

# Kontextfreie Grammatiken

# Was kann man mit kontextfreien Grammatiken anfangen?

Kontextfreie Grammatiken, kurz:

**KFGs**

werden zur Modellierung von

rekursiv definierten baumartigen Strukturen

eingesetzt.

KFGs werden unter Anderen für die Beschreibung von

- **Programmiersprachen** wie etwa C, Java, Pascal oder Python,
- bzw. **Datenaustauschformaten**, d.h. Sprachen (wie etwa HTML oder XML) als Schnittstelle zwischen Software-Werkzeugen,
- bzw. **Bäumen** zur Repräsentation strukturierter Daten (z.B. XML)

angewandt.

Eine kontextfreie Grammatik

$$G = (\Sigma, V, S, P)$$

besteht aus

- einer endlichen Menge  $\Sigma$  von **Terminalen** und einer endlichen Menge  $V$  von **Nichtterminalen** (oder **Variablen**).
  - ▶ Die Mengen  $\Sigma$  und  $V$  sind disjunkt, d.h.  $\Sigma \cap V = \emptyset$  gilt.
  - ▶ Die Menge  $W := \Sigma \cup V$  heißt **Vokabular**, die Elemente in  $W$  nennt man auch Symbole,
- einem Symbol  $S \in V$ , dem **Startsymbol** und
- einer endlichen Menge

$$P \subseteq V \times W^*$$

von **Produktionen**. Für eine Produktion  $(A, x) \in P$  schreiben wir  $A \rightarrow x$ .

Wir möchten alle „wohl-gebildeten“ arithmetische Ausdrücke beschreiben,

- die über den Zahlen 1, 2, 3 gebildet sind und
- die Operatoren +, −, · sowie Klammern (, ) benutzen.

Beispiele für wohl-gebildete arithmetische Ausdrücke sind

$$(1 + 3) \cdot (2 + 2 + 3) - 1$$

und

$$(1 + 3) \cdot ((2 + 2 + 3) - 1).$$

Wir betrachten die KFG  $G_{AA} := (\Sigma, V, S, P)$  mit

- Terminalalphabet  $\Sigma := \{1, 2, 3, +, -, \cdot, (, )\}$
- Nichtterminalalphabet  $V := \{Ausdruck, Operator\}$
- Startsymbol  $S := Ausdruck$
- und der Produktionsmenge

$$P := \left\{ \begin{array}{l} Ausdruck \rightarrow 1, \\ Ausdruck \rightarrow 2, \\ Ausdruck \rightarrow 3, \\ Ausdruck \rightarrow Ausdruck Operator Ausdruck, \\ Ausdruck \rightarrow ( Ausdruck ), \\ \\ Operator \rightarrow +, \\ Operator \rightarrow -, \\ Operator \rightarrow \cdot \end{array} \right\}$$

# Wir sparen Schreibarbeit

Wir fassen Zeilen, die das gleiche Nichtterminal auf der linken Seite des Pfeils aufweisen, zu einer einzigen Zeile zusammen.

Damit können wir die Produktionsmenge  $P$  auch kurz wie folgt beschreiben:

$$P = \left\{ \begin{array}{l} \textit{Ausdruck} \rightarrow 1 \mid 2 \mid 3 , \\ \textit{Ausdruck} \rightarrow \textit{Ausdruck Operator Ausdruck} \mid ( \textit{Ausdruck} ) , \\ \textit{Operator} \rightarrow + \mid - \mid \cdot \end{array} \right\}.$$

Die Produktion  $\textit{Ausdruck} \rightarrow \textit{Ausdruck Operator Ausdruck}$  können wir auffassen

- als **Strukturregel**, die besagt „Ein *Ausdruck* besteht aus einem *Ausdruck*, gefolgt von einem *Operator*, gefolgt von einem *Ausdruck* — oder als
- **Ersetzungsregel**, die besagt, dass das „Symbol *Ausdruck* durch das Wort *Ausdruck Operator Ausdruck* ersetzt werden kann.“

# Ableitungen

# Ein Ableitungsschritt

Sei  $G = (\Sigma, V, S, P)$  eine KFG.

Falls

$$A \rightarrow x$$

eine Produktion in  $P$  ist und  $u \in W^*$  und  $v \in W^*$  beliebige Worte über dem Vokabular  $W = \Sigma \cup V$  sind, so schreiben wir

$$uAv \Longrightarrow_G uxv \quad (\text{bzw. kurz: } uAv \Longrightarrow uxv)$$

und sagen, dass  $uAv$  in einem **Ableitungsschritt** zu  $uxv$  umgeformt werden kann.



# Mehrere Ableitungsschritte: Ableitungen

Eine **Ableitung** ist eine endliche Folge von hintereinander angewendeten Ableitungsschritten.

