$,

Grammatik: $S \to a$,

$S \to aS$

Typ 2: $L = \{a^n b^n \mid n \in \mathbb{N}\}$,

Grammatik: $S \to ab$,

$S \to aSb$

Typ 1: $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$,

Grammatik: $S \to aSXY$,

$S \to aXY$,

$S \to aXY$,

$XY \to YX$,

$aX \to ab$,

$bX \to bb$,

$bY \to bc$,

$cY \to cc$

**<u>Backus-Naur-Form</u>**: Formalismus zur kompakten Darstellung von Typ 2 Grammatiken.

Statt

$$\begin{array}{ccc} A & \to & \beta_1 \\ A & \to & \beta_2 \\ & \vdots & \\ A & \to & \beta_n \end{array}$$

schreibt man $A \to \beta_1|\beta_2|\dots|\beta_n$.

Statt

$$\begin{array}{ccc} A & \to & \alpha\gamma \\ A & \to & \alpha\beta\gamma \end{array}$$

schreibt man $A \to \alpha[\beta]\gamma$.

Statt

$$\begin{array}{ccc} A & \to & \alpha\gamma \\ A & \to & \alpha B\gamma \\ B & \to & \beta \\ B & \to & \beta B \end{array}$$

schreibt man $A \to \alpha\{\beta\}\gamma$.

**Beispiel:** Arithmetische Ausdrücke

$$
\begin{aligned}
<\text{exp}> &\rightarrow <\text{term}> \\
<\text{exp}> &\rightarrow <\text{exp}> + <\text{term}> \\
<\text{term}> &\rightarrow (<\text{exp}>) \\
<\text{term}> &\rightarrow <\text{term}> * <\text{term}> \\
<\text{term}> &\rightarrow a \mid b \mid \ldots \mid z
\end{aligned}
$$

Aufgabe eines Compilers: Prüfe ob ein gegebener String einen gültigen arithmetischen Ausdruck darstellt und, falls ja, zerlege ihn in seine Bestandteile.

---

Abstrakt kann man diese Probleme wie folgt formulieren:

WORTPROBLEM

Gegeben: Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $w \in \Sigma^*$.

Frage: Gilt $w \in L(G)$?

ABLEITUNGSPROBLEM

Gegeben: Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $w \in L(G)$.

Aufgabe: Konstruiere eine Ableitung von $w$, d.h., Worte $w_0, w_1, \ldots, w_n \in (\Sigma \cup V)^*$ mit $w_0 = S$, $w_n = w$ und $w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$.

**Satz:** Für kontextsensitive Sprachen ist das Wortproblem entscheidbar.
Genauer: Es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G$ und eines Wortes $w$ in endlicher Zeit entscheidet, ob $w \in L(G)$.

**Beweisidee:** Angenommen $w \in L(G)$. Dann gibt es $w_0, w_1, \ldots, w_n \in (\Sigma \cup V)^*$ mit $w_0 = S$, $w_n = w$ und $w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$.

WICHTIG: Da $G$ kontextsensitiv ist, gilt $|w_0| \leq |w_1| \leq \ldots \leq |w_n|$.
D.h., es genügt alle Worte in $L(G)$ der Länge $\leq n$ zu erzeugen.

**Formaler Beweis:** Definiere

$$T_m^n := \{w \in (\Sigma \cup V)^* \mid |w| \leq n \text{ und } w \text{ läßt sich aus } S \text{ in höchstens } m \text{ Schritten herleiten } \}$$

Diese Mengen kann man für alle $n$ induktiv wie folgt berechnen:

$$\begin{aligned} T_0^n &:= \{S\} \\ T_{m+1}^n &:= T_m^n \cup \{w \in (\Sigma \cup V)^* \mid |w| \leq n \text{ und } \\ &\qquad w' \Rightarrow w \text{ für ein } w' \in T_m^n\} \end{aligned}$$

Beachte: Für alle $m$ gilt: $|T_m^n| \leq \sum_{i=1}^n |\Sigma \cup V|^i$.
Es muß daher immer ein $m_0$ geben mit $T_{m_0}^n = T_{m_0+1}^n = \ldots$.

