

Die Chomsky–Hierarchie

Hans U. Simon (RUB)

mit Modifikationen von

Maike Buchin (RUB)

Lehrstuhl Mathematik und Informatik

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 Σ werden als Zeichen, Symbole oder Buchstaben bezeichnet.

Beispiele:

1. $\Sigma = \{a, \dots, z\} \cup \{A, \dots, Z\} \cup \{0, \dots, 9\}$.
2. $\Sigma = \{a, b\}$.
3. $\Sigma = \{0, 1\}$.

Zeichenfolgen

Σ^+ = Menge der nicht-leeren Zeichenfolgen über Alphabet Σ .

Die Elemente von Σ^+ 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:

- $\{a, b\}^+ = \{a, b, aa, ab, ba, bb, aaa, \dots\}$.
- $|a| = |ba|_a = 1$, $|ba| = 2$, $|aaa| = 3$.
- $|informatik| = 10$.

Zeichenfolgen (fortgesetzt)

Das **leere Wort** der Länge 0 wird mit ε 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 Σ .

Konkatenation von Wörtern

Wörter konkateniert (= aneinandergelagert) ergeben wieder Wörter:

Sei $u = \sigma_1 \dots \sigma_m$ und $v = \tau_1 \dots \tau_n$, dann ist $uv = \sigma_1 \dots \sigma_m \tau_1 \dots \tau_n$,

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 Σ .

Beispiele:

1. $L = \{p \in \{0, 1\}^* \mid p \text{ ist Binärdarstellung einer Primzahl}\}$.
2. $L = \{\varepsilon, 0, 1, 001\}$.
3. $L = \{\text{alle gültigen JAVA-Programme}\}$.

Formale Sprachen (fortgesetzt)

Neben den üblichen Mengenoperationen (Vereinigung, Durchschnitt, Komplement, Mengendifferenz,...) sind auf Sprachen die folgenden Operationen von Interesse:

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

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

Kleenescher Abschluss $A^+ = \bigcup_{n \geq 1} A^n$ und $A^* = \bigcup_{n \geq 0} A^n$.

Formale Sprachen (fortgesetzt)

Beispiel Betrachte die Sprachen

$$A = \{w \in \{a, b\}^* \mid |w| = 5\} \text{ und } B = \{w \in \{a, b\}^* \mid |w| = 7\} .$$

Dann gilt

$$AB = \{w \in \{a, b\}^* \mid |w| = 12\}$$

$$A^{20} = \{w \in \{a, b\}^* \mid |w| = 100\}$$

$$A^* = \{w \in \{a, b\}^* \mid 5 \text{ ist Teiler von } |w|\}$$

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 $xRyRz$.

Zur Erinnerung: eine Relation R heisst:

1. **reflexiv**, wenn $\forall x \in M : xRx$
2. **transitiv**, wenn $\forall x, y, z \in M : xRyRz \Rightarrow xRz$
3. **symmetrisch**, wenn $\forall x, y \in M : xRy \Rightarrow yRx$

Verknüpfung von Relationen

Ähnlich wie bei formalen Sprachen betrachten wir folgende Operationen:

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

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

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

Es gilt:

1. $R^n = \{(x, y) \in M \times M \mid \exists z_1, \dots, z_{n-1} : xRz_1Rz_2 \cdots Rz_{n-1}Ry\}$ 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

M = Menge von Personen

R = die Relation „ist Schwester von“

S = die Relation „ist Vater oder Mutter von“

Dann gilt:

RS = die Relation „ist Tante von“

S^2 = die Relation „ist Großvater oder Großmutter von“

S^+ = die Relation „ist (echter) Vorfahr von“

Jetzt gehts ...

... zur Sache !

Grammatiken

Eine **Grammatik** besteht aus **vier Komponenten** $G = (V, \Sigma, P, S)$:

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

Dabei sind V, Σ, 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 Σ heißen **Sätze**; **Strings über $V \cup \Sigma$** heißen **Satzformen**.

Schreibweise: Wir verwenden i.d.R. Grossbuchstaben für Variablen und Kleinbuchstaben für Terminale.

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 $y \rightarrow y' \in P$.

Es ist also „ \Rightarrow_G “ eine Relation auf $(V \cup \Sigma)^*$.

