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 $\in$ N	**Startsymbol** (auch Zielsymbol)
P $\subseteq$ N $\times$ V*	**Menge der Produktionen**; (A, x) $\in$ P, mit A $\in$ N und x $\in$ 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:

Terminale **T** = { **(** , **)** }

Nichtterminale **N** = { **Klammern**, **Liste** }

Startsymbol **S** = **Klammern**

Produktionsmenge P =

Name	*N*	*V**
	{	
p1:	**Klammern**	**::= ’(’ Liste ’)’**
p2:	**Liste**	**::= Klammern Liste**
p3:	**Liste**	**::=**
	}	

Bedeutung der Produktionen

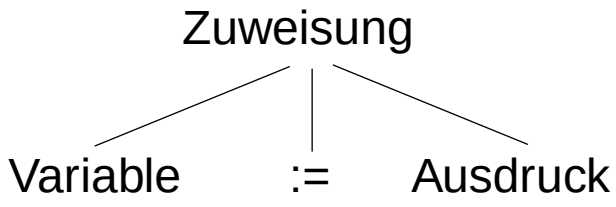
Eine Produktion A ::= x ist eine **Strukturregel**: A **besteht aus** x

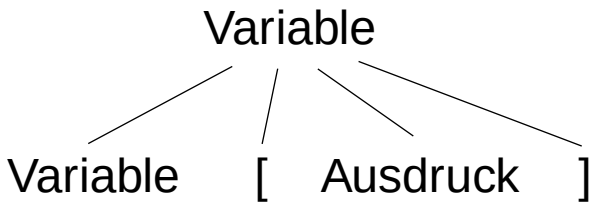
Beispiele:

	DeutscherSatz	::=	Subjekt Prädikat Objekt
Ein	DeutscherSatz	*besteht aus (der Folge)*	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 =>* 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**:

$\{ (), (()), (()()()), ((()())(())), \ldots \} \subseteq L(G)$

Ableitung des Satzes `(()())`:

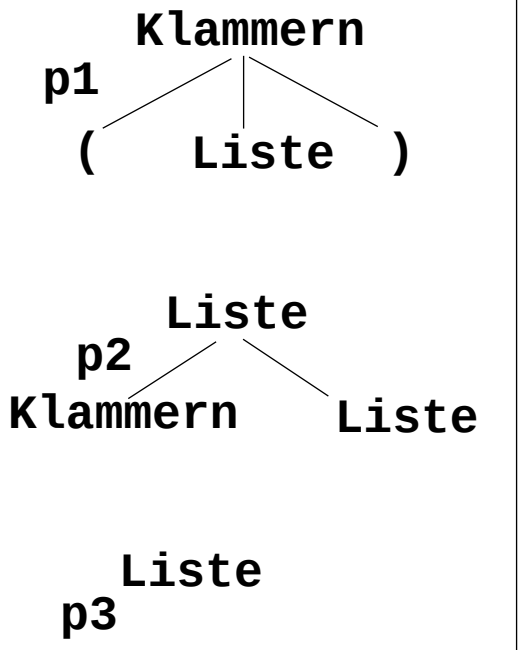
`S` **= Klammern**
=> (Liste)
=> (Klammern Liste)
=> (Klammern Klammern Liste)
=> (Klammern (Liste) Liste)
=> ((Liste) (Liste) Liste)
=> (( ) (Liste) Liste)
=> (( ) ( ) Liste)
=> (( ) ( ))

