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.

Kontextfreie Grammatiken: Die formale Definition

Eine kontextfreie Grammatik

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

besteht aus

- einer endlichen Menge Σ von **Terminalen** und einer endlichen Menge V von **Nichtterminalen** (oder **Variablen**).
  - Die Mengen Σ 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 $A \to x$.

Arithmetische Ausdrücke (1/2)

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

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

Beispiele für wohl-gebildete arithmetische Ausdrücke sind

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

und

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

Arithmetische Ausdrücke (2/2)

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

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

$$
\begin{aligned}
P := \quad \{ \quad & \mathit{Ausdruck} \rightarrow 1 \,, \\
& \mathit{Ausdruck} \rightarrow 2 \,, \\
& \mathit{Ausdruck} \rightarrow 3 \,, \\
& \mathit{Ausdruck} \rightarrow \mathit{Ausdruck}\ \mathit{Operator}\ \mathit{Ausdruck} \,, \\
& \mathit{Ausdruck} \rightarrow (\ \mathit{Ausdruck}\ ) \,, \\
& \\
& \mathit{Operator} \rightarrow + \,, \\
& \mathit{Operator} \rightarrow - \,, \\
& \mathit{Operatur} \rightarrow \cdot \quad \}
\end{aligned}
$$

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:

$$
\begin{array}{lllll}
P = \{ & \textit{Ausdruck} & \to & 1 \mid 2 \mid 3 & , \\
 & \textit{Ausdruck} & \to & \textit{Ausdruck Operator Ausdruck} \mid (\ \textit{Ausdruck}\ ) & , \\
 & \textit{Operator} & \to & + \mid - \mid \cdot & \}.
\end{array}
$$

Die Produktion $\textit{Ausdruck} \to \textit{Ausdruck Operator Ausdruck}$ 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

$$A \to x$$

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

$$uAv \implies_G uxv \quad \text{(bzw. kurz:} \quad uAv \implies uxv\text{)}$$

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

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

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:

$$w \overset{*}{\Longrightarrow}_G w.$$

Ableitungen: Ein Beispiel

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

$$\mathit{Ausdruck} \overset{*}{\Longrightarrow} (1 + 3) \cdot 2,$$

denn

$$\begin{array}{lllll} P = \{ & \mathit{Ausdruck} & \to & 1 \mid 2 \mid 3 & , \\ & \mathit{Ausdruck} & \to & \mathit{Ausdruck}\ \mathit{Operator}\ \mathit{Ausdruck} \mid (\ \mathit{Ausdruck}\ ) & , \\ & \mathit{Operator} & \to & + \mid - \mid \cdot & \}. \end{array}$$

und

$$\begin{array}{rcl} \mathit{Ausdruck} & \Longrightarrow & \mathit{Ausdruck}\ \mathit{Operator}\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ \mathit{Ausdruck}\ )\ \mathit{Operator}\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ \mathit{Ausdruck}\ \mathit{Operator}\ \mathit{Ausdruck}\ )\ \mathit{Operator}\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ \mathit{Ausdruck}\ +\ \mathit{Ausdruck}\ )\ \mathit{Operator}\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ \mathit{Ausdruck}\ +\ \mathit{Ausdruck}\ )\ \cdot\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ 1\ +\ \mathit{Ausdruck}\ )\ \cdot\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ 1\ +\ 3\ )\ \cdot\ \mathit{Ausdruck} \\ & \Longrightarrow & (\ 1\ +\ 3\ )\ \cdot\ 2 \end{array}$$

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 Σ, die aus dem Startsymbol S abgeleitet werden können. D.h.:

$$L(G) := \left\{ w \in \Sigma^* \quad : \quad S \overset{*}{\Longrightarrow}_G w \right\}.$$

(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 $\Sigma^* \Longrightarrow$
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 $+, -, \cdot$ und den Klammersymbolen $(,)$ wohl-geformten arithmetischen Ausdrücken.

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

Welche Worte gehören zu $L(G_{AA})$?

- ? 3,
- ? $(3+1)$,
- ? $()$,
- ? $(3+1$,
- ? $1+2\cdot 3$,
- ? $(3+1)\cdot(2+2+3)-1$,
- ? $2\cdot((3+1)\cdot(2+2+3)-1)$,
- ? $((3+1))$,
- ? *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

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

Eine äquivalente Lösung erhält man mit den Produktionen

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

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

Die Produktion $S \to (S)S$ fügt die äußeren Klammern für den „linkesten“ 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 $\Sigma = \{\texttt{a}, \ldots, \texttt{z}, \texttt{;}, :=, \texttt{begin}, \texttt{end}, \texttt{while}, \texttt{do}\}$ und
- die Variablen S, statements, statement, assign-statement, while-statement, variable, boolean, expression.
- variable, boolean und expression sind im Folgenden nicht weiter ausgeführt.

