

Informatik IV

Theoretische Informatik

Kapitel 1

Grundbegriffe

Sommersemester 2019

Dozent: Prof. Dr. J. Rothe

im Wechsel mit

Jun.-Prof. Dr. D. Baumeister

Prof. Dr. M. Leuschel



HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Alphabet, Wort und Sprache

Definition

- Ein *Alphabet* ist eine endliche, nichtleere Menge Σ von Buchstaben (oder Symbolen).
Beispiel: $\Sigma_1 = \{a, b, c\}$ oder $\Sigma_2 = \{0, 1\}$.
- Ein *Wort über Σ* ist eine endliche Folge von Elementen aus Σ .
Beispiel: $w_1 = abc$ und $w_2 = caba$ sind Wörter über Σ_1 ;
 $w_3 = 1110$ und $w_4 = 0101$ sind Wörter über Σ_2 .

Alphabet, Wort und Sprache

Definition

- Die *Länge eines Wortes* w (bezeichnet mit $|w|$) ist die Anzahl der Symbole in w .
Beispiel: $|010| = 3$.
- Das *leere Wort* ist das eindeutig bestimmte Wort der Länge 0 und wird mit dem griechischen Buchstaben λ („Lambda“) bezeichnet.
(Beachte: λ ist nicht als Symbol eines Alphabets erlaubt.)
Es gilt $|\lambda| = 0$.
- Die *Menge aller Wörter über Σ* bezeichnen wir mit Σ^* .
Beispiel: $\{0, 1\}^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, \dots\}$.

Alphabet, Wort und Sprache

Definition

- Eine (*formale*) *Sprache* (über Σ) ist eine jede Teilmenge von Σ^* .
Beispiel: $L = \{00, 10\} \subseteq \Sigma^* = \{0, 1\}^*$.
- Die *leere Sprache* ist die Sprache, die keine Wörter enthält, und wird mit \emptyset bezeichnet. (Beachte: $\emptyset \neq \{\lambda\}$.)
- Die *Kardinalität einer Sprache* L ist die Anzahl der Wörter von L und wird mit $\|L\|$ bezeichnet.
Beispiel: $\|\{00, 10\}\| = 2$.

Operationen auf Wörtern und Sprachen

Definition

Seien A und B beliebige Sprachen über dem Alphabet Σ , d.h., $A, B \subseteq \Sigma^*$. Definiere

- die *Vereinigung von A und B* durch

$$A \cup B = \{x \in \Sigma^* \mid x \in A \text{ oder } x \in B\};$$

Beispiel: $\{10, 1\} \cup \{1, 0\} = \{10, 1, 0\}$.

- den *Durchschnitt von A und B* durch

$$A \cap B = \{x \in \Sigma^* \mid x \in A \text{ und } x \in B\};$$

Beispiel: $\{10, 1\} \cap \{1, 0\} = \{1\}$.

Operationen auf Wörtern und Sprachen

Definition

- die *Differenz von A und B* durch

$$A - B = \{x \in \Sigma^* \mid x \in A \text{ und } x \notin B\};$$

Beispiel: $\{10, 1\} - \{1, 0\} = \{10\}$.

- das *Komplement von A* durch

$$\bar{A} = \Sigma^* - A = \{x \in \Sigma^* \mid x \notin A\}.$$

Beispiel: $\overline{\{10, 1\}} = \{0, 1\}^* - \{10, 1\} = \{\lambda, 0, 00, 11, 01, 000, \dots\}$.

Operationen auf Wörtern und Sprachen

Definition

- Die *Konkatenation (Verkettung) von Wörtern* $u, v \in \Sigma^*$ ist ein Wort $uv \in \Sigma^*$, das wie folgt definiert ist:
 - Ist $u = v = \lambda$, so ist $uv = vu = \lambda$.
 - Ist $v = \lambda$, so ist $uv = u$.
 - Ist $u = \lambda$, so ist $uv = v$.
 - Sind $u = u_1 u_2 \cdots u_n$ und $v = v_1 v_2 \cdots v_m$ mit $u_i, v_i \in \Sigma$, so ist

$$uv = u_1 u_2 \cdots u_n v_1 v_2 \cdots v_m.$$

Beispiel: $u = 01$, $v = 10$, $uv = 0110$ und $vu = 1001$.

Operationen auf Wörtern und Sprachen

Definition

- Die *Konkatenation (Verkettung) von Sprachen A und B* (also von Mengen von Wörtern) ist die Sprache

$$AB = \{ab \mid a \in A \text{ und } b \in B\}.$$

Beispiel: $\{10, 1\}\{1, 0\} = \{101, 100, 11, 10\}$.