Für Worte  $w \in W^*$  und  $w' \in W^*$  schreiben wir

$$w \xRightarrow{*}_G w' \quad (\text{bzw. kurz: } w \xRightarrow{*} w'),$$

um auszusagen, dass es eine endliche Folge von Ableitungsschritten gibt, die  $w$  zu  $w'$  umformt.

*Spezialfall:* Diese Folge darf auch aus 0 Ableitungsschritten bestehen, d.h. f.a.  $w \in W^*$  gilt:

$$w \xRightarrow{*}_G w.$$

# Ableitungen: Ein Beispiel

Für die Grammatik  $G_{AA} = (\Sigma, V, S, P)$  arithmetischer Ausdrücke gilt

$$\text{Ausdruck} \xRightarrow{*} (1 + 3) \cdot 2,$$

denn

$$P = \left\{ \begin{array}{l} \text{Ausdruck} \rightarrow 1 \mid 2 \mid 3, \\ \text{Ausdruck} \rightarrow \text{Ausdruck Operator Ausdruck} \mid (\text{Ausdruck}), \\ \text{Operator} \rightarrow + \mid - \mid \cdot \end{array} \right\}.$$

und

$$\begin{aligned} \text{Ausdruck} &\Longrightarrow \text{Ausdruck Operator Ausdruck} \\ &\Longrightarrow (\text{Ausdruck}) \text{ Operator Ausdruck} \\ &\Longrightarrow (\text{Ausdruck Operator Ausdruck}) \text{ Operator Ausdruck} \\ &\Longrightarrow (\text{Ausdruck} + \text{Ausdruck}) \text{ Operator Ausdruck} \\ &\Longrightarrow (\text{Ausdruck} + \text{Ausdruck}) \cdot \text{Ausdruck} \\ &\Longrightarrow (1 + \text{Ausdruck}) \cdot \text{Ausdruck} \\ &\Longrightarrow (1 + 3) \cdot \text{Ausdruck} \\ &\Longrightarrow (1 + 3) \cdot 2 \end{aligned}$$

# Kontextfreie Sprachen

# Kontextfreie Sprachen $L(G)$

Sei  $G = (\Sigma, V, S, P)$  eine KFG.

- (a) Die **von  $G$  erzeugte Sprache  $L(G)$**  ist die Menge aller Worte über dem Terminalalphabet  $\Sigma$ , die aus dem Startsymbol  $S$  abgeleitet werden können.  
D.h.:

$$L(G) := \{w \in \Sigma^* : S \xRightarrow{*}_G w\}.$$

- (b) Eine Sprache  $L$  heißt genau dann **kontextfrei**, wenn es eine kontextfreie Grammatik  $G$  gibt mit

$$L = L(G).$$

**Achtung:**  $L(G)$  ist eine Teilmenge von  $\Sigma^* \implies$

In den Worten aus  $L(G)$  kommen keine Nichtterminale vor!

# Die von $G_{AA}$ erzeugte Sprache

Die Sprache  $L(G_{AA})$  besteht aus allen über den Zahlen 1, 2, 3, den Operatoren +, -, · und den Klammersymbolen (, ) wohl-geformten arithmetischen Ausdrücken.

$$P = \left\{ \begin{array}{l} \text{Ausdruck} \rightarrow 1 \mid 2 \mid 3 \text{ ,} \\ \text{Ausdruck} \rightarrow \text{Ausdruck Operator Ausdruck} \mid (\text{Ausdruck}) \text{ ,} \\ \text{Operator} \rightarrow + \mid - \mid \cdot \end{array} \right\}.$$

Welche Worte gehören zu  $L(G_{AA})$ ?

- ? 3,
- ? (3 + 1),
- ? (),
- ? (3 + 1,
- ? 1 + 2 · 3,
- ? (3 + 1) · (2 + 2 + 3) - 1,
- ? 2 · ((3 + 1) · (2 + 2 + 3) - 1),
- ? ((3 + 1)),
- ? Ausdruck Operator Ausdruck

# Wohlgeformte Klammerausdrücke

Sei  $D$  die Sprache aller wohl-geformten Klammerausdrücke über  $\Sigma = \{ (, ) \}$ .

1. **Wohl-geformte Klammerausdrücke** sind

▶  $()$ ,  $((()))$ ,  $((()))()((()))$ ,  $((()))$

2. **Nicht wohl-geformt** sind

▶  $((()))$ ,  $((())$

Es ist  $D = L(G)$  für die kontextfreie Grammatik  $G = (\Sigma, \{S\}, S, P)$  mit den Produktionen

$$S \rightarrow \epsilon \mid (S) \mid SS$$

Eine äquivalente Lösung erhält man mit den Produktionen

$$S \rightarrow \epsilon \mid (S)S.$$

Baue einen wohl-geformten Klammerausdruck **von links nach rechts** auf:

Die Produktion  $S \rightarrow (S)S$  fügt die äußeren Klammern für den „linksten“ Klammerausdruck ein.

