

Theoretische Informatik I



Einheit 3

Kontextfreie Sprachen



1. Kontextfreie Grammatiken
2. Pushdown Automaten
3. Eigenschaften kontextfreier Sprachen

VERARBEITUNG VON PROGRAMMIERSPRACHEN

- **Was ist das einfachste Beschreibungsmodell?**
 - Analyse und Compilation muss formal beschreibbar sein
 - Generisches Modell für “alle” Programmiersprachen erforderlich
- **Typ-3 Sprachen sind einfach und effizient**
 - Beschreibung durch Grammatiken oder reguläre Ausdrücke
 - Beschreibung umwandelbar in endlichen Automaten
 - Erkennung von Wörtern der Sprache in “Echtzeit”
- **Aber Programmiersprachen sind nicht regulär**
 - Die meisten Programmstrukturen enthalten Schachtelungen wie Blöcke, if-then-else, arithmetische Ausdrücke, Klammersausdrücke, ...
 - Korrekte Klammersausdrücke und Schachtelungen sind nicht regulär



Syntaxanalyse und Compilation von Programmiersprachen braucht mehr als reguläre Sprachen

ALLE BEDEUTENDEN SPRACHEN SIND KONTEXTFREI

● Programmiersprachen

- **Compiler** kann kontextfreie Grammatiken effizient verarbeiten
- **Parser kann** aus kontextfreier Grammatik **automatisch erzeugt** werden
 - Standard Unix tool **YACC** unterstützt schnellen Compilerentwurf

● Markup Sprachen

- **HTML**: Formatierung von Dokumenten mit Links zu Programmaufrufen
- **XML**: Einheitliche Beschreibung der Semantik von Dokumenten

Beide Sprachen erfordern die Mächtigkeit von kontextfreien Grammatiken

Mehr in HMU §5.3

● Wichtige Fragen

- **Analyse**: Systematische Rekonstruktion des Syntaxbaums
 - Ist das immer **eindeutig** möglich?
- **Compilation**: Zuweisung von Objectcode an Wörter der Sprache
- **Maschinenmodell**: effektive generische Erkennungsverfahren

Theoretische Informatik I



Einheit 3.1

Kontextfreie Grammatiken



1. Grammatiken und Ableitungen
2. Ableitungsbäume
3. Mehrdeutigkeiten

RÜCKBLICK: KONTEXTFREIE GRAMMATIKEN

- Eine **kontextfreie Grammatik (kfG)** ist ein 4-Tupel $G = (V, T, P, S)$ mit
 - T endliches **Terminalalphabet**
 - V endliches **Hilfsalphabet** mit $V \cap T = \emptyset$
 - $P \subseteq V \times \Gamma^*$ endliche Menge der **Produktionen** (wobei $\Gamma = V \cup T$)
 - $S \in V$ **Startsymbol**

Die übliche Schreibweise für Produktionen $(A, r) \in P$ ist $A \rightarrow r$

Eine kompakte Notation für $A \rightarrow r_1, A \rightarrow r_2, \dots, A \rightarrow r_n$ ist $A \rightarrow r_1 | r_2 | \dots | r_n$

- **Ableitbarkeit in einer kontextfreien Grammatik**

– $w \rightarrow z \equiv \exists x, y \in \Gamma^*. \exists A \rightarrow r \in P. w = x A y \wedge z = x r y$

– $w \xrightarrow{*} z \equiv \exists n \in \mathbb{N}. w \xrightarrow{n} z$

wobei $w \xrightarrow{0} z \equiv w = z$ und $w \xrightarrow{n+1} z \equiv \exists u \in \Gamma^*. w \rightarrow u \wedge u \xrightarrow{n} z$