Ableitungsbäume

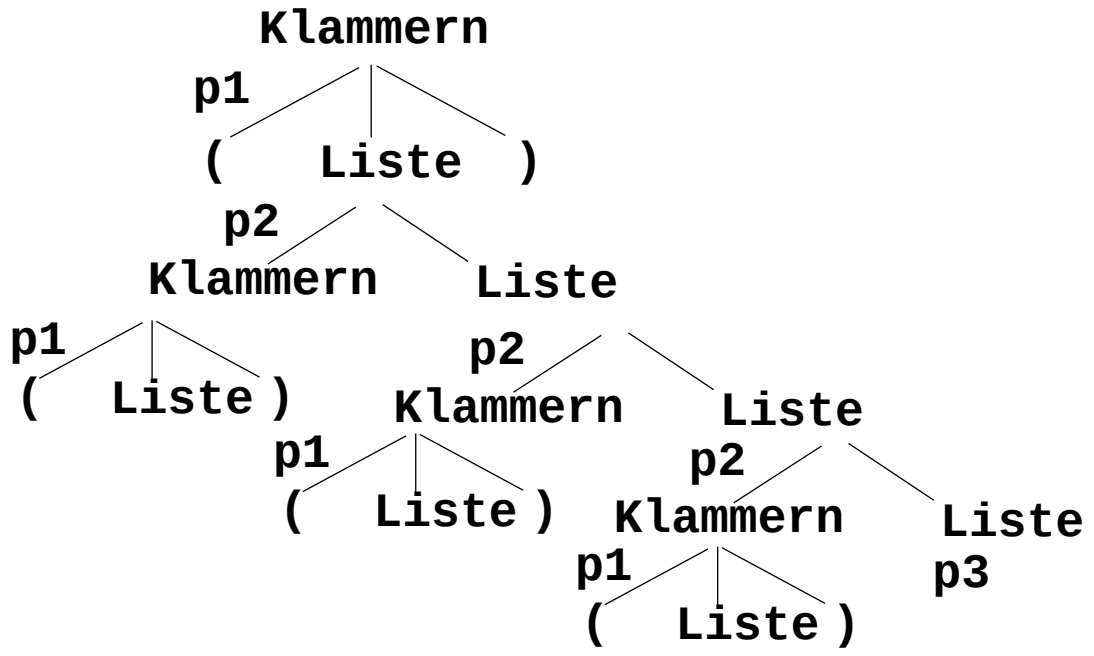
Jede Ableitung kann man als **gewurzelten Baum** darstellen:

- Die **Knoten** mit ihren Marken repräsentieren **Vorkommen von Symbolen**.
- Ein Knoten mit seinen direkten Nachbarn repräsentiert die **Anwendung einer Produktion**.
- Die **Wurzel** ist mit dem **Startsymbol** markiert.
- **Terminale** kommen nur an **Blättern** vor.

Produktionen:



ein Ableitungsbaum:



der Satz dazu: **(()(()))**

Satz zum Baum: Terminale im links-abwärts Durchgang

Beispiel: Ausdrucksgrammatik

p1: Ausdruck ::= Ausdruck BinOpr Ausdruck

p2: Ausdruck ::= Zahl

p3: Ausdruck ::= Bezeichner

p4: Ausdruck ::= ’(’ Ausdruck ’)’

p5: BinOpr ::= ’+’

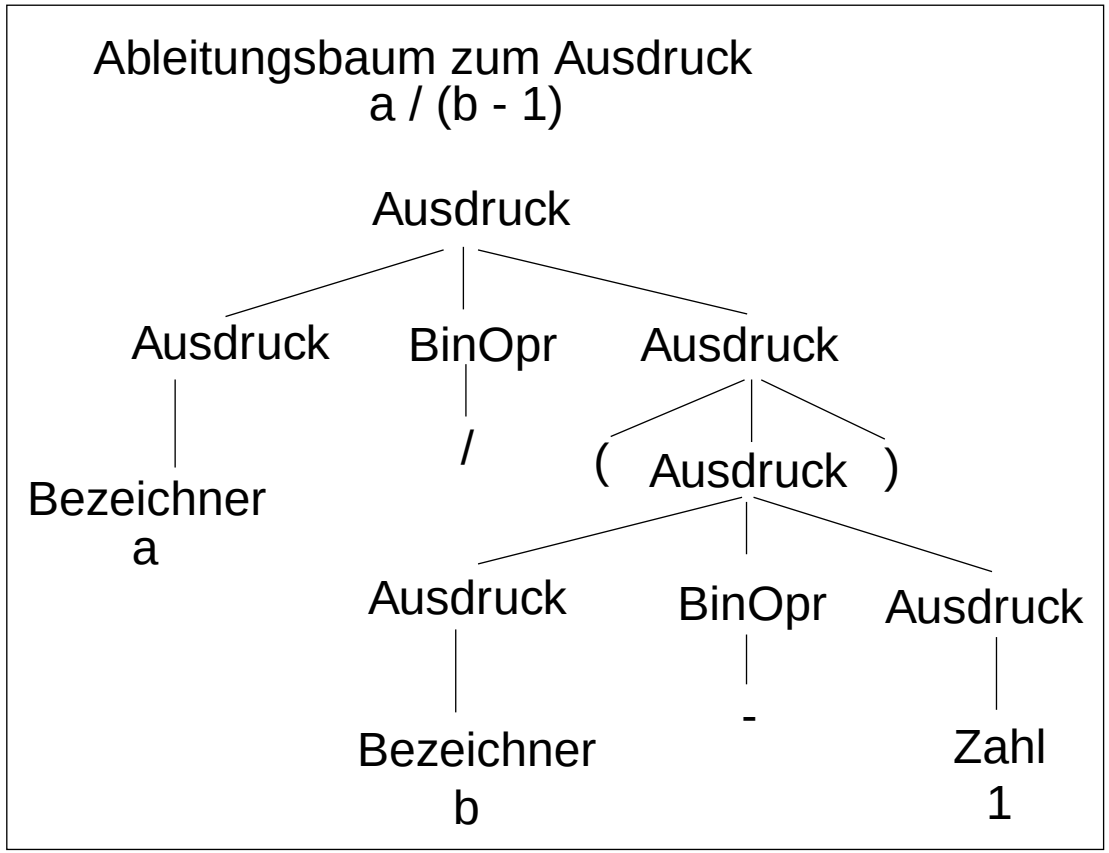
p6: BinOpr ::= ’-’