Algorithmus:

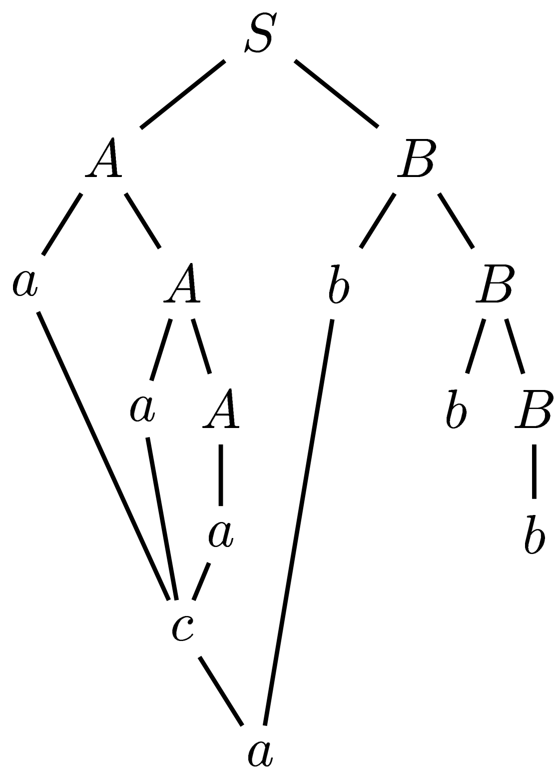
$n := |w|$;
$T_0^n := \{S\}; m := 0;$
**repeat**
  $T_{m+1}^n :=$ <wie oben>; $m := m + 1;$
**until** $T_m^n = T_{m-1}^n$ or $w \in T_m^n$;
**if** $w \in T_m^n$ **then return** ("Ja") **else return** ("Nein").

## Ableitungsgraphen

Grammatik:

$$\begin{aligned} S &\to AB \\ A &\to aA \\ A &\to a \\ B &\to bB \\ B &\to b \\ aaa &\to c \\ cb &\to a \end{aligned}$$

Beispiel:




Die Terminale ohne Kante nach unten entsprechen, von links nach rechts gelesen, dem durch den Ableitungsgraphen dargestellten Wort.

In obigem Beispiel gilt also $abb \in L(G)$. Der Ableitungsgraph entspricht der Ableitung

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aAbB \Rightarrow aaAbB \Rightarrow aaAbbB \Rightarrow aaabbB \Rightarrow aaabbb \Rightarrow cbbb \Rightarrow ab$$

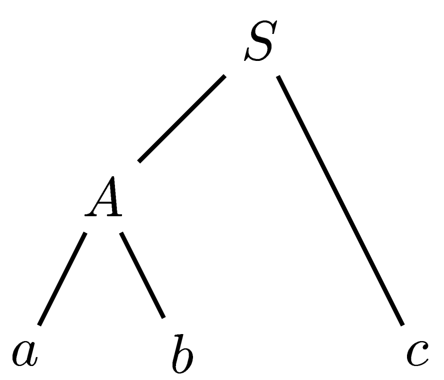
# Ableitungsbäume

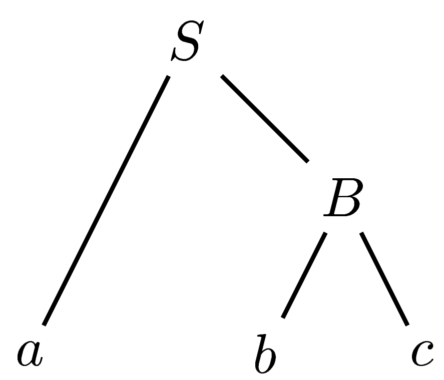
**Beobachtung:** Bei kontextfreien Sprachen sind die Ableitungsgraphen immer Bäume.

**Beispiel:** Grammatik:

$$\begin{array}{lcl} S & \rightarrow & aB \\ S & \rightarrow & Ac \\ A & \rightarrow & ab \\ B & \rightarrow & bc \end{array}$$

Für das Wort $abc$ gibt es zwei verschiedene Ableitungsbäume:



**Definition:** Eine Ableitung $S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$ eines Wortes $w_n$ heißt *Linksableitung*, wenn für jede Anwendung einer Produktion $y \to y'$ auf Worte $w_i = xyz$, $w_{i+1} = xy'z$ gilt: Auf jedes echte Präfix von $xy$ läßt sich *keine* Regel anwenden.