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

\Rightarrow_G^n = n -fache Potenz von \Rightarrow_G

\Rightarrow_G^+ = transitive Hülle von \Rightarrow_G

\Rightarrow_G^* = reflexive–transitive Hülle von \Rightarrow_G

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

$$w_0, w_1, \dots, w_n$$

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 Σ :

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

Beispiel: Korrekt geklammerte Rechenausdrücke

Betrachte die Grammatik G mit den Komponenten

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

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

- $S = E$, 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:

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

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

Chomsky–Hierarchie ohne Sonderregeln für ε

Typ 0: Eine Grammatik mit Regeln der allgemeinen Form $w \rightarrow w'$ 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 $w \rightarrow w'$ mit $w, w' \in (V \cup \Sigma)^+$ und $|w| \leq |w'|$ heißt **kontextsensitive** (oder auch Typ 1) **Grammatik**.

Typ 2: Eine Grammatik mit Regeln der Form $X \rightarrow w$ 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 $X \rightarrow a$ oder $X \rightarrow aY$ 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 ε (schlecht, falls $\varepsilon \in L(G)$ erwünscht ist). Daher folgende Regelung:

Typ 1: Wir erlauben auch die Regel

$$S \rightarrow \varepsilon, \quad S \text{ Startsymbol.}$$

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

Typ 2 bzw. 3: Wir erlauben beliebige „ ε -Regeln“ der Form

$$A \rightarrow \varepsilon, \quad A \in V.$$

Sonderregeln für das leere Wort (fortgesetzt)

Satz:

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

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

Beweis für Typ 1:

„ \Rightarrow “: folgt sofort, denn sei G Grammatik mit Sonderregeln, dann ist G' mit $P' = P \setminus \{(S, \varepsilon)\}$ Grammatik für $L(G) \setminus \{\varepsilon\}$

„ \Leftarrow “: Sei G Grammatik ohne Sonderregeln und $\varepsilon \in L$ erwünscht. Dann füge die Regeln $S' \rightarrow S, S' \rightarrow \varepsilon$ hinzu und verwende S' als neue Startvariable.

Beweis für Typ 2: später.

Sonderregeln für das leere Wort (fortgesetzt)

Folgerung: Für formale Sprachen bleibt die Inklusionskette

regulär \Rightarrow kontextfrei \Rightarrow kontextsensitiv \Rightarrow Typ 0

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

Echtheit der Chomsky–Hierarchie (Übersicht)

1. Die Sprache $\{a^n b^n \mid n \geq 1\}$ ist **kontextfrei** aber **nicht regulär**.
2. Die Sprache $\{a^n b^n c^n \mid n \geq 1\}$ ist **kontextsensitiv** aber **nicht kontextfrei**.
3. Es gibt **Sprachen vom Typ 0**, die **nicht kontextsensitiv** sind (Beweis später).
4. Es gibt **überabzählbar viele formale Sprachen**, aber nur **abzählbar viele vom Typ 0**.

Daher 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.

Beispiel: Die reguläre Sprache $\{a^n b^m \mid n, m \geq 1\}$ wird erzeugt durch die Regeln $S \rightarrow aS, S \rightarrow aB, B \rightarrow b, B \rightarrow bB$.

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

Beweis:

„ \subseteq “: Ein Wort $a^n b^n c^n$ lässt sich ableiten durch

$$S \Rightarrow_G^{n-1} a^{n-1} S(BC)^{n-1} \Rightarrow_G a^n (BC)^n \Rightarrow_G^{n(n-1)/2} a^n B^n C^n \Rightarrow_G a^n b B^{n-1} C^n$$

$$\Rightarrow_G^{n-1} a^n b^n C^n \Rightarrow_G a^n b^n c C^{n-1} \Rightarrow_G^{n-1} a^n b^n c^n.$$

„ \supseteq “: folgt aus

- für Satzform w mit $S \Rightarrow_G^* w$ gilt: $|w|_a = |w|_{Bb} = |w|_{Cc}$
- für Satz w mit $S \Rightarrow_G^* w$ gilt: a 's vor b 's vor c 's.

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 Abl_n aller Satzformen der Maximallänge n , die sich aus S ableiten lassen:

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

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

Dabei kann Abl_n iterativ berechnet werden wie folgt:

Initialisierung $\text{Abl} := \{S\}$.

