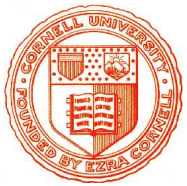
Theoretische Informatik I



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, Klammerausdrücke, ...
  - Korrekte Klammerausdrücke und Schachtelungen sind nicht regulär

$$\Downarrow$$

Syntaxanalyse und Compilation von Programmiersprachen braucht mehr als reguläre Sprachen

Alle bedeutenden Sprachen sind kontextfrei

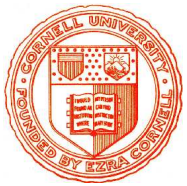
- **Programmiersprachen**
  - Compiler kann kontextfreie Grammatiken effizient verarbeiten
  - Parser kann aus kontextfreier Grammatik automatisch erzeugt werden
    - Standard Unix tool YACC unterstützt schnellen Compilerentwurf
- **Markup Sprachen**
  - HTML: Formatierung von Dokumenten mit Links zu 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 $A \longrightarrow r$

  Eine kompakte Notation für $A \longrightarrow r_1, A \longrightarrow r_2 .. A \longrightarrow r_n$ ist $A \longrightarrow r_1 | r_2 | .. | r_n$
- **Ableitbarkeit in einer kontextfreien Grammatik**
  - $w \longrightarrow z \equiv \exists x, y \in \Gamma^*. \exists A \longrightarrow r \in P. w = x\,A\,y \wedge z = x\,r\,y$
  - $w \overset{*}{\longrightarrow} z \equiv \exists n \in \mathbb{N}. w \overset{n}{\longrightarrow} z$

    wobei $w \overset{0}{\longrightarrow} z \equiv w = z$ und $w \overset{n+1}{\longrightarrow} z \equiv \exists u \in \Gamma^*. w \longrightarrow u \wedge u \overset{n}{\longrightarrow} z$
- **Von G erzeugte Sprache:**

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

Grammatik für geschachtelte Klammerausdrücke

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

- **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
  - $S \xrightarrow{1} w \ \Leftrightarrow\ (S \to w) \in P \ \Leftrightarrow\ w = \epsilon \ \Leftrightarrow\ w = (^0)^0$ ✓

  Induktionsschritt
  - Es gelte $\forall v \in \{(,)\}^*.\ S \xrightarrow{k+1} v \ \Leftrightarrow\ v = (^k)^k$
  - $$\begin{aligned} S \xrightarrow{k+2} w &\Leftrightarrow S \to (S) \xrightarrow{k+1} w \\ &\Leftrightarrow \exists v \in \{(,)\}^*.\ S \xrightarrow{k+1} v \ \wedge\ w = (v) \\ &\Leftrightarrow \exists v \in \{(,)\}^*.\ v = (^k)^k \ \wedge\ w = (v) \qquad \text{(Annahme)} \\ &\Leftrightarrow w = (^{k+1})^{k+1} \end{aligned}$$ ✓

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

Kontextfreie Grammatik für Palindrome

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

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

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

- **Beweise 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

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

Induktionsschritt

- Es gelte $\forall v \in \{0,1\}^*.\ S \xrightarrow{k+1} v \ \Leftrightarrow\ v = v^R \wedge |v| \in \{2k, 2k{+}1\}$
- $S \xrightarrow{k+2} w \ \Leftrightarrow\ S \to 0S0 \xrightarrow{k+1} w \ \vee\ S \to 1S1 \xrightarrow{k+1} w$
  $\Leftrightarrow\ \exists v \in \{0,1\}^*.\ S \xrightarrow{k+1} v \ \wedge\ (w = 0v0 \ \vee\ w = 1v1)$
  $\Leftrightarrow\ \exists v \in \{0,1\}^*.\ v{=}v^R \wedge |v| \in \{2k, 2k{+}1\} \ \wedge\ (w{=}0v0 \ \vee\ w{=}1v1)$
  $\Leftrightarrow\ w = w^R \wedge |w| \in \{2k{+}2, 2k{+}3\}$ ✓