- Die *Iteration einer Sprache $A \subseteq \Sigma^*$* (die *Kleene-Hülle von A*) ist die Sprache A^* , die induktiv definiert ist durch

$$A^0 = \{\lambda\}, \quad A^n = AA^{n-1}, \quad A^* = \bigcup_{n \geq 0} A^n.$$

Definiere die *λ -freie Iteration von A* durch $A^+ = \bigcup_{n \geq 1} A^n$.

Operationen auf Wörtern und Sprachen

Es gilt: $A^+ \cup \{\lambda\} = A^*$.

Es gilt nicht: $A^+ = A^* - \{\lambda\}$. Warum nicht?

Beispiel:

$$\{10, 1\}^0 = \{\lambda\},$$

$$\{10, 1\}^1 = \{10, 1\},$$

$$\{10, 1\}^2 = \{10, 1\}\{10, 1\} = \{1010, 101, 110, 11\},$$

$$\{10, 1\}^3 = \{10, 1\}\{10, 1\}^2 = \{101010, \dots\}.$$

Operationen auf Wörtern und Sprachen

Definition

- Die *Spiegelbildoperation* für ein Wort $u = u_1 u_2 \cdots u_n \in \Sigma^*$ ist definiert durch

$$sp(u) = u_n \cdots u_2 u_1.$$

Beispiel: $sp(0100101) = 1010010$.

- Die *Spiegelung einer Sprache* $A \subseteq \Sigma^*$ ist definiert durch

$$sp(A) = \{sp(w) \mid w \in A\}.$$

Beispiel: $sp(\{100, 011\}) = \{001, 110\}$.

Operationen auf Wörtern und Sprachen

Definition

- Die *Teilwortrelation* auf Σ^* ist definiert durch

$$u \sqsubseteq v \iff (\exists v_1, v_2 \in \Sigma^*) [v_1 u v_2 = v].$$

Gilt $u \sqsubseteq v$, so bezeichnen wir u als ein *Teilwort* (einen *Infix*) von v .

Beispiel: $100 \sqsubseteq 1010010$.

- Die *Anfangswortrelation* auf Σ^* ist definiert durch

$$u \sqsubseteq_a v \iff (\exists w \in \Sigma^*) [u w = v].$$

Gilt $u \sqsubseteq_a v$, so bezeichnen wir u als ein *Anfangswort* (einen *Präfix*) von v .

Beispiel: $101 \sqsubseteq_a 1010010$.

Grammatik

Definition

Eine *Grammatik* ist ein Quadrupel $G = (\Sigma, N, S, P)$, wobei

- Σ ein Alphabet (von so genannten **Terminalsymbolen**) ist,
- N eine endliche Menge (von so genannten **Nichtterminalen**) mit $\Sigma \cap N = \emptyset$,
- $S \in N$ das **Startsymbol** und
- $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$ die endliche Menge der **Produktionen (Regeln)**.

Dabei ist $(N \cup \Sigma)^+ = (N \cup \Sigma)^* - \{\lambda\}$.

Regeln (p, q) in P schreiben wir auch so:

$$p \rightarrow q.$$

Grammatik

Für Grammatikregeln mit gleicher linker Seite $A \in N$ schreiben wir auch kurz

$$A \rightarrow q_1 \mid q_2 \mid \cdots \mid q_n \quad \text{statt} \quad \begin{array}{l} A \rightarrow q_1 \\ A \rightarrow q_2 \\ \vdots \\ A \rightarrow q_n \end{array}$$

(aus der so genannten *BNF (Backus-Naur-Form)*).

Ableitungsrelation, Sprache einer Grammatik

Definition

Sei $G = (\Sigma, N, S, P)$ eine Grammatik, und seien u und v Wörter in $(N \cup \Sigma)^*$.

- Definiere die *unmittelbare Ableitungsrelation bzgl. G* so:

$$u \vdash_G v \iff u = xpz, v = xqz,$$

wobei $x, z \in (N \cup \Sigma)^*$ und $p \rightarrow q$ eine Regel in P ist.

- Durch n -malige Anwendung von \vdash_G erhalten wir \vdash_G^n . Das heißt:

$$u \vdash_G^n v \iff u = x_0 \vdash_G x_1 \vdash_G \cdots \vdash_G x_n = v$$

für $n \geq 0$ und für eine Folge von Wörtern $x_0, x_1, \dots, x_n \in (\Sigma \cup N)^*$.
Insbesondere ist $u \vdash_G^0 u$.