# KFGs und Programmiersprachen

# Ein Fragment von Pascal

Wir beschreiben einen (allerdings sehr kleinen) Ausschnitt von Pascal durch eine kontextfreie Grammatik.

- Wir benutzen das Alphabet  $\Sigma = \{a, \dots, z, ;, :=, \text{begin}, \text{end}, \text{while}, \text{do}\}$  und
- die Variablen  $S$ , `statements`, `statement`, `assign-statement`, `while-statement`, `variable`, `boolean`, `expression`.
- `variable`, `boolean` und `expression` sind im Folgenden nicht weiter ausgeführt.

$S$	→	<code>begin statements end</code>
<code>statements</code>	→	<code>statement   statement ; statements</code>
<code>statement</code>	→	<code>assign-statement   while-statement</code>
<code>assign-statement</code>	→	<code>variable := expression</code>
<code>while-statement</code>	→	<code>while boolean do statements</code>



Lassen sich die **syntaktisch korrekten** Programme einer modernen Programmiersprache durch eine kontextfreie Sprache definieren?

- **1. Antwort: Nein.** In Pascal muss zum Beispiel sichergestellt werden, dass Anzahl und Typen von formalen und aktuellen Parameter übereinstimmen.
  - ▶ Die Sprache  $\{ww : w \in \Sigma^*\}$  wird sich als **nicht** kontextfrei herausstellen.
- **2. Antwort: Im Wesentlichen ja,** wenn man „Details“ wie Typ-Deklarationen und Typ-Überprüfungen ausklammert:
  - ▶ Man beschreibt die Syntax durch eine kontextfreie Grammatik, die alle syntaktisch korrekten Programme erzeugt.
  - ▶ Allerdings werden auch syntaktisch inkorrekte Programme (z.B. aufgrund von Typ-Inkonsistenzen) erzeugt.

# Eine „kontextfreie“ Grammatik für Python

Eine Grammatik für Python wird beschrieben in

<https://docs.python.org/3/reference/grammar.html>

(zuletzt besucht am 30.01.2019)

Die Grammatik ist „*im Wesentlichen*“ kontextfrei: In der Beschreibung der Grammatik werden zum Beispiel folgende Notationen als hilfreiche Abkürzungen verwendet:

- der Kleene-Stern und das Kleene-Plus,
- eckige Klammern [ ... ] für optionale Strings

Lässt man einen „**Lexer**“ in einem Vorverarbeitungsschritt über das Anwender-Programm laufen, um

- Einrückungen zu „verstehen“ (NEWLINE, INDENT, DEDENT),
- Schlüsselwörter (if, for, else, ...) zu entdecken,
- Kommentare zu entfernen, ...

dann hat der **Parser** im eigentlichen Verarbeitungsschritt nur noch ein kontextfreies Sprachenproblem zu lösen.

# Die Backus-Naur-Form und die Java-Syntax

Die Backus-Naur-Form (BNF) wird zur Formalisierung der Syntax von Programmiersprachen genutzt.

- BNF ist ein „Dialekt“ der kontextfreien Grammatiken. Produktionen der Form

$$X \rightarrow aYb$$

(mit  $X, Y \in V$  und  $a, b \in \Sigma$ ) werden in BNF notiert als

$$\langle X \rangle ::= a \langle Y \rangle b$$

- **Beispiel:** Eine Beschreibung der Syntax von **Java** in einer BNF-Variante wird beschrieben in

<https://docs.oracle.com/javase/specs/jls/se11/html/index.html>  
(zuletzt besucht am 05.02.2019)

Für kontextfreie Sprachen ist eine effiziente **Syntaxanalyse** möglich.

**Frage:** Was ist eine Syntaxanalyse?

**Antwort:** Die Bestimmung einer Ableitung bzw. eines Ableitungsbaums.

Und was ist ein Ableitungsbaum?

