

## E2. Symbole und Syntax Überblick

Grundbegriffe zur **formalen Definition von Sprachen**:

- **Alphabet**: Menge von Zeichen
- **Wort**: Folge von Zeichen aus Alphabet - gebildet nach bestimmten Regeln (Ebene 1)  
**Satz**: Folge von Symbolen aus Vokabular - gebildet nach bestimmten Regeln (Ebene 2)  
(Wort, Satz), (Zeichen, Symbol) und (Alphabet, Vokabular) paarweise synonym
- **Sprache**: Menge von Worten bzw. Sätzen

**Kalküle** zur Bildung von Worten und Sätzen:

- **reguläre Ausdrücke**

zur Definition der **Notation von Grundsymbolen**  
(Ebene 1 der Spracheigenschaften)

zur Definition von **Textmustern**

angewandt zum Suchen und Ersetzen von Zeichenfolgen  
programmiert in PHP, Perl, Unix-sh

- **kontextfreie Grammatik**

zur Definition der **Menge der syntaktisch korrekten Sätze** einer Sprache  
(Ebene 2 der Spracheigenschaften)

**Struktur** der syntaktisch korrekten Sätze einer Sprache

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 301

#### Ziele:

Zusammenhang der Begriffe erkennen

#### in der Vorlesung:

- Die Begriffe werden kurz erklärt und
- der Zusammenhang gezeigt.
- Auf der lexikalischen Ebene werden meist die Bezeichnungen Wort, Zeichen, Alphabet benutzt; aus der syntaktischen Ebene spricht man stattdessen von Satz, Symbol, Vokabular - bei gleicher Bedeutung.
- Die Anwendungen der Kalküle werden erklärt.

## Alphabete und Zeichenfolgen

Ein **Alphabet** ist eine nicht-leere **Menge von Zeichen** zur Bildung von Zeichenfolgen.

Wir betrachten hier nur Alphabete mit endlich vielen Zeichen.

Alphabete werden in Formeln häufig mit  $\Sigma$  bezeichnet;  
sonst gibt man ihnen individuelle Namen oder benutzt sie unbenannt.

Beispiele:

$\Sigma =$	{T, F}
Dualziffern =	{0, 1}
Dezimalziffern =	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
Kleinbuchstaben =	{a, b, ..., z}
Nukleotide =	{A, C, G, U}
ASCII =	standardisierter Zeichensatz mit 128 Zeichen

Ein **Wort über einem Alphabet A** ist eine Folge von Zeichen aus A.

formal: eine Folge  $a_1 a_2 \dots a_n$ , mit  $a_i \in A$ , für  $i = 1, \dots, n$ .

n ist die **Länge der Folge** bzw. die **Länge des Wortes**.

Beispiele: Wort der Länge 7 über dem Alphabet Dualziffern: 1001101

Die **leere Folge** bzw. das **leere Wort** wird mit  $\varepsilon$  (epsilon) bezeichnet.

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 302

**Ziele:**

Wörter über Alphabeten

**in der Vorlesung:**

Erläuterungen und Beispiele dazu

## Reguläre Ausdrücke

Ein **regulärer Ausdruck**  $R$

definiert eine **Mengen von Worten** über einem Alphabet  $A$ , die **Sprache**  $L(R)$ .

Ein regulärer Ausdruck kann aus folgenden 8 Formen rekursiv zusammengesetzt sein.  
 $F$  und  $G$  seien reguläre Ausdrücke.

regulärer Ausdruck $R$	Sprache $L(R)$	Erklärung
$a$	$\{a\}$	Zeichen $a$ als Wort
$FG$	$\{fg \mid f \in L(F), g \in L(G)\}$	Zusammenfügen von 2 Worten
$F G$	$\{f \mid f \in L(F)\} \cup \{g \mid g \in L(G)\}$	Alternativen
$\varepsilon$	$\{\varepsilon\}$	das leere Wort
$(F)$	$L(F)$	Klammerung
$F^+$	$\{f_1 f_2 \dots f_n \mid f_i \in L(F), \text{ für } n \geq 1, i=1, \dots, n\}$	nicht-leere Folgen von Worten aus $L(F)$
$F^*$	$\{\varepsilon\} \cup L(F^+)$	Folgen von Worten aus $L(F)$
$F^n$	$\{f_1 f_2 \dots f_n \mid f_i \in L(F), \text{ für } i = 1, \dots, n\}$	Folgen von genau $n$ Worten aus $L(F)$

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 303

#### Ziele:

Definition von regulären Ausdrücken verstehen

#### in der Vorlesung:

Erläuterungen zu

- rekursiver Definition,
- Hintereinanderschreibung von Zeichen und Teilworten,
- Alternativen,
- leer und Klammern,
- Folgen von Worten