- **Von G erzeugte Sprache:**

$$L(G) \equiv \{w \in T^* \mid S \xrightarrow{*} w\}$$

GRAMMATIK FÜR GESCHACHELTE KLAMMERAUSDRÜCKE

$$G_5 = (\{S\}, \{ (,) \}, \{ S \rightarrow (S), S \rightarrow \epsilon \}, S)$$

- Zeige: $L(G_5) = \{ ({}^k) {}^k \mid k \in \mathbb{N} \}$
- Beweise durch Induktion über Länge der Ableitung
 - $\forall k \in \mathbb{N}. \forall w \in \{ (,) \}^*. S \xrightarrow{k+1} w \Leftrightarrow w = ({}^k) {}^k$

Basisfall

$$- S \xrightarrow{1} w \Leftrightarrow (S \rightarrow w) \in P \Leftrightarrow w = \epsilon \Leftrightarrow w = ({}^0) {}^0 \quad \checkmark$$

Induktionsschritt

$$- \text{Es gelte } \forall v \in \{ (,) \}^*. S \xrightarrow{k+1} v \Leftrightarrow v = ({}^k) {}^k$$

$$\begin{aligned} - S \xrightarrow{k+2} w &\Leftrightarrow S \rightarrow (S) \xrightarrow{k+1} w \\ &\Leftrightarrow \exists v \in \{ (,) \}^*. S \xrightarrow{k+1} v \wedge w = (v) \\ &\Leftrightarrow \exists v \in \{ (,) \}^*. v = ({}^k) {}^k \wedge w = (v) \quad (\text{Annahme}) \\ &\Leftrightarrow w = ({}^{k+1}) {}^{k+1} \quad \checkmark \end{aligned}$$

$$\{ ({}^k) {}^k \mid k \in \mathbb{N} \} \in \mathcal{L}_2 - \mathcal{L}_3$$

KONTEXTFREIE GRAMMATIK FÜR PALINDROME

$$L(G_6) = \{w \in \{0, 1\}^* \mid w = w^R\}$$

$$G_6 = (\{S\}, \{0, 1\}, P, S)$$

$$\text{mit } P = \{S \rightarrow \epsilon, S \rightarrow 0, S \rightarrow 1, S \rightarrow 0S0, S \rightarrow 1S1\}$$

- **Beweis durch Induktion über Länge der Ableitung**
 - $\forall k \in \mathbb{N}. \forall w \in \{0, 1\}^*. S \xrightarrow{k+1} w \Leftrightarrow w = w^R \wedge |w| \in \{2k, 2k+1\}$

Basisfall

$$- S \xrightarrow{1} w \Leftrightarrow (S \rightarrow w) \in P \Leftrightarrow w \in \{0, 1, \epsilon\} \Leftrightarrow w = w^R \wedge |w| \in \{0, 1\} \quad \checkmark$$

Induktionsschritt

$$- \text{Es gelte } \forall v \in \{0, 1\}^*. S \xrightarrow{k+1} v \Leftrightarrow v = v^R \wedge |v| \in \{2k, 2k+1\}$$

$$\begin{aligned} - S \xrightarrow{k+2} w &\Leftrightarrow S \rightarrow 0S0 \xrightarrow{k+1} w \vee S \rightarrow 1S1 \xrightarrow{k+1} w \\ &\Leftrightarrow \exists v \in \{0, 1\}^*. S \xrightarrow{k+1} v \wedge (w = 0v0 \vee w = 1v1) \\ &\Leftrightarrow \exists v \in \{0, 1\}^*. v = v^R \wedge |v| \in \{2k, 2k+1\} \wedge (w = 0v0 \vee w = 1v1) \\ &\Leftrightarrow w = w^R \wedge |w| \in \{2k+2, 2k+3\} \quad \checkmark \end{aligned}$$

GRAMMATIK FÜR ARITHMETISCHE AUSDRÜCKE

- **Ausdrücke über Operatoren $+$ und $*$**

- **Bezeichner** (*I*dentifier):

- Buchstabe gefolgt von Buchstaben/Ziffern
- Buchstaben ***a, b, c***, Ziffern ***0, 1***

- **Ausdrücke** (*E*xpressions):

- Schachtelung mit $+$, $*$ und Klammern

- $G_7 = (\{E, I\}, \{a, b, c, 0, 1, +, *, (,)\}, P, E)$

mit $P = \{ E \rightarrow I \mid E + E \mid E * E \mid (E)$

$I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$

Kann man die Struktur eines arithmetischen Ausdrucks mit G_7 immer rekonstruieren?