$$
\begin{array}{rcl}
S & \to & \texttt{begin}\ \text{statements}\ \texttt{end} \\
\text{statements} & \to & \text{statement} \mid \text{statement}\ \texttt{;}\ \text{statements} \\
\text{statement} & \to & \text{assign-statement} \mid \text{while-statement} \\
\text{assign-statement} & \to & \text{variable} := \text{expression} \\
\text{while-statement} & \to & \texttt{while}\ \text{boolean}\ \texttt{do}\ \text{statements}
\end{array}
$$

Programmiersprachen und kontextfreie Sprachen

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

$$X \to aYb$$

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

```
<X> ::= a <Y> 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 und Ableitungsbäume (1/2)

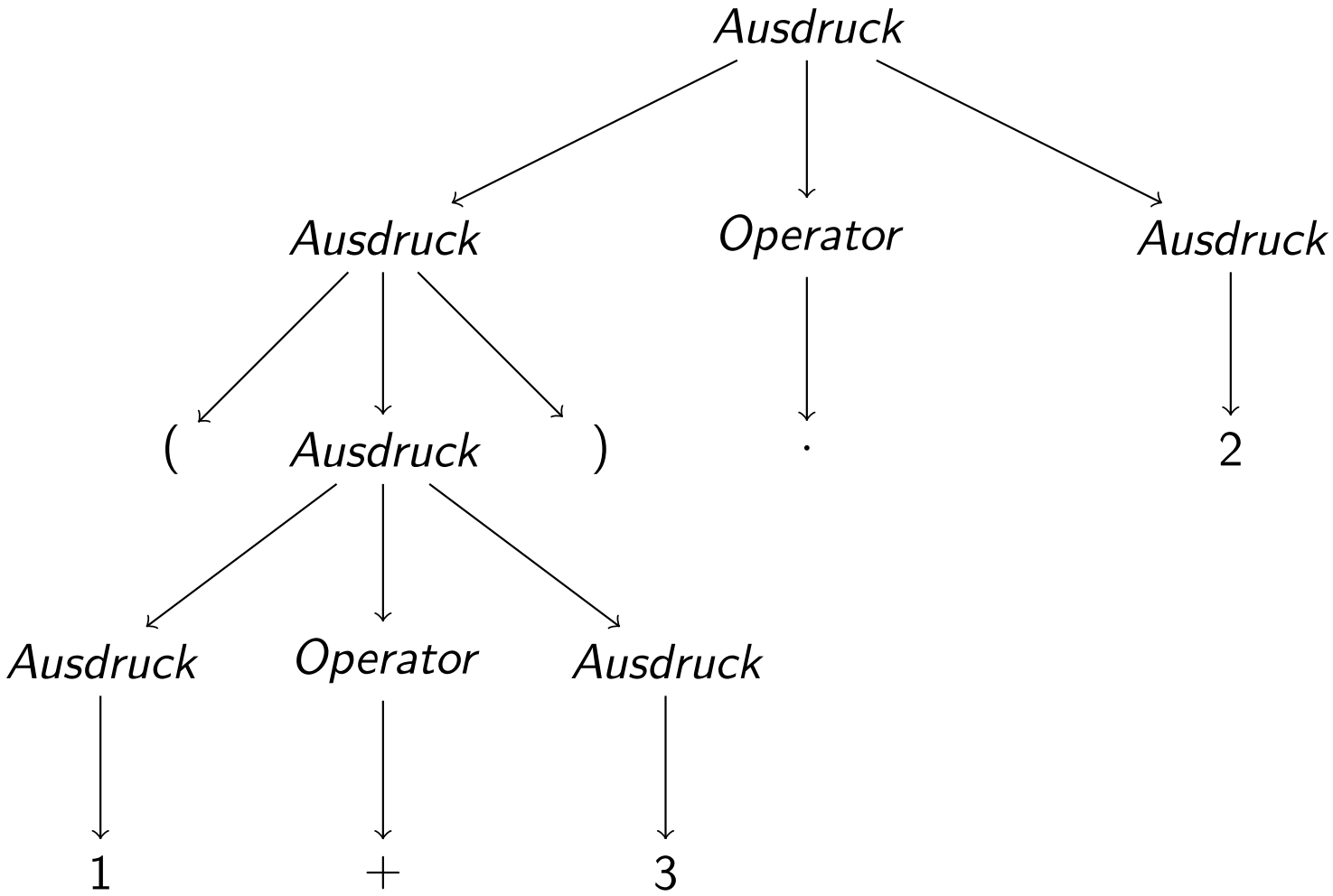
Ableitungen lassen sich am besten mit **Ableitungsbäumen** veranschaulichen.

Betrachte dazu in G_{AA} die Ableitung

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

Ableitungen und Ableitungsbäume (2/2)

Diese Ableitung hat den folgenden **Ableitungsbaum**:



Beachte: Ein Ableitungsbaum kann mehrere Ableitungen repräsentieren.

Ableitungsbäume: Die formale Definition (1/2)