p7: BinOpr ::= ’*’

p8: BinOpr ::= ’/’

Startsymbol: Ausdruck

Terminale:
T = { Zahl, Bezeichner, (,), +, -, *, / }



Schreibweise der Terminale
Zahl und Bezeichner wird nicht in der KFG definiert.

Grammatik ist mehrdeutig: Es gibt **Sätze, die mehrere Ableitungsbäume** haben.

Beispiel: Tabellen in HTML

HTML: Hypertext Markup Language zur Darstellung von verzeigerten Dokumenten, insbesondere im WWW verwendet.
typisch: geklammerte Strukturen mit Klammern der Form **<x>...</x>**.
hier: vereinfachter Ausschnitt aus der Sprache zur Darstellung von Tabellen.

Produktionen der kontextfreien Grammatik:

Table	::= '<table>' Rows '</table>'
Rows	::= Row *****
Row	::= '<tr>' Cells '</tr>'
Cells	::= Cell *****
Cell	::= '<td>' Text '</td>'
Cell	::= '<td>' Table '</td>'

Erweiterung der Notation von KFGn:

X * auf der rechten Seite einer Produktion steht für eine **beliebig lange Folge von X**

(gleiche Bedeutung wie bei Wertebereichen)

Beispieltext in HTML:

```
<table>
    <tr> <td>Tag</td>
         <td>Zeit</td>
         <td>Raum</td></tr>
    <tr> <td>Mo</td>
         <td>11:00-12.30</td>
         <td>AM</td></tr>
    <tr> <td>Fr</td>
         <td>9:15-10:45</td>
         <td>AM</td></tr>
</table>
```

Darstellung der Tabelle:

Tag	Zeit	Raum
Mo	11:00-12.30	AM
Fr	9:15-10:45	AM

6.2 Baumstrukturen in XML
Übersicht

XML (Extensible Markup Language, dt.: Erweiterbare Auszeichnungssprache)

- seit 1996 vom W3C definiert, in Anlehnung an SGML
- Zweck: Beschreibungen **allgemeiner Strukturen** (nicht nur Web-Dokumente)
- **Meta-Sprache** (“erweiterbar”):
  Die Notation ist festgelegt (Tags und Attribute, wie in HTML),
  Für beliebige Zwecke kann **jeweils eine spezielle syntaktische Struktur** definiert werden (DTD)
  Außerdem gibt es Regeln (XML-Namensräume), um XML-Sprachen in andere **XML-Sprachen zu importieren**
- X**HTML** ist so als XML-Sprache definiert
- Viele **Sprachen sind aus XML abgeleitet,** z.B. SVG, MathML, SMIL, RDF, WML
- **individuelle XML-Sprachen** werden definiert, um strukturierte Daten zu speichern, die von **Software-Werkzeugen geschrieben und gelesen** werden
- XML-Darstellung von strukturierten Daten kann mit verschiedenen Techniken **in HTML transformiert** werden, um sie **formatiert anzuzeigen:**
  XML+CSS, XML+XSL, SAX-Parser, DOM-Parser

Dieser Abschnitt orientiert sich eng an **SELFHTML** (Stefan Münz), http://de.selfhtml.org

