Theoretische Informatik

Eduard Mehofer

Fakultät für Informatik

Währinger Straße 29

Universität Wien

Inhalt

- Formale Sprachen und kontextfreie Grammatiken
- Reguläre Grammatiken, endliche Automaten und reguläre Ausdrücke
- Chomsky Hierarchie: Sprachen, Grammatiken, Automaten
- Kellerautomaten
- Turingmaschinen
- Aussagenlogik und Prädikatenlogik
- Semantik und Verifikation
- Berechenbarkeit, Entscheidbarkeit, Komplexität

Literatur (weitere Literatur bei Kapitel)

- U. Schöning: Theoretische Informatik - kurz gefasst. Spektrum Akademischer Verlag, 5. Aufl. 2008.

- G. Vossen, K.-U. Witt: Grundkurs Theoretische Informatik. Vieweg+Teubner Verlag, 5. Aufl. 2011.

- J. E. Hopcroft, R. Motwani, J. D. Ullman: Einführung in die Automatentheorie, Formale Sprachen und Komplexität. Pearson Studium, 3. Aufl. 2011.
  Auch auf engl. verfügbar: Introduction to Automata Theory, Languages and Computation.

Anregungen zu Sprachen (1)

Problemstellungen:

- Wie läßt sich die Syntax (Form) einer Programmiersprache präzise beschreiben?
- Wie läßt sich die Semantik (Bedeutung) einer Programmiersprache beschreiben?
- Wie läßt sich die Korrektheit eines Programms nachweisen?
- Übersetzung eines Programms in Sprache A nach (einem semantisch äquivalenten Programm in) Sprache B

Anregungen zu Sprachen (2)

Eine Sprachbeschreibung erfolgt i.w. in 3 Ebenen:

1. Lexikalische Ebene
   - Definiert Schreibweise: Rechtschreibung.
   - Deutsch: Duden-Rechtschreibung (alle Worte),
     C++: Liste der Keywords, Zahlenformate, etc.

2. Syntaktische Ebene
   - Definiert wohlgeformte Sätze: Grammatik.
   - Englisch: Wordorder “Subject Predicate Object”,
     C++: Syntaxregeln für Schleifen, etc.

3. Semantische Ebene
   - Definiert Bedeutung der Sätze.
   - C++: Sortieren von Zahlen

Anregungen zu Sprachen (3)

Beispiele:

1. Deutsch: “Hörsal.” - lexikalisch falsch.

2. Deutsch: “Hund Mauer schnell.” - lexikalisch richtig aber syntaktisch falsch.

3. C++: “while; if(;;) for @fred” - lexikalisch falsch.

4. C++: “while; if(;;) ffor fred” - lexikalisch richtig aber syntaktisch falsch.

5. Deutsch: “Ein Baum ist ein Säugetier.” - syntaktisch richtig aber semantisch falsch.

6. C++: “float I; I=’’Hello”; I=3.14; I++;’’ - syntaktisch richtig aber semantisch falsch.

7. Deutsch: “Ein Hund ist ein Säugetier.” - semantisch korrekt.

Anregungen zu Sprachen (4)

Semantik von Programmiersprachen

- Semantik:
  - Laufzeitverhalten von Programmen
  - Semantikregeln beschreiben Bedeutung von Programmen
- Semantik kann in statische Semantik und dynamische Semantik unterteilt werden
- statische Semantik
  - Semantikregeln, die vom Compiler überprüft werden
  - Typüberprüfungen sind ein wichtiger Teil
- dynamische Semantik
  - Semantikregeln, die zur Laufzeit überprüft werden
  - Effekte der Ausführung - “Ausführungssemantik“

Sprachen und kontextfreie Grammatiken

Definition: Ein **Alphabet** ist eine endliche, nichtleere Menge von Symbolen (Zeichen).

Beispiele:

$\{0, 1\}$	Binärzeichen
$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$	Dezimalziffern
$\{a, b, c, \ldots, z\}$	Kleinbuchstaben
$\{a, \ldots, z, 0, \ldots, 9\}$	Alphanumerische Zeichen

Definition:

Sei Σ ein Alphabet. Eine **Zeichenkette** (Wort, String) über Σ ist eine endliche Sequenz von Symbolen aus Σ und lässt sich rekursiv wie folgt spezifizieren:

1. ε ist eine Zeichenkette über Σ.
   ε heißt die **leere Zeichenkette** (Leerwort).