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

Jede Ableitung

$$S \overset{*}{\Longrightarrow}_G w$$

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.

Ableitungsbäume: Die formale Definition (2/2)

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

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

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. $i \in \{1, \ldots, |x|\}$).
- Wenn $x = \varepsilon$, dann hat v genau ein Kind, das mit ε markiert ist.

Syntaxanalyse

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**.*

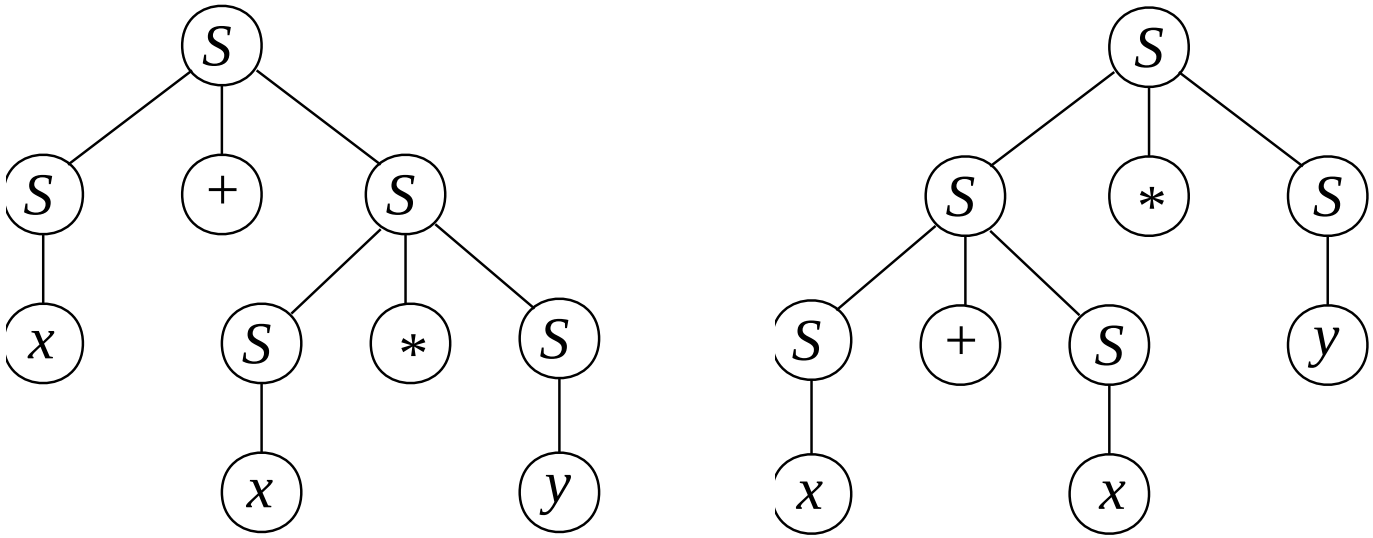
Arithmetische Ausdrücke: Eindeutige Grammatiken (1/2)

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!

Arithmetische Ausdrücke: Eindeutige Grammatiken (2/2)

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

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T * F \mid F \\ F &\to (S) \mid x \mid y. \end{aligned}$$

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

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

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