Man sagt: eine Grammatik ist *eindeutig* wenn es für jedes Wort $w \in L(G)$ genau eine eindeutige bestimmte Linksableitung gibt. Nicht eindeutige Grammatiken nennt man auch *mehrdeutig*.

**1. Beispiel:** Die zugehörige Linksableitung ist:
$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaaB \Rightarrow cB \Rightarrow cbB \Rightarrow aB \Rightarrow abB \Rightarrow abb$
Eine andere Linksableitung ist:

$$S \Rightarrow AB \Rightarrow aB \Rightarrow abB \Rightarrow abb.$$

Die Grammatik ist also mehrdeutig.

**2. Beispiel:** Beide Ableitungsbäume entsprechen Linksableitungen:

$$S \Rightarrow Ac \Rightarrow abc \quad \text{bzw.} \quad S \Rightarrow aB \Rightarrow abc.$$

Die Grammatik ist also ebenfalls mehrdeutig.

**Definition.** Ein *deterministischer endlicher Automat* (englisch: deterministic finite automata, kurz DFA) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, z_0, E)$ beschrieben, das folgende Bedingungen erfüllt:

– $Z$ ist eine endliche Menge von *Zuständen*.

– $\Sigma$ ist eine endliche Menge, das *Eingabealphabet*, wobei $Z \cap \Sigma = \emptyset$.

– $z_0 \in Z$ ist der *Startzustand*.

– $E \subseteq Z$ ist die Menge der *Endzustände*.

– $\delta : Z \times \Sigma \to Z$ heißt *Übergangsfunktion*.

Die von $M$ *akzeptierte* Sprache ist

$$L(M) := \{w \in \Sigma^* \mid \hat{\delta}(z_0, w) \in E\},$$

wobei $\hat{\delta} : Z \times \Sigma^* \to Z$ induktiv definiert ist durch

$$\begin{aligned} \hat{\delta}(z, \epsilon) &= z \\ \hat{\delta}(z, ax) &= \hat{\delta}(\delta(z, a), x) \end{aligned}$$

**Bem.:** Endliche Automaten können durch (gerichtete und beschriftete) *Zustandsgraphen* veranschaulicht werden:

Knoten $\hat{=}$ Zustände

Kanten $\hat{=}$ Übergänge

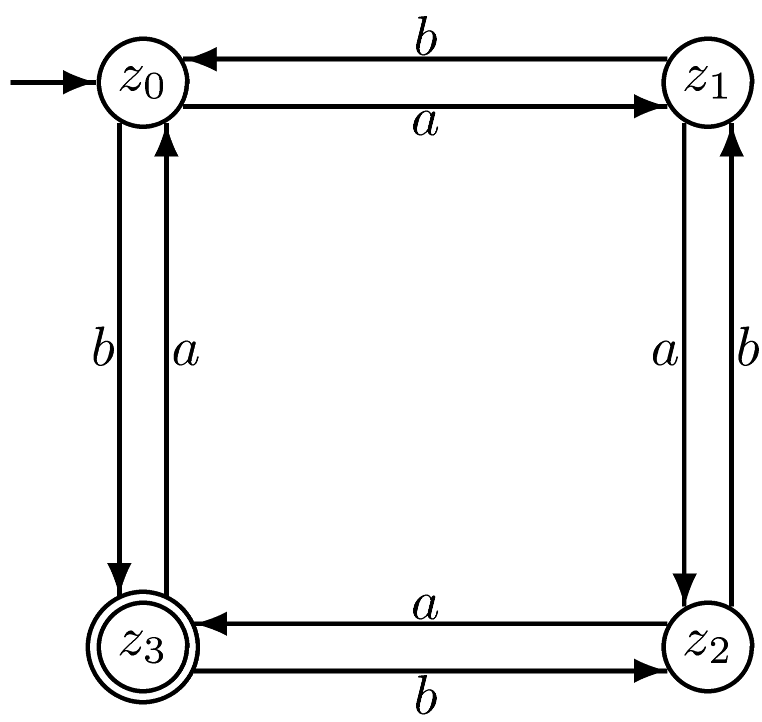
[genauer: Kante $(u, v)$, beschriftet mit $a \in \Sigma$, entspricht $\delta(u, a) = v$]

Endzustände werden durch doppelte Kreise gekennzeichnet.