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

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

existiert, nimm auch w in Abl auf.

Das Wortproblem (fortgesetzt)

Skizze der Korrektheit:

- In jedem Schritt der Iteration, enthält Abl nur Satzformen, die sich aus S ableiten lassen.
- Da es nur endlich (exponentiell) viele Wörter der Länge $\leq n$ über $\Sigma \cup V$ gibt, stoppt die Iteration.
- Jeder Iterationsschritt dauert endliche Zeit, da endlich viele Teilwörter auf endlich viele Regeln geprüft werden.

Beispiel: Um zu prüfen, ob $aabb$ und $abbb$ in der von $S \rightarrow ab$, $S \rightarrow aSb$ erzeugten Sprache liegen, berechnen wir iterativ $Abl_4 = \{S, ab, aSb, aabb\}$, und stellen fest, dass $aabb \in L(G)$ und $abbb \notin L(G)$.

Syntaxbäume

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

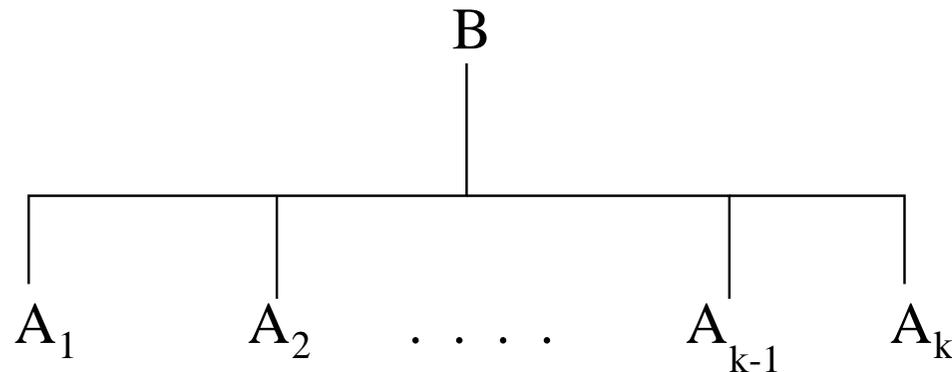


Abbildung 1: Visualisierung einer Regel $B \rightarrow A_1 \cdots A_k$.

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 Σ (oder mit ε) markiert.
5. Von links nach rechts gelesen ergeben die Blattmarkierungen das Wort w .