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

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

Die Aussagenlogik als kontextfreie Sprache (1/3)

Wir konstruieren eine KFG

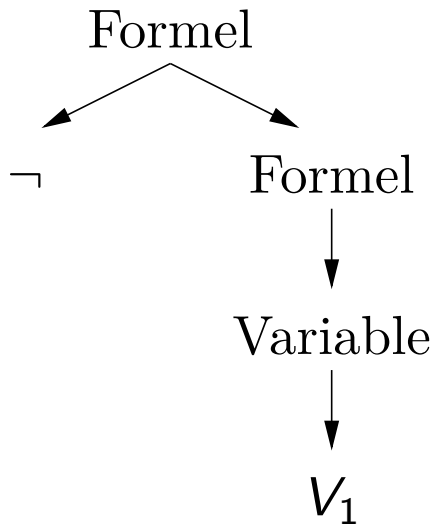
$$G_{\mathrm{AL}} \;=\; (\Sigma, V, S, P),$$

deren Sprache $L(G_{\mathrm{AL}})$ 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:** $V := \{\; \mathit{Formel},\; \mathit{Variable},\; \mathit{Junktor}\;\}$,
- **Startsymbol:** $S := \mathit{Formel}$,
- **Produktionsmenge** $P :=$

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

Die Aussagenlogik als kontextfreie Sprache (2/3)

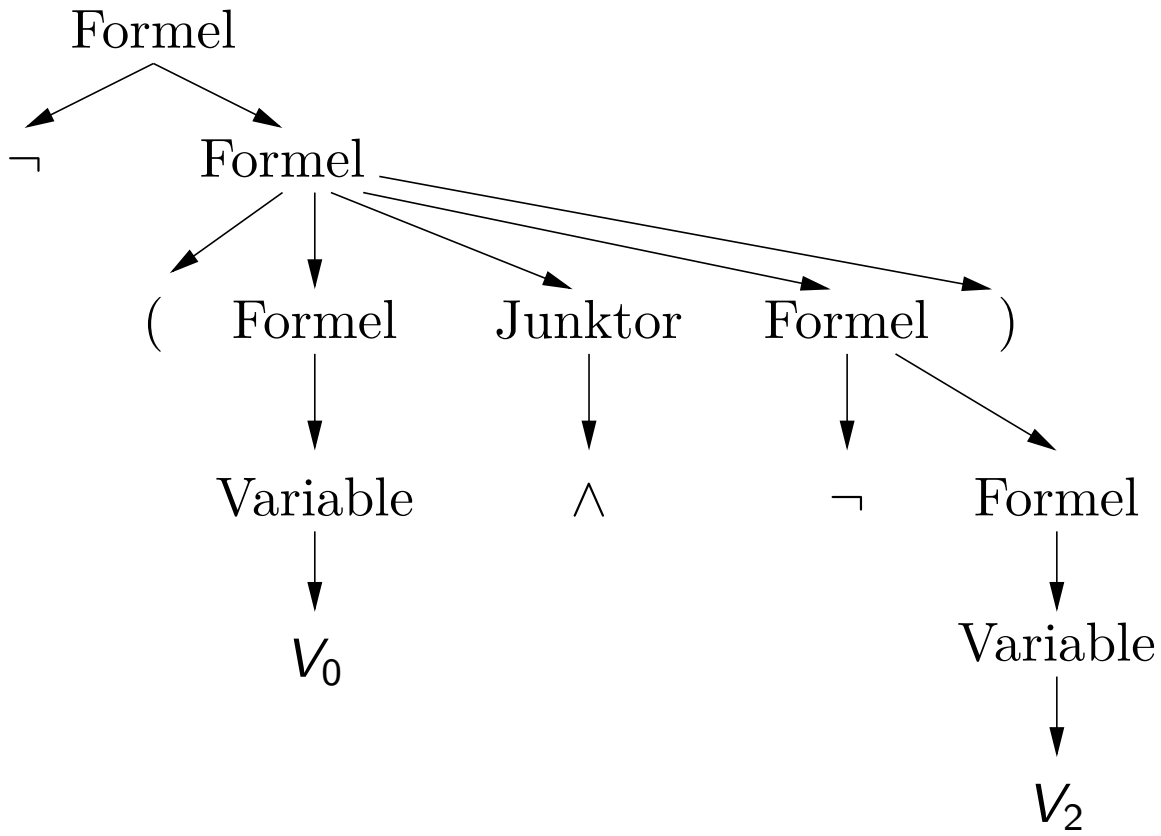


Der Ableitungsbaum repräsentiert die Ableitung

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

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

Die Aussagenlogik als kontextfreie Sprache (3/3)



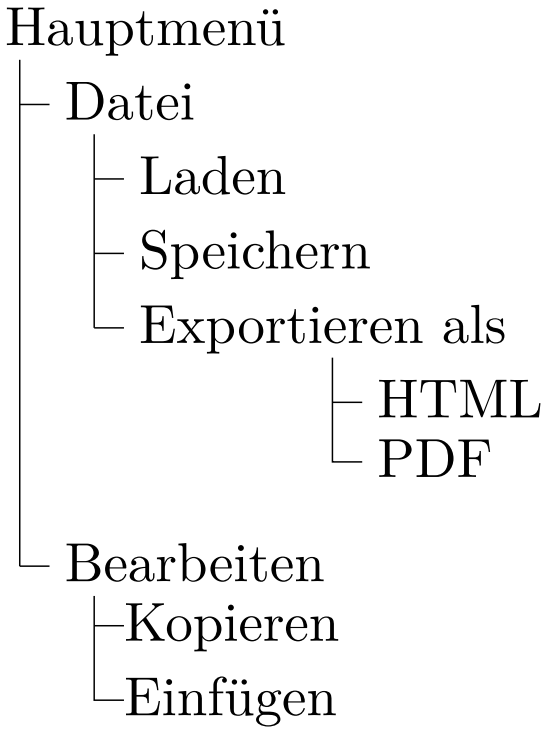
Dieser Ableitungsbaum repräsentiert die Ableitung der Formel $\neg(V_0 \wedge \neg V_2)$

$$
\begin{aligned}
\mathit{Formel} &\Longrightarrow \neg\,\mathit{Formel} \Longrightarrow \neg\,(\mathit{Formel}\ \mathit{Junktor}\ \mathit{Formel}) \\
