# Die Chomsky–Hierarchie

**Hans U. Simon (RUB)**

Email: simon@lmi.rub.de

Homepage: http://www.ruhr-uni-bochum.de/lmi

# Vorgeplänkel: Mathematische Grundlagen

**Voraussetzung:** Wissen aus mathematischen Grundvorlesungen

**hier:** weitere Grundlagen zu Wörtern, Sprachen und Relationen

# Alphabet und Zeichen

Alphabet = nichtleere, endliche Menge

Die Elemente eines Alphabets $\Sigma$ werden als Zeichen, Symbole oder Buchstaben bezeichnet.

**Beispiele:**

1. $\Sigma = \{a, \ldots, z\} \cup \{A, \ldots, Z\} \cup \{0, \ldots, 9\}$.

2. $\Sigma = \{a, b\}$.

# Zeichenfolgen

$$\Sigma^+ = \text{Menge der nicht–leeren Zeichenfolgen über Alphabet } \Sigma.$$

Die Elemente von $\Sigma^+$ werden auch als Wörter, Sätze oder Strings bezeichnet.

Die Länge eines Strings $w$ ist gegeben durch

$$|w| = \text{Anzahl der Zeichen (mit Vielfachheit) in } w.$$

Für ein Zeichen $a \in \Sigma$ schreiben wir

$$|w|_a = \text{Anzahl der Vorkommen von } a \text{ in } w.$$

**Beispiel:**

- $\{a, b\}^+ = \{a, b, aa, ab, ba, bb, aaa, \ldots\}$.
- $|a| = |ba|_a = 1$, $|ba| = 2$, $|aaa| = 3$.

## Zeichenfolgen (fortgesetzt)

Das leere Wort der Länge 0 wird mit $\varepsilon$ bezeichnet.
(Spielt eine ähnliche Rolle wie die „Null“ beim Addieren.)

$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$ bezeichnet dann die Menge aller Strings (inklusive dem leeren) über Alphabet $\Sigma$.

# Konkatenation von Wörtern

Wörter konkateniert (= aneinandergehängt) ergeben wieder Wörter. Es gilt:

$$|uv| = |u| + |v|$$

**Beispiel:** Für $u = ab$ und $v = aab$ gilt $uv = abaab$ und $|uv| = 2 + 3 = 5$.

Die $n$–fache Konkatenation eines Wortes $w$ mit sich selbst notieren wir mit $w^n$. Es gilt $|w^n| = n \cdot |w|$.

**Beispiel:** Für $w = ab$ gilt $w^3 = ababab$ und $|w^3| = 3 \cdot 2 = 6$.

# Formale Sprachen

Eine Menge $A \subseteq \Sigma^*$ heißt (formale) Sprache über Alphabet $\Sigma$. Neben den üblichen Mengenoperationen (Vereinigung, Durchschnitt, Komplement, Mengendifferenz,. . .) sind auf Sprachen die folgenden Operationen von Interesse:

**Konkatenation** $AB := \{uv |\ u \in A, v \in B\}$.

**Potenz** $A^n := \underbrace{A \cdots A}_{n-mal}$, wobei $A^0 = \{\varepsilon\}$ und $A^1 = A$.

**Kleenescher Abschluss** $A^+ = \cup_{n\geq1} A^n$ und $A^* = \cup_{n\geq0} A^n$.

# Formale Sprachen (fortgesetzt)

**Beispiel** Betrachte die Sprachen

$$A = \{w \in \{a,b\}^* |\ |w|_a = 5\} \text{ und } B = \{w \in \{a,b\}^* |\ |w|_b = 5\} \ .$$

Dann gilt

$$w = \overbrace{baaabbababbbb}^{\in A}\overbrace{aaabaabbbaabaaa}^{\in B} \in AB \ .$$

Weiter gilt:

$$\begin{aligned} A^{20} &= \{w \in \{a,b\}^* |\ |w|_a = 100\} \\ A^+ &= \{w \in \{a,b\}^+ |\ 5 \text{ ist Teiler von } |w|_a\} \end{aligned}$$

# Relationen

Wir betrachten Relationen $R$ über einer Grundmenge $M$.

**Formal:** $R \subseteq M \times M$.

Statt $(x, y) \in R$ schreiben wir oft $xRy$.

Statt $(x, y) \in R$ und $(y, z) \in R$ schreiben wir mitunter $xRyRz$.