Ableitungsrelation, Sprache einer Grammatik

Definition

- Die Folge (x_0, x_1, \dots, x_n) , $x_i \in (\Sigma \cup N)^*$, $x_0 = S$ und $x_n \in \Sigma^*$ heißt *Ableitung von x_n in G* , falls $x_0 \vdash_G x_1 \vdash_G \dots \vdash_G x_n$.
- Definiere $\vdash_G^* = \bigcup_{n \geq 0} \vdash_G^n$. Man kann zeigen, dass \vdash_G^* die reflexive und transitive Hülle von \vdash_G ist, d.h. die kleinste binäre Relation, die reflexiv und transitiv ist und \vdash_G umfasst.
- Die *von der Grammatik G erzeugte Sprache* ist definiert als

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

- Zwei Grammatiken G_1 und G_2 heißen *äquivalent*, falls

$$L(G_1) = L(G_2).$$

Welche Sprachen interessieren uns?

- Alle korrekten Programme in Java, Python, C, ... :
keine Typfehler, keine Laufzeitfehler, Terminierung, ...
- Die Eingabe-Ausgabepaare, die ein Computerprogramm generiert (siehe Fib.py)
- Alle Tautologien oder erfüllbaren Formeln der Aussagenlogik, Prädikatenlogik, ...
- Alle Zeilen eines Textes, die ein gewisses Suchwort (oder kleine Abänderungen davon) beinhalten
- Andere Gebiete (z. B. Bioinformatik): Genomsequenzen

Wie erzeugen wir diese Sprachen? Mit Grammatiken.

Beschreibung aus dem Handbuch – Python

The if Statement

```
if test:
    suite
[elif test:
    suite]*
[else:
    suite]
```

The if statement selects from among one or more actions (statement blocks), and it runs the suite associated with the first if or elif test that is true, or the else suite if all are false.

The while Statement

```
while test:
    suite
[else:
    suite]
```

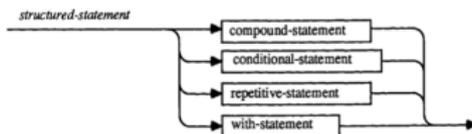
The while loop is a general loop that keeps running the first suite while the test at the top is true. It runs the else suite if the loop exits without hitting a break statement.

Abbildung: Quelle: O'Reilly. Python Pocket Reference.

Beschreibung aus dem Handbuch – Pascal

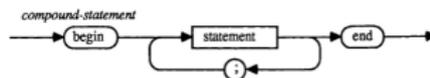
6.2 Structured-Statements

Structured-statements are made up of other statements that are to be executed either conditionally (conditional-statements), repeatedly (repetitive-statements), or in sequence (compound-statement or with-statement).



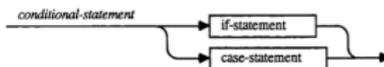
6.2.1 Compound-Statement

The compound-statement specifies the execution of its statement-sequence in its textual order.



6.2.2 Conditional-Statements

A conditional-statement selects one or none of its component statements for execution.



6.2.2.1 If-Statement

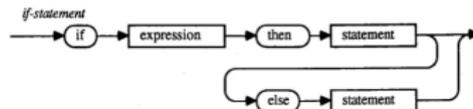


Abbildung: Quelle: TML Systems. TML Pascal: User's Guide and Reference

Beschreibung aus dem Handbuch – C

386

SYNTAX OF THE C LANGUAGE

```
hexadecimal-constant : (LEXICAL)
    hex-marker
    hexadecimal-constant hex-digit

identifier : (LEXICAL)
    underscore
    letter
    identifier following-character

if-else-statement :
    if ( expression ) statement else statement

if-statement :
    if ( expression ) statement

indirect-component-selection :
    postfix-expression -> name

indirection-expression :
    * unary-expression

initialized-declaration :
    declaration-specifiers initialized-declarator-list ;

initialized-declarator :
    declarator initializer-part_opt

initialized-declarator-list :
    initialized-declarator
    initialized-declarator-list , initialized-declarator

initializer :
    expression
    { initializer-list , opt }

initializer-list :
    initializer
    initializer-list , initializer
```

Abbildung: Quelle: Harbison, Steele Jr. C: A Reference Manual. Prentice-Hall.

Beschreibung aus dem Handbuch – Java

14.9. The `if` Statement

The `if` statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

```
IfThenStatement:  
    if ( Expression ) Statement  
  
IfThenElseStatement:  
    if ( Expression ) StatementNoShortIf else Statement  
  
IfThenElseStatementNoShortIf:  
    if ( Expression ) StatementNoShortIf else StatementNoShortIf
```

The `Expression` must have type `boolean` or `Boolean`, or a compile-time error occurs.

14.9.1. The `if-then` Statement

An `if-then` statement is executed by first evaluating the `Expression`. If the result is of type `boolean`, it is subject to unboxing conversion (§5.1.8).

If evaluation of the `Expression` or the subsequent unboxing conversion (if any) completes abruptly for some reason, the `if-then` statement completes abruptly for the same reason.

Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the contained `Statement` is executed; the `if-then` statement completes normally if and only if execution of the `Statement` completes normally.
- If the value is `false`, no further action is taken and the `if-then` statement completes normally.

14.9.2. The `if-then-else` Statement

An `if-then-else` statement is executed by first evaluating the `Expression`. If the result is of type `boolean`, it is subject to unboxing conversion (§5.1.8).

If evaluation of the `Expression` or the subsequent unboxing conversion (if any) completes abruptly for some reason, then the `if-then-else` statement completes abruptly for the same reason.

Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the first contained `Statement` (the one before the `else` keyword) is executed; the `if-then-else` statement completes normally if and only if execution of that statement completes normally.
- If the value is `false`, then the second contained `Statement` (the one after the `else` keyword) is executed; the `if-then-else` statement completes normally if and only if execution of that statement completes normally.

Abbildung: Quelle: Gosling et al.: The Java(TM) Language Specification.

Grammatik, Ableitung, erzeugte Sprache

Beispiel: Sei $G_1 = (\Sigma_1, \Gamma_1, S_1, R_1)$ die folgende Grammatik:

- das terminale Alphabet ist $\Sigma_1 = \{a, b\}$,
- das nichtterminale Alphabet ist $\Gamma_1 = \{S_1\}$ und
- die Menge der Regeln ist gegeben durch

$$R_1 = \{S_1 \rightarrow aS_1b \mid \lambda\}.$$

Eine Folge von Ableitungsschritten für G_1 :

$$S_1 \vdash_{G_1} aS_1b \vdash_{G_1} aaS_1bb \vdash_{G_1} aabb$$

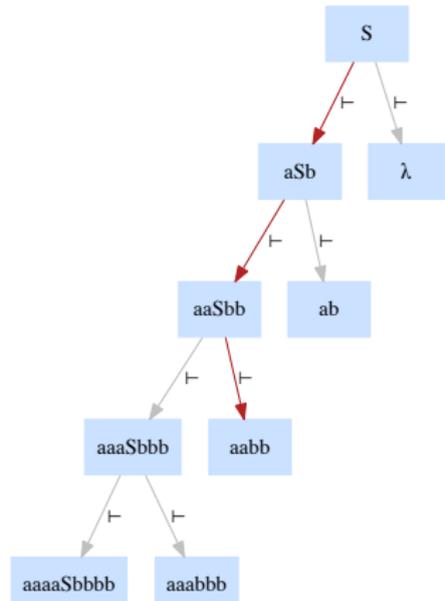
$$S_1 \vdash_{G_1}^1 aS_1b$$

$$S_1 \vdash_{G_1}^2 aaS_1bb$$

$$S_1 \vdash_{G_1}^3 aabb$$

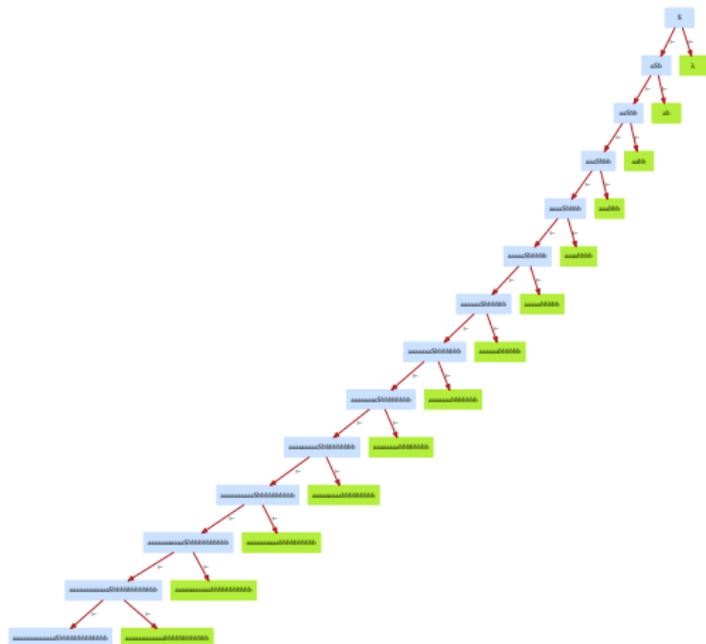
Grammatik, Ableitung, erzeugte Sprache

$$R_1 = \{S_1 \rightarrow aS_1b \mid \lambda\}$$



Grammatik, Ableitung, erzeugte Sprache

$$R_1 = \{S_1 \rightarrow aS_1b \mid \lambda\}$$



Grammatik, Ableitung, erzeugte Sprache

Beispiel: Sei $G_1 = (\Sigma_1, \Gamma_1, S_1, R_1)$ die folgende Grammatik:

- das terminale Alphabet ist $\Sigma_1 = \{a, b\}$,
- das nichtterminale Alphabet ist $\Gamma_1 = \{S_1\}$ und
- die Menge der Regeln ist gegeben durch

$$R_1 = \{S_1 \rightarrow aS_1b \mid \lambda\}.$$

Eine Ableitung für G_1 :

$$S_1 \vdash_{G_1} aS_1b \vdash_{G_1} aaS_1bb \vdash_{G_1} aabb \Rightarrow aabb \in L(G_1).$$

Offenbar erzeugt G_1 die Sprache $L(G_1) = \{a^n b^n \mid n \geq 0\}$.