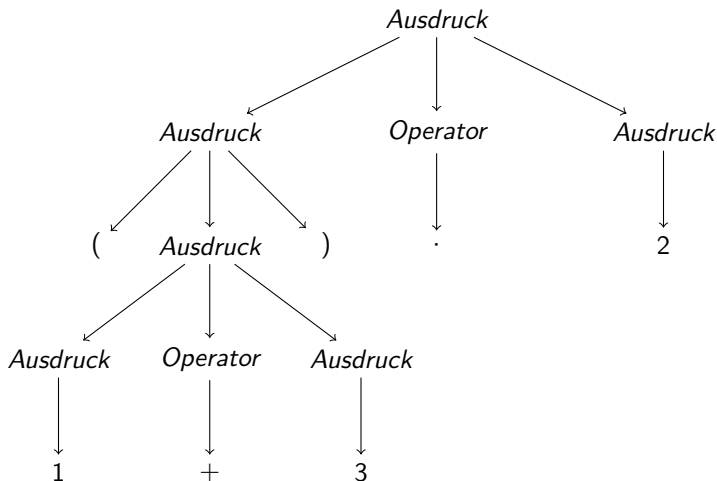
# Ableitungsbäume

Ableitungen lassen sich am besten mit **Ableitungsbäumen** veranschaulichen.

Betrachte dazu in  $G_{AA}$  die Ableitung

$$\begin{aligned} \text{Ausdruck} &\implies \text{Ausdruck Operator Ausdruck} \\ &\implies ( \text{Ausdruck} ) \text{ Operator Ausdruck} \\ &\implies ( \text{Ausdruck Operator Ausdruck} ) \text{ Operator Ausdruck} \\ &\implies ( \text{Ausdruck} + \text{Ausdruck} ) \text{ Operator Ausdruck} \\ &\implies ( \text{Ausdruck} + \text{Ausdruck} ) \cdot \text{Ausdruck} \\ &\implies ( 1 + \text{Ausdruck} ) \cdot \text{Ausdruck} \\ &\implies ( 1 + 3 ) \cdot \text{Ausdruck} \\ &\implies ( 1 + 3 ) \cdot 2 , \end{aligned}$$

Diese Ableitung hat den folgenden **Ableitungsbaum**:



Beachte: Ein Ableitungsbaum kann mehrere Ableitungen repräsentieren.

Sei  $G = (\Sigma, V, S, P)$  eine KFG und sei  $w \in L(G)$ .

Jede Ableitung

$$S \xRightarrow{*}_G w$$

lässt sich als gerichteter Baum darstellen, bei dem

1. jeder Knoten mit einem Symbol aus  $\Sigma \cup V \cup \{\varepsilon\}$  markiert ist und
2. die Kinder jedes Knotens eine festgelegte Reihenfolge haben.
  - ▶ In der Zeichnung eines Ableitungsbaums werden von links nach rechts zunächst das „erste Kind“ dargestellt, dann das zweite, dritte etc.
  - ▶ Der Ableitungsbaum ist also ein geordneter Baum.

3. Die Wurzel des Baums ist mit dem Startsymbol  $S$  markiert.
4. Jeder Knoten mit seinen Kindern repräsentiert die Anwendung einer Produktion aus  $P$ , also einer Produktion

$$A \rightarrow x \text{ mit } A \in V, x \in (V \cup \Sigma)^*.$$

Die Anwendung der Produktion wird im Ableitungsbaum repräsentiert durch einen Knoten  $v$ , der mit dem Symbol  $A$  markiert ist.

- ▶ Wenn  $x \in (V \cup \Sigma)^+$ , dann hat  $v$  genau  $|x|$  viele Kinder, so dass das  $i$ -te Kind mit dem  $i$ -ten Symbol von  $x$  markiert ist (f.a.  $i \in \{1, \dots, |x|\}$ ).
- ▶ Wenn  $x = \varepsilon$ , dann hat  $v$  genau ein Kind, das mit  $\varepsilon$  markiert ist.



1. Die Bedeutung eines syntaktisch korrekten Programms  $p$  wird durch **den** Ableitungsbaum von  $p$  bestimmt.
2. Und wenn es **mehrere** Ableitungsbäume für  $p$  gibt?

Die Spezifikation der Programmiersprache – also die KFG – muss garantieren, dass es für alle syntaktisch korrekten Programme nur **einen** Ableitungsbaum gibt.  
*Solche KFGs heißen **eindeutig**.*

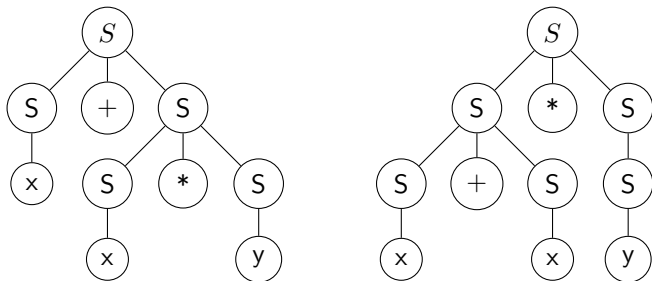
# Arithmetische Ausdrücke: Eindeutige Grammatiken (1/2)

Die Produktionen  $S \rightarrow S + S \mid S * S \mid (S) \mid x \mid y$  definieren arithmetische Ausdrücke auf **mehrdeutige** Art und Weise.