&\Longrightarrow \neg\,(\mathit{Variable}\ \mathit{Junktor}\ \mathit{Formel}) \Longrightarrow \neg\,(V_0\ \mathit{Junktor}\ \mathit{Formel}) \\
&\Longrightarrow \neg\,(V_0 \wedge \mathit{Formel}) \Longrightarrow \neg\,(V_0 \wedge \neg \mathit{Formel}) \Longrightarrow \neg\,(V_0 \wedge \neg \mathit{Variable}) \\
&\Longrightarrow \neg\,(V_0 \wedge \neg V_2).
\end{aligned}
$$

Menüs in Benutzungsoberflächen (1/4)

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

Beispiel:



Menüs in Benutzungsoberflächen (2/4)

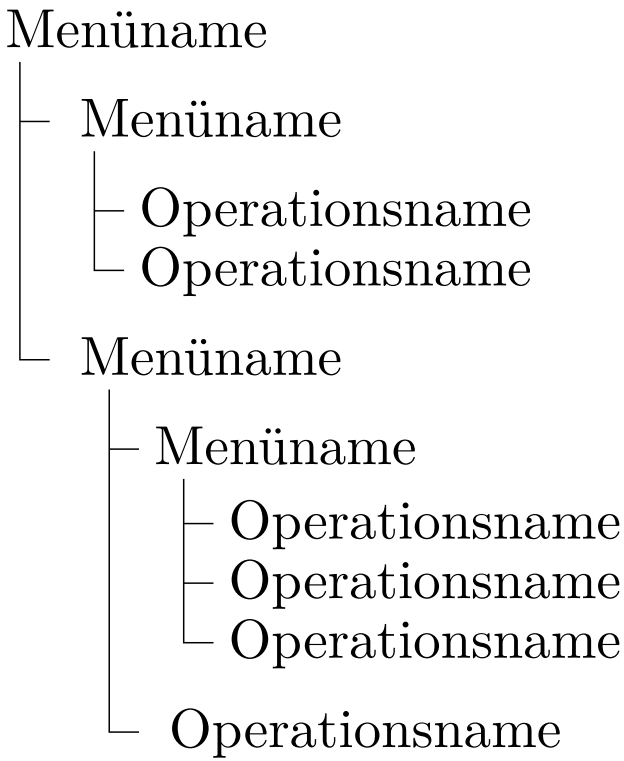
Ein **Menü** besteht aus einem Menünamen und einer Folge von Einträgen: Ein Eintrag besteht aus einem Operationsnamen oder selbst wieder einem Menü.

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

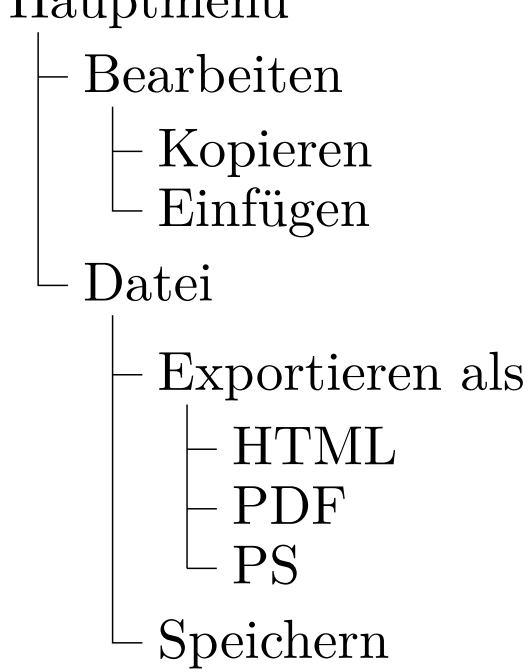
- $\Sigma := \{ \texttt{Menüname}, \texttt{Operationsname} \}$,
- $V := \{ \mathit{Menü}, \mathit{Eintragsfolge}, \mathit{Eintrag} \}$,
- $S := \mathit{Menü}$,
- $P := \{ \begin{array}{rcl} \mathit{Menü} & \to & \texttt{Menüname}\ \mathit{Eintragsfolge}\ , \\ \mathit{Eintragsfolge} & \to & \mathit{Eintrag} \mid \mathit{Eintrag}\ \mathit{Eintragsfolge}\ , \\ \mathit{Eintrag} & \to & \texttt{Operationsname} \mid \mathit{Menü} \end{array} \}$.

