Lektion 8

Zunächst gibt es hier noch einen kurzen Ausblick auf verschiedene Anwendungen, die reguläre Ausdrücke verwenden, um auch deren praktische Bedeutung zu verdeutlichen.

Danach werden dann sog. *kontextfreie Grammatiken* eingeführt. Dies ist eine weitere formale Methode, mit der Sprachen definiert werden können. Später werden Sie sehen, dass kontextfreie Grammatiken mächtiger als reguläre Ausdrücke sind, d.h. es können auch manche Sprachen damit spezifiziert werden, die nicht durch reguläre Ausdrücke darstellbar sind, z.B. typische Sprachkonstrukte von Programmiersprachen.

5.5 Ausblick: Anwendung regulärer Ausdrücke

Reguläre Ausdrücke als Suchmuster in Texten

- Für alle modernen Programmiersprachen gibt es Bibliotheken, die das Arbeiten mit regulären Ausdrücken ermöglichen.

  Java: Bietet Klassen `Pattern` und `Matcher` in Paket `java.util.regex` sowie verschiedene Methoden mit regulären Ausdrücken als Musterangabe in Klasse `String`.

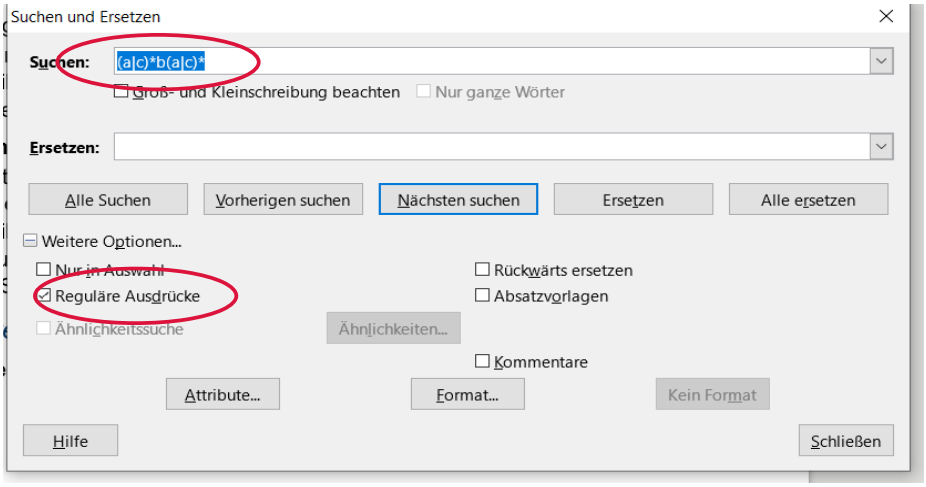
  Klasse `String`:

`boolean`	**`matches(`*`regex`*`)`** prüft, ob der String entsprechend dem regulären Ausdruck *regex* aufgebaut ist.
`String`	**`replaceAll(String `*`regex`*`, String `*`replacement`*`)`** ersetzt alle Vorkommen des Musters *regex* durch *replacement*
`String`	**`replaceFirst(String `*`regex`*`, String `*`replacement`*`)`** ersetzt das erste Vorkommen des Musters *regex* durch *replacement*
`String[]`	**`split`**`(String regex, `**`int limit`**`)` Splits this string around matches of the given regular expression

  C: GNU regex library

  C++: Mit Version C++11 wurde Unterstützung für reguläre Ausdrücke eingeführt.

- Die **Suche nach Zeichenketten** ist in Textverarbeitungssystemen wie LibreOffice, OpenOffice oder Word auch mit regulären Ausdrücken möglich.



Auch manche Texteditoren, z.B. Notepad++, erlauben es, beim Suchen und Ersetzen reguläre Ausdrücke zu verwenden.