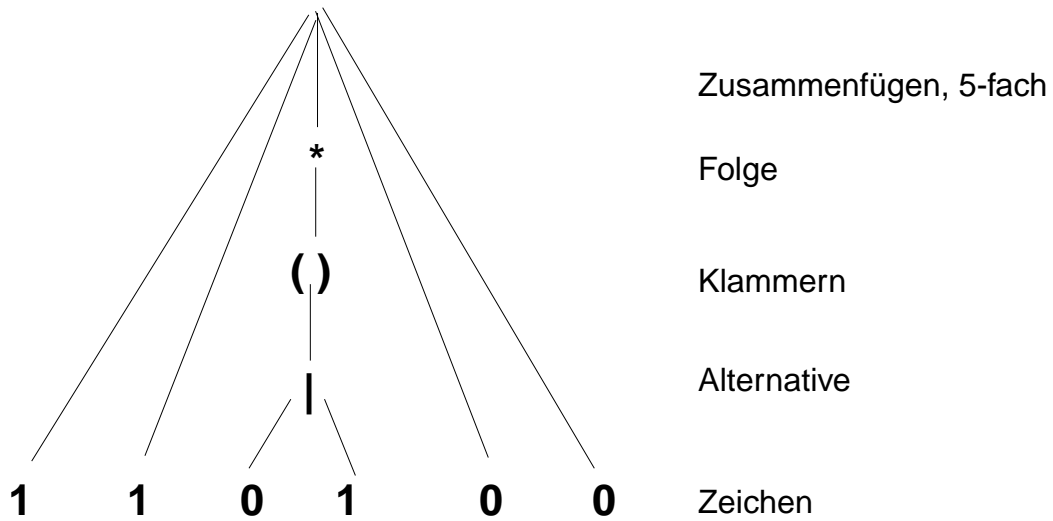
#### Verständnisfragen:

Unterscheiden Sie:

- das leere Wort,
- die leere Menge,
- die Menge, die nur das leere Wort enthält.

## Struktur eines regulären Ausdruckes

1 1 (0 | 1)\* 0 0



verbal:

Jedes Wort aus der Sprache dieses regulären Ausdruckes besteht aus zwei 1, gefolgt von beliebig vielen 0 oder 1, gefolgt von zwei 0.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 304

### Ziele:

Aufbau von regulären Ausdrücken verstehen

### in der Vorlesung:

Erläuterungen:

- rekursive Konstruktion,
- Struktur erkennen.
- Bedeutung folgt der Struktur.
- Verbale Beschreibung folgt der Struktur.
- Das Alphabet ergibt sich aus den Zeichen, die in dem regulären Ausdruck vorkommen.

## Beispiele für reguläre Ausdrücke

Name	regulärer Ausdruck $A$	Worte aus seiner Sprache $L(A)$
$Abc =$	$(a   b) (c   d   \epsilon)$	ac bc ad bd a b
$Anrede =$	Sehr geehrte(r   $\epsilon$ ) (Frau   Herr)	Sehr geehrte Frau
$Dig =$	$0   1   \dots   9$	7
$sLet =$	$a   b   \dots   z$	x
$cLet =$	$A   B \dots   Z$	B
$Let =$	$sLet   cLet$	m N
$Bezeichner =$	$Let ( Let   Dig )^*$	Maximum min3 a
$GeldBetrag =$	$Dig^+, Dig^2$	23,95 0,50
$KFZ =$	$(cLet   cLet^2   cLet^3) - (cLet   cLet^2) - (Dig   Dig^2   Dig^3   Dig^4)$	PB-AX-123
$Dual =$	$1^3 (1   0)^* 0^3$	1111000 111000 1111101010000

Wenn **Namen** von regulären Ausdrücken **in regulären Ausdrücken** verwendet werden, müssen sie von den Zeichen unterschieden werden können; hier *Namen* kursiv.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 305

### Ziele:

Reguläre Ausdrücke anwenden lernen

### in der Vorlesung:

Einen regulären Ausdruck verstehen:

- Struktur erkennen,
- Bedeutung erkennen, verbal beschreiben,
- Sprache herleiten,
- Beispiele für Worte aus der Sprache geben. Einen regulären Ausdruck entwerfen:
- Beispiele für Worte aus der Sprache geben.
- Sprache herleiten, verbal beschreiben,
- Struktur entwerfen,
- regulären Ausdruck aufschreiben.

## Beispiele für Definitionen von Grundsymbolen

*Pascal\_Identifier* =  $Let (Let | Dig)^*$

*C\_Identifier* =  $(Let | \_)(Let | \_ | Dig)^*$

*ADA\_Identifier* =  $Let ( (\_ | \epsilon) (Let | Dig) )^*$

*PHP\_Var\_Identifier* =  $\$ (Let | \_)(Let | \_ | Dig)^*$

*Pascal\_Real* =  $((Dig^+ \cdot Dig^+) ((e | E) (+ | - | \epsilon) Dig^+) | \epsilon) | (Dig^+ (e | E) (+ | - | \epsilon) Dig^+)$

*HexDig* =  $Dig | a | b | c | d | e | f | A | B | C | D | E | F$

*HTML\_CharRef* =  $\& ( Let^+ | \# Dig^+ | \#x HexDig^+ ) ;$

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 306

**Ziele:**

Definitionen verstehen

**in der Vorlesung:**

- Reguläre Ausdrücke untersuchen,
- Beispiele für Worte seiner Sprache angeben

## Reguläre Ausdrücke als Textmuster

In Sprachen, die zur **Textverarbeitung** eingesetzt werden, benutzt man **reguläre Ausdrücke, um Textmuster zu definieren.**