Rekonstruierbare Auswahl von Produktionen

- **Beliebige Ableitung**

$$\begin{aligned}
 E &\longrightarrow E * E \longrightarrow I * E \longrightarrow I * (E) \\
 &\longrightarrow I * (E + E) \longrightarrow I * (I + E) \longrightarrow I * (I + I) \longrightarrow I * (a + I) \\
 &\longrightarrow I * (a + I0) \longrightarrow I * (a + I00) \longrightarrow I * (a + b00) \longrightarrow a * (a + b00)
 \end{aligned}$$

- **Linksseitige Ableitung $w \xrightarrow{L} z$**

– In w wird die am weitesten links stehende Variable ersetzt

$$\begin{aligned}
 E &\xrightarrow{L} E * E \xrightarrow{L} I * E \xrightarrow{L} a * E \xrightarrow{L} a * (E) \\
 &\xrightarrow{L} a * (E + E) \xrightarrow{L} a * (I + E) \xrightarrow{L} a * (a + E) \xrightarrow{L} a * (a + I) \\
 &\xrightarrow{L} a * (a + I0) \xrightarrow{L} a * (a + I00) \xrightarrow{L} a * (a + b00)
 \end{aligned}$$

- **Rechtsseitige Ableitung $w \xrightarrow{R} z$**

– In w wird die am weitesten rechts stehende Variable ersetzt

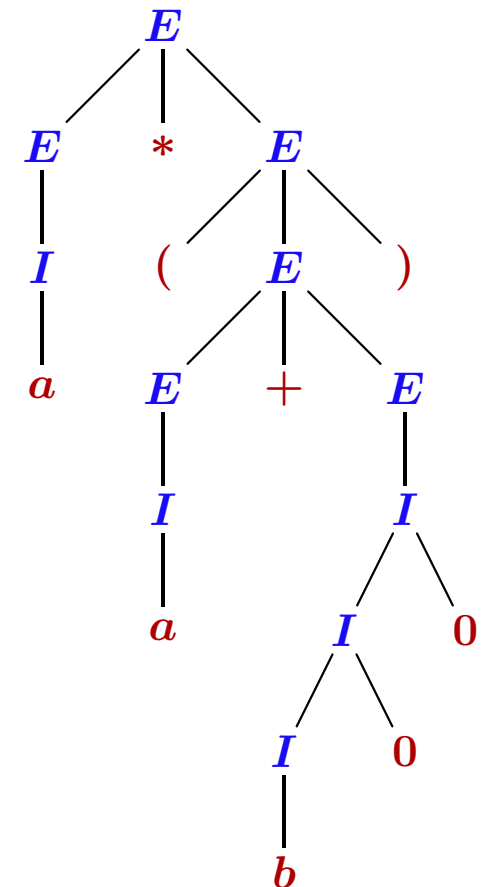
$$\begin{aligned}
 E &\xrightarrow{R} E * E \xrightarrow{R} E * (E) \xrightarrow{R} E * (E + E) \xrightarrow{R} E * (E + I) \\
 &\xrightarrow{R} E * (E + I0) \xrightarrow{R} E * (E + I00) \xrightarrow{R} E * (E + b00) \\
 &\xrightarrow{R} E * (I + b00) \xrightarrow{R} E * (a + b00) \xrightarrow{R} I * (a + b00) \xrightarrow{R} a * (a + b00)
 \end{aligned}$$

ABLEITUNGSBÄUME (PARSEBÄUME)

Baumdarstellung von Ableitungen

● Geordneter markierter Baum

- Innere Knoten mit Variablen $A \in V$ markiert
- Wurzel markiert mit Startsymbol
- Blätter mit Terminalsymbolen $a \in T$ oder mit ϵ markiert
- Hat ein innerer Knoten Markierung A und Nachfolger mit Markierungen $v_1 \dots v_n$, so ist $A \rightarrow v_1 \dots v_n \in P$