- **fgrep**-Kommando in Linux: Mit diesem Kommando können alle Zeilen einer Textdatei herausgefiltert werden, die eine Zeichenkette enthalten, deren Aufbau durch einen regulären Ausdruck spezifiziert werden kann.
- **Skriptsprachen** (awk, perl, PHP, …). Diese Sprachen enthalten viele Befehle, um auf der Basis von regulären Ausdrücken Zeichenketten bequem verarbeiten zu können.
- **Scanner-Genratoren im Compilerbau**: Der Aufbau von Token (z.B. Zahlen, Namen von Variablen, Methoden und Klassen, ...) kann mit regulären Ausdrücken beschreiben werden. Sog. Scanner-Generatoren (z.B. lex, Antlr) generieren daraus dann Code für die Zerlegung des Programmquelltexts in einzelne Token.

Erweiterte reguläre Ausdrücke

Werkzeuge und Bibliotheken bieten meist viele Erweiterungen, um sehr bequem mit regulären Ausdrücken arbeiten zu können. Hier sind einige typische Beispiele dafür angegeben [nicht klausurrelevant, kann aber für das Informatikerleben hilfreich sein].

- **Muster für genau ein Zeichen (Zeichenklassen)**

[egh]	eines der Zeichen 'e', 'g' oder 'h'
[0-6]	eine Ziffer im Bereich von '0' bis '6'
[A-Za-z0-9]	ein Buchstabe von A bis Z (groß oder klein) oder eine Ziffer
[^a]	ein beliebiges Zeichen *außer* Zeichen *a*, d.h. „^“ am Anfang einer Zeichenklasse negiert selbige.
.	Punkt steht für ein beliebiges Zeichen. (Tipp: Will man in einem Muster nur den Punkt als Zeichen angeben, kann man das mit **[.]** bewerkstelligen)

- **Quantoren für reguläre Ausdrücke:**

R*	R darf beliebig oft (auch keinmal) vorkommen
R+	R muss mindestens einmal vorkommen
R?	R ist optional (null- oder einmal)
R{*n*}	R muss exakt *n*-mal vorkommen.
R{*n*,*m*}	R muss mindestens *n*-mal und darf maximal *m*-mal vorkommen.

Kap. 6 Kontextfreie Grammatiken

Inhalt

- Kontextfreie Grammatiken
- Ableitungen und Ableitungsbäume
- Mehrdeutigkeit von Grammatiken
- Äquivalenz von Grammatiken
- Alternative Formen der Syntaxbeschreibung
  - Backus-Naur-Form (BNF)
  - Erweiterte Backus-Naur-Form (EBNF)
  - Syntaxdiagramme
- Allgemeinere Formen von Grammatiken: Chomsky-Grammatiken
- Exkurs: Strukturdefinitionen für XML-Bäume mittels DTD

6.1 Motivation

Im vorigen Kapitel haben Sie gelernt, wie man mit Hilfe regulärer Ausdrücke Sprachen bzw. Textmuster definieren kann. Wie an den Beispielen zu erkennen war, sind sie eine sehr elegante und übersichtliche Methode (wenn man etwas Erfahrung damit hat) dafür und werden deswegen an vielen Stellen eingesetzt. Allerdings haben reguläre Ausdrücke auch Grenzen, was damit beschrieben werden kann.

Aufgabe 6.1 - Klammerfolgen

In der Informatik hat man es an vielen Stellen mit geklammerten Strukturen zu tun (denken Sie an Programmiersprachen oder XML). Wir betrachten hier nur einen sehr einfachen Fall von Klammerung:

Versuchen Sie, durch einen regulären Ausdruck alle Folgen aus öffnenden und schließenden Klammern zu definieren, die aus einer beliebigen Anzahl öffnender Klammern am Anfang und der gleichen Zahl schließender Klammern danach bestehen, d.h.

$$L = \{ (^n)^n \mid n \geq 0 \}$$

also z.B. ε, `()`, `(())`, `((()))`.

Beispiel 6.2 - Palindrome

Palindrome sind Wörter, die von vorne nach hinten gelesen gleich sind wie von hinten nach vorne gelesen.