Grammatik, Ableitung, erzeugte Sprache

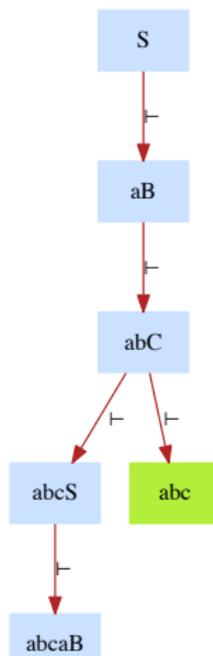
Beispiel: Sei $G'_2 = (\Sigma_2, \Gamma_2, S'_2, R'_2)$ die folgende Grammatik:

- das terminale Alphabet ist $\Sigma_2 = \{a, b, c\}$,
- das nichtterminale Alphabet ist $\Gamma_2 = \{S'_2, B, C\}$ und
- die Menge der Regeln ist gegeben durch

$$R'_2 = \left\{ \begin{array}{l} S'_2 \rightarrow aB, \\ B \rightarrow bC, \\ C \rightarrow cS'_2 | c \end{array} \right\}.$$

Grammatik, Ableitung, erzeugte Sprache

$$R'_2 = \{S'_2 \rightarrow aB, B \rightarrow bC, C \rightarrow cS'_2, C \rightarrow c\}$$



Grammatik, Ableitung, erzeugte Sprache

$$R'_2 = \{S_2 \rightarrow aB, B \rightarrow bC, C \rightarrow cS'_2, C \rightarrow c\}$$



Offenbar erzeugt G'_2 die Sprache $L(G'_2) = \{(abc)^n \mid n \geq 1\}$.

Grammatik, Ableitung, erzeugte Sprache

Beispiel: Sei $G_2 = (\Sigma_2, \Gamma_2, S_2, R_2)$ die folgende Grammatik:

- das terminale Alphabet ist $\Sigma_2 = \{a, b, c\}$,
- das nichtterminale Alphabet ist $\Gamma_2 = \{S_2, B, C\}$ und
- die Menge der Regeln ist gegeben durch

$$R_2 = \{ S_2 \rightarrow aS_2BC \mid aBC, \\ CB \rightarrow BC, \\ aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc \}.$$

Grammatik, Ableitung, erzeugte Sprache

Beispiel: Eine Ableitung für G_2 :

$$\begin{array}{lll}
 S_2 \vdash_{G_2} aS_2BC & \vdash_{G_2} aaS_2BCBC & \vdash_{G_2} aaaBCBCBC \\
 \vdash_{G_2} aaaBBCCBC & \vdash_{G_2} aaaBBCBCC & \vdash_{G_2} aaaBBBCCC \\
 \vdash_{G_2} aaabBBCCC & \vdash_{G_2} aaabbBCCC & \vdash_{G_2} aaabbbCCC \\
 \vdash_{G_2} aaabbbcCC & \vdash_{G_2} aaabbbccC & \vdash_{G_2} aaabbbccc
 \end{array}$$

$\Rightarrow aaabbbccc \in L(G_2)$.

Die von G_2 erzeugte Sprache ist

$$L(G_2) = \{a^n b^n c^n \mid n \geq 1\}.$$

Grammatik, Ableitung, erzeugte Sprache

Beispiel: Sei $G_3 = (\Sigma_3, \Gamma_3, S_3, R_3)$ die folgende Grammatik:

- das terminale Alphabet ist $\Sigma_3 = \{*, +, (,), a\}$,
- das nichtterminale Alphabet ist $\Gamma_3 = \{S_3\}$ und
- die Menge der Regeln ist gegeben durch

$$R_3 = \{S_3 \rightarrow S_3 + S_3 \mid S_3 * S_3 \mid (S_3) \mid a\}.$$

Grammatik, Ableitung, erzeugte Sprache

Beispiel: Eine Ableitung für G_3 :

$$\begin{array}{ll}
 S_3 \vdash_{G_3} S_3 + S_3 & \vdash_{G_3} S_3 * S_3 + S_3 \\
 \vdash_{G_3} (S_3) * S_3 + S_3 & \vdash_{G_3} (S_3 + S_3) * S_3 + S_3 \\
 \vdash_{G_3} \dots & \vdash_{G_3} (a + a) * a + a
 \end{array}$$

$\Rightarrow (a + a) * a + a \in L(G_3)$.

