

6 Modellierung von Strukturen 6.1 Kontextfreie Grammatiken

Kontextfreie Grammatik (KFG): formaler Kalkül, Ersetzungssystem; definiert

- **Sprache** als Menge von Sätzen; jeder **Satz** ist eine **Folge von Symbolen**
- **Menge von Bäumen**; jeder Baum repräsentiert die **Struktur eines Satzes** der Sprache

Anwendungen:

- Programme einer **Programmiersprache** und deren Struktur, z. B. Java, Pascal, C
- Sprachen als Schnittstellen zwischen Software-Werkzeugen, **Datenaustauschformate**, z. B. HTML, XML
- Bäume zur Repräsentation **strukturierter Daten**, z. B. in HTML
- Struktur von **Protokollen** beim Austausch von Nachrichten zwischen Geräten oder Prozessen

Beispiel zu HTML:

```
<table>
  <tr>
    <td>Mo</td>
    <td>11-13</td>
    <td>AM</td>
  </tr>
  <tr>
    <td>Fr</td>
    <td>9-11</td>
    <td>AM</td>
  </tr>
</table>
```

Kontextfreie Grammatik

Eine kontextfreie Grammatik $G = (T, N, P, S)$ besteht aus:

- T** Menge der **Terminalsymbole** (kurz: Terminale)
- N** Menge der **Nichtterminalsymbole** (kurz: Nichtterminale)
T und N sind disjunkte Mengen
- S** ∈ N **Startsymbol** (auch Zielsymbol)
- P** ⊆ N × V* Menge der **Produktionen**; (A, x) ∈ P, mit A ∈ N und x ∈ V*;
statt (A, x) schreibt man A ::= x
- $V = T \cup N$ heißt auch **Vokabular**, seine Elemente heißen **Symbole**

Man sagt „In der Produktion A ::= x steht A auf der **linken Seite** und x auf der **rechten Seite**.“
Man gibt Produktionen häufig **Namen**: p1: A ::= x
In Symbolfolgen aus V* werden die Elemente nur durch Zwischenraum getrennt: A ::= B C D

Beispiel:		Produktionsmenge P =	
Terminale	T = { (,) }	Name	N V*
Nichtterminale	N = { Klammern, Liste }	p1:	Klammern ::= '(Liste)'
Startsymbol	S = Klammern	p2:	Liste ::= Klammern Liste
		p3:	Liste ::=

Bedeutung der Produktionen

Eine Produktion A ::= x ist eine **Strukturregel**: A **besteht aus** x

Beispiele:

DeutscherSatz ::=	Subjekt Prädikat Objekt
EinDeutscherSatz <i>besteht aus (der Folge)</i>	Subjekt Prädikat Objekt
Klammern ::=	'(Liste)'
Zuweisung ::=	Variable ':=' Ausdruck
Variable ::=	Variable '[' Ausdruck]'

Produktion graphisch als gewurzelter Baum

mit geordneten Kanten und mit Symbolen als Knotenmarken:



Ableitungen

Produktionen sind **Ersetzungsregeln**: Ein Nichtterminal A in einer Symbolfolge u A v kann durch die rechte Seite x einer Produktion A ::= x ersetzt werden.
Das ist ein **Ableitungsschritt**; er wird notiert als u A v => u x v

z. B. Klammern Klammern Liste => Klammern (Liste) Liste
mit Produktion p1

Beliebig viele **Ableitungsschritte nacheinander** angewandt heißen **Ableitung**;
notiert als $u \Rightarrow^* v$

Eine kontextfreie Grammatik **definiert eine Sprache**; das ist eine Menge von Sätzen.
Jeder Satz ist eine Folge von Terminalsymbolen, die aus dem Startsymbol ableitbar ist:
 $L(G) = \{ w \mid w \in T^* \text{ und } S \Rightarrow^* w \}$

Grammatik auf Mod-6.2 **definiert** geschachtelte Folgen paariger Klammern als **Sprache**:
 $\{ (), (()), (() ()), ((()) ()), \dots \} \subseteq L(G)$