Beispiele für Palindrome:

anna

gnudung

reliefpfeiler

die liebe ist sieger, stets rege ist sie bei leid
(von Leer- und Satzzeichen abgesehen)

00100

Fragestellung: Wie lässt sich die Sprache aller Palindrome über einem gegebenen Alphabet Σ formal exakt definieren? Palindrome lassen sich nicht durch einen regulären Ausdruck beschreiben.

- **Ansatz 1 (analytisch)**: Man definiert eine Eigenschaft *istPalindrom*(w) und formuliert dann:

  $L_{Palindrom} = \{ w \in \Sigma^* \mid istPalindrom(w) \}$

  wobei

  $istPalindrom(a_1 a_2 ... a_n)$ genau dann, wenn $a_i = a_{n+1-i}$ für alle i = 1, …, n.

- **Ansatz 2 (konstruktiv)**: Man gibt Regeln an, wie sich alle Palindrome bilden lassen. Die Regeln für alle Palindrome zum Alphabet $\Sigma = \{0,1\}$ könnten z.B. so aussehen (die Notation wird später genauer erläutert):

$P \rightarrow \varepsilon$	ε kann genommen werden, um ein Palindrom zu bilden
$P \rightarrow 0$	0 kann genommen werden, um ein Palindrom zu bilden
$P \rightarrow 1$	1 kann genommen werden, um ein Palindrom zu bilden
$P \rightarrow 0P_a0$	hat man schon ein Palindrom P_a gebildet, dann ist auch $0P_a0$, d.h. mit 0 davor und danach, ein Palindrom
$P \rightarrow 1P_a1$	hat man schon ein Palindrom P_a gebildet, dann ist auch $1P_a1$ ein Palindrom

  Lassen sich nach diesen Regeln z.B. die Palindrome 10101 und 0110 bilden? Ja:

  - 1 ist Palindrom, also ist auch 010 ein Palindrom und somit 10101
  - ε ist Palindrom, also ist auch $1\varepsilon1 = 11$ ein Palindrom und somit 0110

  Wie unschwer zu erkennen ist, ist alles, was nach diesen Regeln gebildet werden kann, ein Palindrom und jedes Palindrom lässt sich damit bilden.

Beispiel 6.3 - Arithmetische Ausdrücke

Der syntaktische Aufbau arithmetischer Ausdrücke mit Zahlen, Variablen und arithmetischen Operatoren (+, -, *, …), wie z.B.

```
2 * (x + 7 - y)
```

soll definiert werden. Ein analytischer Ansatz wie im vorigen Beispiel wäre hier schwierig. Ein konstruktiver Ansatz mit Bildungsregeln ist dagegen relativ einfach:

- Regeln für den Aufbau arithmetischer Ausdrücke über Alphabet
  Σ = {*Zahl*, *Variable*, `+`, `-`, `*`, `/`, `(`, `)`} . Zahlen und Variablen werden hier als ein elementare Symbole betrachtet (als ein "Token").

  Ausdruck → *Zahl* (1)
  Ausdruck → *Variable* (2)
  Ausdruck → *Ausdruck Operator Ausdruck* (3)
  Ausdruck → `(` *Ausdruck* `)` (4)

  Operator → `+` (5)
  Operator → `-` (6)
  Operator → `*` (7)
  …

- Mit diesen Regeln kann dann beispielsweise der Ausdruck `2 * (x + 7 - y)` folgendermaßen gebildet werden:

  `x` ist eine Variable und somit eine Ausdruck (2)
  `7` ist eine Zahl und somit ein Ausdruck (1)
  `+` ist ein Operator (5)
  somit kann Ausdruck `x+7` gebildet werden (3)
  `-` ist Operator (6)
  `y` ist eine Variable und somit ein Ausdruck (2)
  somit kann `x+7-y` als Ausdruck gebildet werden (3)
  somit kann auch `(x+7-y)` als Ausdruck gebildet werden (4)
  `2` ist Variable, bildet somit einen Ausdruck (2)
  `*` ist Operator (7)
  somit kann auch `2 * (x+7-y)` als Ausdruck gebildet werden (3)

Entsprechend können mit diesen recht einfachen Regeln beliebige andere Ausdrücke aus Variablen, Zahlen, Operatoren und Klammern gebildet werden.