### Beispiele:

suche alle Dateinamen der Form `ews(0|1|2|3|4|5|6|7|8|9)3.html`

in der Schreibweise von Unix-sh `ls ews[0-9][0-9][0-9].html`

Aufruf einer PHP-Funktion `preg_match ("/[dD]aß/", $absatz)`  
sucht ein Textmuster in einer Zeichenreihe      Muster      Zeichenreihe

```
$d = "[0-9]";  
preg_match ("/ews$d$d$d\.html/", $files)
```

Reguläre Ausdrücke als Textmuster sind umfassend in der Skriptsprache Perl definiert und so in PHP übernommen.

Auf den vorigen Folien haben wir die **grundlegenden Begriffe** für reguläre Ausdrücke mit der **dafür üblichen Notation** eingeführt. In **PHP, Perl**, Unix-sh werden die gleichen Begriffe aber in anderer **Notation** verwendet.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 307

### Ziele:

Anwendung von regulären Ausdrücke als Textmuster

### in der Vorlesung:

- Beispiele erläutern.
- Neue Notation begründen.

## Notation von regulären Ausdrücken in PHP

$a$	das Zeichen $a$
$FG$	Zusammenfügen von 2 Worten
$F G$	Alternativen
$(F)$	Klammerung
$F?$	Option; wie $F \epsilon$
$F^+$	nicht-leere Folge von Worten aus $L(F)$
$F^*$	beliebig lange Folge von Worten aus $L(F)$
$F\{m,n\}$	Folge mit mindestens $m$ und höchstens $n$ von Worten aus $L(F)$
$F\{m\}$	Folge mit genau $m$ Worten aus $L(F)$
$[abc]$	alternativ ein Zeichen aus der Klammer
$[\^abc]$	alternativ ein anderes Zeichen als die in der Klammer
$[a-zA-Z]$	alternativ ein Zeichen aus Zeichenbereichen
$.$	beliebiges Zeichen
$^$	Anfang der Zeichenfolge (nichts darf vorangehen)
$\$$	Ende der Zeichenfolge (nichts darf darauf folgen)

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 308

**Ziele:**

Notation lernen

**in der Vorlesung:**

- Notationen vergleichen,
- bisher definierte reguläre Ausdrücke umschreiben,
- weitere Beispiele angeben.



## Beispiele für reguläre Ausdrücke in PHP-Notation

Reguläre Ausdrücke kommen in PHP-Programmen immer als Zeichenreihenliterale (Strings) vor. Diese werden in " eingeschlossen, z. B. "1|0".

Statt Namen für reguläre Ausdrücke zu definieren, weisen wir die Zeichenreihe einer Variablen zu, z. B. `$binDig = "(1|0)";`

**Variable = regulärer Ausdruck als PHP-String**

`$Abc = "(a|b)(c|d)?"`;

`$Anrede = "Sehr geehrte(r)? (Frau|Herr)";`

`$Dig = "[0-9]";`

`$sLet = "[a-z]";`

`$cLet = "[A-Z]";`

`$Let = "[a-zA-Z]";`

`$Bezeichner = "[a-zA-Z][a-zA-Z0-9]*";`

`$GeldBetrag = "$Dig+,$Dig{2}";`

`$KFZ = "$cLet{1,3}-$cLet{1,2}-$Dig{1,4}";`

`$Dual = "1{3}[10]*0{3}";`

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 309

### Ziele:

Notationen üben

### in der Vorlesung:

- Notationen vergleichen,
- bisher definierte reguläre Ausdrücke umschreiben.

## Beispiele für Definitionen von Grundsymbolen in PHP-Notation

```
$Pascal_Identifier = "[a-zA-Z][a-zA-Z0-9]*";
$C_Identifier = "[a-zA-Z_][a-zA-Z_0-9]*";
$ADA_Identifier = "[a-zA-Z](_[a-zA-Z0-9])*";
$PHP_Var_Identifier = "\$[a-zA-Z_][a-zA-Z_0-9]*";
$Pascal_Exponent = "((e|E)(\+|-)?[0-9]+)";
$Pascal_Real = "(([0-9]+\.[0-9]+)$Exponent?)|([0-9]+$Exponent)";
$HexDig = "[0-9a-fA-F]";
$HTML_CharRef = "&(([a-zA-Z]+)|(#[0-9]+)|(#x$HexDig+));";
```

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 310

**Ziele:**

Notationen üben

**in der Vorlesung:**

- Notationen vergleichen,
- bisher definierte reguläre Ausdrücke umschreiben.