**Denn** das Wort

$$x + x * y$$

hat die beiden Ableitungsbäume:



Der erste Baum führt zur Auswertung  $x + (x * y)$ , der zweite zu  $(x + x) * y$ .

Wir brauchen eine eindeutige Grammatik!

Die neue Grammatik  $G$  legt fest, dass Multiplikation stärker "bindet" als Addition.

- $V := \{S, T, F\}$ :  $S$  ist das Startsymbol,  $T$  „erzeugt“ Terme,  $F$  „erzeugt“ Faktoren.
- Die Produktionen von  $G$  haben die Form

$$S \rightarrow S + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (S) \mid x \mid y.$$

Warum ist diese Grammatik eindeutig?

Eine Antwort wird in der Veranstaltung „**Theoretische Informatik**“ gegeben.

# Klammerausdrücke: Eine eindeutige Grammatik

Die „**Klammersprache**“  $L$  wird durch die Produktionen

$$S \rightarrow \epsilon \mid SS \mid (S).$$

erzeugt.

1. Die Grammatik ist **mehrdeutig**, denn zum Beispiel besitzt das leere Wort mehrere Ableitungsbäume. Welche?
2. Wir erhalten eine **eindeutige** Grammatik mit den Produktionen

$$S \rightarrow \epsilon \mid (S)S.$$

- ▶ Ein Klammerausdruck wird **zwangsweise** von links nach rechts aufgebaut.
- ▶ Die Produktion  $S \rightarrow (S)S$  fügt die äußeren Klammern für den linkesten Klammerausdruck ein.

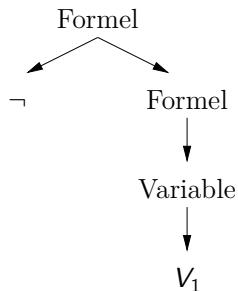
Wir konstruieren eine KFG

$$G_{AL} = (\Sigma, V, S, P),$$

deren Sprache  $L(G_{AL})$  gerade die Menge aller aussagenlogischen Formeln ist, in denen nur Variablen aus  $\{V_0, V_1, V_2\}$  vorkommen.

- **Terminale:**  $\Sigma := \{ V_0, V_1, V_2, \mathbf{0}, \mathbf{1}, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, \oplus, (, ) \}$ ,
- **Nichtterminale:**  $V := \{ \text{Formel}, \text{Variable}, \text{Junktor} \}$ ,
- **Startsymbol:**  $S := \text{Formel}$ ,
- **Produktionsmenge**  $P :=$

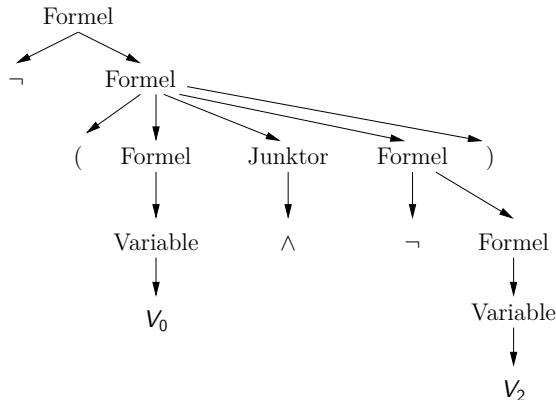
$$\left\{ \begin{array}{l} \text{Formel} \rightarrow \mathbf{0} \mid \mathbf{1} \mid \text{Variable} , \\ \text{Formel} \rightarrow \neg \text{Formel} \mid ( \text{Formel} \text{ Junktor } \text{Formel} ) , \\ \text{Variable} \rightarrow V_0 \mid V_1 \mid V_2 , \\ \text{Junktor} \rightarrow \wedge \mid \vee \mid \rightarrow \mid \leftrightarrow \mid \oplus \end{array} \right\}.$$



Der Ableitungsbaum repräsentiert die Ableitung

$$\begin{aligned} \textit{Formel} &\Longrightarrow \neg \textit{Formel} \\ &\Longrightarrow \neg \textit{Variable} \\ &\Longrightarrow \neg V_1 \end{aligned}$$

Das durch diese(n) Ableitung(sbaum) erzeugte Wort in der Sprache  $L(G_{AL})$  ist die Formel  $\neg V_1$ .