● Exkurs: Notation für Bäume

- **Baum**: Sammlungen von **Knoten** mit **Nachfolgerrelation**
- **Nachfolger** sind geordnet
- Ein Knoten hat **maximal einen Vorgänger**
- **Wurzel**: Knoten ohne Vorgänger
- **Blatt** / **Innerer Knoten**: Knoten ohne/mit Nachfolger
- **Nachkommen**: transitive Hülle der Nachfolgerrelation

ABLEITUNGSBÄUME REPRÄSENTIEREN ABLEITUNGEN

- **Blätter repräsentieren Terminalwörter**

- Auslesen durch Tiefensuche von links nach rechts
- $a * (a + b00)$

- **Baum repräsentiert Ableitungen**

- Rekursive Erzeugung beginnend mit Wurzel
- Vorrang für tiefe linke Knoten ergibt Linksableitung

$$\begin{aligned}
 E &\xrightarrow{L} E * E \xrightarrow{L} I * E \xrightarrow{L} a * E \xrightarrow{L} a * (E) \\
 &\xrightarrow{L} a * (E + E) \xrightarrow{L} a * (I + E) \xrightarrow{L} a * (a + E) \\
 &\xrightarrow{L} a * (a + I) \xrightarrow{L} a * (a + I0) \xrightarrow{L} a * (a + I00) \\
 &\xrightarrow{L} a * (a + b00)
 \end{aligned}$$

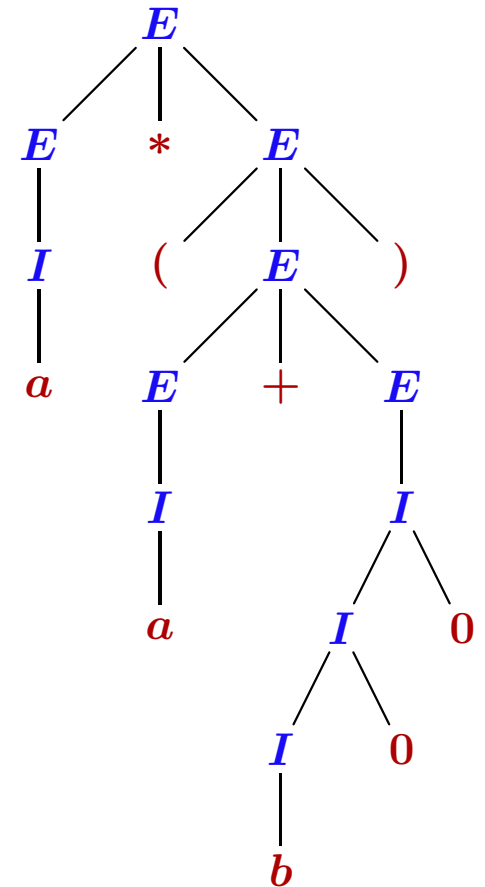
- Vorrang für tiefe rechte Knoten ergibt Rechtsableitung

$S \xrightarrow{*} w \Leftrightarrow$ es gibt einen Ableitungsbaum mit Blattmarkierung w

\Rightarrow : Konstruiere Baum induktiv aus Linksableitung von w

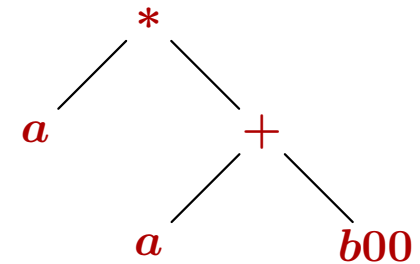
\Leftarrow : Extrahiere Linksableitung von w induktiv aus Baum

Details in HMU §5.2



● Parser erzeugen **Syntaxbaum**

- Rekonstruktion des Ableitungsbaumes aus dem Wort
- Entferne Blätter mit Klammern
- Gruppiere Identifier zu lexikalischen Einheiten
- Entferne Vorgänger von Identifiern, die mit E markiert sind
- Entferne E -Vorgänger von $+/*$