## E2.2 Kontextfreie Grammatiken

**Kontextfreie Grammatik (KFG):** formaler Kalkül zur Definition einer

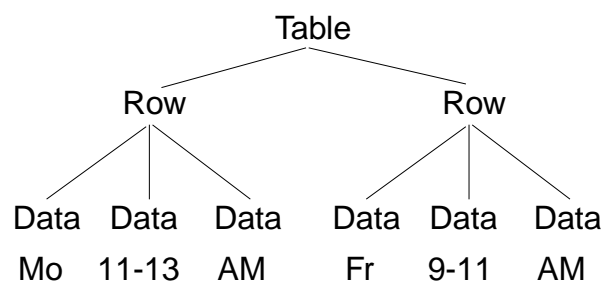
- **Sprache** als Menge von Sätzen; jeder **Satz** ist eine **Folge von Symbolen**
- **Menge von Bäumen**; jeder Baum repräsentiert die **Struktur eines Satzes** der Sprache

**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

### Beispiel: Tabellen in 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>
```



## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 311

### Ziele:

Einsatz von KFGn kennenlernen

### in der Vorlesung:

Erläuterungen zu den Anwendungen:

- Struktur von HTML-Sätzen,
- Struktur von Tabellen.
- Bäume werden noch definiert.

## Definition: 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 disjunkt)
- S **Startsymbol**; S ist ein Nichtterminal:  $S \in N$
- P **Menge der Produktionen**  
jede Produktion hat die Form  $A ::= x$   
A ist ein Nichtterminal, d. h.  $A \in N$  und  
x ist eine (evtl. leere) Folge von Terminalen und Nichtterminalen,  
d. h.  $x \in (T \cup N)^* = V^*$

$V = T \cup N$  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:  $A ::= B C D$

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 312

### Ziele:

KFG Definition lernen

### in der Vorlesung:

- Erläuterung der Begriffe am Beispiel der nächsten Folie EWS-3.13,
- Erläuterung der Notation von Produktionen
- Wir benutzen jetzt die Bezeichnungen Satz, Symbol und Vokabular statt Wort, Zeichen und Alphabet.
- Die Angabe  $V^*$  ist als regulärer Ausdruck zu verstehen, also "eine (evtl. auch leere) Folge beliebiger Symbole aus V."

## Beispiel zur Definition einer KFG

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

**Nichtterminale**  $N = \{ \text{Klammern}, \text{Liste} \}$

**Startsymbol**  $S = \text{Klammern}$

**Produktionen**  $P =$

{

p1:  $\text{Klammern} ::= \text{'(' Liste '}'$

p2:  $\text{Liste} ::= \text{Klammern Liste}$

p3:  $\text{Liste} ::=$

}

**Vokabular**  $V = T \cup N =$   
 $\{ (, ), \text{Klammern}, \text{Liste} \}$

Unbenannte Terminale werden in ' eingeschlossen, um Verwechslungen mit KFG-Zeichen zu vermeiden: '( '

	$\cap$	$\cap$
<b>Namen</b>	<b>N</b>	<b>V*</b>

Diese Grammatik definiert eine Sprache, deren Sätze Folgen von geschachtelten Klammerpaaren sind, z. B.

$(( ))$        $(( ( )) ( ( ) ( ) ) )$

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 313

### Ziele:

Kleines Beispiel zum Verstehen der Definition

### in der Vorlesung:

Die Begriffe der Definition von KFGn werd an diesem Beispiel erklärt:

- Im Rahmen stehen die formal notwendigen Angaben T, N, S, P.
- Vokabular und Namen von Produktionen sind zusätzliche Angaben.
- Die Menge N wird aus den Symbolen auf den linken Seiten der Produktionen gebildet.
- Symbole, die in Produktionen vorkommen, aber nicht in N sind, sind Terminale in T.
- Die Anwendung der Produktionen als Regeln zur Definition der Sprache der Grammatik folgt auf den nächsten Folien.