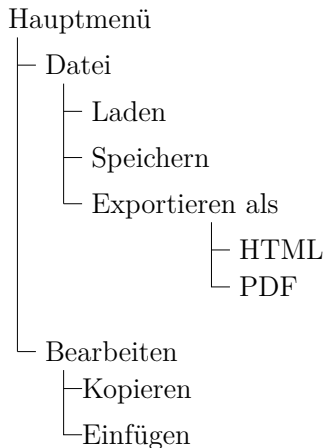


Dieser Ableitungsbaum repräsentiert die Ableitung der Formel  $\neg(V_0 \wedge \neg V_2)$

$$\begin{aligned}
 \text{Formel} &\implies \neg \text{Formel} \implies \neg(\text{Formel Junktor Formel}) \\
 &\implies \neg(\text{Variable Junktor Formel}) \implies \neg(V_0 \text{ Junktor Formel}) \\
 &\implies \neg(V_0 \wedge \text{Formel}) \implies \neg(V_0 \wedge \neg \text{Formel}) \implies \neg(V_0 \wedge \neg \text{Variable}) \\
 &\implies \neg(V_0 \wedge \neg V_2).
 \end{aligned}$$

Ein **Menü** besteht aus einem Menünamen und einer Folge von Einträgen:  
Ein Eintrag besteht aus einem Operationsnamen oder selbst wieder einem Menü.

*Beispiel:*



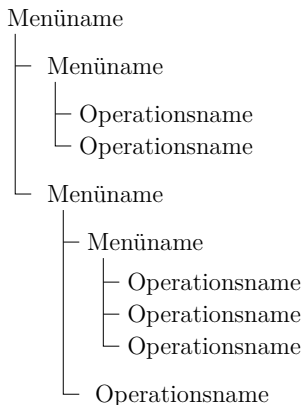


Ein **Menü** besteht aus einem Menünamen und einer Folge von Einträgen: Ein Eintrag besteht aus einem Operationsnamen oder selbst wieder einem Menü.

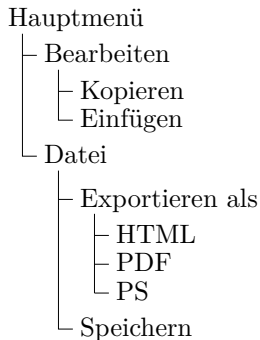
Benutze z.B. die Grammatik  $G_{\text{Menü}} = (\Sigma, V, S, P)$  mit

- $\Sigma := \{ \text{Menüname, Operationsname} \},$
- $V := \{ \text{Menü, Eintragsfolge, Eintrag} \},$
- $S := \text{Menü},$
- $P := \left\{ \begin{array}{ll} \text{Menü} & \rightarrow \text{Menüname Eintragsfolge} , \\ \text{Eintragsfolge} & \rightarrow \text{Eintrag} \mid \text{Eintrag Eintragsfolge} , \\ \text{Eintrag} & \rightarrow \text{Operationsname} \mid \text{Menü} \end{array} \right\}.$

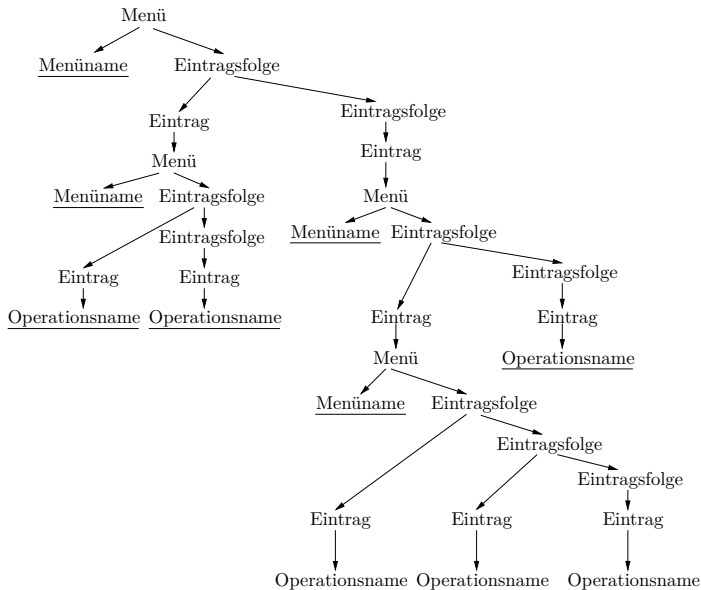
## Die Struktur eines Menüs:



## Beispiel für Namen:



Und wie sieht der Ableitungsbaum für das Beispiel aus?



## **HTML** (HyperText Markup Language)

ist ein Format zur Beschreibung von verzweigten Dokumenten im Internet.

Uns interessiert hier der HTML-Code zur Erzeugung von HTML-Tabellen.

