Grundlagen Theoretischer Informatik 2

WiSe 2011/12 in Trier

Henning Fernau
Universität Trier
fernau@uni-trier.de

Grundlagen Theoretischer Informatik 2 Gesamtübersicht

- Organisatorisches; Einführung
- Ersetzungsverfahren: Grammatiken und Automaten
- Rekursionstheorie-Einführung
- Komplexitätstheorie-Einführung

Organisatorisches

Vorlesungen FR 12.15-13.45 im F59
Vorschlag: 12.25-13.55 ab 2. Semesterwoche

Übungsbetrieb in Form von einer Übungsgruppe
BEGINN: als Sprechstunde in der ersten Semesterwoche
FR 14.00-15.30, F 59

Dozentensprechstunde DO 13-14 in meinem Büro H 410 (4. Stock)

Mitarbeitersprechstunde (Stefan Gulan) DO 13-14 H 413

Tutorensprechstunde (Martin Puppe) DO 13-14 H 407

Benotung

Die Note ergibt sich ausschließlich aus der gezeigten Leistung bei der Abschlussprüfung.

Die Abschlussprüfung erfolgt mündlich (n.V.); unterschiedlicher Umfang für BEd bzw. BSc.-Studenten.

Zulassungskriterien

Um an der Abschlussprüfung teilnehmen zu dürfen, wird vorausgesetzt:

(1) Mindestens zweimaliges erfolgreiches Vorrechnen von Hausaufgaben sowie
(2) 40% der Hausaufgabenpunkte (Zweier-/ Dreiergruppenabgabe möglich) sowie
(3) Bestehen der Zwischenklausur.

Abgabe von Hausaufgaben

Diese sollte in Gruppen zu 2-3 Personen erfolgen.

Abgabeschluss ist MI 14 Uhr, im mit GTI 2 beschrifteten Kasten im 4. Stock vor dem gemeinsamen Sekretariat von Prof. Näher.
Verspätete Abgaben gelten als nicht abgegeben und werden dementsprechend mit 0 Punkten bewertet

In der darauffolgenden Übung (am Freitag) werden die korrigierten Lösungen zurückgegeben. Wir nutzen die Zeit (i.d.R. zwei Tage) auch dazu, uns die Namen derjenigen herauszusuchen, die gewisse Aufgaben dann vorzurechnen haben.
Wer die Aufgaben auf Aufforderung **nicht erläuternd vorrechnen** kann, bekommt die durch Hausaufgabenabgabe erzielten Punkte für das gesamte Aufgabenblatt abgezogen.
Selbstverständlich gilt das Vorrechnen dann nicht als “erfolgreich” im Sinne der vorigen Folie.
Sollte ein Vorrechnen nicht erfolgreich sein, wird i.d.R. ein weiterer Studierender Gelegenheit zum Vorrechnen bekommen.

Die Lösungen sind handschriftlich anzufertigen; weder Schreibmaschinen- noch Computerausdrucke werden akzeptiert, erst recht keine Kopien.

Probleme ? Fragen ?

Klären Sie bitte Schwierigkeiten mit Vorlesungen oder Übungen möglichst umgehend in den zur Verfügung gestellten Sprechzeiten.

In der Tutorensprechstunde stehen Ihnen (alternierend) Studenten höherer Semester zu Rückfragen bereit.

Wir helfen Ihnen gerne!
. . . wir sind aber keine Hellseher, die Ihnen Ihre Schwierigkeiten an der Nasenspitze ansehen. . .

Erinnerung: *Mengenprodukt* $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Mengenpotenzen und Wörter

Ist X eine Menge und $n \in \mathbb{N}$, $n > 1$, so definiere: $X^1 := X$ und $X^n = X \times X^{n-1}$.
Dies ist ein (bekanntes) Beispiel für eine *rekursive Definition*.
Ein Element aus X^n heißt auch *Folge der Länge n über X*.
Ist X ein Alphabet, so nennen wir eine Folge auch ein *Wort* (der Länge n).

Lemma: Gilt $|X| < \infty$, so ist $|X^n| = |X|^n$ für beliebige $n \in \mathbb{N}, n \geq 1$.

Verkettung: eine Verknüpfung auf X^n ?