Ableitung des Satzes (()): S = Klammern
=> (Liste)
=> (Klammern Liste)
=> (Klammern Klammern Liste)
=> (Klammern (Liste) Liste)
=> ((Liste) (Liste) Liste)
=> (() (Liste) Liste)
=> (() () Liste)
=> (() ())

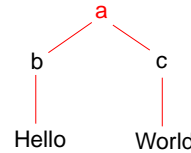
3 elementare Prinzipien

Die XML-Notation basiert auf 3 elementaren Prinzipien:

A: Vollständige Klammerung durch Tags

```
<a>
  <b>Hello</b>
  <c>World</c>
</a>
```

B: Klammerstruktur ist äquivalent zu gewurzelterm Baum



C: Kontextfreie Grammatik definiert Bäume;
eine DTD ist eine KFG

```
a ::= b c
b ::= PCDATA
c ::= PCDATA
```

Notation und erste Beispiele

Ein Satz in einer XML-Sprache ist ein Text, der durch **Tags** strukturiert wird.

Tags werden immer in **Paaren von Anfangs- und End-Tag** verwendet:

```
<ort>Paderborn</ort>
```

Anfangs-**Tags** können Attribut-Wert-Paare enthalten:

```
<telefon typ="dienst">05251606686</telefon>
```

Die **Namen von Tags und Attributen** können für die XML-Sprache **frei gewählt** werden.

Mit **Tags** gekennzeichnete Texte können geschachtelt werden.

```
<adressBuch>
<adresse>
  <name>
    <nachname>Mustermann</nachname>
    <vorname>Max</vorname>
  </name>
  <anschrift>
    <strasse>Hauptstr 42</strasse>
    <ort>Paderborn</ort>
    <plz>33098</plz>
  </anschrift>
</adresse>
</adressBuch>
```

(a+b)² in MathML:

```
<msup>
  <mfenced>
    <mrow>
      <mi>a</mi>
      <mo>+</mo>
      <mi>b</mi>
    </mrow>
  </mfenced>
  <mn>2</mn>
</msup>
```

Ein vollständiges Beispiel

Kennzeichnung des Dokumentes als XML-Datei

Datei mit der Definition der Syntaktischen Struktur dieser XML-Sprache (DTD)

Datei mit Angaben zur Transformation in HTML

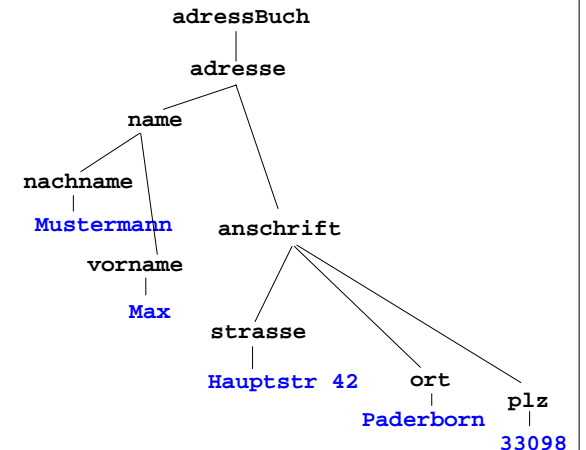
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE adressBuch SYSTEM "adressBuch.dtd">
<?xml-stylesheet type="text/xsl" href="adressBuch.xsl" ?>
<adressBuch>
<adresse>
  <name>
    <nachname>Mustermann</nachname>
    <vorname>Max</vorname>
  </name>
  <anschrift>
    <strasse>Hauptstr 42</strasse>
    <ort>Paderborn</ort>
    <plz>33098</plz>
  </anschrift>
</adresse>
</adressBuch>
```

Baumdarstellung von XML-Texten

Jeder XML-Text ist durch Tag-Paare **vollständig geklammert** (wenn er *wohlgeformt* ist).

Deshalb kann er eindeutig **als Baum dargestellt** werden. (Attribute betrachten wir hier nicht) Wir markieren die inneren Knoten mit den Tag-Namen; die **Blätter** sind die elementaren Texte:

```
<adressBuch>
<adresse>
  <name>
    <nachname>Mustermann
  </nachname>
    <vorname>Max
  </vorname>
  </name>
  <anschrift>
    <strasse>Hauptstr 42
  </strasse>
    <ort>Paderborn</ort>
    <plz>33098</plz>
  </anschrift>
</adresse>
</adressBuch>
```



XML-Werkzeuge können die Baumstruktur eines XML-Textes ohne weiteres ermitteln und ggf. anzeigen.

Wohlgeformte XML-Texte

XML-Texte sind **wohlgeformt** (well-formed), wenn sie folgende Regeln erfüllen:

1. Ein Element beginnt mit einem Anfangs-Tag und endet mit einem gleichnamigen End-Tag. Dazwischen steht eine evtl. leere Folge von Elementen und elementaren Texten.
2. Elementare Texte können beliebige Zeichen, aber keine Tags enthalten.
3. ein XML-Text ist ein Element.

wohlgeformt	wohlgeformt	nicht wohlgeformt
<code><a></code>	<code><a></code>	<code><a></code>
<code></code>	1	<code></code>
<code><c>1</c></code>	<code></code>	<code><c>1</code>
<code><d>2</d></code>	2	<code></c></code>
<code></code>	<code><c>3</c></code>	<code></code>
<code><e>3</e></code>	4	
<code></code>	<code><d>5</d></code>	
	<code></code>	
	<code><e>6</e></code>	
	<code></code>	

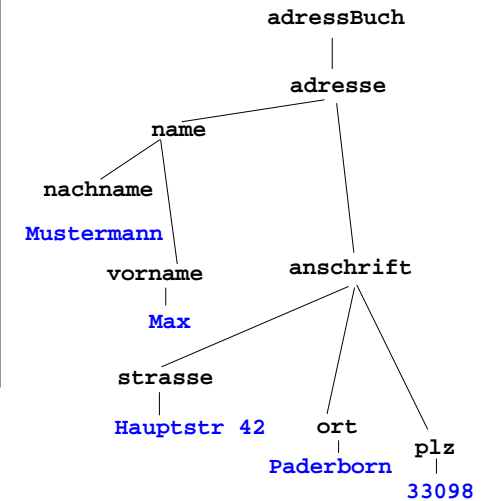
Grammatik definiert die Struktur der XML-Bäume

Mit **kontextfreien Grammatiken (KFG)** kann man **Bäume definieren**.

Folgende KFG definiert korrekt strukturierte Bäume für das Beispiel Adressbuch:

```

adressBuch ::= adresse*
adresse  ::= name anschrift
name     ::= nachname vorname
Anschrift ::= strasse ort plz
nachname ::= PCDATA
vorname  ::= PCDATA
strasse  ::= PCDATA
ort      ::= PCDATA
plz     ::= PCDATA
    
```



Document Type Definition (DTD) statt KFG

Die Struktur von XML-Bäumen und -Texten wird in der **DTD-Notation** definiert. Ihre Konzepte entsprechen denen von KFGn:

KFG	DTD
<code>adressBuch ::= adresse*</code>	<code><!ELEMENT adressBuch(adresse)* ></code>
<code>adresse ::= name anschrift</code>	<code><!ELEMENT adresse (name, anschrift) ></code>
<code>name ::= nachname vorname</code>	<code><!ELEMENT name (nachname, vorname) ></code>
<code>Anschrift ::= strasse ort plz</code>	<code><!ELEMENT anschrift (strasse, ort, plz) ></code>
<code>nachname ::= PCDATA</code>	<code><!ELEMENT nachname (#PCDATA) ></code>
<code>vorname ::= PCDATA</code>	<code><!ELEMENT vorname (#PCDATA) ></code>
<code>strasse ::= PCDATA</code>	<code><!ELEMENT strasse (#PCDATA) ></code>
<code>ort ::= PCDATA</code>	<code><!ELEMENT ort (#PCDATA) ></code>
<code>plz ::= PCDATA</code>	<code><!ELEMENT plz (#PCDATA) ></code>

weitere Formen von DTD-Produktionen:

<code>(Y)+</code>	nicht-leere Folge
<code>(A B)</code>	Alternative
<code>(A)?</code>	Option
<code>EMPTY</code>	leeres Element