Beispiel 6.4 - "Natürliche" Sprache

Wir definieren dazu als Beispiel eine sehr einfache Sprache *Deutsch*$^{--}$ in folgender Weise. Es sollen Sätze wie

"Die kleine Katze schläft."

"Das Klavier spielt."

korrekt sein, aber

"Katze . schläft Die kleine"

unzulässig.

Unsere Sprache *Deutsch*$^{--}$ wird durch folgende Regeln definiert:

Satz	→	*Nominalsatz Verb .*
Nominalsatz	→	*Artikel Substantiv*
Nominalsatz	→	*Artikel Adjektiv Substantiv*
Artikel	→	Der
Artikel	→	Die
Artikel	→	Das
Substantiv	→	Hund
Substantiv	→	Katze
Substantiv	→	Klavier
Adjektiv	→	große
Adjektiv	→	kleine
Adjektiv	→	nette
Verb	→	schläft
Verb	→	spielt
Verb	→	bellt

Damit kann dann eine ganze Reihe von Sätzen gebildet werden, z.B. "Der Hund bellt.", oder auch weniger sinnvolle, wie "Das nette Klavier schläft."

Aufgabe 6.5 - Deutsch^{--}

- Geben Sie noch drei weitere Sätze an, die in Deutsch^{--} nach obigen Regeln gebildet werden können.

Anmerkung

Das Beispiel *Deutsch*$^{--}$ ist zwar selbst wenig realistisch, aber lässt zumindest erahnen, dass mit Hilfe solcher Regeln auch die Grammatik natürlicher Sprachen beschrieben werden kann (mit gewissen Einschränkungen, da natürliche Sprachen äußerst komplex sind).

Der Bezug zu natürlichen Sprachen wird hier hergestellt, da die Methode der Beschreibung von Sprachen mit Hilfe von Erzeugungsregeln in den 1950er Jahren im Bereich der Linguistik, d.h. der Sprachwissenschaften für natürliche Sprachen, entwickelt wurde. Diese Methode, genannt *kontextfreie Grammatiken*, wurde dann sehr schnell übernommen, um auch die Syntax von Programmiersprachen zu

definieren.

Die erste Programmiersprache, für die die Syntax formal in dieser Weise beschrieben wurde, war Ende der 1950er-Jahre die Programmiersprache Algol 60. Damit konnte erstmals eine Programmiersprache unabhängig von einer Implementierung definiert werden. Die formale Beschreibung mit kontextfreien Grammatiken war dann wiederum die Basis dafür, effiziente Syntaxanalyseverfahren entwickeln zu können und Programmiersprachen systematisch implementieren zu können.

6.2 Kontextfreie Grammatiken

Definition 6.6 - Kontextfreie Grammatik

Eine **kontextfreie Grammatik** (engl. *context-free grammar*)
$G = (N, T, P, S)$
besteht aus:

N	endliche Menge von sog. **nichtterminalen Symbolen**,	
T	endliche Menge von sog. **terminalen Symbolen** (mit $N \cap T = \varnothing$)	
P	endliche Menge von **Produktionen** (Ersetzungsregeln)	
	$L \rightarrow r$,	
	wobei	
	$L \in N$	*linke Seite*, nichtterminales Symbol
	$r \in (N \cup T)^*$	*rechte Seite*, evtl. leere Folge aus terminalen und nichtterminalen Symbolen
S	***Startsymbol*** $S \in N$	eines der nichtterminalen Symbole

Anmerkungen

- Die nichtterminalen Symbole (oft auch nur kurz *Nichtterminale* genannt) sind die Symbole, die zur als grammatikalische Hilfsbegriffe dienen, um die Sprache zu beschreiben (z.B. *Satz*, *Nominalsatz* oder *Substantiv* im Beispiel oben).
- Die terminalen Symbole (auch kurz *Terminale* genannt) sind das Alphabet der Sprache, die definiert werden soll, also das, was am Ende damit damit angegeben wird (deswegen "terminal").
- $N \cap T = \varnothing$ bedeutet, dass terminal und nichtterminale Symbol voneinander unterschieden werden können, d.h. es kann kein Symbol geben, das sowohl terminales als auch nichtterminales Symbol ist.
- Das Startsymbol ist eines der nichtterminalen Symbole, das festlegt, was mit der Grammatik beschrieben wird (z.B. *Satz* im Beispiel oben).