G_3 erzeugt die Sprache aller verschachtelten Klammersausdrücke mit den Operationen $+$ und $*$ und einem Zeichen a .

Grammatik, Ableitung, erzeugte Sprache

Bemerkung:

- Der Prozess des Ableitens von Wörtern ist inhärent nichtdeterministisch, denn im Allgemeinen kann mehr als eine Regel in einem Ableitungsschritt angewandt werden.
- Verschiedene Grammatiken können dieselbe Sprache erzeugen.
- Tatsächlich hat jede durch eine Grammatik definierbare Sprache unendlich viele sie erzeugende Grammatiken.
- Das heißt, eine Grammatik ist ein syntaktisches Objekt, das ein semantisches Objekt erzeugt, nämlich ihre Sprache.

Grammatik, Ableitung, erzeugte Sprache

Bemerkung:

- Es gibt viele unterschiedliche Notationen für Grammatiken.
- Aufgabe: Schauen wir uns die Handbücher der Programmiersprachen noch einmal an. Was sind die Terminale, Nichtterminale, Regeln? Gibt es Unterschiede?
- Hier eine Grammatik für eine ganz kleine Untermenge der deutschen Sprache:
 - $\Sigma = \{a, b, c, \dots, A, B, C, \dots, -\}$, $N = \{S, P, V, O\}$.
 - $S \rightarrow P \ V \ O$
 - $P \rightarrow Er \ | \ Sie \ | \ Es$
 - $V \rightarrow angelt \ | \ baut \ | \ isst$
 - $O \rightarrow Fische \ | \ Schiffe \ | \ Himbeeren$

Grammatik, Ableitung, erzeugte Sprache

- Franz Kafka. Auf der Galerie. Satz 1 von 2:

Wenn irgendeine hinfällige, lungensüchtige Kunstreiterin in der Manege auf schwankendem Pferd vor einem unermüdlichen Publikum vom peitschenschwingenden erbarmungslosen Chef monatelang ohne Unterbrechung im Kreise rundum getrieben würde, auf dem Pferde schwirrend, Küsse werfend, in der Taille sich wiegend, und wenn dieses Spiel unter dem nichtaussetzenden Brausen des Orchesters und der Ventilatoren in die immerfort weiter sich öffnende graue Zukunft sich fortsetzte, begleitet vom vergehenden und neu anschwellenden Beifallsklatschen der Hände, die eigentlich Dampfhämmer sind ? vielleicht eilte dann ein junger Galeriebesucher die lange Treppe durch alle Ränge hinab, stürzte in die Manege, rief das: Halt! durch die Fanfaren des immer sich anpassenden Orchesters.

Grammatik, Ableitung, erzeugte Sprache

- Marcel Proust. À l'ombre des jeunes filles en fleurs. Erster Satz:
Ma mère, quand il fut question d'avoir pour la première fois M. de Norpois à dîner, ayant exprimé le regret que le Professeur Cottard fût en voyage et qu'elle-même eût entièrement cessé de fréquenter Swann, car l'un et l'autre eussent sans doute intéressé l'ancien Ambassadeur, mon père répondit qu'un convive éminent, un savant illustre, comme Cottard, ne pouvait jamais mal faire dans un dîner, mais que Swann, avec son ostentation, avec sa manière de crier sur les toits ses moindres relations, était un vulgaire esbrouffeur que le Marquis de Norpois eût sans doute trouvé selon son expression, «puant».
- Die Struktur und benötigte Grammatik bei Programmiersprachen ist zum Glück einfacher.

Chomsky-Hierarchie

Definition

$G = (\Sigma, N, S, P)$ sei eine Grammatik.

- G ist *Typ-0-Grammatik*, falls P keinerlei Einschränkungen unterliegt.
- G ist *Typ-1-Grammatik* (bzw. *kontextsensitiv* bzw. *nichtverkürzend* oder *monoton*), falls für alle Regeln $p \rightarrow q$ in P gilt: $|p| \leq |q|$.
- Eine Typ-1-Grammatik G ist *vom Typ 2* (bzw. *kontextfrei*), falls für alle Regeln $p \rightarrow q$ in P gilt: $p \in N$.
- Eine Typ-2-Grammatik G ist *vom Typ 3* (bzw. *regulär* bzw. *rechtslinear*), falls für alle Regeln $p \rightarrow q$ in P gilt: $p \in N$ und $q \in \Sigma \cup \Sigma N$.