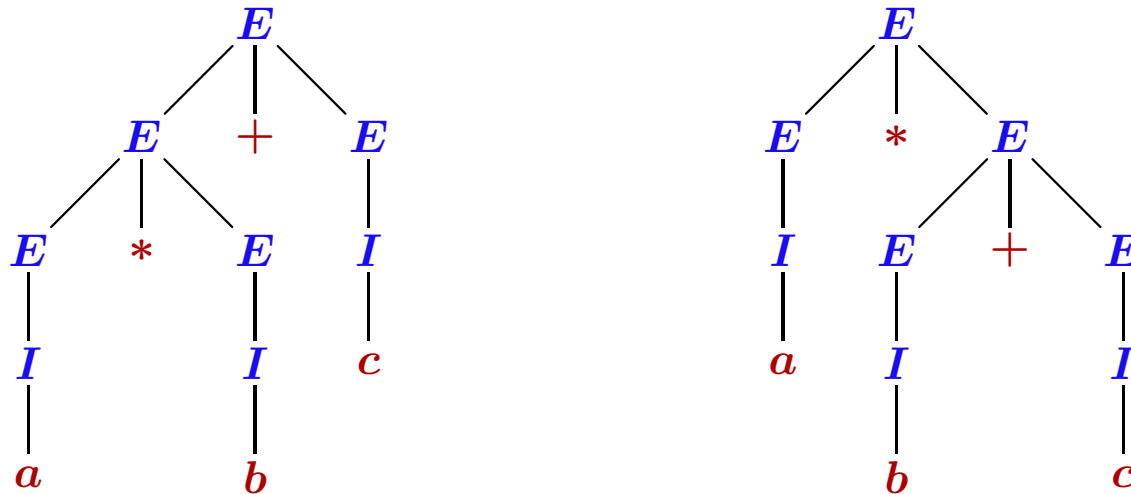
● Compiler erzeugt **Maschinencode**

- Übersetze Identifier in Registernamen
- $+(x, y)$:
 - Bestimme Wert von y und lege ihn im Register ab
 - Bestimme Wert von x
 - Addiere Registerwert
- Coderzeugung für $*(x, y)$ analog

Details in Vossen/Witt §7.2

Nur möglich, wenn Ableitungsbaum eindeutig

WANN IST DER ABLEITUNGSBAUM EINDEUTIG?



- Das Wort $a * b + c$ hat zwei Ableitungen in G_7

$E \longrightarrow E + E \longrightarrow E * E + E \longrightarrow I * E + E \longrightarrow a * E + E \longrightarrow a * I + E$
 $\longrightarrow a * b + E \longrightarrow a * b + I \longrightarrow a * b + c$

$E \longrightarrow E * E \longrightarrow I * E \longrightarrow a * E \longrightarrow a * E + E \longrightarrow a * I + E$
 $\longrightarrow a * b + E \longrightarrow a * b + I \longrightarrow a * b + c$

Beide Ableitungen sind Linksableitungen

- Die Grammatik G_7 ist **mehrdeutig**

– Wörter der Sprache können nicht eindeutig analysiert werden

MEHRDEUTIGKEIT

- **Eindeutige Grammatik** $G = (V, T, P, S)$
 - Jedes Wort $w \in L(G)$ hat genau einen Ableitungsbaum
 - Andernfalls ist G **mehrdeutig**
(ein $w \in L(G)$ hat mindestens zwei verschiedene Ableitungsbäume)
 - G_7 ist mehrdeutig
- **Eindeutige Sprache** L
 - Es gibt eine eindeutige Grammatik G mit $L = L(G)$
 - Andernfalls ist L **inhärent mehrdeutig**
(eine eindeutige Grammatik kann nicht angegeben werden)
 - Die Sprache von G_7 ist eindeutig ↪ nächste Folie
 - $\{0^i 1^j 2^k \mid i=j \vee j=k\}$ ist inhärent mehrdeutig Details in Wegener §6.7

Programmiersprachen müssen eindeutig sein

AUFLÖSUNG VON MEHRDEUTIGKEITEN

- Was fehlt bei $G_7 = (\{E, I\}, \{a, b, c, 0, 1, +, *, (,)\}, P, E)$
mit $P = \{ E \rightarrow I \mid E+E \mid E*E \mid (E), I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$?