## Bedeutung der Produktionen

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

### Beispiele:

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

**Produktion** graphisch dargestellt als gewurzelter **Baum** mit geordneten Kanten und mit Symbolen als Knotenmarken (zum Begriff „Baum“ siehe nächste Folie):



## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 314

### Ziele:

Produktionen verstehen

### in der Vorlesung:

Erläuterungen der beiden Rollen von Produktionen:

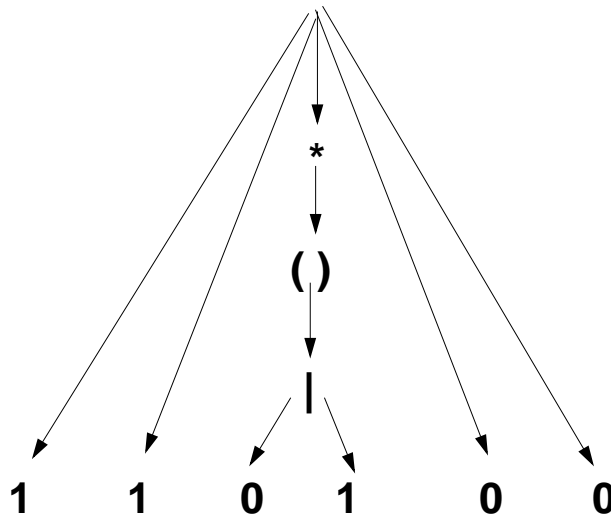
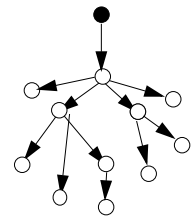
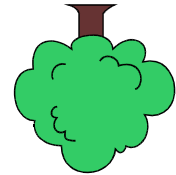
- Definition von Struktur: "besteht aus"
- Definition von Ersetzungen
- Siehe auch spätere Folie zur graphischen Darstellung von Produktionen.

## Bäume als Abstraktion

**Bäume** bezeichnen in der Informatik **abstrakte Datenstrukturen** mit speziellen Eigenschaften.

Sie werden zur abstrakten Beschreibung bestimmter Zusammenhänge eingesetzt. Ein besonders wichtiger ist die „**besteht-aus**“-Beziehung zwischen einem Objekt und seinen Teilen.

Z. B. ein regulärer Ausdruck besteht aus Teilausdrücken:



### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 315

**Ziele:**

Bäume als Abstraktion verstehen

**in der Vorlesung:**

Am Beispiel wird gezeigt, wie die besteht-aus Relation durch einen Baum dargestellt wird.

## Begriffe zu Bäumen

**Definition:** Ein (gewurzelter, gerichteter) **Baum** besteht aus

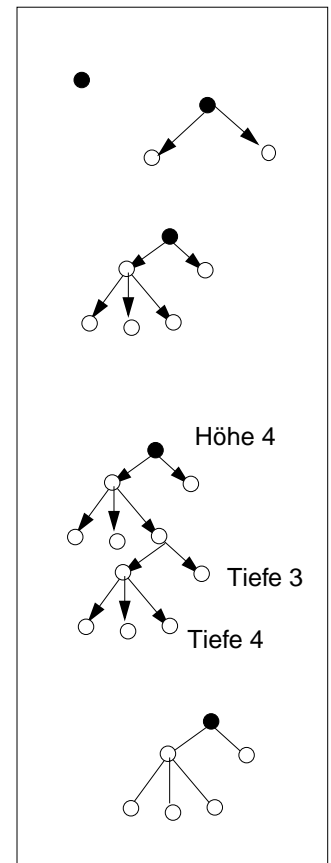
- einem **Knoten**  $k$  und
- einer (evtl. leeren) **Folge von Bäumen** und **Kanten** von  $k$  zu jedem Element der Folge.
- In **einen** Knoten mündet keine Kante, in alle anderen genau eine.

Begriffe und Eigenschaften:

- **Wurzel:** Knoten in den keine Kante mündet.
- **Blätter:** Knoten, von denen keine Kante ausgeht
- **innere Knoten:** es mündet eine Kante und es gehen welche aus
- Es gibt genau eine Wurzel.
- Wenn ein Baum  $n$  Knoten hat, dann hat er  $n-1$  Kanten.
- **Tiefe eines Blattes:** Anzahl der Kanten auf dem Weg von der Wurzel zu dem Blatt.
- **Höhe des Baumes:** größte Tiefe aller seiner Blätter.

Knoten und/oder Kanten können **beschriftet** werden.

Man kann die Pfeilspitzen weglassen, wenn die Wurzel bekannt ist.



## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 316

### Ziele:

Definition und Begriffe zu Bäumen verstehen

### in der Vorlesung:

Die Definition und die Begriffe werden an Beispielen erklärt.



## 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  $u A v \Rightarrow u x v$

z. B. **Klammern Klammern** Liste  $\Rightarrow$  Klammern ( Liste ) Liste  
mit Produktion p1: **Klammern** ::= ( Liste )

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 EWS-3.13 definiert geschachtelte Folgen paariger Klammern als Sprachmenge:

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

Ableitung des Satzes $( ( ) ( ) )$ :	<b>S</b>	= <b>Klammern</b>	p1
		$\Rightarrow$ ( <b>Liste</b> )	p2
		$\Rightarrow$ ( Klammern <b>Liste</b> )	p2
		$\Rightarrow$ ( Klammern <b>Klammern</b> Liste )	p1