## Beispiel

Sei $M = (Z, \Sigma, \delta, z_0, E)$, wobei

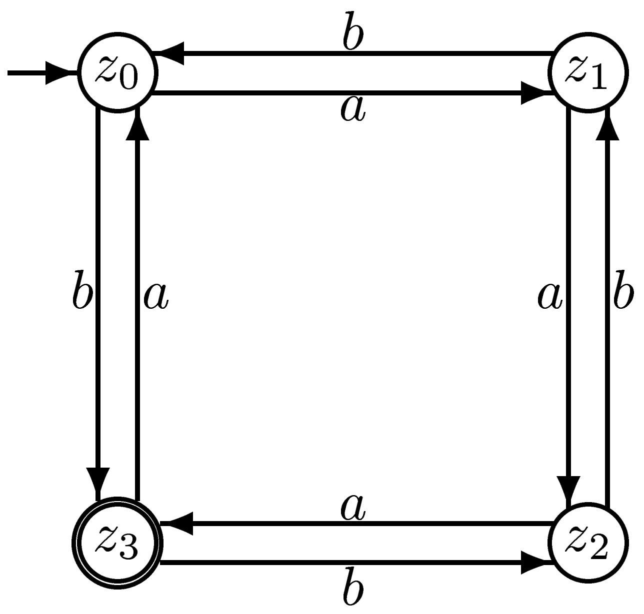
$$\begin{aligned}
Z &= \{z_0, z_1, z_2, z_3\} \\
\Sigma &= \{a, b\} \\
E &= \{z_3\} \\
\delta(z_0, a) &= z_1 \\
\delta(z_0, b) &= z_3 \\
\delta(z_1, a) &= z_2 \\
\delta(z_1, b) &= z_0 \\
\delta(z_2, a) &= z_3 \\
\delta(z_2, b) &= z_1 \\
\delta(z_3, a) &= z_0 \\
\delta(z_3, b) &= z_2
\end{aligned}$$



**Satz.** Ist $M = (Z, \Sigma, \delta, z_0, E)$ ein deterministischer endlicher Automat, so ist die durch

$$P \quad := \quad \{z \to az' \mid \delta(z,a) = z'\} \cup \{z \to a \mid \delta(z,a) \in E\}$$

gegebene Grammatik $G = (Z, \Sigma, z_0, P)$ regulär.




Produktionen:

$$z_0 \to az_1, \quad z_0 \to bz_3, \quad z_1 \to az_2,$$

$$z_1 \to bz_0, \quad z_2 \to az_3, \quad z_2 \to bz_1,$$

$$z_3 \to az_0, \quad z_3 \to bz_2,$$

$$z_2 \to a, \quad z_0 \to b.,$$

Beispiel: $z_0 \Rightarrow az_1 \Rightarrow abz_0 \Rightarrow abaz_1 \Rightarrow abaaz_2 \Rightarrow abaaa \in L(G)$

**Definition.** Ein *nichtdeterministischer endlicher Automat* (englisch: nondeterministic finite automata, kurz NFA) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, S, E)$ beschrieben, das folgende Bedingungen erfüllt:

- $Z$ ist eine endliche Menge von *Zuständen*.
- $\Sigma$ ist eine endliche Menge, das *Eingabealphabet*, wobei $Z \cap \Sigma = \emptyset$.
- $S \subseteq Z$ ist die Menge der *Startzustände*.
- $E \subseteq Z$ ist die Menge der *Endzustände*.
- $\delta : Z \times \Sigma \to \mathcal{P}(Z)$ heißt *Übergangsfunktion*.

Die von $M$ *akzeptierte* Sprache ist

$$L(M) := \{w \in \Sigma^* \mid \hat{\delta}(S, w) \cap E \neq \emptyset\},$$

wobei $\hat{\delta} : Z \times \Sigma^* \to \mathcal{P}(Z)$ wieder induktiv definiert ist durch

$$\begin{aligned} \hat{\delta}(Z', \epsilon) &= Z' \quad \forall Z' \subseteq Z \\ \hat{\delta}(Z', ax) &= \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x) \end{aligned}$$