Beispiel: Nach der rekursiven Definition z.B. für $\{a, b, c\}^3$ gilt:

$$(a, (a, b)) \in \{a, b, c\}^3.$$

Die Folge $(a, (a, b))$ der Länge drei entsteht durch *Verkettung* (oder *Aneinanderreihung*, *Hintereinanderschreiben*, *(Kon-)Katenation*) des Wortes a der Länge eins mit dem Wort (a, b) der Länge zwei, und letzteres wieder durch Verkettung der Wörter a und b der Länge eins.

Problem: X^n ist bezüglich der $\cdot$ geschriebenen Operation “Verkettung” nicht abgeschlossen.

Wir lieben jedoch Operationen, bezüglich derer unsere Grundmenge abgeschlossen ist.

Verkettung: eine Verknüpfung auf X^+ !

Lösung: Betrachte $X^+ := \bigcup_{n \geq 1} X^n$.

Satz: Die Menge X^+ der Wörter beliebiger positiver Länge über dem Alphabet X ist bezüglich der Verkettung abgeschlossen. Überdies handelt es sich um eine assoziative Operation.

Im Beweis benötigt man:
X^n und $X^{n-\ell} \times X^\ell$ bezeichnen für jedes $1 \leq \ell < n$ "dasselbe."
Deshalb (Assoziativität!) kann man auch die vielen Klammern bei der Notation von Wörtern fortlassen:
Wir schreiben also aab statt $(a, (a, b))$.

Verkettung: eine Verknüpfung auf X^+ !

Vornehmere Sprechweise: Eine Menge M zusammen mit einer Verknüpfung $\circ$ auf M, d.h., einer Abbildung $\circ : M \times M \to M$, heißt *Halbgruppe*, falls $\circ$ assoziativ ist.

Satz: $(X^+, \cdot)$ ist eine Halbgruppe, die sogenannte *frei erzeugte Halbgruppe (über X)*.

Beispiel: $LASS \in \{A, D, S, L\}^4$ und $DAS \in \{A, D, S, L\}^3$, also gilt für die Konkatenation $LASSDAS \in \{A, D, S, L\}^7$.

Eine Struktur $(M, \circ, e)$ ist ein *Monoid* gdw. $(M, \circ)$ eine Halbgruppe ist und e ein neutrales Element von $\circ$ ist.

Verkettung: eine Verknüpfung auf X^* !

Problem: X^+ ist kein Monoid ?!

Lösung: Betrachte $X^* := \bigcup_{n \geq 0} X^n = X^+ \cup \{\lambda\}$.
λ (andere Notationen: ϵ oder e) ist *das leere Wort*, formal ein künstlich hinzugefügtes neutrales Element.

Satz: $(X^*, \cdot, \lambda)$ ist ein Monoid, das so genannte *frei erzeugte Monoid (über X)*.

Komplexprodukte

Erinnerung: Ist X eine Menge, so bezeichnet 2^X ihre Potenzmenge.

Ist $\circ$ eine Verknüpfung auf X, so wird $\circ$ durch $A \circ B = \{a \circ b \mid a \in A, b \in B\}$ eine Verknüpfung auf 2^X, genannt *Komplexprodukt*.

Satz: Ist $(H, \circ)$ eine Halbgruppe, so auch $(2^H, \circ)$.

So wird insbesondere die Konkatenation zwischen Sprachen definiert:

Beispiel: $\{ab, ba\} \cdot \{ca, cb\} = \{abca, baca, abcb, bacb\}$.