Menüs in Benutzungsoberflächen (3/4)

Die Struktur eines Menüs:

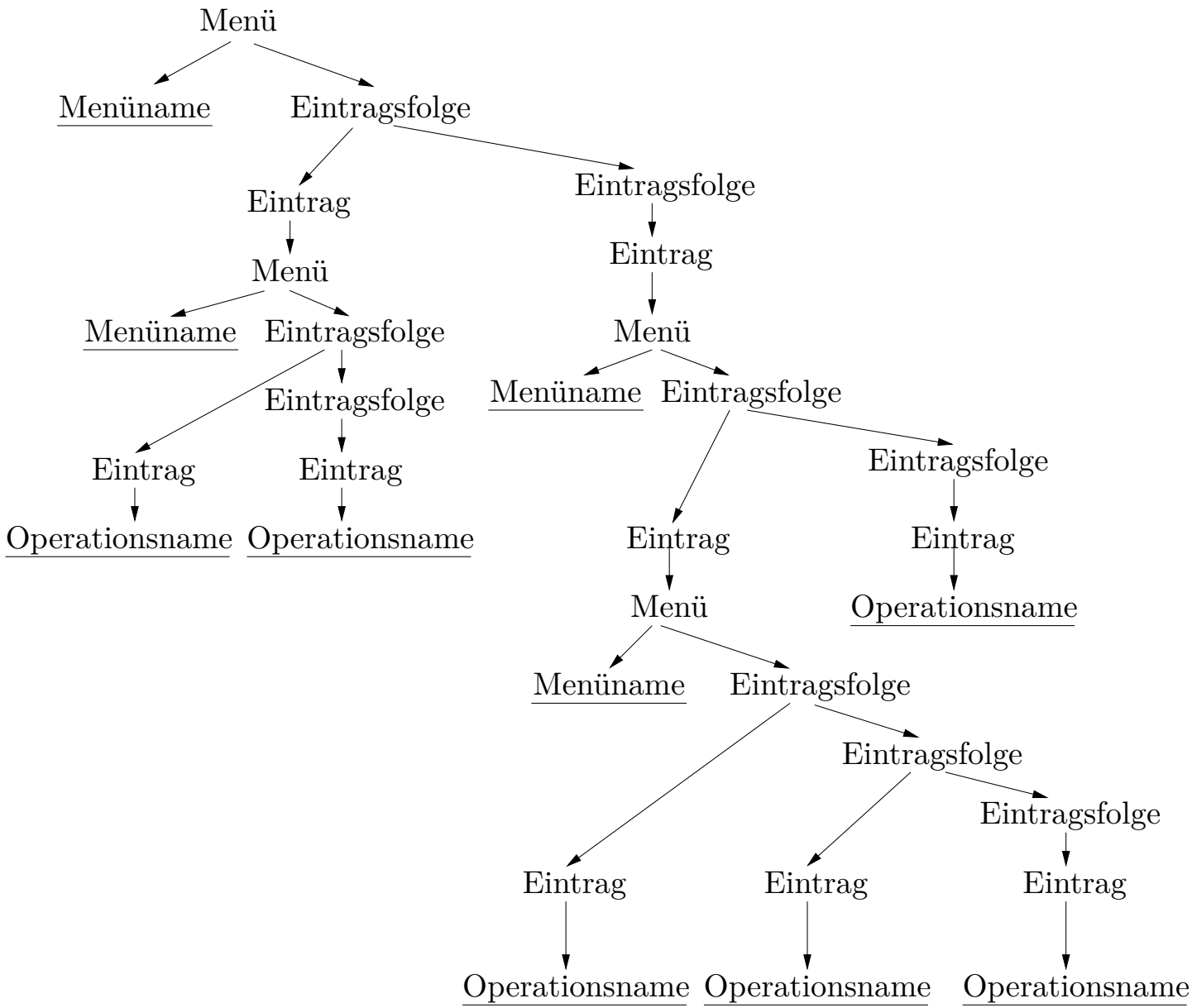


Beispiel für Namen:



Und wie sieht der Ableitungsbaum für das Beispiel aus?

Menüs in Benutzungsoberflächen (4/4)



HTML-Tabellen und KFGs

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	Hoersaal VI
Do	8:00-10:00	Hoersaal 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> Hoersaal VI </td>
  </tr>
  <tr>
    <td> Do </td>
    <td> 8:00-10:00 </td>
    <td> Hoersaal 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. für Anfang und Ende eines **Eintrags in einer Zelle der Tabelle**.

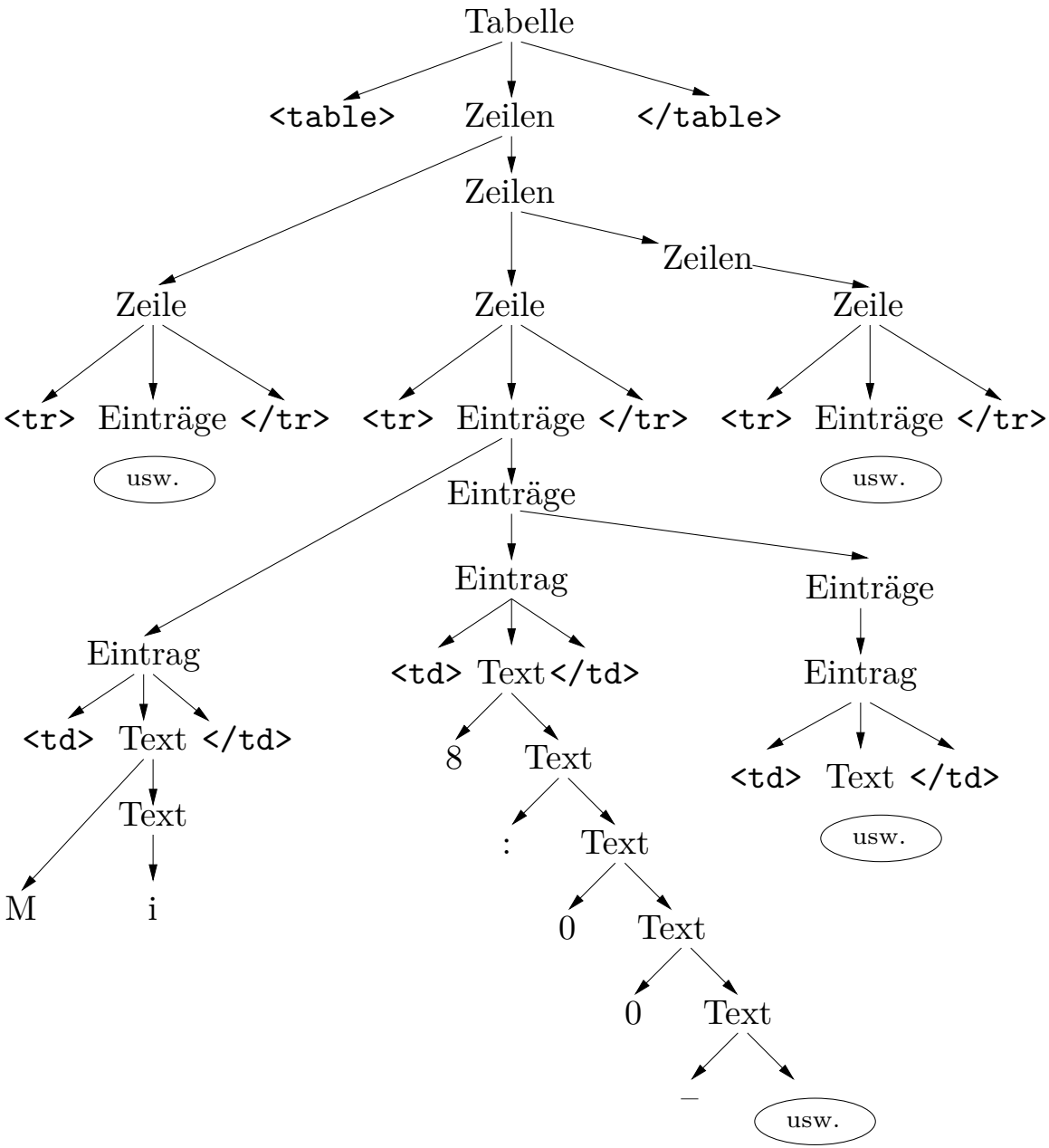
Eine KFG zur Erzeugung von HTML-Tabellen

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

- $\Sigma := \{$ `<table>`, `</table>`, `<tr>`, `</tr>`, `<td>`, `</td>`, $\mathtt{a}, \ldots, \mathtt{z}$, $\mathtt{A}, \ldots, \mathtt{Z}$, $\mathtt{0}, \mathtt{1}, \ldots, \mathtt{9}$, :, -, $\sqcup$ $\}$
- $V := \{$ *Tabelle*, *Zeilen*, *Zeile*, *Einträge*, *Eintrag*, *Text* $\}$