- Die Produktionen dienen als Ersetzungsregeln, mit denen etwas erzeugt ("produziert") werden kann. Die linke Seite, immer ein Nichtterminal, kann durch die rechte Seite ersetzt werden. Wie sie angewendet werden, wird nachfolgend genauer erklärt.
- Die rechte Seite einer Produktion kann auch die leere Folge ε sein. Eine solche Produktion wird dann als ε-Produktion bezeichnet.

Beispiel 6.7 - Kontextfreie Grammatik

Grammatik G = (N, T, P, S) mit

- Nichtterminale Symbole $N = \{A, D, S\}$
- Terminale Symbole $T = \{a, b, c, d\}$
- Produktionen = {

  $S \rightarrow AD,$
  $A \rightarrow aAc,$
  $A \rightarrow b,$
  $D \rightarrow dD,$
  $D \rightarrow \varepsilon$ }
- Startsymbol: S

Notationskonventionen 6.8 - kontextfreie Grammatiken

- Produktionen mit dem gleichen Nichtterminal auf der linken Seite werden meist zusammengefasst. Alternative rechte Seiten werden durch | getrennt. D.h. statt

  $L \rightarrow r_1$
  $L \rightarrow r_2$
  ...
  $L \rightarrow r_k$

  wird kürzer so notiert:

  $L \rightarrow r_1 \mid r_2 \mid \ldots \mid r_k$

  bzw. mit übersichtlicherem Layout:

  $L \rightarrow r_1$
  $\mid r_2$
  ...
  $\mid r_k$
- In den abstrakten Beispielen werden für *terminale* Symbole üblicherweise *Kleinbuchstaben* verwendet und für *nichtterminale* Symbole *Großbuchstaben*. Üblicherweise wird für das Startsymbol S verwendet, sofern nicht anders angegeben oder aus dem Kontext erkennbar.
- Es wird meist nur die Menge der Produktionen angegeben (und ggf. was

Startsymbol ist), da daraus erkennbar ist, was terminale und was nichtterminale Symbole sind.

Beispiel 6.9 - Kontextfreie Grammatik für Palindrome

- Die kontextfreie Grammatik für Palindrome über {0, 1} hat nur ein einziges Nichtterminal *P*, das auch das Startsymbol ist.

$$
\begin{aligned}
P \rightarrow\ & \varepsilon \\
\mid\ & 0 \\
\mid\ & 1 \\
\mid\ & 0P0 \\
\mid\ & 1P1
\end{aligned}
$$

6.2.1 Ableitungsschritte und Ableitbarkeit

Die Anwendung einer Produktion auf eine Folge von Symbolen nennt man einen direkten Ableitungsschritt. Wendet man eine Produktion L → *r* auf eine Folge von (terminalen oder nichtterminalen) Symbolen an, wird an einer Stelle ein Vorkommen des Nichtterminals *L* der linken Seite durch die rechte Seite *r* der Produktion ersetzt.

Definition 6.10 - Direkte Ableitungsrelation

Gegeben sei eine kontextfreie Grammatik $G = (N, T, P, S)$.