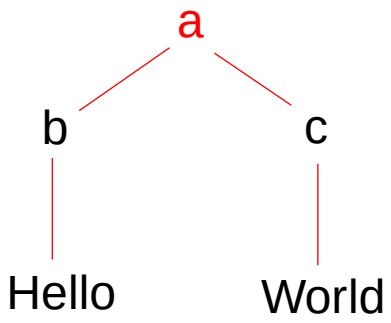
3 elementare Prinzipien

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 gewurzeltem 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

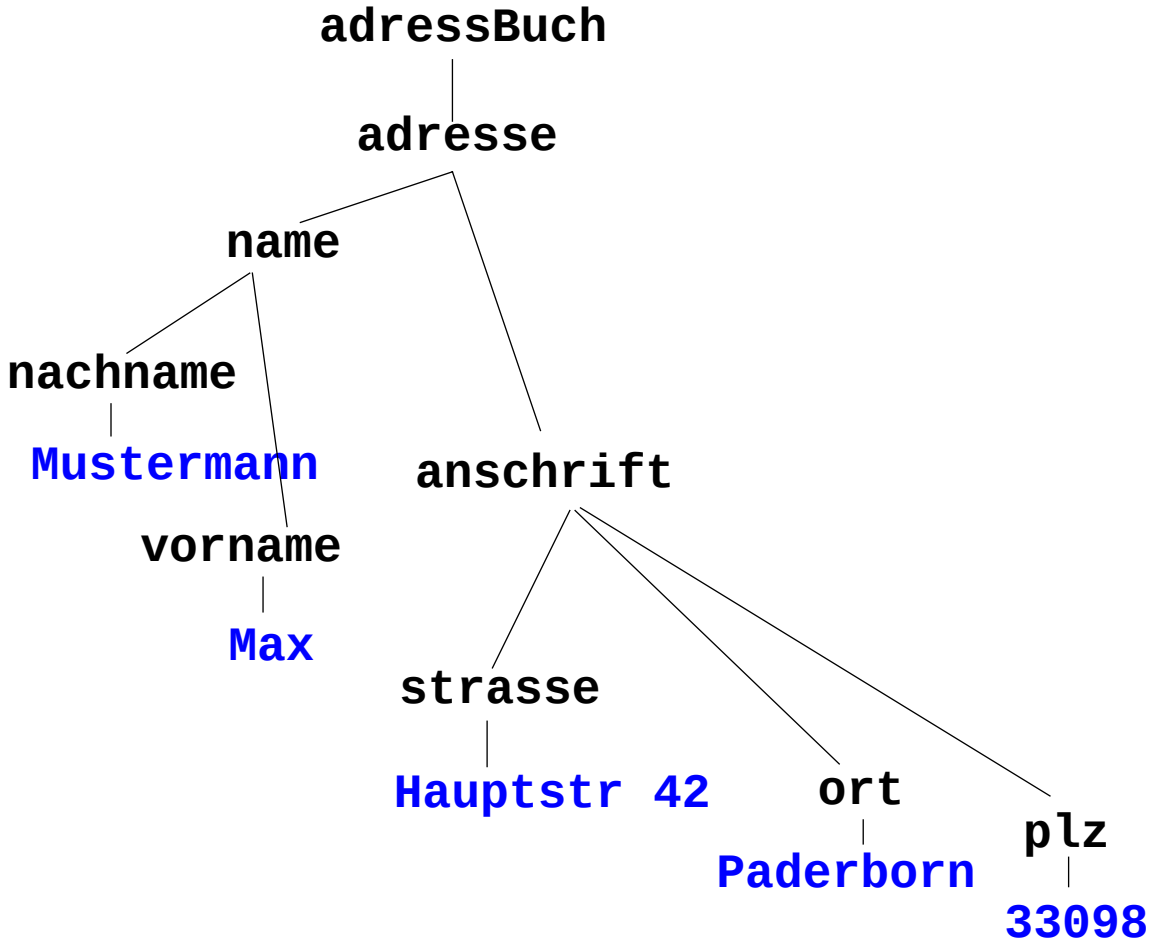
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE adressBuch SYSTEM "adressBuch.dtd">
<?xml-stylesheet type="text/xsl" href="adressBuch.xsl" ?>
<adressBuch>
<adresse>
   <name>
      <nachname>Mustermann</nachname>
      <vorname>Max</vorname>
   </name>
   <anschrift>
      <strasse>Hauptstr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

```
<a>
   <b>
      <c>1</c>
      <d>2</d>
   </b>
   <e>3</e>
</a>
```