- $S :=$ *Tabelle*
- $P :=$

$$
\begin{array}{rcl}
\textit{Tabelle} & \to & \texttt{<table>}\ \textit{Zeilen}\ \texttt{</table>}\ , \\
\textit{Zeilen} & \to & \textit{Zeile} \mid \textit{Zeile Zeilen}\ , \\
\textit{Zeile} & \to & \texttt{<tr>}\ \text{Einträge}\ \texttt{</tr>}\ , \\
\textit{Einträge} & \to & \textit{Eintrag} \mid \textit{Eintrag Einträge}\ , \\
\textit{Eintrag} & \to & \texttt{<td>}\ \textit{Text}\ \texttt{</td>} \mid \texttt{<td>}\ \textit{Tabelle}\ \texttt{</td>}\ , \\
\textit{Text} & \to & a\,\textit{Text} \mid b\,\textit{Text} \mid \cdots \mid z\,\textit{Text} \mid A\textit{Text} \mid B\textit{Text} \mid \cdots \mid Z\textit{Text} \\
\textit{Text} & \to & 0\,\textit{Text} \mid \cdots 9\,\textit{Text} \mid :\,\textit{Text} \mid -\,\textit{Text} \mid \sqcup \textit{Text}, \\
\textit{Text} & \to & a \mid b \mid \cdots \mid z \mid A \mid B \mid \cdots \mid Z \mid 0 \mid \cdots \mid 9 \mid : \mid - \mid \sqcup
\end{array}
\Big\}.
$$

Der Ableitungsbaum der HTML-Tabelle



Reguläre und kontextfreie Sprachen

Links- und Rechtsreguläre Grammatiken