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 $P = \{\ E \to I \mid E{+}E \mid E{*}E \mid (E)$

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

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

Rekonstruierbare Auswahl von Produktionen

- **Beliebige Ableitung**

$$E \longrightarrow E*E \longrightarrow I*E \longrightarrow I*(E)$$
$$\longrightarrow I*(E+E) \longrightarrow I*(I+E) \longrightarrow I*(I+I) \longrightarrow I*(a+I)$$
$$\longrightarrow I*(a+I0) \longrightarrow I*(a+I00) \longrightarrow I*(a+b00) \longrightarrow a*(a+b00)$$

- **Linksseitige Ableitung $w \longrightarrow_L z$**
  - In w wird die am weitesten links stehende Variable ersetzt

$$E \longrightarrow_L E*E \longrightarrow_L I*E \longrightarrow_L a*E \longrightarrow_L a*(E)$$
$$\longrightarrow_L a*(E+E) \longrightarrow_L a*(I+E) \longrightarrow_L a*(a+E) \longrightarrow_L a*(a+I)$$
$$\longrightarrow_L a*(a+I0) \longrightarrow_L a*(a+I00) \longrightarrow_L a*(a+b00)$$

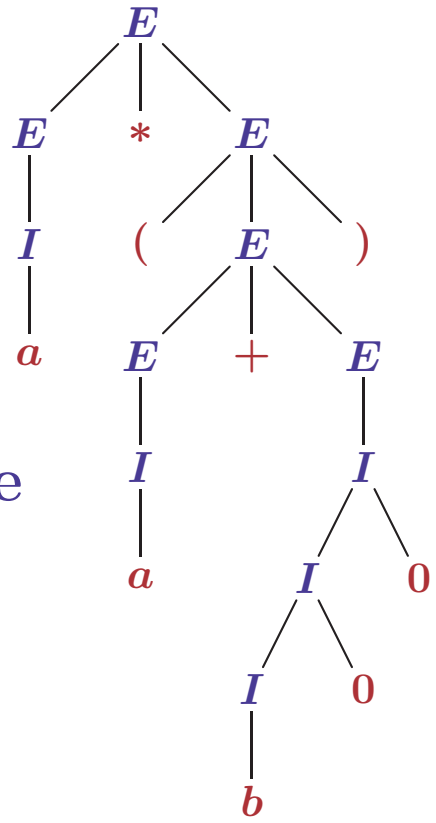
- **Rechtsseitige Ableitung $w \longrightarrow_R z$**
  - In w wird die am weitesten rechts stehende Variable ersetzt

$$E \longrightarrow_R E*E \longrightarrow_R E*(E) \longrightarrow_R E*(E+E) \longrightarrow_R E*(E+I)$$
$$\longrightarrow_R E*(E+I0) \longrightarrow_R E*(E+I00) \longrightarrow_R E*(E+b00)$$
$$\longrightarrow_R E*(I+b00) \longrightarrow_R E*(a+b00) \longrightarrow_R I*(a+b00) \longrightarrow_R a*(a+b00)$$

Ableitungsbäume (Parsebäume)

Baumdarstellung von Ableitungen

- **Exkurs: Notation für (geordnete) Bäume**
  - Baum: Sammlung 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