Chomsky-Hierarchie

Definition

- Eine Sprache $A \subseteq \Sigma^*$ ist genau dann vom Typ $i \in \{0, 1, 2, 3\}$, wenn es eine Typ- i -Grammatik G gibt mit $L(G) = A$.
- Die *Chomsky-Hierarchie* besteht aus den vier Sprachklassen:

$$\mathcal{L}_i = \{L(G) \mid G \text{ ist Typ-}i\text{-Grammatik}\},$$

wobei $i \in \{0, 1, 2, 3\}$. Übliche Bezeichnungen:

- \mathcal{L}_0 ist die Klasse aller Sprachen, die durch eine Grammatik erzeugt werden können;
- $\mathcal{L}_1 = \text{CS}$ ist die *Klasse der kontextsensitiven Sprachen*;
- $\mathcal{L}_2 = \text{CF}$ ist die *Klasse der kontextfreien Sprachen*;
- $\mathcal{L}_3 = \text{REG}$ ist die *Klasse der regulären Sprachen*.

Chomsky-Hierarchie

Fakt

$$\text{REG} \subseteq \text{CF} \subseteq \text{CS} \subseteq \mathcal{L}_0.$$

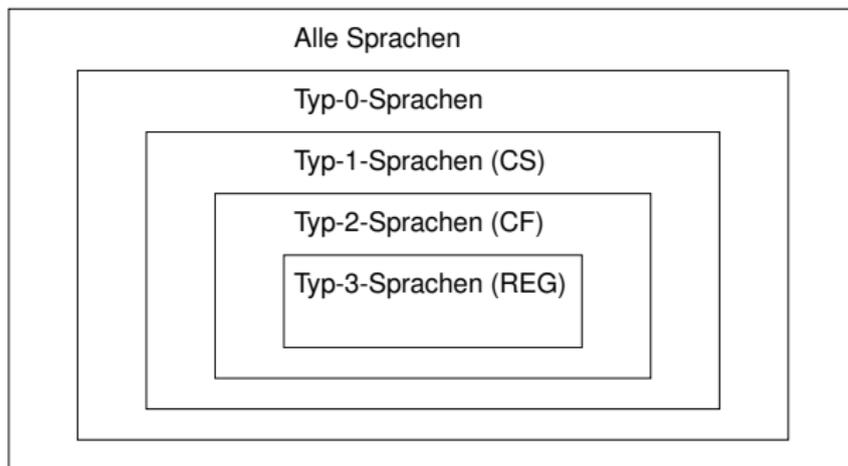


Abbildung: Die Chomsky-Hierarchie

Chomsky-Hierarchie

Beispiel:

- Offensichtlich sind die Grammatiken G_2 und G_3 aus dem vorigen Beispiel kontextsensitiv.
- Die Grammatik G_1 enthält die Regel

$$S_1 \rightarrow \lambda$$

und ist wegen $|S_1| = 1 > 0 = |\lambda|$ nach Definition nicht kontextsensitiv (vgl. „Sonderregelung für das leere Wort“). Da G_1 eine Grammatik ist, ist G_1 eine Typ-0-Grammatik.

- Offensichtlich ist die Grammatik G_3 aus dem vorigen Beispiel sogar kontextfrei.

Sonderregelung für das leere Wort

Um das leere Wort in Sprachen von Typ- i -Grammatiken für $i \in \{1, 2, 3\}$ aufzunehmen, treffen wir die folgende Vereinbarung:

(a) Die Regel

$$S \rightarrow \lambda$$

ist als einzige verkürzende Regel für Grammatiken vom Typ 1, 2, 3 zugelassen.

(b) Tritt die Regel $S \rightarrow \lambda$ auf, so darf S auf keiner rechten Seite einer Regel vorkommen.

Sonderregelung für das leere Wort

Dies ist keine Beschränkung der Allgemeinheit, denn:

Gibt es in P Regeln mit S auf der rechten Seite, so verändern wir die Regelmenge P zur neuen Regelmenge P' wie folgt:

- 1 In allen Regeln der Form $S \rightarrow u$ aus P mit $u \in (N \cup \Sigma)^*$ wird jedes Vorkommen von S in u durch ein neues Nichtterminal S' ersetzt.
- 2 Zusätzlich enthält P' alle Regeln aus P , mit S ersetzt durch S' .
- 3 Die Regel $S \rightarrow \lambda$ wird hinzugefügt.

Die so modifizierte Grammatik $G' = (\Sigma, N \cup \{S'\}, S, P')$ ist (bis auf $S \rightarrow \lambda$) vom selben Typ wie G und erfüllt $L(G') = L(G) \cup \{\lambda\}$.