		$\Rightarrow$ ( <b>Klammern</b> ( Liste ) Liste )	p1
		$\Rightarrow$ ( ( <b>Liste</b> ) ( Liste ) Liste )	p3
		$\Rightarrow$ ( ( ) ( <b>Liste</b> ) Liste )	p3
		$\Rightarrow$ ( ( ) ( ) <b>Liste</b> )	p3
		$\Rightarrow$ ( ( ) ( ) )	

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 317

### Ziele:

Ableitungsbegriff verstehen

### in der Vorlesung:

Erläuterungen dazu

- Beispiele für Ableitungen
- Beispiele für Sprachen

# 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**.

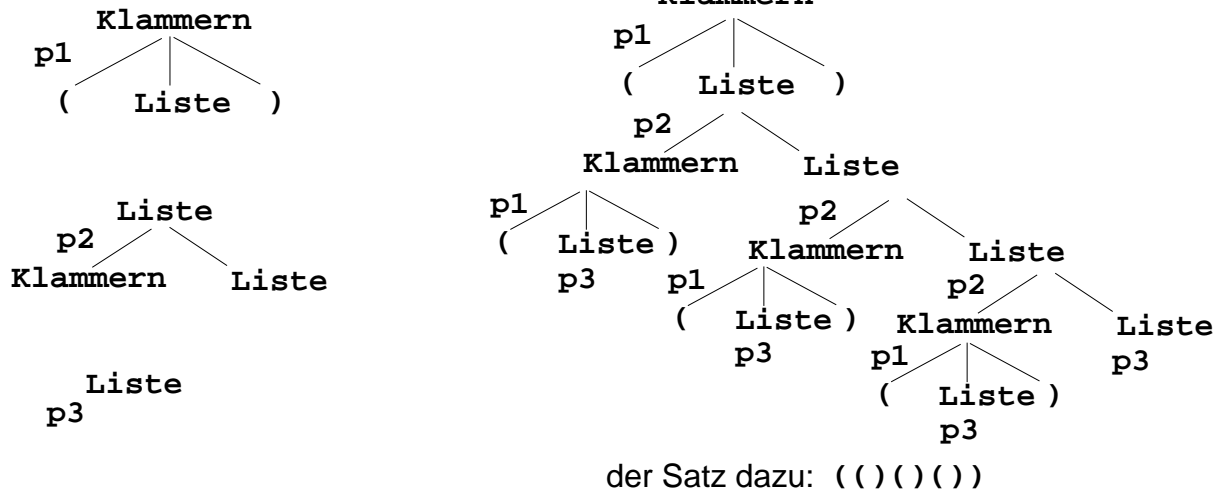
Die **Wurzel** ist mit dem **Startsymbol** markiert.

**Terminale** kommen nur an **Blättern** vor.

Ein Ableitungsbaum entsteht durch „Zusammensetzen von Kopien der Produktionsbäume“.

**Produktionen:**

ein **Ableitungsbaum:**



© 2004 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 318

**Ziele:**  
Ableitungsbaum verstehen

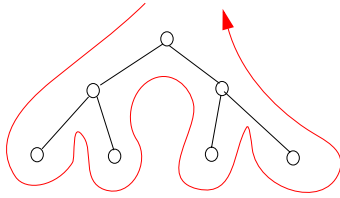
- in der Vorlesung:**
- Konstruktion des Baumes durch Zusammensetzen von Produktionsanwendungen,

## Satz zum Ableitungsbaum

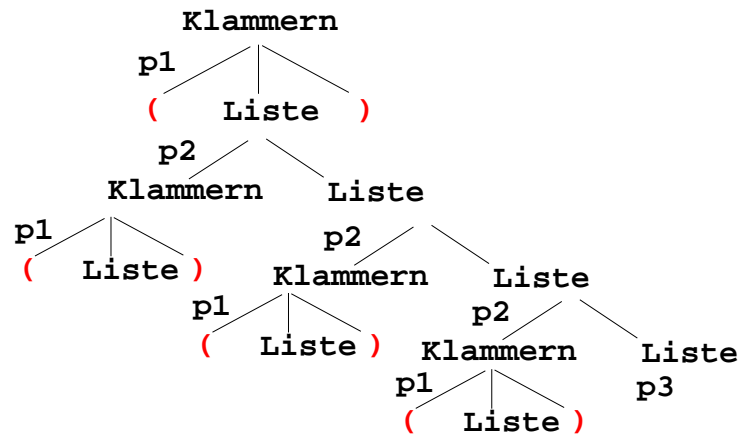
Man erhält den **Satz** aus einem **Ableitungsbaum**, wenn man seine **Terminale** in der Reihenfolge eines **links-abwärts Durchlaufes** aufschreibt.

### Ein links-abwärts Durchlauf

- **beginnt** mit einem Besuch der **Wurzel**,
- bei dem Besuch eines Knotens werden nacheinander für jeden seiner **Unterbäume von links nach rechts** jeweils ein **links-abwärts Durchlauf** durchgeführt.



ein **Ableitungsbaum**:



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

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 319

### Ziele:

Satz aus Ableitungsbaum herleiten

### in der Vorlesung:

An Beispielen wird erklärt:

- rekursiv definierter Baumdurchlauf,
- Zusammenhang zum Satz der Sprache

# Grammatik für arithmetische Ausdrücke

## Grammatik für arithmetische Ausdrücke Produktionen:

p1: Expr ::= Expr AddOpr Fact  
p2: Expr ::= Fact  
p3: Fact ::= Fact MulOpr Opd  
p4: Fact ::= Opd  
p5: Opd ::= '(' Expr ')'  
p6: Opd ::= Ident  
p7: AddOpr ::= '+'  
p8: AddOpr ::= '-'  
p9: MulOpr ::= '\*'  
p10: MulOpr ::= '/'

## Beispiele:

b + c  
a \* (b + c)  
a \* b + c  
a + b \* c

$T = \{ (, ), +, -, *, /, \text{Ident} \}$

$N = \{ \text{Expr}, \text{Fact}, \text{Opd}, \text{AddOpr}, \text{MulOpr} \}$

$S = \text{Expr}$

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 320

### Ziele:

Kleine, realistische Grammatik verstehen

### in der Vorlesung:

An Beispielen wird erklärt:

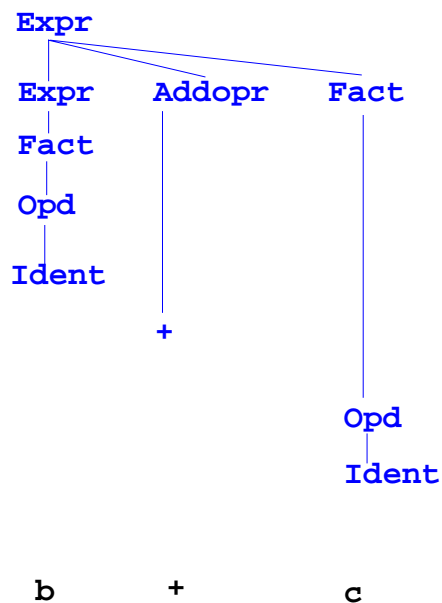
- Ableitungen und Bäume dazu bilden (mit EWS-3.21),
- Rollen der Produktionen verstehen.

# Beispiel für eine Ableitung zur Ausdrucksgrammatik

Satz der Ausdrucksgrammatik  $b + c$   
**Ableitung:**

**Ableitungsbaum:**

		<b>Expr</b>		
p1	=>	<b>Expr</b>	<b>Addopr</b>	<b>Fact</b>
p2	=>	<b>Fact</b>	<b>Addopr</b>	<b>Fact</b>
p4	=>	<b>Opd</b>	<b>Addopr</b>	<b>Fact</b>
p6	=>	<b>Ident</b>	<b>Addopr</b>	<b>Fact</b>
p7	=>	<b>Ident</b>	<b>+</b>	<b>Fact</b>