Die **direkte Ableitungsrelation** $\Rightarrow_G$ auf der Menge $(N \cup T)^*$ der Wörter aus terminalen und nichtterminalen Symbolen ist definiert durch:

$$u \Rightarrow_G v$$

genau dann, wenn es Wörter x, *y* und eine Produktion $L \rightarrow r \in P$ gibt mit

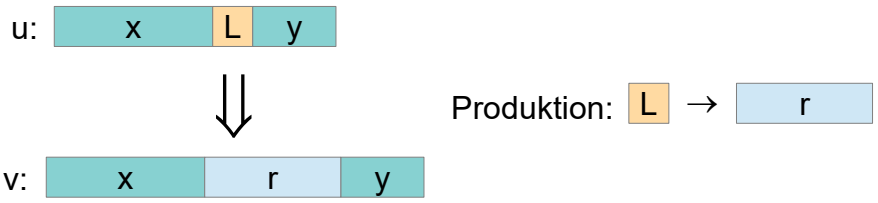
$u = xLy$

$v = xry.$

Wir sagen dazu: "*u* kann direkt nach *v* abgeleitet werden".

Anmerkungen

- D.h. das Ausgangswort *u* enthält *L*, der Teil *x* vor *L* bleibt stehen und der Teil *y* nach *L* bleibt auch unverändert, nur *L* wird durch die rechte Seite *r* der Produktion ersetzt.



- Was vor und nach L steht, hat keinen Einfluss auf die Ersetzung, daher die Bezeichnung "kontextfrei".
- Die Teile *x* und *y* können auch das leere Wort sein, d.h. das zu ersetzende Symbol *L* kann auch ganz am Anfang oder ganz am Ende stehen.
- Meist wird nur $u \Rightarrow v$ statt $u \Rightarrow_G v$ geschrieben, wenn klar ist, welche Grammatik gemeint ist.

Beispiel 6.11 - Direkte Ableitungsschritte

- Gegeben sei die Grammatik aus *Beispiel 6.7 - Kontextfreie Grammatik*

$$S \rightarrow AD$$
$$A \rightarrow aAc \mid b$$
$$D \rightarrow dD \mid \varepsilon$$

Dann sind folgendes direkte Ableitungsschritte. Die Anwendungsstellen der Produktionen (d.h. welches nichtterminale Symbol ersetzt wird) sind hier durch Unterstreichung gekennzeichnet.

a<u>A</u>cdD ⇒ a**aAc**cdC	(mit Produktion $A \rightarrow aAc$)
a<u>A</u>cdD ⇒ a**b**cdD	(mit Produktion $A \rightarrow b$)
aAcd<u>D</u> ⇒ aAcd**dD**	(mit Produktion $D \rightarrow dD$)
aAcd<u>D</u> ⇒ aAcd	(mit Produktion $D \rightarrow \varepsilon$)

Anmerkung

- Da es zu einer Zeichenfolge meist viele Möglichkeiten für einen Ableitungsschritt gibt (z.B. wie im vorigen Beispiel), handelt es sich bei ⇒ um eine *Relation* zwischen Symbolfolgen.

Mehrere solche Ableitungsschritte kann man nacheinander ausführen, das nennt man dann eine *Ableitung*.

Definition 6.12 - Ableitbarkeit

w ist aus *u* **ableitbar**, notiert

$$\boldsymbol{u \Rightarrow_G^* w}$$

genau dann, wenn entweder

$u = w$ (0 direkte Ableitungsschritte)

oder wenn es eine Folge direkter Ableitungsschritte

$u \Rightarrow_G v_1 \Rightarrow_G v_2 \Rightarrow_G \ldots \Rightarrow_G v_{n-1} \Rightarrow_G w$ (*n* direkte Ableitungsschritt)

von *u* nach *w* gibt.

Der Stern bei ⇒* wird im Sinn von "beliebig viele" (Schritte) verwendet. In der Mathematik wird dies als *reflexiver und transitiver Abschluss* der direkten Ableitungsrelation ⇒ bezeichnet.

