

Kontextfreie Grammatiken

Was kann man mit kontextfreien Grammatiken anfangen?

Kontextfreie Grammatiken, kurz:

KFGs

werden zur Modellierung von

beliebig tief geschachtelten baumartigen Strukturen

eingesetzt.

KFGs werden unter Anderen angewandt,

- um **Programmiersprachen** wie etwa Java, C oder Pascal zu beschreiben und spielen auch beim „Compilerbau“ eine zentrale Rolle,
- um **Datenaustauschformate**, d.h. Sprachen (wie etwa HTML oder XML) als Schnittstelle zwischen Software-Werkzeugen, zu beschreiben,
- um **Bäume** zur Repräsentation strukturierter Daten (z.B. XML) zu beschreiben.

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 meistens $A \rightarrow x$.

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:

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

Die Produktion $\text{Ausdruck} \rightarrow \text{Ausdruck Operator Ausdruck}$ 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

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.

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: $w \Longrightarrow_G w$.

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

Die von einer KFG G erzeugte Sprache $L(G)$

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

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.:

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

Achtung:

$L(G)$ ist eine Teilmenge von $\Sigma^* \Rightarrow$

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

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

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

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

Eine Sprache $L \subseteq \Sigma^*$ heißt genau dann

kontextfrei,

wenn es eine kontextfreie Grammatik $G = (\Sigma, V, S, P)$ gibt mit

$$L = L(G).$$

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	\rightarrow	<code>begin statements end</code>
<code>statements</code>	\rightarrow	<code>statement statement ; statements</code>
<code>statement</code>	\rightarrow	<code>assign-statement while-statement</code>
<code>assign-statement</code>	\rightarrow	<code>variable := expression</code>
<code>while-statement</code>	\rightarrow	<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 der 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.

Die Backus-Naur Normalform

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

- Sie 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 Variante der BNF auf <http://docs.oracle.com/javase/specs/jls/se8/html/index.html> (zuletzt besucht am 29.01.2015)

Eine effiziente **Syntaxanalyse** ist möglich.

Frage: Was ist eine Syntaxanalyse?

Antwort: Die Bestimmung einer Ableitung bzw. eines Ableitungsbaums.

Und was ist ein Ableitungsbaum?

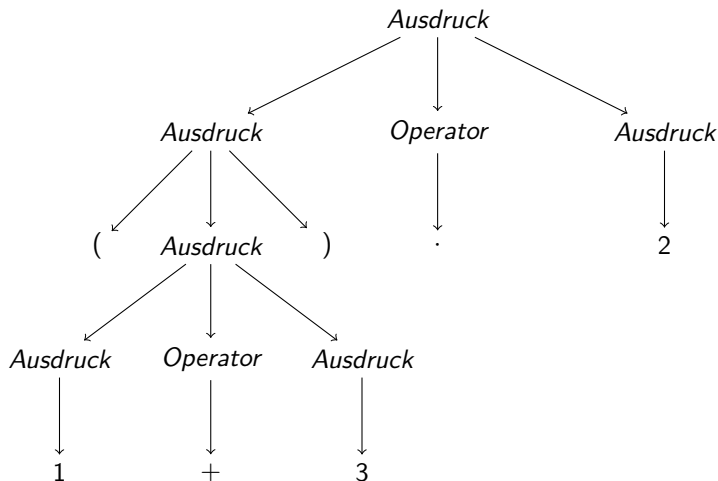
Ableitungsbäume

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 ε markiert ist.

Wie versteht ein Compiler ein syntaktisch korrektes Programm p ?

Indem der Compiler den Ableitungsbaum von p bestimmt.

Und wenn es mehrere Ableitungsbäume für p gibt?

Die Spezifikation der Programmiersprache, –also die KFG im Fall von Java– **muss garantieren**, dass es stets nur einen Ableitungsbaum gibt.

*Solche KFGs heißen **eindeutig**.*

Details: In der Veranstaltung „**Theoretische Informatik 2**“.