# Verknüpfung von Relationen

Ähnlich wie bei formalen Sprachen betrachten wir folgende Operationen:

**Konkatenation** $RS := \{(x, y) \in M \times M |\ \exists z \in M : xRz \text{ und } zSy\}$.

**Potenz** $R^n := \underbrace{R \cdots R}_{n-mal}$, wobei $R^0 = \{(x, x)|\ x \in M\}$ und $R^1 = R$.

**(reflexive) transitive Hülle** $R^+ = \cup_{n \geq 1} R^n$ und $R^* = \cup_{n \geq 0} R^n$.

Es gilt:

1. $R^n = \{(x, y) \in M \times M |\ \exists z_1, \ldots, z_{n-1} : xRz_1Rz_2 \cdots Rz_{n-1}Ry\}$ für $n \geq 1$.
2. $R^+$ ist die kleinste transitive $R$ umfassende Relation.
3. $R^*$ ist die kleinste reflexive und transitive $R$ umfassende Relation.

## Beispiele

$$\begin{array}{rcl} M & = & \text{Menge von Personen} \\ R & = & \text{die Relation „ist Schwester von“} \\ S & = & \text{die Relation „ist Vater oder Mutter von“} \end{array}$$

Dann gilt:

$$\begin{array}{rcl} RS & = & \text{die Relation „ist Tante von“} \\ S^2 & = & \text{die Relation „ist Großvater oder Großmutter von“} \\ S^+ & = & \text{die Relation „ist (echter) Vorfahr von“} \end{array}$$

# Jetzt gehts ...

# ... zur Sache !

# Grammatiken

Eine Grammatik besteht aus vier Komponenten:

- $V$, die Menge der Variablen
- $\Sigma$, das Terminalalphabet
- $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$, das Regel– oder Produktionensystem
- $S \in V$, die Startvariable

Dabei sind $V, \Sigma, P$ endliche Mengen und $V \cap \Sigma = \emptyset$. Ein Paar $(y, y')$ aus $P$ notieren wir meist in der Form $y \rightarrow y'$.

**Intuition:** In einer grammatischen Ableitung darf $y$ durch $y'$ ersetzt werden.

Strings über $\Sigma$ heißen Sätze; Strings über $V \cup \Sigma$ heißen Satzformen.

# Grammatische Ableitungen

Die Notation $u \Rightarrow_G v$ bedeutet, dass Satzform $u$ unter **einer** Regelanwendung der Grammatik $G$ in Satzform $v$ übergehen kann:

u,v haben die Form $u = xyz$, $v = xy'z$, und $y \rightarrow y' \in P$.

Mit Hilfe von Relation „$\Rightarrow_G$“ können folgende Relationen gebildet werden:

$$
\begin{aligned}
\Rightarrow_G^n &= n\text{–fache Potenz von } \Rightarrow_G \\
\Rightarrow_G^+ &= \text{transitive Hülle von } \Rightarrow_G \\
\Rightarrow_G^* &= \text{reflexive–transitive Hülle von } \Rightarrow_G
\end{aligned}
$$

# Grammatische Ableitungen (fortgesetzt)

- $u \Rightarrow_G^n v$ bedeutet, dass Satzform $u$ unter $n$ Anwendungen von Regeln der Grammatik $G$ in Satzform $v$ übergehen kann.
- $u \Rightarrow_G^+ v$ bedeutet, dass Satzform $u$ unter einer iterierten Anwendung (mindestens einmal) von Regeln der Grammatik $G$ in Satzform $v$ übergehen kann.
- $u \Rightarrow_G^* v$ bedeutet, dass Satzform $u$ unter einer iterierten Anwendung (auch null-mal) von Regeln der Grammatik $G$ in Satzform $v$ übergehen kann.

# Grammatische Ableitungen (fortgesetzt)

Eine Folge

$$w_0, w_1, \ldots, w_n$$

von Satzformen mit

$$w_0 = S,\ w_n \in \Sigma^* \text{ und } w_i \Rightarrow_G w_{i+1}$$

heißt Ableitung (von $w_n$ mit Regeln aus $G$).