Beispiel 6.13 - Ableitbarkeit

Gegeben ist die Grammatik aus Beispiel 6.7:

$S \rightarrow AD$
$A \rightarrow aAc \mid b$
$D \rightarrow dD \mid \varepsilon$

Dann gilt:

(1) S $\Rightarrow^*$ a*A*cd*D*

(2) S $\Rightarrow^*$ aabccdd

eine Ableitung zu (1):

S	$\Rightarrow$ AD	(mit Produktion S$\rightarrow$ AD)
	$\Rightarrow$ aAcD	(mit Produktion A$\rightarrow$ aAc)
	$\Rightarrow$ aAcdD	(mit Produktion D$\rightarrow$ dD)

eine Ableitung zu (2):

S $\Rightarrow$ AD $\Rightarrow$ aAcD $\Rightarrow$ aAcdD $\Rightarrow$ aAcddD $\Rightarrow$ aaAccddD
$\Rightarrow$ aabccddD $\Rightarrow$ aabccdd

Aufgabe 6.14 - Ableitbarkeit mit Grammatik für Palindrome

Gegeben ist die Grammatik für Palindrome aus Beispiel 6.9:

$P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$

Zeigen Sie, dass

0PP0 $\Rightarrow^*$ 001110111110

gilt.

6.2.2 Sprache einer Grammatik

Bei der vorigen Aufgabe 6.14 ist Ihnen vielleicht aufgefallen, dass mit den Produktionen am Ende das Wort 001110111110 produziert wurde, was offensichtlich kein Palindrom ist. Stimmt da irgend etwas nicht, ist die Ableitung oder die Grammatik falsch?

Nein - denn wir müssen erst genau definieren, was die Sprache der Grammatik ist: Alle Folgen von terminalen Symbolen, die *aus dem Startsymbol ableitbar* sind. Da aber 0PP0 und folglich auch 001110111110 nicht aus dem Startsymbol P ableitbar sind, gehört 001110111110 nicht zur Sprache der Grammatik für Palindrome.

Definition 6.15 - Sprache einer Grammatik

Die von einer Grammatik $G = (N, T, P, S)$ **erzeugte Sprache** $L(G)$ ist

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

D.h. die Sprache der Grammatik sind alle Wörter, die aus dem Startsymbol *S* durch endlich viele Ableitungsschritte ableitbar sind und die nur aus terminalen Symbolen bestehen.

Beispiel 6.16 - Sprache einer Grammatik

Welche Sprache wird durch folgende Grammatik erzeugt?

$$S \rightarrow \varepsilon \mid aSb$$

Betrachten wir erst mal ein Beispiel für eine Ableitung:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

Mit der Produktion $S \rightarrow aSb$ wird für jedes a vor dem S auch ein b nach dem S produziert. Man kann diese Produktion beliebig oft anwenden, bis einmal die ε-Produktion verwendet wird und dann eine Folge von terminalen Zeichen übrig bleibt.

Die Sprache der Grammatik ist also:

$$L(G) = \{a^n b^n \mid n \geq 0\}$$

Anmerkung

Diese Sprache $\mathbf{a^n b^n}$ ist das einfachste Beispiel für eine Sprache, die zwar, so wie wir gerade gesehen haben, mit kontextfreien Grammatiken beschrieben werden kann, die aber nicht mit regulären Ausdrücken dargestellt werden kann.

Erinnern Sie sich an das Beispiel mit Klammerfolgen (((…))) vom Kapitelanfang? Das ist das gleiche Problem, wenn man statt **a** die öffnende Klammer und statt **b** die schließende Klammer nimmt.

Beispiel 6.17 - Sprache einer Grammatik

Welche Sprache wird durch die Grammatik

$S \rightarrow AD$
$A \rightarrow aAc \mid b$
$D \rightarrow dD \mid \varepsilon$

aus Beispiel 6.7 erzeugt?

Man erkennt, dass man aus D eine beliebig lange, ggf. auch leere Folgen von d produzieren kann. Mit A kann man für jedes a am Anfang ein zugehöriges c am Ende erzeugen, solange, bis man mit einem b aufhört. Als Sprache ergibt sich somit:

$$L(G) = \{ a^n b\, c^n d^k \mid n \geq 0, k \geq 0 \}$$

Aufgabe 6.18 - Grammatik für Dyck-Sprache

Wie kann eine Grammatik definiert werden, die genau die korrekt geklammerten Folgen aus Klammern (und) erzeugt. Zu jeder öffnenden Klammer muss es

danach eine zugehörige schließend Klammer geben. Klammergruppen dürfen ineinander geschachtelt sein oder nebeneinander stehen. Das leere Wort soll auch als korrekt geklammert betrachtet werden. Korrekt geklammert sind z.B.