Sonderregelung für das leere Wort

Beispiel: Wir betrachten die Grammatik $G = (\Sigma, N, S, P)$ mit

- das terminale Alphabet ist $\Sigma = \{a, b\}$,
- das nichtterminale Alphabet ist $N = \{S\}$ und
- die Menge der Regeln ist gegeben durch

$$P = \{S \rightarrow aSb \mid ab\}.$$

Man sieht leicht, dass G kontextfrei ist und die Sprache $L = \{a^n b^n \mid n \geq 1\}$ erzeugt, d.h., $L(G) = L$.

Wir modifizieren die Grammatik G nun gemäß der Sonderregelung für das leere Wort, um eine kontextfreie Grammatik für die Sprache

$$\{a^n b^n \mid n \geq 0\} = L \cup \{\lambda\}$$

zu gewinnen.

Sonderregelung für das leere Wort

Beispiel: Nach obiger Konstruktion erhalten wir eine kontextfreie Grammatik $G' = (\Sigma, \{S, S'\}, S, P')$ mit

$$P' = \left\{ \begin{array}{ll} S \rightarrow aS'b \mid ab & \text{gemäß 1.)} \\ S' \rightarrow aS'b \mid ab & \text{gemäß 2.)} \\ S \rightarrow \lambda & \text{gemäß 3.)} \end{array} \right\}$$

und $L(G') = L(G) \cup \{\lambda\}$.

Syntaxbaum

Um Ableitungen von Worten in Grammatiken vom Typ 2 oder 3 anschaulich darzustellen, verwenden wir einen wie folgt definierten **Syntaxbaum**.

- Sei $G = (\Sigma, N, S, P)$ eine Grammatik vom Typ 2 oder 3, und sei $S = w_0 \vdash_G w_1 \vdash_G \cdots \vdash_G w_n = w$ eine Ableitung von $w \in L(G)$.
- Die **Wurzel** (d.h. die **Etage 0**) des Syntaxbaums ist das Startsymbol S .
- **Etage i** : Wird wegen der Regel $A \rightarrow z$ im i -ten Ableitungsschritt ($w_{i-1} \vdash_G w_i$) das Nichtterminal A in w_{i-1} durch das Teilwort z von w_i ersetzt, so hat der Knoten A im Syntaxbaum $|z|$ Söhne, die v.l.n.r. mit den Symbolen aus z beschriftet sind.
- Die **Blätter** des Baumes ergeben von links nach rechts gelesen w .

Syntaxbaum

Beispiel:

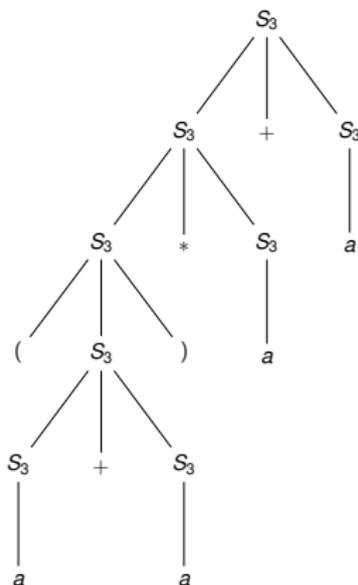


Abbildung: Syntaxbaum für die Ableitung des Wortes $(a + a) * a + a$ in der Typ 2-Grammatik G_3 aus dem vorigen Beispiel

Mehrdeutige Grammatik

Definition

- Eine Grammatik G heißt *mehrdeutig*, falls es ein Wort $w \in L(G)$ gibt, das zwei verschiedene Syntaxbäume hat.
- Eine Sprache A heißt *inhärent mehrdeutig*, falls jede Grammatik G mit $A = L(G)$ mehrdeutig ist.

Mehrdeutige Grammatik

Beispiel:

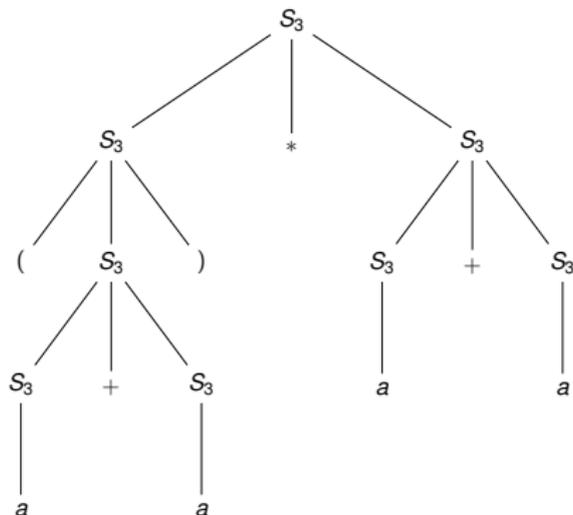


Abbildung: Ein weiterer Syntaxbaum für die Ableitung des Wortes $(a + a) * a + a$ in der Typ 2-Grammatik G_3 aus dem vorigen Beispiel