Die von $G$ erzeugte Sprache ist die Menge aller aus Startsymbol $S$ ableitbaren Wörter über Terminalalphabet $\Sigma$:

$$L(G) := \{w \in \Sigma^* |\ S \Rightarrow_G^* w\}$$

## Beispiel: Korrekt geklammerte Rechenausdrücke

Betrachte die Grammatik $G$ mit den Komponenten

- $V = \{E, T, F\}$.
- $\Sigma = \{(, ), a, +, *\}$.
- $P$ enthalte die Regeln

$$E \rightarrow T \ , \ E \rightarrow E + T \ , \ T \rightarrow F \ , \ T \rightarrow T * F \ , \ F \rightarrow a \ , \ F \rightarrow (E) \ .$$

- $S = E$, d.h., $E$ ist die Startvariable.

**Intuition:** $E$ steht für EXPRESSION (arithmetischer Ausdruck), $T$ für TERM und $F$ für FACTOR. Terminalzeichen $a$ repräsentiert einen atomaren Ausdruck (etwa eine Konstante oder eine Variable). Die Grammatik soll gerade die korrekt geklammerten Rechenausdrücke erzeugen.

# Beispiel (fortgesetzt)

Es gilt

$$a * (a + a) + a \in L(G)$$

wie folgende Ableitung zeigt:

$$\begin{array}{cccccccc}
E & \Rightarrow & E+T & \Rightarrow & T+T & \Rightarrow & T*F+T \\
 & \Rightarrow & F*F+T & \Rightarrow & a*F+T & \Rightarrow & a*(E)+T \\
 & \Rightarrow & a*(E+T)+T & \Rightarrow & a*(T+T)+T & \Rightarrow & a*(F+T)+T \\
 & \Rightarrow & a*(a+T)+T & \Rightarrow & a*(a+F)+T & \Rightarrow & a*(a+a)+T \\
 & \Rightarrow & a*(a+a)+F & \Rightarrow & a*(a+a)+a & &
\end{array}$$

Es wurde in jedem Schritt die am weitesten links stehende Variable ersetzt: wir sprechen von einer Linksableitung.

## Chomsky–Hierarchie ohne Sonderregeln für $\varepsilon$

**Typ 0:** Eine Grammatik mit Regeln der allgemeinen Form $w \to w'$ mit $w \in (V \cup \Sigma)^+$ und $w' \in (V \cup \Sigma)^*$ heißt Grammatik vom Typ 0.

**Typ 1:** Eine Grammatik mit Regeln der Form $w \to w'$ mit $w, w' \in (V \cup \Sigma)^+$ und $|w| \le |w'|$ heißt kontextsensitive (oder auch Typ 1) Grammatik.

**Typ 2:** Eine Grammatik mit Regeln der Form $X \to w$ mit $X \in V$ und $w \in (V \cup \Sigma)^+$ heißt kontextfreie (oder auch Typ 2) Grammatik.

**Typ 3:** Eine Grammatik mit Regeln der Form $X \to a$ oder $X \to aY$ mit $X, Y \in V$ und $a \in \Sigma$ heißt reguläre (oder auch Typ 3) Grammatik.

Wir übertragen diese Bezeichnungen auch auf die von den Grammatiken generierten Sprachen.

Offensichtlich gilt:

regulär $\Rightarrow$ kontextfrei $\Rightarrow$ kontextsensitiv $\Rightarrow$ Typ 0 $\Rightarrow$ formale Sprache

# Sonderregeln für das leere Wort

Jetzige Definition von Typ 1,2,3 Grammatiken $G$ erlaubt nicht die Generierung des leeren Wortes $\varepsilon$ (schlecht, falls $\varepsilon \in L(G)$ erwünscht ist). Daher folgende Regelung:

**Typ 1:** Wir erlauben auch die Regel

$$S \to \varepsilon \ , \ S \text{ Startsymbol} \ .$$

In diesem Falle darf aber $S$ auf keiner rechten Seite einer Regel auftreten.

**Typ 2 bzw. 3:** Wir erlauben beliebige „$\varepsilon$-Regeln“ der Form

$$A \to \varepsilon \ , \ A \in V \ .$$

# Sonderregeln für das leere Wort (fortgesetzt)

**Man kann zeigen (Beweis später):**

$L$ ist vom Typ 1 (bzw. 2 oder 3) unter Einsatz der Sonderregeln

**dund**

$L \setminus \{\varepsilon\}$ ist vom Typ 1 (bzw. 2 oder 3) ohne Einsatz der Sonderregeln.