Beispiel

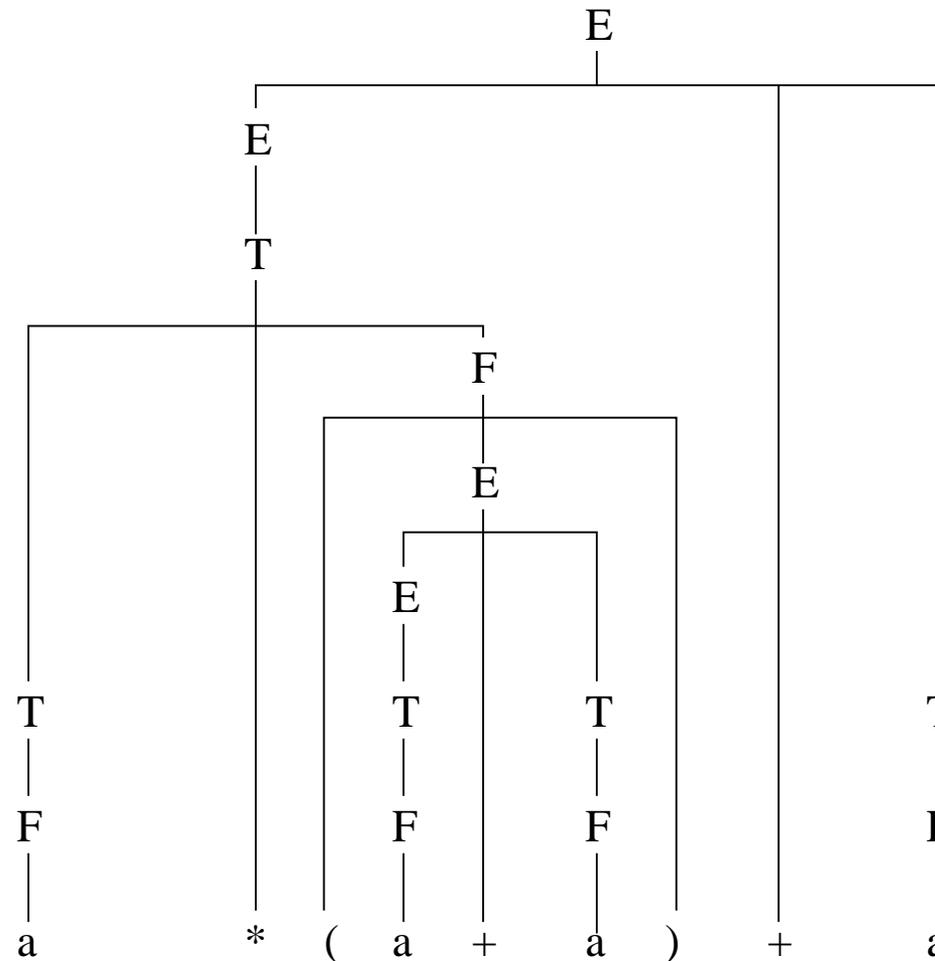


Abbildung 2: Der Syntaxbaum zur Ableitung von $a * (a + a) + a$ mit der Grammatik für korrekt geklammerte Rechenausdrücke.

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:

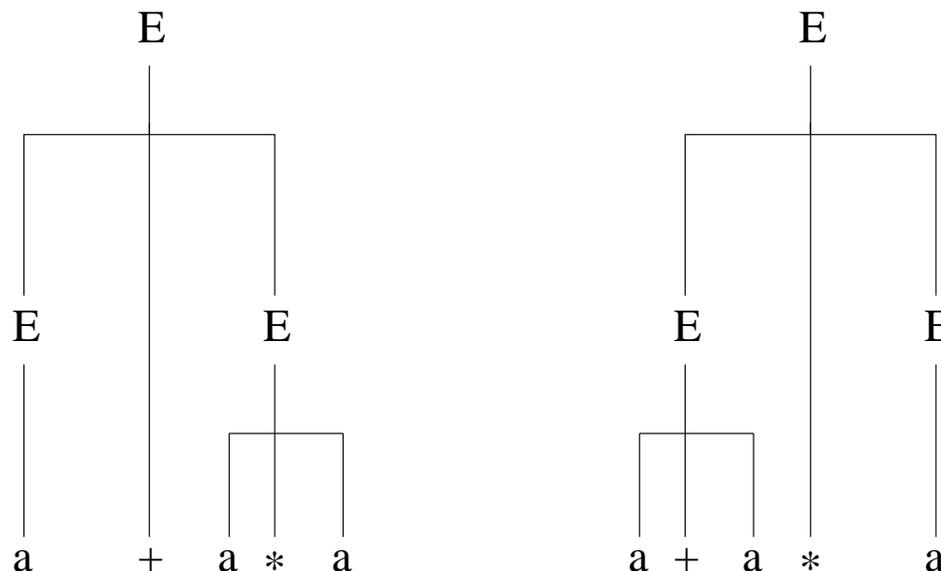


Abbildung 3: Zwei verschiedene Syntaxbäume für den Rechenausdruck $a + a * a$.

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 P_1

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

erzeugen die Sprache L der korrekten Klammersausdrücke.

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 Klammersausdrücke durchlaufen nicht-negative Niveaus und führen am Ende zu Niveau 0.
- Ein atomarer Klammersausdruck ist einer der erst am Ende, aber nicht zwischendurch, Niveau 0 erreicht.
- Jeder Klammersausdruck zerfällt eindeutig in eine Konkatenation atomarer Klammersausdrücke.

Beispiele für (korrekte und falsche) Klammersausdrücke

- Korrekt geklammerte Ausdrücke:

	()		(()	()		()
Niveau	1	0		1	2	1	2	1		1	0

- Falsch geklammerte Ausdrücke:

	(())		(
Niveau	1	2	1	0		1

Korrektheit der Grammatik

Behauptung: Die Grammatik G_1 mit den Regeln P_1 erzeugt L .

Beweis: Wir zeigen $L(G_1) = L$ in zwei Schritten.

\subseteq : Induktion (#Ableitungsschritte):

$n = 1$: $K \rightarrow \varepsilon$ liefert korrekten Klammerausdruck.