2. Ist x eine Zeichenkette über dem Alphabet Σ und $a \in \Sigma$, dann ist auch xa eine Zeichenkette über Σ.

3. Jede Zeichenkette über Σ wird ausschließlich durch Anwendung der Regeln 1 und 2 gebildet.

Beispiele für Zeichenketten

Bemerkung: Nicht alle dieser Zeichenketten müssen eine Bedeutung haben, siehe die beiden untersten Beispiele.

- **Zeichenketten über dem Binäralphabet {0, 1}:**
  ε, 0, 1, 00, 01, 10, 11, 000, 001, ...

- **Zeichenketten über dem Alphabet der Kleinbuchstaben:**
  haus, baum, uni, inu, muab, ...

- **Zeichenketten über dem unären Alphabet {I}: ε, I, II, III, IIII, ...**

- **Zeichenketten über dem Alphabet {a,...,z, 0, ... 9, :, =,+,∗, (,), ... }**
  a:=b∗(c+∗(d-e)∗3.141592), abc+)))(x

- **Zeichenketten über dem Alphabet der römischen Ziffern {M, D, C, L, X, V, I}: MCMXCV, MM, IIICCCMMM**

Zeichenkette: Notationen

Sei Σ ein beliebiges Alphabet.

- Für ein beliebiges Zeichen $a \in \Sigma$ definiert $a^0 = \varepsilon$, $a^1 = a$ und $a^n = \overbrace{a \ldots a}^{\text{n-mal}}$ ('a' n-fach wiederholt) für $n \geq 1$.
- Die Länge einer Zeichenkette x wird mit $|x|$ bezeichnet.

Es gilt:

$$|x| = \begin{cases} 0, \text{ falls } x = \varepsilon \\ n, \text{ falls } x = a_1 a_2 \ldots a_n, \ n > 0 \end{cases}$$

Alphabet: Notationen

Sei Σ ein beliebiges Alphabet.

- Dann bezeichnet die Potenz eines Alphabets Σ^k , $k \geq 0$, die Menge aller Zeichenketten der Länge k, wobei $\Sigma^0 = \{\varepsilon\}$ für jedes beliebige Alphabet.

  Beispiel: $\Sigma = \{0,1\}$
  $\Sigma^0 = \{\varepsilon\}$, $\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00,01,10,11\}$, $\Sigma^3 = \{000,001,\ldots\}$, ...

- Σ^* bezeichnet die Menge aller Zeichenketten über dem Alphabet Σ, d.h. $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$

- Σ^+ bezeichnet die Menge aller Zeichenketten über dem Alphabet Σ ohne dem Leerwort, d.h. $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

- Somit gilt: $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$

Konkatenation von Zeichenketten

Definition: Sei Σ ein Alphabet und seien x, y Zeichenketten über Σ, wobei $x = a_1a_2\ldots a_m$, $y = b_1b_2\ldots b_n$, $m, n \geq 0$, $a_i, b_i \in \Sigma$.
Die **Konkatenation** (Verkettung) von x und y ist die Zeichenkette

$$xy = a_1a_2\ldots a_m b_1b_2\ldots b_n.$$

Spezialfall: Für jede Zeichenkette x gilt: $x\varepsilon = \varepsilon x = x$,
d.h. ε ist das neutrale Element für Konkatenation.

Definition: Sei $x \in \Sigma^*$ mit $x = a_1a_2\ldots a_m$ und $m > 0$.

- Jedes Wort $x' = a_1\ldots a_k$ $(0 \leq k \leq m)$ ist ein **Präfix** von x.
- Jedes Wort $x'' = a_k\ldots a_m$ $(0 \leq k \leq m)$ ist ein **Suffix** von x.

Bei dieser Definition sind ε und x sowohl Präfix als auch Suffix von x.
Ist $x' \neq x$ bzw. $x'' \neq x$, so sprechen wir von einem **echten** Präfix bzw. **echten** Suffix von x.

Für die Konkatenation von Zeichenketten gilt das Assoziativgesetzt:

$$\forall_{x,y,z \in \Sigma^*}: (xy)z = x(yz)$$

Definition: Sei x, y, z $\in \Sigma^*$.

Dann ist y ein Teilwort von xyz.
Falls y $\neq \varepsilon$, spricht man von einem echten Teilwort.

Sprache

Definition: Sei Σ ein Alphabet.
Eine Sprache L über Σ ist eine Teilmenge $L \subseteq \Sigma^*$.