**Folgerung:** Für formale Sprachen bleibt die Inklusionskette

$$\text{regulär} \Rightarrow \text{kontextfrei} \Rightarrow \text{kontextsensitiv} \Rightarrow \text{Typ } 0$$

gültig, auch wenn Sonderregeln für das leere Wort eingesetzt werden dürfen.

# Echtheit der Chomsky–Hierarchie (Übersicht)

1. Die Sprache $\{(a^n b^n | \ n \geq 1\}$ ist kontextfrei aber nicht regulär.
2. Die Sprache $\{a^n b^n c^n | \ n \geq 1\}$ ist kontextsensitiv aber nicht kontextfrei.
3. Es gibt überabzählbar viele formale Sprachen, aber nur abzählbar viele vom Typ 0.

Da es auch nicht kontextsensitive Sprachen vom Typ 0 gibt, sind alle Inklusionen der Chomsky–Hierarchie echt.

# Echtheit der Chomsky–Hierarchie (fortgesetzt)

Die kontextfreien Regeln

$$S \rightarrow ab,\ S \rightarrow aSb$$

erzeugen die Sprache

$$\{a^n b^n \mid n \geq 1\}\ .$$

Wir werden später sehen, dass diese Sprache nicht regulär ist.

# Echtheit der Chomsky–Hierarchie (fortgesetzt)

**Satz:** Die kontextsensitive Grammatik mit den Regeln

$$S \rightarrow aSBC,\ S \rightarrow aBC,\ CB \rightarrow BC$$
$$aB \rightarrow ab,\ bB \rightarrow bb,\ bC \rightarrow bc,\ cC \rightarrow cc$$

erzeugt die Sprache $\{a^n b^n c^n |\ n \geq 1\}$ (Beweisskizze an der Tafel).

Wir werden später sehen, dass diese Sprache nicht kontextfrei ist.

## Das Wortproblem

Das Wortproblem für eine Sprache $L \subseteq \Sigma^*$ ist folgendes Problem:

**Eingabe:** $w \in \Sigma^*$

**Frage:** $w \in L$?

Es heißt entscheidbar, wenn ein Algorithmus existiert, der für jede Eingabe $w$ die richtige Antwort liefert.

**Satz:** Das Wortproblem für eine kontextsensitive Sprache $L$ ist stets entscheidbar.

**Idee:** Verwende eine kontextsensitive Grammatik $G$, die $L$ generiert.
Wegen der Monotonie–Eigenschaft
„rechte Seite einer Regel ist nicht kürzer als die linke Seite“
sind nur die endlich vielen Satzformen mit einer Maximallänge von $n = |w|$ für die Ableitung von $w$ relevant.

# Das Wortproblem (fortgesetzt)

**Implementierung der Idee:**

1. Setze $n := |w|$.
2. Berechne die Menge $\mathrm{Abl}_n$ aller Satzformen der Maximallänge $n$, die sich aus $S$ ableiten lassen:

$$\mathrm{Abl}_n := \{w \in (V \cup \Sigma)^* \mid |w| \leq n \text{ und } S \Rightarrow_G^* w\}$$

3. Falls $w \in \mathrm{Abl}_n$, dann akzeptiere $w$; andernfalls verwerfe $w$.

Dabei kann $\mathrm{Abl}_n$ iterativ berechnet werden wie folgt:

**Initialisierung** $\mathrm{Abl} := \{S\}$.

**Iteration** Solange ein Wort $w \notin \mathrm{Abl}$ mit

$$|w| \leq n, \exists u \in \mathrm{Abl} : u \Rightarrow_G w$$

existiert, nimm auch $w$ in Abl auf.

# Syntaxbäume

Betrachte wieder eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$. Jede Regel $B \to A_1 \cdots A_k$ kann durch eine Verzweigung visualisiert werden:

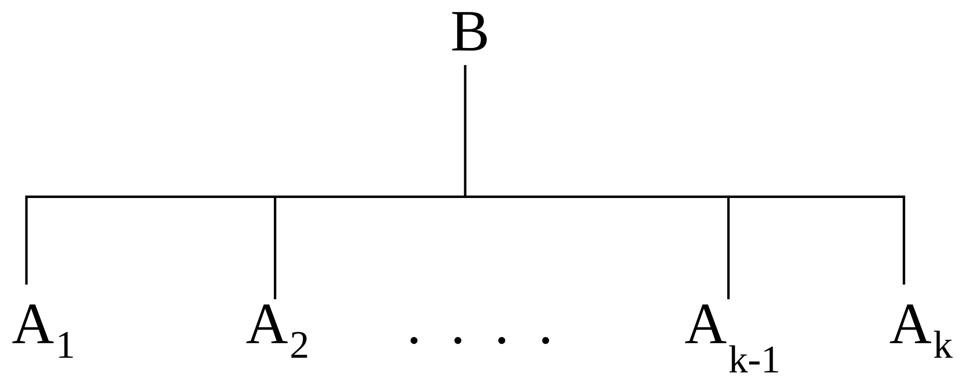


Abbildung 1: Visualisierung einer Regel $B \to A_1 \cdots A_k$.

# Syntaxbäume (fortgesetzt)

Ein Syntaxbaum zur kontextfreien Grammatik $G$ mit Beschriftung $w$ ist ein geordneter Wurzelbaum mit folgenden Eigenschaften:

1. Die Wurzel ist mit dem Startsymbol $S$ markiert.
2. Die inneren Knoten sind mit Variablen aus $V$ markiert.
3. Jede Verzweigung entspricht einer Regel aus $P$.
4. Die Blätter sind mit Terminalzeichen aus $\Sigma$ (oder mit $\varepsilon$) markiert.
5. Von links nach rechts gelesen ergeben die Blattmarkierungen das Wort $w$.

# Beispiel (fortgesetzt)

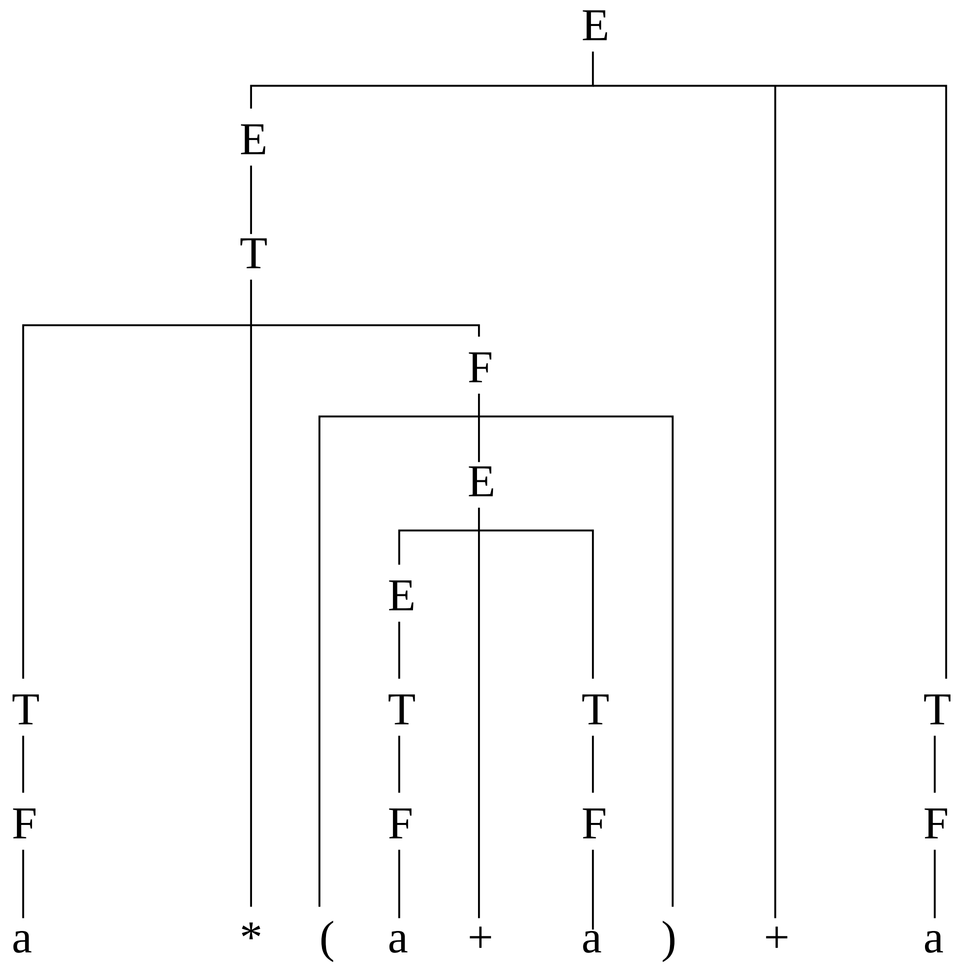


Abbildung 2: Der Syntaxbaum zur Ableitung von $a * (a + a) + a$ mit der Grammatik für korrekt geklammerte Rechenausdrücke.