$n > 1$: $K \rightarrow KK$ sowie $K \rightarrow (K)$ führen nach I.V. zu korrekten Ausdrücken.

\supseteq : Induktion ($|w|$):

$n = 0$: $w = \varepsilon$ lässt sich durch $K \rightarrow \varepsilon$ ableiten.

$n > 1$: betrachte erste Verzweigung im Syntaxbaum: falls w aus nur einem atomaren Klammerausdruck besteht, muss diese der Regel $K \rightarrow (K)$ entsprechen; andernfalls der Regel $K \rightarrow KK$. Die Behauptung folgt dann mit I.V.

Demonstration der Mehrdeutigkeit

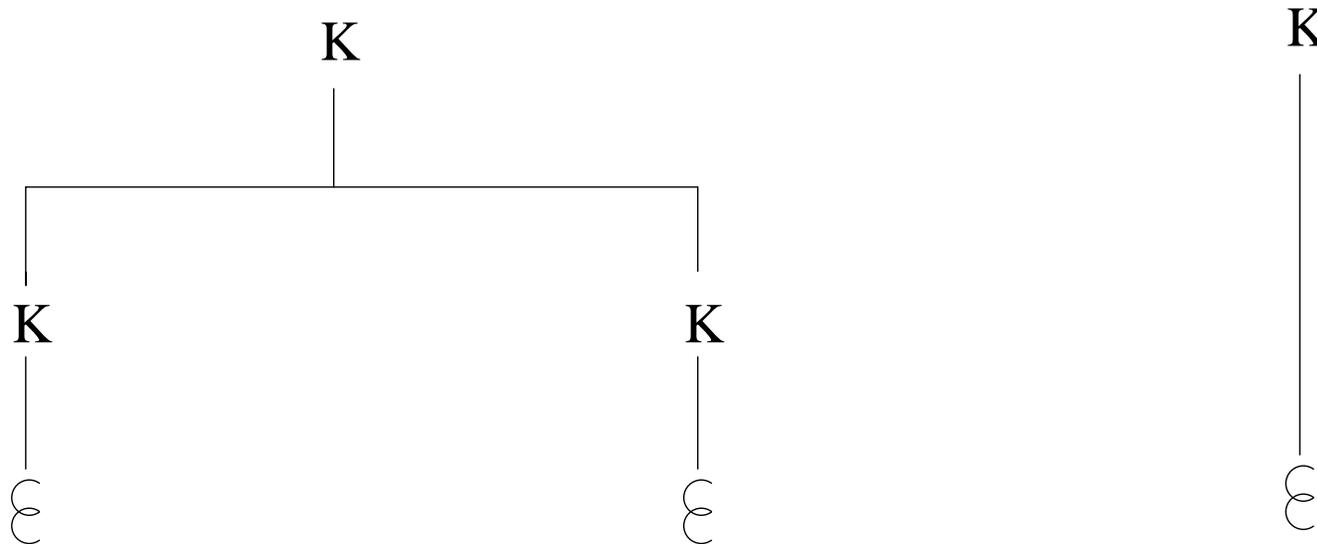


Abbildung 4: Zwei verschiedene Syntaxbäume für den leeren Klammersausdruck.

Eindeutige Grammatik für dieselbe Sprache

$$K \rightarrow \varepsilon, K \rightarrow (K)K$$

Behauptung: Die Grammatik G_2 mit obigen Regeln ist eindeutig.

Beweis: Sei $w \in L(G_2)$ beliebig.

Induktion ($|w|$):

$n = 0$: $w = \varepsilon$ hat eindeutigen Syntaxbaum entsprechend der Regel $K \rightarrow \varepsilon$.

$n > 1$: w hat die Gestalt $w = (\alpha)\beta$. Die erste Verzweigung im Syntaxbaum muss also der Regel $K \rightarrow (K)K$ entsprechen. Die Behauptung folgt dann mit I.V. angewandt auf α, β .

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 Programmiersprachen 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.

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

$$A \rightarrow \beta_1$$

$$A \rightarrow \beta_2$$

$$\cdots \quad \cdots$$

$$A \rightarrow \beta_n$$

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 β zwischen α und γ 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 β kann zwischen α und γ 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