p4	=>	<b>Ident</b>	<b>+</b>	<b>Opd</b>
p6	=>	<b>Ident</b>	<b>+</b>	<b>Ident</b>
		<b>b</b>	<b>+</b>	<b>c</b>



## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 321

**Ziele:**

Zusammenhang zw. Ableitung und Ableitungsbaum

**in der Vorlesung:**

An dem Beispiel wird erklärt:

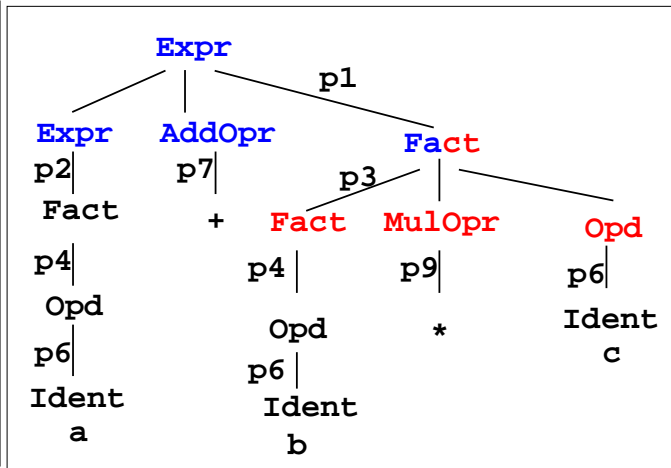
- Baum stellt Ableitungsschritte dar.
- Ableitungsschritte können in unterschiedlicher Reihenfolge ausgeführt werden,
- die Bäume dazu sind identisch.

# Präzedenz und Assoziativität in Ausdrucksgrammatiken

Die Struktur eines Satzes wird durch seinen Ableitungsbaum bestimmt. Ausdrucksgrammatiken legen dadurch die **Präzedenz** und **Assoziativität** von Operatoren fest.

Im Beispiel hat **AddOpr** geringere Präzedenz als **MulOpr**, weil er höher in der Hierarchie der Kettenproduktionen **Expr ::= Fact**, **Fact ::= Opd** steht.

Name	Produktion
p1:	Expr ::= Expr AddOpr Fact
p2:	Expr ::= Fact
p3:	Fact ::= Fact MulOpr Opd
p4:	Fact ::= Opd
p5:	Opd ::= '(' Expr ')'
p6:	Opd ::= Ident
p7:	AddOpr ::= '+'
p8:	AddOpr ::= '-'
p9:	MulOpr ::= '*'
p10:	MulOpr ::= '/'



Im Beispiel sind **AddOpr** und **MulOpr** links-assoziativ, weil ihre **Produktionen links-rekursiv** sind, d. h.  $a + b - c$  entspricht  $(a + b) - c$ .

© 2006 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 322

**Ziele:**

Struktur von Ausdrucksgrammatiken verstehen

**in der Vorlesung:**

Erläuterungen dazu am Beispiel:

- Präzedenz (Bindungsstärke) von Operatoren,
- Assoziativität von Operatoren,
- Kettenproduktion: genau ein Nichtterminal auf der rechten Seite.
- Zusammenhang zu Produktionen zeigen,
- Variation des Beispiels

**Verständnisfragen:**

- Wie ändert sich die Sprache, wenn Produktion p1 durch  $Expr ::= Fact '+' Fact$  ersetzt wird? Für welche Art von Operatoren wäre das sinnvoll?

## Schemata für Ausdrucksgrammatiken

**Ausdrucksgrammatiken** konstruiert man **schematisch**, sodass **strukturelle Eigenschaften** der Ausdrücke definiert werden:

**eine Präzedenzstufe**, binärer Operator, linksassoziativ:

$$A ::= A \text{ Opr } B$$

$$A ::= B$$

eine Präzedenzstufe, binärer Operator, **rechtsassoziativ**:

$$A ::= B \text{ Opr } A$$

$$A ::= B$$

eine Präzedenzstufe, **unärer Operator**, präfix:

$$A ::= \text{Opr } A$$

$$A ::= B$$

eine Präzedenzstufe, unärer Operator, **postfix**:

$$A ::= A \text{ Opr}$$

$$A ::= B$$

**Elementare Operanden**: nur aus dem Nichtterminal der höchsten Präzedenzstufe (sei hier H) abgeleitet:

$$H ::= \text{Ident}$$

**Geklammerte Ausdrücke**: nur aus dem Nichtterminal der höchsten Präzedenzstufe (sei hier H) abgeleitet; enthalten das Nichtterminal der niedrigsten Präzedenzstufe (sei hier A)

$$H ::= '( A )'$$

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 323

**Ziele:**

Schemata erkennen können

**in der Vorlesung:**

Erläuterungen dazu

**Übungsaufgaben:**

Anwenden der Schemata zum Verstehen von Ausdrucksgrammatiken

## Nützliche Grammatik-Konstrukte

**beliebig lange Folgen** von `stmt` z. B. in:

```
Block ::= '{' stmts '}'
stmts ::= stmts stmt
stmts ::=
```

Abkürzung für **beliebig lange Folgen**:

```
Block ::= '{' stmt* '}'
```

Die 2 Produktionen für `stmts` entfallen.

**nicht leere Folgen** von `stmt`:

```
stmts ::= stmts stmt
stmts ::= stmt
```

Entsprechend für **nicht-leere Folgen**:

```
Block ::= '{' stmt+ '}'
```

nicht-leere Folgen von `stmt`  
**getrennt durch ;**:

```
stmts ::= stmts ';' stmt
stmts ::= stmt
```

**Alternative Produktionen** für dasselbe  
Nichtterminal mit `|` zusammenfassen, z. B.

```
stmts ::= stmts ';' stmt |
        stmt
```

**Optionale Parameter** z. B. in:

```
Call ::= Name '(' ParamOpt ')'
ParamOpt ::= Parameter
ParamOpt ::=
```

**Optionales** mit `[ ]` klammern, z. B.

```
Call ::= Name '(' [ Parameter ] ')'
Die 2 Produktionen für ParamOpt entfallen.
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 324

### Ziele:

Produktionen für Folgen und Optionen verstehen

### in der Vorlesung:

Die Konstrukte und Abkürzungen werden an Beispielen erklärt:

- Beispiele für Sätze aus `Block` und `Call` ableiten.
- `*` und `+` mit regulären Ausdrücken vergleichen.



## Ausschnitte aus einer HTML-Grammatik

Die **Syntax von HTML** ist im Kalkül der „Document Type Definition (DTD)“ formal definiert. Man kann **Ausschnitte zu einigen Aspekten** von HTML in **KFGn** übertragen. Es folgen Beispiele dafür.

### Grundstruktur (ohne Attribute innerhalb von Tags):

```
HTMLDoc ::= '<html>' '<head>' HeadContent '</head>'
          '<body>' Block '</body>'
          '</html>'

Block ::= Paragraph | Table | List | Heading | ...

Paragraph ::= '<p>' Inline ['</p>']

Inline ::= ... Fließtext mit Auszeichnungen ohne Blockstrukturen ...

Flow ::= Block | Inline
```

```
Tabellen: Table ::= '<table>' Row* '</table>'
           Row  ::= '<tr>' Cell* '</tr>'
           Cell ::= '<td>' Flow '</td>'
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 325

### Ziele:

Produktionen aus einer größeren Grammatik verstehen

### in der Vorlesung:

An bekannten Beispielen aus HTML wird erklärt:

- Bedeutung der Produktionen,
- Verwendung von Grammatikkonstrukten,
- Klassifikation in Block und Inline und Zusammenfassung zu Flow,
- Tabellenstruktur,
- Hinweis auf DTD für HTML beim W3C.

# HTML-Grammatik: Listen und Attribute

## Listen:

```
List          ::= '<ol>' ListElement+ '</ol>' |  
              '<ul>' ListElement+ '</ul>'  
  
ListElement  ::= '<li>' Flow '</li>'
```

## Attribute in Anfangs-Tags.

In dieser Grammatik werden Tags weiter zerlegt:

```
AnfangsTag   ::= '<' TagName Attribute* '>'  
  
Attribute    ::= AttributeName '=' AttributeValue  
  
AttributeName ::= Identifier  
  
AttributeValue ::= StringLiteral | Identifier | Number | ...
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 326

### Ziele:

HTML-Konstrukte an Grammatikregeln verstehen

### in der Vorlesung:

An bekannten Beispielen aus HTML wird erklärt:

- Bedeutung der Produktionen,
- Verwendung von Grammatikkonstrukten,
- Struktur von Listen in HTML,
- Notation der Attribute in HTML.