# HTML-Tabellen: Ein Beispiel

Die HTML-Tabelle

Tag	Zeit	Raum
Di	8:00-10:00	Hörsaal VI
Do	8:00-10:00	Hörsaal VI

besitzt den HTML-Code:

```
<table>
  <tr>
    <td> Tag </td>
    <td> Zeit </td>
    <td> Raum </td>
  </tr>
  <tr>
    <td> Di </td>
    <td> 8:00-10:00 </td>
    <td> Hörsaal VI </td>
  </tr>
  <tr>
    <td> Do </td>
    <td> 8:00-10:00 </td>
    <td> Hörsaal VI </td>
  </tr>
</table>
```

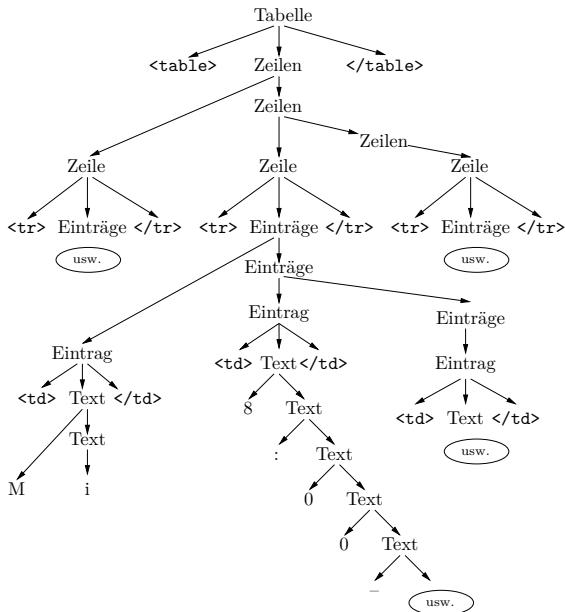
Die Symbole `<table>` und `</table>`, `<tr>` und `</tr>` bzw. `<td>` und `</td>` stehen für Anfang und Ende einer **Tabelle**, für Anfang und Ende einer **Zeile der Tabelle** bzw. Anfang und Ende eines **Eintrags in einer Zelle der Tabelle**.

# Eine KFG zur Erzeugung von HTML-Tabellen

Wir konstruieren eine Grammatik  $G_{\text{HTML}} = (\Sigma, V, S, P)$ , so dass  $G_{\text{HTML}}$  genau die (möglicherweise geschachtelten) HTML-Tabellen erzeugt.

- $\Sigma := \{ \langle \text{table} \rangle, \langle / \text{table} \rangle, \langle \text{tr} \rangle, \langle / \text{tr} \rangle, \langle \text{td} \rangle, \langle / \text{td} \rangle, a, \dots, z, A, \dots, Z, 0, 1, \dots, 9, :, -, \_ \}$
- $V := \{ \text{Tabelle}, \text{Zeilen}, \text{Zeile}, \text{Einträge}, \text{Eintrag}, \text{Text} \}$
- $S := \text{Tabelle}$
- $P := \{ \begin{array}{l} \text{Tabelle} \rightarrow \langle \text{table} \rangle \text{Zeilen} \langle / \text{table} \rangle, \\ \text{Zeilen} \rightarrow \text{Zeile} \mid \text{Zeile Zeilen}, \\ \text{Zeile} \rightarrow \langle \text{tr} \rangle \text{Einträge} \langle / \text{tr} \rangle, \\ \text{Einträge} \rightarrow \text{Eintrag} \mid \text{Eintrag Einträge}, \\ \text{Eintrag} \rightarrow \langle \text{td} \rangle \text{Text} \langle / \text{td} \rangle \mid \langle \text{td} \rangle \text{Tabelle} \langle / \text{td} \rangle, \\ \text{Text} \rightarrow a \text{Text} \mid b \text{Text} \mid \dots \mid z \text{Text} \mid A \text{Text} \mid B \text{Text} \mid \dots \mid Z \text{Text} \\ \text{Text} \rightarrow 0 \text{Text} \mid \dots \mid 9 \text{Text} \mid : \text{Text} \mid - \text{Text} \mid \_ \text{Text}, \\ \text{Text} \rightarrow a \mid b \mid \dots \mid z \mid A \mid B \mid \dots \mid Z \mid 0 \mid \dots \mid 9 \mid : \mid - \mid \_ \end{array} \}.$

# Der Ableitungsbaum der HTML-Tabelle



# Reguläre und kontextfreie Sprachen



Kann jede reguläre Sprache von einer kontextfreien Grammatik erzeugt werden?

Sei  $G = (\Sigma, V, S, P)$  eine kontextfreie Grammatik.

(a)  $G$  heißt **rechtsregulär**, wenn alle Produktionen die Form

$$A \rightarrow aB \text{ oder } A \rightarrow \varepsilon$$

für Variable  $A, B \in V$  und ein Terminal  $a \in \Sigma$  besitzen.

(b)  $G$  heißt **linksregulär**, wenn alle Produktionen die Form

$$A \rightarrow Ba \text{ oder } A \rightarrow \varepsilon$$

für Variable  $A, B \in V$  und ein Terminal  $a \in \Sigma$  besitzen.