G_7 beinhaltet nicht die üblichen Konventionen für $*$ und $+$

$*$ bindet stärker als $+$

$*$ und $+$ werden als linkssassoziativ angesehen

Alle anderen Lesarten benötigen Klammern

- **Prioritätsregeln können Eindeutigkeit erzeugen**

– Niedrigste Priorität $+$ steht linkssassoziativ und außen \mapsto **T**erme

– Höhere Priorität $*$ steht linkssassoziativ und innen \mapsto **F**aktoren

– Faktoren können Bezeichner oder Ausdrücke in Klammern sein

$G'_7 = (\{E, T, F, I\}, \{a, b, c, 0, 1, +, *, (,)\}, P', E)$

mit $P' = \{ E \rightarrow T \mid E+T, T \rightarrow F \mid T*F, F \rightarrow I \mid (E)$
 $I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$

G'_7 ist äquivalent zu G_7 und eindeutig

BEGRÜNDUNG DER EINDEUTIGKEIT VON G'_7

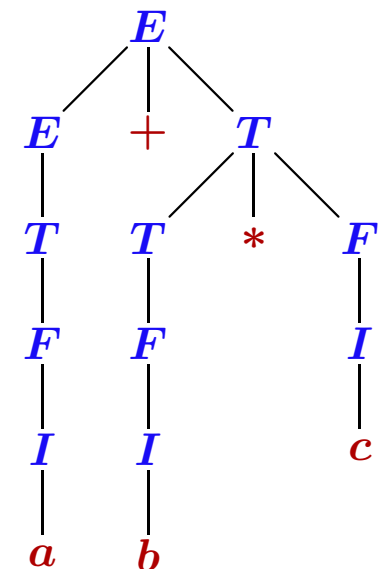
$$P' = \{ E \rightarrow T \mid E+T, \quad T \rightarrow F \mid T*F, \quad F \rightarrow I \mid (E) \\ I \rightarrow a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \}$$

- **Jeder Ausdruck muss aus einer Termfolge bestehen**

- Termfolge muss von rechts nach links erzeugt werden
- Terme haben keine Ausdrücke als direkte Teile
- Es gibt nur einen Parsebaum für $t_1 + t_2 + \dots + t_k$

- **Jeder Term muss aus einer Faktorenfolge bestehen**

- Faktorenfolge muss von rechts nach links erzeugt werden
- Faktoren haben keine Terme als direkte Teile
- Es gibt nur einen Parsebaum für $f_1 * f_2 * \dots * f_n$



Einziger Ableitungsbaum
für $a + b * c$

- **Jeder Faktor ist Bezeichner oder geklammerter Ausdruck**

● **Beschreibungsform für Programmiersprachen**

- Äquivalente Beschreibungsformen: Details in Vossen/Witt §7.3, 7.4
Backus-Naur Form, Syntaxdiagramme, Definitionsgleichungen, ...

● **Compiler benötigt Ableitungsbaum**

- Rekonstruktion nur möglich für eindeutige Grammatiken
- Mehrdeutige Grammatiken können evtl. eindeutig gemacht werden
- Manche kontextfreie Sprache haben keine eindeutige Grammatik

● **Noch zu klärende Fragen**

- Welches Maschinenmodell erkennt genau die kontextfreien Sprachen?
- Wie kann man den Ableitungsbaum rekonstruieren? (Syntaxanalyse)
- Welche Abschlusseigenschaften gelten für kontextfreie Sprachen?
- Welche Spracheigenschaften lassen sich nicht kontextfrei beschreiben?