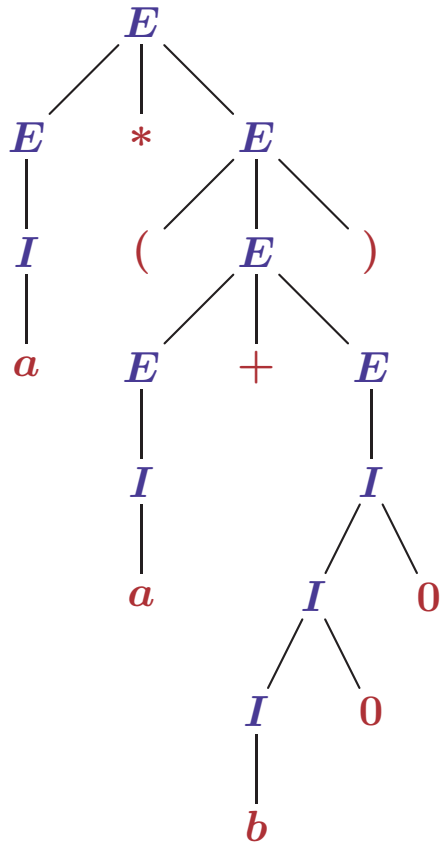
- **Ableitungsbäume sind markierte Bäume**
  - Innere Knoten mit Variablen $A \in V$ markiert
  - Wurzel markiert mit Startsymbol
  - Blätter markiert mit Terminalsymbolen $a \in T$ oder mit ϵ
  - Hat ein innerer Knoten Markierung A und Nachfolger mit Markierungen $v_1 ... v_n$, so ist $A \rightarrow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

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

$$E \longrightarrow_L E*E \longrightarrow_L I*E \longrightarrow_L a*E \longrightarrow_L a*(E)$$
$$\longrightarrow_L a*(E+E) \longrightarrow_L a*(I+E) \longrightarrow_L a*(a+E)$$
$$\longrightarrow_L a*(a+I) \longrightarrow_L a*(a+I0) \longrightarrow_L a*(a+I00)$$
$$\longrightarrow_L a*(a+b00)$$

  - Vorrang für tiefe rechte Knoten ergibt Rechtsableitung



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

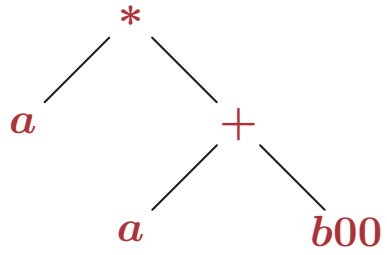
$\Rightarrow$: Konstruiere Baum induktiv aus Linksableitung von w

$\Leftarrow$: Extrahiere Linksableitung von w induktiv aus Baum

Details in HMU §5.2

Syntaxanalyse und Compilation

- **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 $+/*$

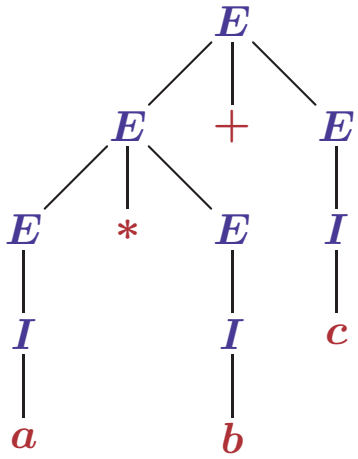


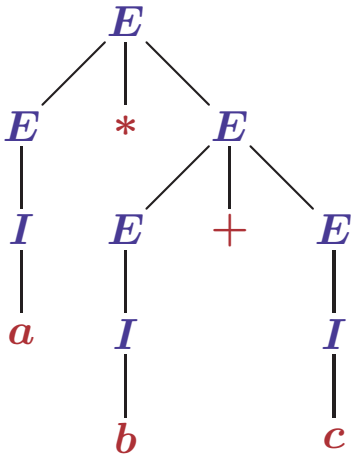
- **Compiler erzeugt Maschinencode**
  - Übersetze Identifier in Registernamen
  - $+(x, y)$: · Bestimme Wert von y und lege ihn im Register ab
    · Bestimme Wert von x
    · Addiere Registerwert
  - Coderzeugung für $*(x, y)$ analog

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 $L = L(G)$
  - Die Sprache von G_7 ist eindeutig