## Syntaxbäume (fortgesetzt)

Folgende Aussagen sind äquivalent:

1. Es gibt eine Ableitung von $w$ mit Regeln aus $G$.
2. Es gibt einen Syntaxbaum zu $G$ mit Beschriftung $w$.
3. Es gibt eine Linksableitung von $w$ mit Regeln aus $G$.

Ein formaler Beweis könnte durch einen Ringschluss

$$1. \Rightarrow 2. \Rightarrow 3. \Rightarrow 1.$$

erfolgen.

## Eindeutige und mehrdeutige Grammatiken

Eine kontextfreie Grammatik $G$ heißt mehrdeutig, wenn sie verschiedene Syntaxbäume mit derselben Beschriftung zulässt; andernfalls heißt sie eindeutig.

Eine kontextfreie Sprache $L$ heißt eindeutig, wenn eine $L$ generierende eindeutige kontextfreie Grammatik $G$ existiert; andernfalls heißt sie inhärent mehrdeutig.

Programmiersprachen sollten eindeutig sein, damit jedes Programm eindeutig interpretiert werden kann.

# Illustration an einem abschreckenden Beispiel

Die Grammatik mit den Regeln

$$E \rightarrow a \ , \ E \rightarrow E + E \ , \ E \rightarrow E * E$$

erzeugt Rechenausdrücke mit den Operationen $+$ und $*$.
Sie ist mehrdeutig:

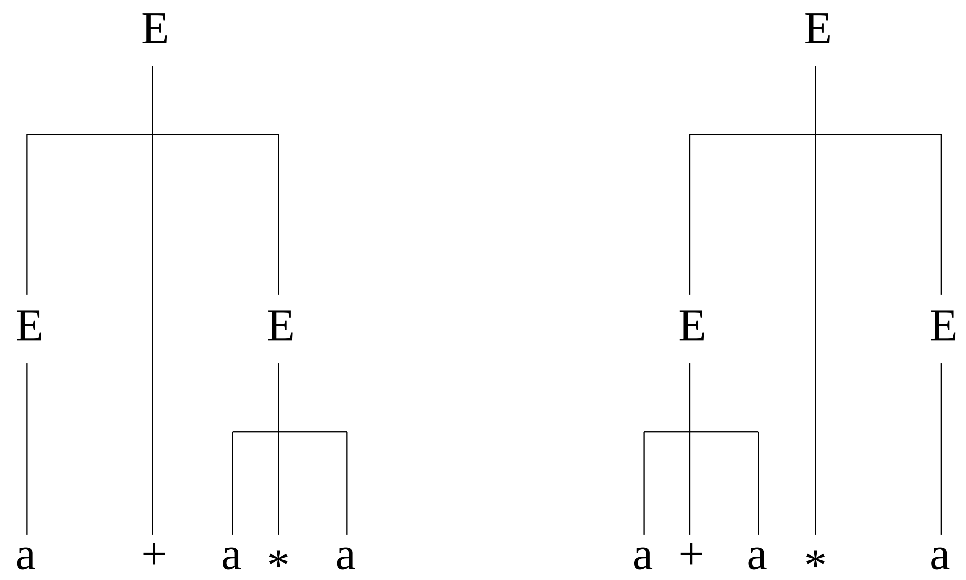


Abbildung 3: Zwei verschiedene Syntaxbäume für den Rechenausdruck $a + a * a$.

# Fortsetzung des abschreckenden Beispiels

- Der linke Syntaxbaum „berechnet“ $a + (a * a)$; der rechte „berechnet“ $(a + a) * a$.
- Die Mehrdeutigkeit führt zu unterschiedlichen Berechnungen (schlecht!)
- Die eindeutige Grammatik, die wir früher für Rechenausdrücke eingeführt hatten, ist daher vorzuziehen.