Formale Sprachen—eine Annäherung

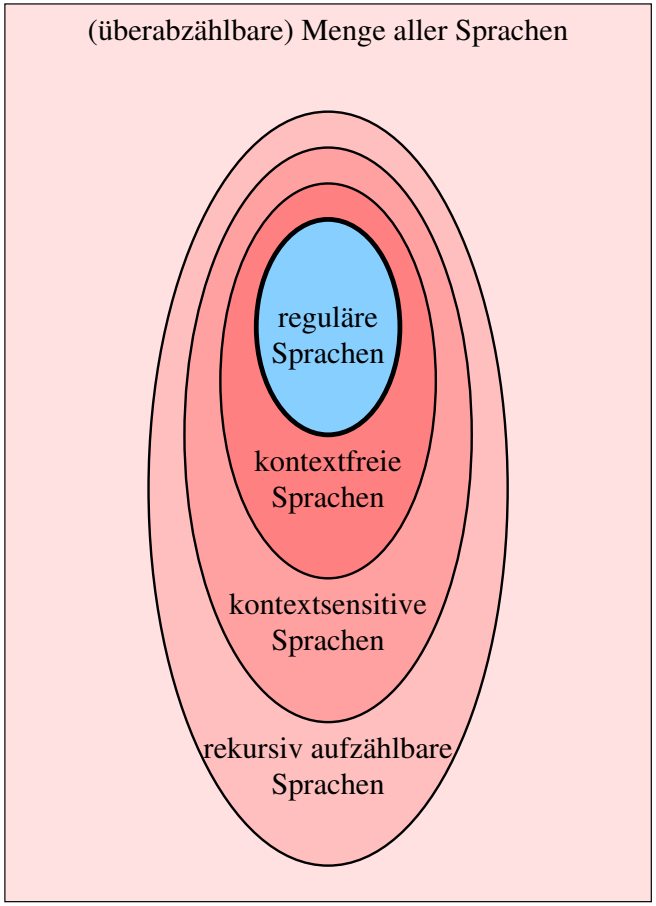
Was ist eine Sprache ?
Eine *Sprache* ist eine Menge von Wörtern.

Das sollte jetzt auch mathematisch klar sein:

Eine Menge Σ heißt *Alphabet*, falls Σ eine endliche, nicht-leere Menge ist, i.Z.: $|\Sigma| < \infty$ und $\Sigma \neq \emptyset$.

Kurz gesagt: Eine Sprache L (über Σ) ist eine Teilmenge des von der Menge Σ frei erzeugten Monoids, i.Z.: $L \subseteq \Sigma^*$.

Bei formalen Sprachen unterscheidet man zunächst vier Grundtypen:



Chomsky-Hierarchie:

- Typ-0-Sprachen: *rekursiv-aufzählbar*
  Beschreibung der Möglichkeiten (und der Grenzen) des Programmierens
- Typ-1-Sprachen: *kontextsensitiv*
- Typ-2-Sprachen: *kontextfrei*
  Syntax (“Grammatik”) von Programmiersprachen
- Typ-3-Sprachen: *regulär/rechtslinear*
  lexikalische Analyse bei Programmiersprachen, Suchalgorithmen in Texten,
  u.v.m.

Chomsky-Grammatik

Eine *Grammatik* ist ein 4-Tupel $G = (N, T, P, S)$ mit:

- N ist eine endliche Menge von *Variablen* oder *Nicht-Terminal-Zeichen*.
- T ist eine endliche Menge, das *Terminalalphabet*, wobei $N \cap T = \emptyset$.
- P ist eine endliche Menge von *Regeln* oder *Produktionen*
  $P \subseteq (N \cup T)^+ \times (N \cup T)^*$
- $S \in N$ ist das *Startsymbol*
- Entfällt die Unterscheidung zwischen N und T, d.h., es ist nur ein *(Gesamt-) Alphabet* Σ gegeben, und gilt $P \subseteq \Sigma^* \times \Sigma^*$, $|P| < \infty$, so heißt (Σ, P) ein *(sequentielles) Ersetzungssystem*.

Seien nun u, v aus $(N \cup T)^*$.
Wir schreiben $u \Rightarrow_G v$, falls

$$u = xyz, \quad v = xy'z$$

für $x, z \in (N \cup T)^*$, wobei $y \to y'$ eine Regel aus P ist.
Sprechweise: *v kann direkt aus u abgeleitet werden*.

In der Folge werden wir das tiefgestellte G auch weglassen.

Eine Folge von direkten Ableitungen

$$w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$$

heißt *Ableitung von w_n aus w_1*.

Ohne Angabe der 'Zwischenwörter' schreiben wir: $w_1 \Rightarrow^* w_n$.

$\Rightarrow^*$ ist die reflexiv-transitive Hülle von $\Rightarrow$.

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

Wir nennen $L(G)$ die *von der Grammatik G erzeugte Sprache*.