    ↦ nächste Folie
- **Inhärent mehrdeutige Sprache L**
  - Eine eindeutige Grammatik für L kann nicht angegeben werden
  - $\{0^i 1^j 2^k \mid i{=}j \vee j{=}k\}$ ist inhärent mehrdeutig

    ↦ Beweisskizze folgt

Programmiersprachen müssen eindeutig sein

Auflösung von Mehrdeutigkeiten

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

  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

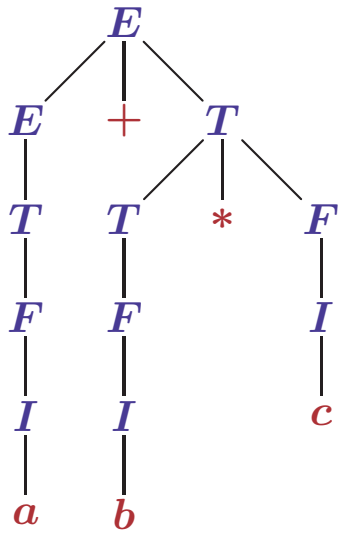
  $\boldsymbol{G'_7 = (\{E, T, F, I\},\ \{a, b, c, 0, 1, +, *, (,)\},\ P',\ E)}$
  mit $\boldsymbol{P' = \{\, E \to T \mid E{+}T,\ \ T \to F \mid T{*}F,\ \ F \to I \mid (E)}$
  $\boldsymbol{I \to a \mid b \mid c \mid Ia \mid Ib \mid Ic \mid I0 \mid I1 \,\}}$

  G'_7 ist äquivalent zu G_7 und eindeutig

Begründung der Eindeutigkeit von G'_7

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

- **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 $t_1 + t_2 + \ldots + t_k$
- **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 $f_1 * f_2 * \ldots * f_n$
- **Jeder Faktor ist Bezeichner oder geklammerter Ausdruck**



Einziger Ableitungsbaum für $a + b * c$

$L = \{0^i 1^j 2^k \mid i{=}j \vee j{=}k\}$ IST INHÄRENT MEHRDEUTIG

- **L ist Vereinigung zweier kontextfreier Sprachen**
  - $L_1 = \{0^i 1^i 2^k \mid i, k \in \mathbb{N}\}$

    $G_1 = (\{S_1, A\},\ \{0, 1, 2\},\ \{S_1 \to S_1 2, S_1 \to A, A \to 0A1, A \to \epsilon\},\ S_1)$
  - $L_2 = \{0^i 1^j 2^j \mid i, j \in \mathbb{N}\}$

    $G_2 = (\{S_2, B\},\ \{0, 1, 2\},\ \{S_2 \to 0S_2, S_2 \to B, B \to 1A2, B \to \epsilon\},\ S_2)$
  - Wörter von L_1 und L_2 haben verschiedene Ableitungsbäume
- **Manche Wörter von L gehören zu $L_1 \cap L_2$**
  - Wörter aus $L_1 \cap L_2$ haben eine Ableitung in G_1 und eine in G_2
  - Wörter aus L, die zu $L_1 \cap L_2$ gehören, haben keine eindeutige Ableitung
- **$L_1 \cap L_2 = \{0^i 1^i 2^i \mid i \in \mathbb{N}\}$ ist selbst nicht kontextfrei**

  Beweis in §3.3
  - Man kann keine einheitliche kfG zur Beschreibung von $L_1 \cap L_2$ angeben
  - Damit läßt sich auch keine bessere (eindeutige) kfG für L angeben

  Intuitives Argument. Präziser Beweis benötigt verallgemeinertes Pumping-Lemma (siehe Wegener §6.7)

Kontextfreie Grammatiken – Zusammenfassung

- **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 rekonstruktieren? (Syntaxanalyse)
  - Welche Abschlusseigenschaften gelten für kontextfreie Sprachen?
  - Welche Spracheigenschaften lassen sich nicht kontextfrei beschreiben?