# Eine weitere mehrdeutige Beispiel-Grammatik

Die Regeln

$$K \to \varepsilon,\ K \to KK\ ,\ K \to (K)$$

erzeugen die Sprache der korrekten Klammerausdrücke. Diese können auch durch den Klammerniveautest erkannt werden:

- Für jedes Vorkommen von “(” zähle das Niveau um 1 hoch.
- Für jedes Vorkommen von “)” zähle das Niveau um 1 runter.
- Korrekte Klammerausdrücke durchlaufen nicht-negative Niveaus und führen am Ende zu Niveau 0.

## Beispiele für (korrekte und falsche) Klammerausdrücke

- Korrekt geklammerte Ausdrücke:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ( | ) | \| | ( | ( | ) | ( | ) | ) | ( | ) |
| Niveau | 1 | 0 | \| | 1 | 2 | 1 | 2 | 1 | 0 | 1 | 0 |

- Falsch geklammerte Ausdrücke:

| | | | | | | |
|---|---|---|---|---|---|---|
| | ( | ( | ) | \| | ) | ( |
| Niveau | 1 | 2 | 1 | \| | -1 | 0 |

## Demonstration der Mehrdeutigkeit

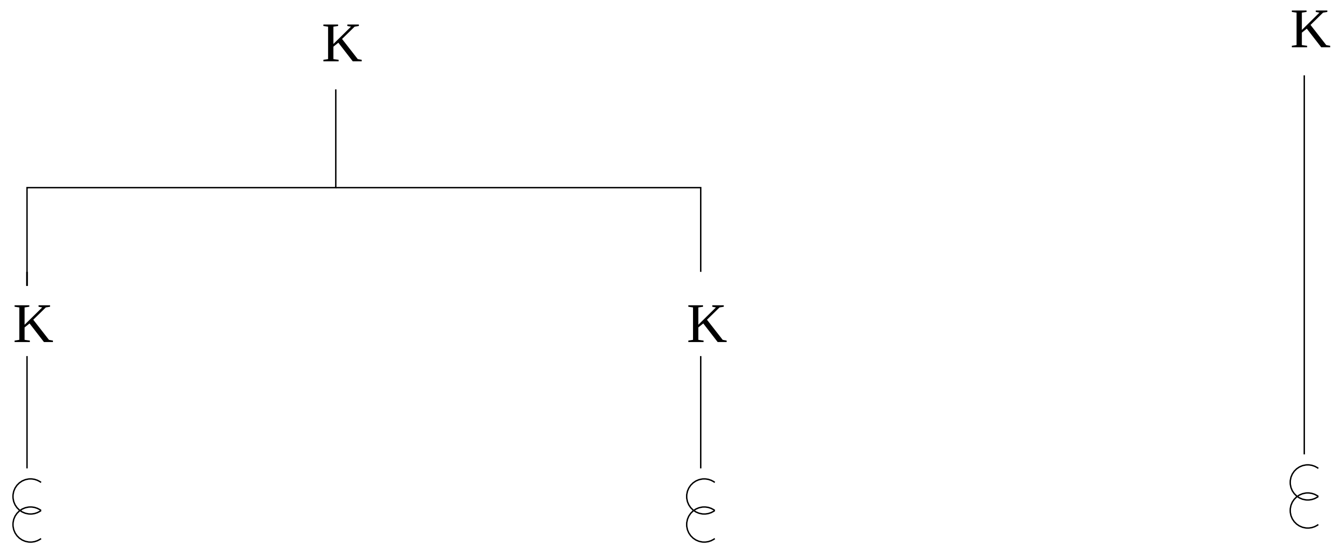


Abbildung 4: Zwei verschiedene Syntaxbäume für den leeren Klammerausdruck.

## Eindeutige Grammatik für dieselbe Sprache

$$K \to \varepsilon, K \to (K)K$$

- Ein atomarer Klammerausdruck ist einer der erst am Ende, aber nicht zwischendurch, Niveau 0 erreicht.
- Jeder Klammerausdruck zerfällt eindeutig in eine Konkatenation atomarer Klammerausdrücke.
- Bei Anwendung der Regel $K \to (K)K$ endet hinter $(K)$ der erste atomare Klammerausdruck.

Aufbauend hierauf lässt sich die Eindeutigkeit der obigen kontextfreien Grammatik leicht nachweisen.

# Beispiel einer inhärent mehrdeutigen Sprache

Die Sprache

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

ist inhärent mehrdeutig (ohne Beweis).