Endliche Automaten als Ersetzungssysteme

Ersetzungssysteme sind allgemeiner als Grammatiken.
So kann man endliche Automaten wie folgt beschreiben:
Ein Ersetzungssystem $E = (\Sigma, P)$ heißt *endlicher Automat*, falls
$\Sigma = Z \cup T$, $Z \cap T = \emptyset$, $P \subseteq ZT \times Z$.
Z: Zustände; T: Eingabezeichen
Sei $z_0 \in Z$ Anfangszustand und $Z_f \subseteq Z$ Endzustände, so *akzeptiert* (E, z_0, Z_f) die Sprache:

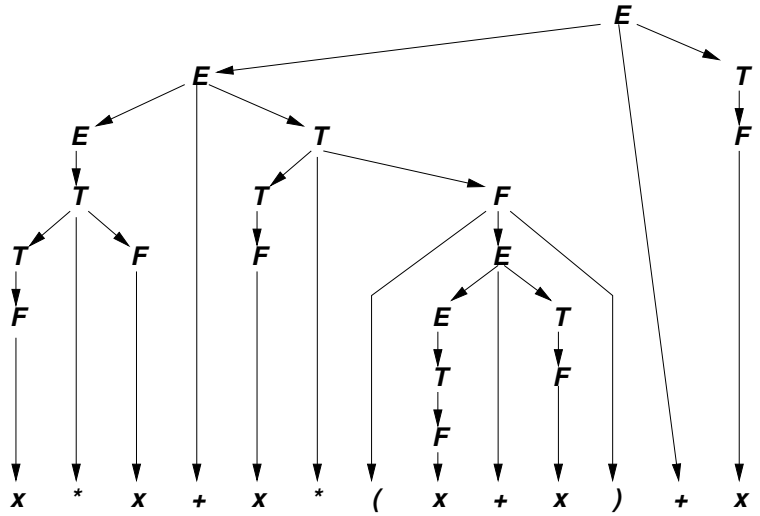
$$L_a(E, z_0, Z_f) = \{w \in T^* \mid \exists z_f \in Z_f : z_0 w \Rightarrow^* z_f\}$$

Aufgaben: 1. Geben Sie in diesem Formalismus einen endlichen Automaten an, der die Sprache $\{a\}^*\{b\}\{b\}^*\{c\}^*$ beschreibt.
2. Wie kann man in diesem Formalismus einen deterministischen endlichen Automaten beschreiben?

arithmetische Ausdrücke Sei $G = (\{E, T, F\}, \{(,), x, +, *\}, P, E)$ mit

$$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to x, F \to (E)\}$$

Gegeben sei das Wort: $x * x + x * (x + x) + x$



$$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to x, F \to (E)\}$$

Der *Syntaxbaum* beschreibt die einzelnen Ableitungsschritte vom Startsymbol bis zum Wort:

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

Aufbau der Syntaxbäume:

- innere Knoten: Nichtterminale
- Blätter: Terminale
- abgeleitetes Wort
  durch Konkatenation der Blätter von links nach rechts
- genauer: Syntaxbaum ausgehend von S sukzessive aufgebaut
- Anwendung von Regel $A \rightarrow z_1.....z_n$:
  Syntaxbaum durch n Kinder von A in geordneter Form von links nach rechts ergänzt.
- Jedes Kind wird mit zugehörigem Zeichen z_i beschriftet.
- Wichtig: Syntaxbäume kann man nur zeichnen, wenn alle linken Regel-Seiten aus Einzelzeichen bestehen!

Beobachte

- Im Beispiel wird immer das am weitesten links stehende Nichtterminalzeichen abgeleitet
  $\Longrightarrow$ *Linksableitung*
- Analog: Rechtsableitung...
- Hier stets: gleiche Form des Syntaxbaumes!
- Ableitungsregeln der Form $A \rightarrow z_1.....z_n$ mit $A \in N, +z_i \in (N \cup T)^*$,
  $\Rightarrow$ Grammatik heißt *kontextfrei*.
- Kontextfreiheit ist notwendig für Syntaxbäume...