wohlgeformt

```
<a>
   1
   <b>
      2
      <c>3</c>
      4
      <d>5</d>
   </b>
   <e>6</e>
</a>
```

nicht wohlgeformt

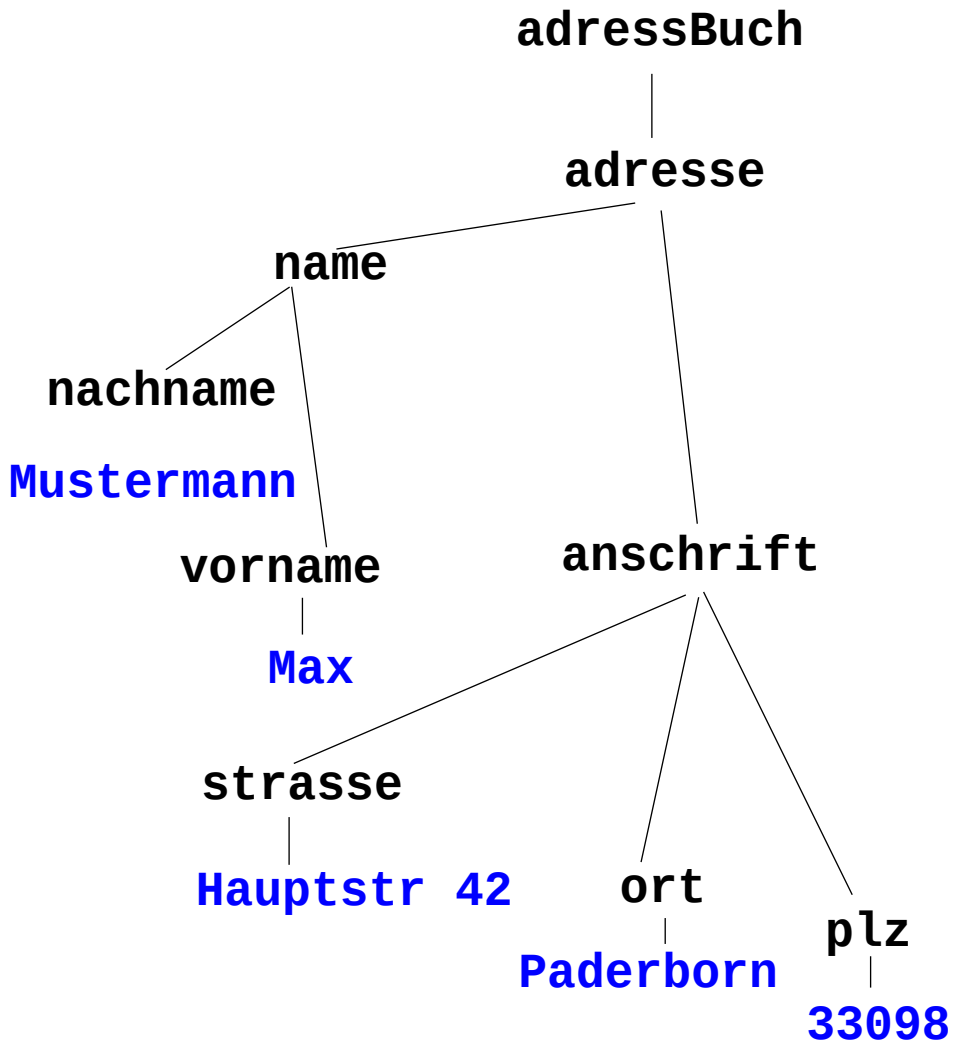
```
<a>
   <b>
      <c>1</b>
   </c>
</a>
```

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

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

DTD

```
<!ELEMENT adressBuch(adresse)*          >
<!ELEMENT adresse   (name, anschrift)  >
<!ELEMENT name      (nachname, vorname)>
<!ELEMENT anschrift (strasse, ort, plz)>
<!ELEMENT nachname  (#PCDATA)          >
<!ELEMENT vorname   (#PCDATA)          >
<!ELEMENT strasse   (#PCDATA)          >
<!ELEMENT ort       (#PCDATA)          >
<!ELEMENT plz       (#PCDATA)          >
```

weitere Formen von DTD-Produktionen:

`(Y)+`	nicht-leere Folge
`(A | B)`	Alternative
`(A)?`	Option
`EMPTY`	leeres Element