## Kontextfreiheit und Programmiersprachen

**Klammerausdrücke:** Als Bestandteil von Rechenausdrücken (oder auch durch die mit **begin** und **end** angezeigte Blockstruktur) kommen sie in Programmen vor.
Die Sprache der Klammerausdrücke ist kontextfrei (wie wir wissen).

**Wortduplikate:** Die Sprache

$$\{w\$w : \ w \in \Sigma^*\}$$

kommt als „Muster“ in Programmieprachen vor, bei denen Variable deklariert werden müssen (1. Vorkommen im Deklarationsteil; 2. Vorkommen bei der ersten Wertzuweisung). Sie ist aber, wie wir später sehen werden, nicht kontextfrei.

**Bemerkung:** Obwohl Programmiersprachen i.A. nicht vollständig kontextfrei sind, sind sie „im Wesentlichen“ kontextfrei.

# Ein weiteres nicht kontextfreies Textmuster

Die Sprache

$$\{a^n b^n c^n : \; n \geq 1\}$$

ist (wie früher schon erwähnt wurde) nicht kontextfrei. Sie entspricht dem Textmuster unterstrichener Wörter:

Für den String

$$\underline{\text{Abrakadabra}}$$

könnte man im Quelltext die Symbole

$$\underbrace{\text{Abrakadabra}}_{11\ Buchstaben}\underbrace{\langle backspace\rangle\cdots\langle backspace\rangle}_{11-mal}\underbrace{\langle underline\rangle\cdots\langle underline\rangle}_{11-mal}$$

vorfinden, was dem Textmuster $a^{11}b^{11}c^{11}$ (in der offensichtlichen Weise) entspricht.

## Backus–Naur–Form

Backus–Naur–Form (BNF) erlaubt flexiblere Formen von kontextfreien Regeln:

1. Die Metaregel

$$A \rightarrow \beta_1 | \beta_2 | \cdots | \beta_n$$

   (unter Verwendung des „Metasymbols" |) steht für

$$\begin{array}{ccc} A & \rightarrow & \beta_1 \\ A & \rightarrow & \beta_2 \\ \cdots & & \cdots \\ A & \rightarrow & \beta_n \end{array}$$

   Dadurch lassen sich alle $A$-Regeln in einer Zeile zusammenfassen.

2. $A \rightarrow \alpha[\beta]\gamma$ steht für $A \rightarrow \alpha\gamma|\alpha\beta\gamma$:
   man darf $\beta$ zwischen $\alpha$ und $\gamma$ einfügen, muss es aber nicht.

3. $A \rightarrow \alpha\{\beta\}\gamma$ steht für $A \rightarrow \alpha\gamma|\alpha B\gamma, B \rightarrow \beta|\beta B$: das Wort $\beta$ kann zwischen $\alpha$ und $\gamma$ iteriert (auch null–mal) eingefügt werden.

# Beispiel zur Backus–Naur–Form

Die Regeln

$$E \rightarrow T \ , \ E \rightarrow E + T \ , \ T \rightarrow F \ , \ T \rightarrow T * F \ , \ F \rightarrow a \ , \ F \rightarrow (E)$$

für korrekt geklammerte Rechenausdrücke können in BNF kompakt notiert werden wie folgt:

$$E \rightarrow T|E + T \ , \ T \rightarrow F|T * F \ , \ F \rightarrow a|(E)$$

Oder noch kompakter:

$$E \rightarrow [E+]T \ , \ T \rightarrow [T*]F \ , \ F \rightarrow a|(E)$$

bzw. in der Form:

$$E \rightarrow \{T+\}T \ , \ T \rightarrow \{F*\}F \ , \ F \rightarrow a|(E)$$

# Exemplarische Lernziele zur Chomsky–Hierarchie

- Grundbegriffe kennen (Vokabeln lernen!) und intellektuell beherrschen
- bei Operationen auf Sprachen die Ergebnissprache beschreiben
- bei Operationen auf Relationen die Ergebnisrelation beschreiben
- zu einer gegebenen Sprache eine passende Grammatik finden
- zu einer gegebenen Grammatik die davon erzeugte Sprache „intelligent erraten“
- zu einem Wort aus einer Sprache die passende grammatische Ableitung (im kontextfreien Fall auch den Syntaxbaum) finden
- Mehrdeutigkeit einer kontextfreien Grammatik erkennen und, sofern möglich, vermeiden können