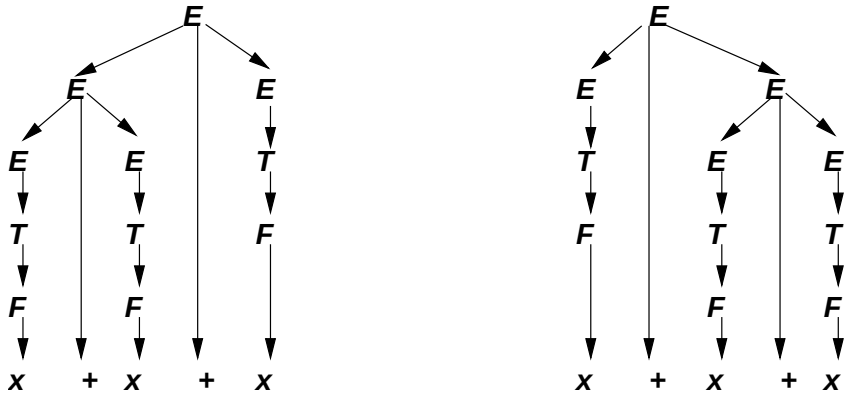
Modifikation der Grammatik: $G' = \{E, T, F\}, \{(,), x, +, *\}, P', E)$

$$P' = \{E \to T, E \to E + E, T \to F, T \to T * F, F \to x, F \to (E)\}$$

statt

$$P = \{E \to T, E \to E + T, T \to F, T \to T * F, F \to x, F \to (E)\}$$

$\Longrightarrow$ gleiche Sprache,
aber z.B. $x + x + x$ mit unterschiedlichen Ableitungsbäumen:



- G' nennt man auch *mehrdeutig*:
  es gibt Worte w aus G' mit verschiedenen Ableitungsbäumen

- Ansonsten: Grammatik heißt *eindeutig*

- 'Mehrdeutigkeit' ist Eigenschaft der Grammatik,
  Sprachen können sowohl eindeutige als auch mehrdeutige Grammatiken haben

- Kontextfreie Sprachen L heißen *inhärent mehrdeutig*, wenn jede Grammatik G, die L erzeugt, mehrdeutig ist.
  Schlecht für Anwendungen wie Compilerbau.

Gegeben sei $G = (\{S\}, \{\ [\ ,\ ]\ \}, P, S)$ mit

$$P = \{S \to\ [\ S\ ]\ , S \to\ [\ ]\ \}$$

Warum gilt: $L(G) := \{\ [^n\ ]^n \mid n > 0\}$?

Behauptung: $L_n := \{x \in \{S, [,]\}^* \mid S \Rightarrow^n x\}$ erfüllt $L_n = L'_n$ mit $L'_n := \{[^n S]^n, [^n]^n\}$, falls $n > 0$, und $L_0 = \{S\}$.
Beweis durch vollständige Induktion.
$n = 0$ klar, ebenso $n = 1$.
Induktionsschritt: Betrachte Ableitung $S \Rightarrow^n w \Rightarrow v$.
Nach Induktionshypothese gilt: $w = [^n S]^n$ oder $w = [^n]^n$.
Nur im ersten Fall kann die Ableitung fortgesetzt werden.
Eine Anwendung der Regel $S \to\ [\ S\ ]$ liefert $v = [^{n+1} S]^{n+1}$.
Eine Anwendung der Regel $S \to\ [\ ]$ liefert $v = [^{n+1}]^{n+1}$.
Da dies die einzigen Regeln sind, folgt $L_{n+1} = L'_{n+1}$.

Aufgabe Gegeben sei $G = (\{S, B, C\}, \{a, b, c\}, P, S)$ mit

$$\begin{aligned} P \quad = \{ \quad & S \to aSBC, S \to aBC, CB \to BC, \\ & aB \to ab, bB \to bb, bC \to bc, cC \to cc\} \end{aligned}$$

Wir wollen zeigen, dass gilt:

$$L(G) := \{a^n b^n c^n | n > 0\}$$

(Achtung: Hier gibt es keine Syntaxbäume, da G nicht kontextfrei ist!)

Wie beweist man so eine Behauptung $L(G) = L$?

In der Regel ist es sinnvoll, die Behauptung in zwei Schritten sich zu überlegen:
(1) $L \subseteq L(G)$ und
(2) $L(G) \subseteq L$.

Der erste Schritt ist oft der einfachere.
Beim zweiten Schritt hat man mit möglichen bösen Ableitungen zu kämpfen.

Der Nachweis jedes der Schritte erfordert in der Regel mehrere (kleinere) Induktionsbeweise.
Manchmal (siehe oben) lassen sich die beiden Schritte "zusammenlegen."
Dann überlegt man sich zumeist, wie die Menge der in n Schritten ableitbaren Wörter (über dem Gesamtalphabet) aussieht.