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

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

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

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

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

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

  $$S \to aSb \mid \epsilon$$

  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**“.

Zusammenfassung

(a) KFGs werden eingesetzt, um

rekursiv definierte Strukturen

zu modellieren.

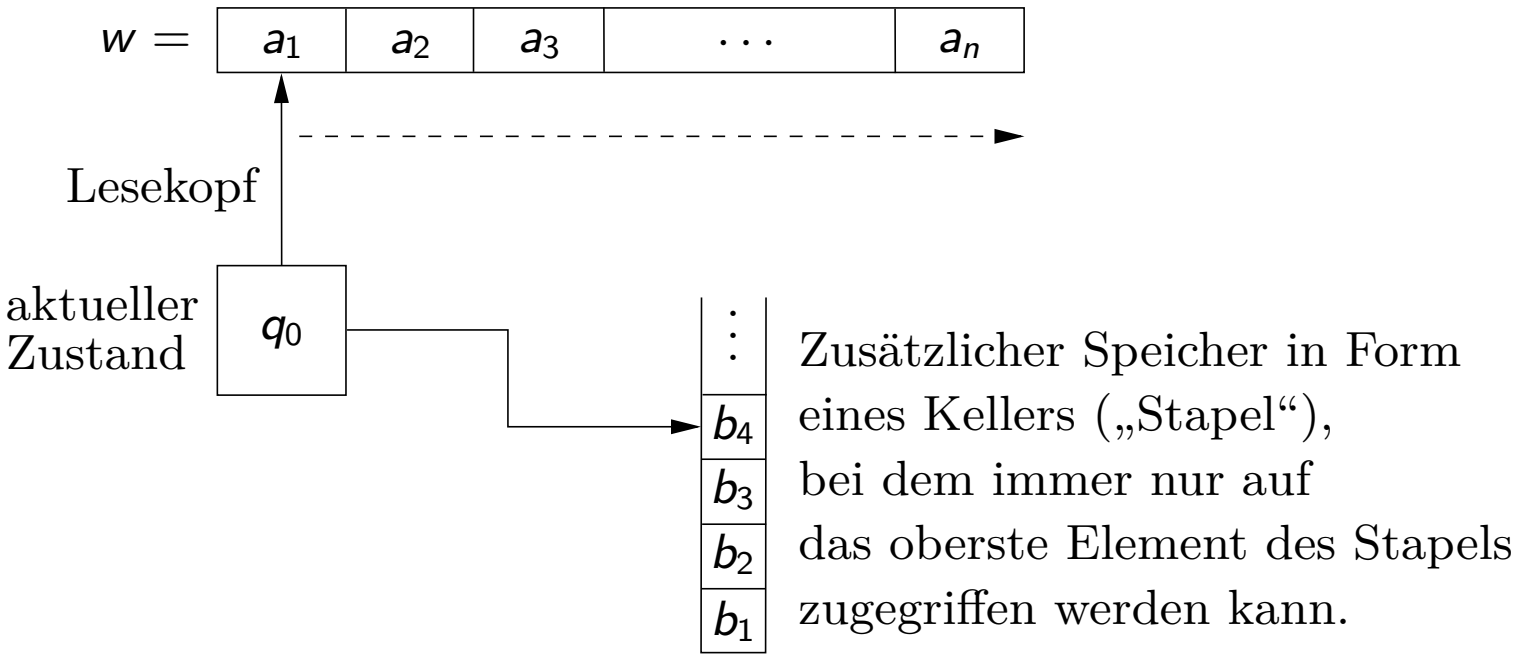
(b) Es gibt z. B. wichtige Anwendungen in der Spezifikation von
- Programmiersprachen,
- HTML,
- XML.

(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.

Deterministisch kontextfreie Sprachen

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?“

sprich: „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 2**“.

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

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

ist kontextfrei!

Details in der Veranstaltung „**Theoretische Informatik 2**“.

Kontextfreie Sprachen sind nicht unter Komplementbildung abgeschlossen!