# Reguläre und kontextfreie Sprachen

- (a) Jede reguläre Sprache wird von einer rechtsregulären Grammatik erzeugt.
  - ▶ Siehe **Tafel**.
- (b) Die Klasse der kontextfreien Sprachen ist eine **echte** Obermenge der regulären Sprachen, denn

$$L = \{ a^n b^n : n \in \mathbb{N} \}$$

- ▶ ist kontextfrei: Die kontextfreie Grammatik  $G = (\{a, b\}, \{S\}, S, P)$  mit den Produktionen

$$S \rightarrow aSb \mid \epsilon$$

erzeugt  $L$ .

- ▶ aber  $L$  ist nicht regulär.

- (c) Eine Sprache  $L$  ist genau dann regulär, wenn es eine links- oder rechtsreguläre Grammatik  $G$  gibt mit

$$L = L(G).$$

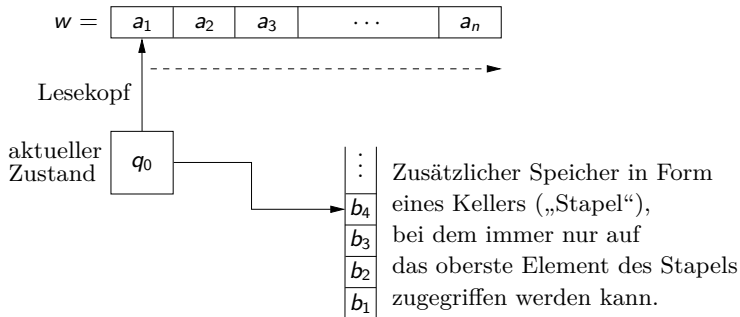
*Details* in der Veranstaltung „**Theoretische Informatik**“.

- (a) KFGs werden eingesetzt, um  
rekursiv definierte Strukturen  
zu modellieren.
- (b) Es gibt z. B. wichtige Anwendungen in der Spezifikation von
  - ▶ Programmiersprachen,
  - ▶ HTML,
  - ▶ XML.
- (c) Der Ableitungsbaum legt die Semantik eines syntaktisch korrekten Programms fest.

# Ausblick

# Kellerautomaten

Schematische Darstellung der Verarbeitung eines Eingabeworts durch einen **Kellerautomaten**:



In der „Theoretischen Informatik“ wird gezeigt, dass kontextfreie Grammatiken und nichtdeterministische Kellerautomaten genau die Klasse der kontextfreien Sprachen erzeugen, bzw. akzeptieren.

Eine Sprache  $L$  heißt **deterministisch kontextfrei**, wenn  $L$  von einem deterministischen Kellerautomaten akzeptiert wird.

- Die Sprache

$$L_1 = \{ a^n b^n : n \in \mathbb{N} \}$$

ist deterministisch kontextfrei,

- die Sprache

$$L_2 = \{ a^n b^n c^m : n, m \in \mathbb{N} \} \cup \{ a^m b^n c^n : n, m \in \mathbb{N} \}$$

ist kontextfrei, aber nicht deterministisch kontextfrei.

*Details* in der Veranstaltung „**Theoretische Informatik**“.

# Das Wort-Problem

Das Wortproblem:

„Erzeugt eine KFG  $G = (\Sigma, V, S, P)$  das Wort  $w$ ?“

spricht: „Ist das Programm  $w$  syntaktisch korrekt?“

Ein Parser muss das Wortproblem für jedes Eingabeprogramm  $w$  lösen, deshalb kommt dem Wortproblem eine große Bedeutung zu.

- (a) Der CYK-Algorithmus löst das Wortproblem für eine Grammatik  $G = (\Sigma, V, S, P)$  in Zeit proportional zu  $|w|^3 \cdot |P|$ .
- (b) **Kubische** Laufzeit ist völlig inakzeptabel, das Wortproblem für **deterministisch kontextfreie Sprachen** ist hingegen in **Linearzeit** lösbar.
  - ▶ Die Syntax vieler Programmiersprachen wird deshalb von deterministisch kontextfreien Grammatiken definiert.

*Details* in der Veranstaltung „**Theoretische Informatik**“.

# Eine nicht-kontextfreie Sprache

(a) Die Sprache

$$L = \{a^n b^n c^n : n \in \mathbb{N}\}$$

ist **nicht** kontextfrei,

(b) aber ihre Komplement-Sprache

$$\bar{L} = \{w \in \{a, b, c\}^* : w \notin L\}$$

ist kontextfrei!

*Details* in der Veranstaltung „**Theoretische Informatik**“.

Kontextfreie Sprachen sind nicht unter Komplementbildung abgeschlossen!