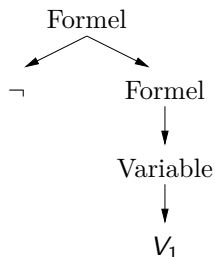
Wir konstruieren eine KFG

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

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

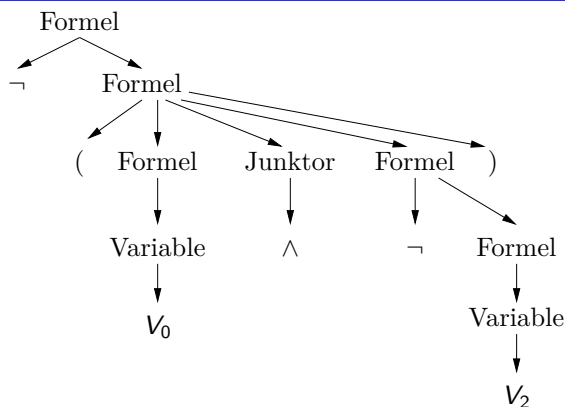
$$\left\{ \begin{array}{l} \text{Formel} \rightarrow \mathbf{0} \mid \mathbf{1} \mid \text{Variable} , \\ \text{Formel} \rightarrow \neg \text{Formel} \mid (\text{Formel} \text{ Junktor } \text{Formel}) , \\ \text{Variable} \rightarrow V_0 \mid V_1 \mid V_2 , \\ \text{Junktor} \rightarrow \wedge \mid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



Der Ableitungsbaum repräsentiert die Ableitung

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

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

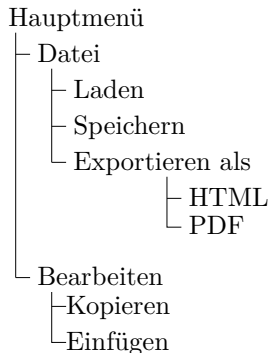


Dieser Ableitungsbaum repräsentiert die Ableitung der Formel $\neg(V_0 \wedge \neg V_2)$

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

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

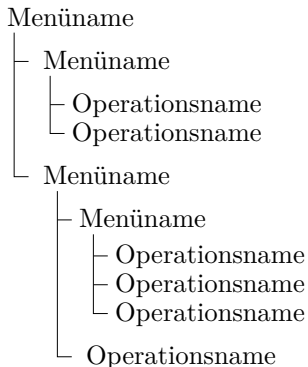
Beispiel:



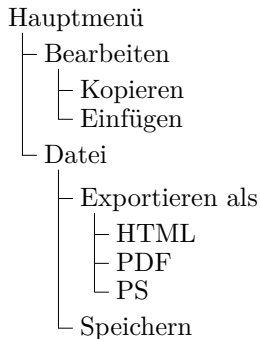
Zur Spezifizierung solcher Menüs kann man die Grammatik $G_{\text{Menü}} = (\Sigma, V, S, P)$ verwenden, wobei

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

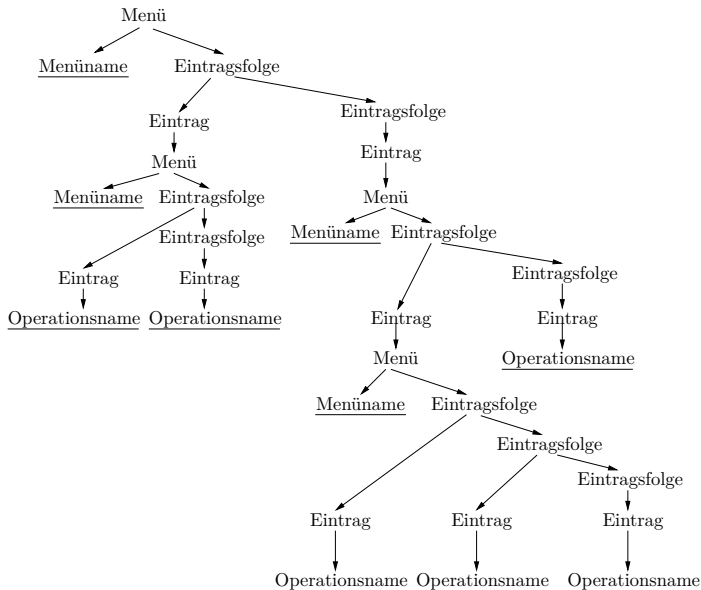
Die Struktur eines Menüs:



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	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>
```

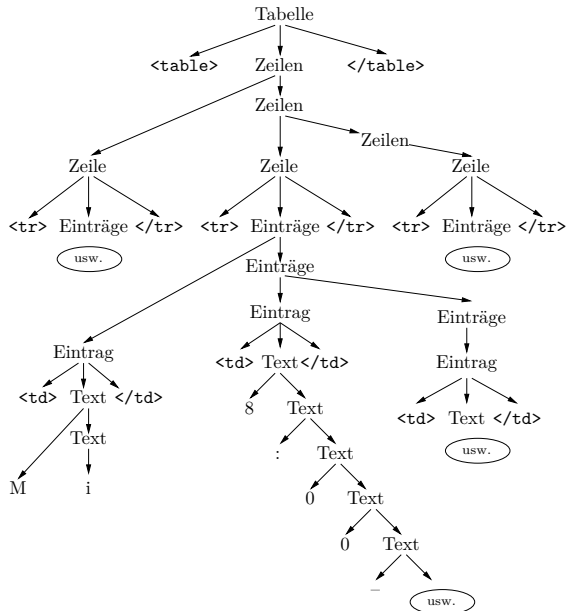
Die Symbole `<table>` und `</table>`, `<tr>` und `</tr>` bzw. `<td>` und `</td>` stehen für Anfang und Ende einer **Tabelle**, für Anfang und Ende einer **Zeile der Tabelle** bzw. für Anfang und Ende eines **Eintrags in einer Zelle der Tabelle**.

Eine KFG zur Erzeugung von HTML-Tabellen

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

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

Der Ableitungsbaum der HTML-Tabelle



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

- (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 nicht regulär.

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

$$L = L(G).$$

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

Zusammenfassung und Ausblick

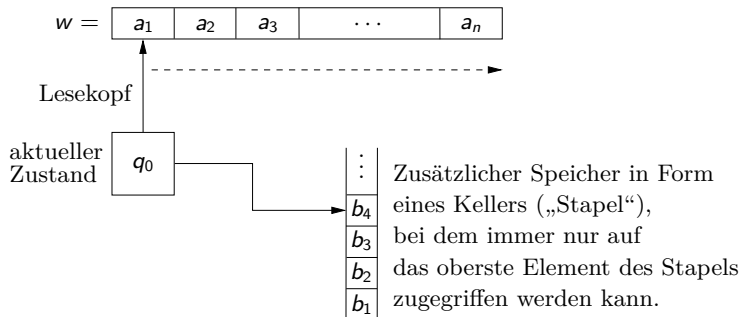
KFGs werden eingesetzt, um

rekursiv definierte Strukturen

zu modellieren.

Kellerautomaten

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



In der „Theoretischen Informatik 2“ 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 nicht deterministisch kontextfrei.

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

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 2**“.

Viele Parser übersetzen ein Anwenderprogramm in einen Ableitungsbaum, um damit die Semantik des Programms auszudrücken.

(a) Nenne eine kontextfreie Grammatik

eindeutig,

wenn jedes erzeugbare Wort genau einen Ableitungsbaum besitzt.

(b) Ein deterministischer Kellerautomat „besitzt“ eine äquivalente eindeutige Grammatik.

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

Die Syntax von Programmiersprachen und deterministisch kontextfreie Sprachen:

„A marriage made in heaven“.

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 2**“.

Kontextfreie Sprachen sind nicht unter Komplementbildung abgeschlossen!