$L(G) = \{a^n b^n c^n | n > 0\} =: L$! 1. Schritt: $L \subseteq L(G)$.

- $(n-1)$-faches Anwenden der Regel $S \to aSBC$:

$$S \Rightarrow^* a^{n-1} S (BC)^{n-1}$$

- einmaliges Anwenden von $S \to aBC$:

$$a^{n-1} S (BC)^{n-1} \Rightarrow a^n (BC)^n$$

- mehrfaches Vertauschen $CB \to BC$:

$$a^n (BC)^n \Rightarrow^* a^n B^n C^n$$

- Zusammengesetzt $S \Rightarrow^* a^n B^n C^n$
- einmal $aB \to ab$, $(n-1)$-mal $bB \to bb$:$S \Rightarrow^* a^n b^n C^n$
- einmal $bC \to bc$, $(n-1)$-mal $cC \to cc$:$S \Rightarrow^* a^n b^n c^n$

Ein Induktionsbeweis en detail

Behauptung: Für alle $n \geq 1$ gilt: $S \Rightarrow^* a^{n-1}S(BC)^{n-1}$.
Präzisierung: Für alle $n \geq 1$ gilt: $S \Rightarrow^{n-1} a^{n-1}S(BC)^{n-1}$.
Induktionsanfang (IA) für $n = 1$: $S \Rightarrow^0 S$. OK!
(Unnötiger Test für $n = 2$: $S \Rightarrow aSBC$. Stimmt auch!)
Induktionshypothese (Induktionsvoraussetzung IV):
Angenommen, die Aussage(form) gelte für $n = m$.
Formaler IV: $S \Rightarrow^{m-1} a^{m-1}S(BC)^{m-1}$
Wir müssen daraus die Gültigkeit der Behauptung für $n = m + 1$ ableiten.
Formale IB: $S \Rightarrow^m a^m S(BC)^m$.
Eine $(m + 1)$-Schritt-Ableitung lässt sich schreiben als eine m-Schritt-Ableitung, gefolgt von einem weiteren Ableitungsschritt. Nach IV und nach einer (weiteren) Anwendung der Regel $S \to aSBC$ gilt: $S \Rightarrow^{m-1} a^{m-1}S(BC)^{m-1} \Rightarrow a^m S(BC)^m$.
Also gilt die Induktionsbehauptung.
Nach dem Prinzip der vollständigen Induktion folgt die “Präzisierung” und damit die ursprüngliche Behauptung.

Umkehrung: 2. Schritt: $L(G) \subseteq L$.

- Balancebedingung per Induktion:
  $S \Rightarrow^* w$ impliziert $\#_a w = \#_b w + \#_B w = \#_c w + \#_C w$
- Zudem per Induktion: $S \Rightarrow^* w$ impliziert

  $$w = a^i S v \text{ oder } w = a^i b^j c^k v$$

  für geeignete $i, j, k \in \mathbb{N}$, $i > 0$, und $v \in \{B, C\}^*$.
- Zusammengesetzt: $S \Rightarrow^* w$ und $w \in T^+$ impliziert:

  $$w = a^i b^j c^k v \text{ mit } i = j = k > 0 \text{ und } v = \lambda.$$

Chomsky-Hierarchie Eine Chomsky-Grammatik $G = (N, T, P, S)$ heißt

- *Typ-3-Grammatik* oder *rechtslinear*,
  wenn alle Produktionen von einer der Formen $A \to w$ oder $A \to wB$ sind ($w \in T^*, A, B \in N$).
- *Typ-2-Grammatik* oder *kontextfrei*,
  wenn alle Produktionen von der Form $A \to \gamma$ sind (mit $\gamma \in (N \cup T)^*, A \in N$).
- *Typ-1-Grammatik* oder *kontextsensitiv*,
  wenn alle Produktionen von der Form $\alpha A \beta \to \alpha \gamma \beta$ sind (mit $A \in N$, $\alpha, \beta, \gamma \in (N \cup T)^*, \gamma \neq \lambda$).
  $S \to \lambda$ nur dann gelten, falls S auf keiner rechten Seite einer Regel auftritt.
- *Typ-0-Grammatik* oder *allgemeine Chomsky-Grammatik*,
  wenn die Form der Produktionen nicht weiter eingeschränkt ist.