ε
( )
(( ))
( ) ( )
(( )) (( ) (( )))

aber nicht folgende Klammerfolgen:

( ))
) (
(( ) ( )

Diese Sprache wird nach dem Mathematiker Walther von Dyck auch als *Dyck*-Sprache bezeichnet.

Anmerkung

In der Informatik hat man es an vielen Stellen mit geklammerten/geschachtelten Strukturen zu tun, z.B. bei arithmetischen und logischen Ausdrücken, bei geklammerten Sprachkonstrukten in Programmiersprachen (z.B. Böcke { … } bei Schleifen und Fallunterscheidungen) oder auch bei der Klammerung von XML-Elementen durch Anfangs- und Endtags. Solche Konstrukte lassen sich leicht mit kontextfreien Grammatiken beschreiben, aber nicht mit regulären Ausdrücken.

6.2.3 Links- und Rechtsableitungen

Wie Sie in den vorigen Beispielen gesehen haben, gibt es bei Ableitungen meist mehrere Möglichkeiten, an welchen Stellen die nächste Produktion angewendet wird. Nimmt man immer die erste mögliche Stelle, ergibt sich eine *Linksableitung*, nimmt man immer die letzte mögliche Stelle, bekommt man eine *Rechtsableitung*.

Definition 6.19 – Links- und Rechtsableitung

- Eine Ableitung $v_0 \Rightarrow_G v_1 \Rightarrow_G v_2 \Rightarrow_G \ldots \Rightarrow_G v_k$ heißt **Linksableitung**, wenn in jedem Ableitungsschritt das *am weitesten links stehende* nichtterminale Symbol ersetzt wird.
- Eine Ableitung $v_0 \Rightarrow_G v_1 \Rightarrow_G v_2 \Rightarrow_G \ldots \Rightarrow_G v_k$ heißt **Rechtsableitung**, wenn in jedem Ableitungsschritt das *am weitesten rechts stehende* nichtterminale Symbol ersetzt wird.

Beispiel 6.20 - Links- und Rechtsableitungen

- gegeben: Grammatik für Dyck-Sprache (Klammerfolgen), s.o.

S → ε
| (S)
| SS

- Linksableitung für (( )) ( ) :

  S ⇒ SS ⇒ (S)S ⇒ ((S))S ⇒ (())S ⇒ (())(S) ⇒ (())()

Anmerkung

- Für die Fragestellung, ob ein Wort ableitbar ist, ist es egal, ob man beliebige Ableitungen betrachtet oder sich auf Links- bzw. Rechtsableitungen beschränkt. Gibt es eine Ableitung, dann gibt es auch eine Links- sowie eine Rechtsableitung, und umgekehrt gilt dies auch.
- Die Begriffe Links- bzw. Rechtsableitung spielt später wieder eine Rolle bei der Syntaxanalyse. Es gibt Verfahren zur Syntaxanalyse, die eine Linksableitung liefern und andere, die eine Rechtsableitung bestimmen.

Aufgabe 6.21 - Rechtsableitungen

- gegeben: Grammatik für Dyck-Sprache (Klammerfolgen), s.o.:

  S → ε | (S) | SS

- Geben Sie eine Rechtsableitung für (( )) ( ) an.

Fazit zu Lektion 8

Diese Fragen sollten Sie nun beantworten können

- Wo werden reguläre Ausdrücke praktisch eingesetzt?
- Was ist eine kontextfreie Grammatik?
- Was bedeutet Ableitbarkeit bei einer kontextfreien Grammatik?
- Wie ist die Sprache einer kontextfreien Grammatik definiert?
- Was ist eine Links- bzw. eine Rechtsableitung?