Man beachte eine Sprache kann aus einer unendlichen Menge von Zeichenketten bestehen, Σ selbst jedoch ist endlich.

Zum Beispiel sind bei geeigneter Wahl des Alphabets Σ: C, Pascal, Fortran, Java, Deutsch und Chinesisch Sprachen.

Die *Syntax* (Form) einer Programmiersprache wird in der Regel durch eine kontextfreie Grammatik beschrieben.

Die *Semantik* einer Programmiersprache ordnet den Elementen der Sprache eine Bedeutung zu. Dies kann auf viele verschiedene Arten durchgeführt werden (z.B. formale Systeme wie operationale Semantik, axiomatische Semantik, denotationale Semantik oder auch mittels Prosatext).

Kontextfreie Grammatiken (1)

Zur Beschreibung einer Sprache benötigt man 4 Komponenten

1) N: Menge von Nonterminalsymbole oder Variablen, die für eine Menge von Zeichenketten stehen.

2) Σ: Menge von Terminalsymbole — das Alphabet der Sprache.

3) Regeln oder Produktionen, mit denen man rekursiv die Sprache definieren kann.

4) Eine Startvariable, mit der man mit den Produktionen beginnt.

Kontextfreie Grammatiken (2)

Definition: Eine **kontextfreie Grammatik (KFG)** ist ein Quadrupel

$$\boxed{G = (N, \Sigma, P, S)}$$

wobei

- N: Menge der **Nichtterminalsymbole**, endlich
- Σ: Menge der **Terminalsymbole**, endlich
- $P \subseteq N \times (N \cup \Sigma)^*$: Menge der **Regeln**, endlich (x … kartesisches Produkt)
- $S \in N$: **Startsymbol** ($\cup$ und $\cap$: Vereinigung und Durchschnitt von 2 Mengen) ($\subseteq$ Teilmenge)

Wir verlangen $N \cap \Sigma = \phi$ und führen zusätzlich ein:

- $\Gamma := N \cup \Sigma$: Gesamtalphabet

KFG: Notationen

- A, B, C ... bezeichnen Nichtterminalsymbole ($\in N$)
- a, b, c ... bezeichnen Terminalsymbole ($\in \Sigma$)
- α, β, γ ... bezeichnen Worte über dem Gesamtalphabet ($\in \Gamma^*$)

- Regeln sind geordnete Paare der Form $(A, \alpha) \in P$. Sie werden aus Gründen der einfacheren Lesbarkeit meistens als $A \rightarrow \alpha$ geschrieben.

- $(A,\alpha_1), (A,\alpha_2), \ldots, (A, \alpha_n)$, wird zusammengefasst zu $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$.

- Jede Regel aus *P* erhält eine eineindeutige Nummer zwischen 1 und $p = |P|$.

Kontextfreie Grammatiken: Beispiele (1)

Beispiel 1: Gesucht ist eine Grammatik für die Sprache bestehend aus mindestens einem Pluszeichen, i.e. $\underline{+++\ldots+}$, n-mal mit $n>0$.

A) $G = (N, \Sigma, P, S)$ mit :

i) $N = \{S\}$, $\Sigma = \{+\}$ und $P = \{S \to +S,\ S \to +\}$ *Rechtsrekursion*

oder

ii) $N = \{S\}$, $\Sigma = \{+\}$ und $P = \{S \to S+,\ S \to +\}$ *Linksrekursion*

B) Oder etwas komplizierter mit $G = (N, \Sigma, P, S)$ wobei

$N = \{S, A\}$, $\Sigma = \{+\}$ und $P = \{\ 1: S \to +A,\ 2: A \to +A,\ 3: A \to \varepsilon\ \}$

Ableitung: $S \overset{1}{\Rightarrow} +A \overset{2}{\Rightarrow} ++A \overset{2}{\Rightarrow} +++A \overset{3}{\Rightarrow} +++$

Das heißt, eine Grammatik ist nicht eindeutig, sondern es gibt unterschiedliche Lösungen.

Kontextfreie Grammatiken: Beispiele (2)

Beispiel 2: Gesucht ist eine Grammatik für die Sprache aller Binärworte beliebiger Länge größer Null, i.e. die Sprache $\{0,1\}^* - \{\varepsilon\}$ bzw. $\{0,1\}^+$.

$G = (N, \Sigma, P, S)$ mit:

$N = \{S\}$, $\Sigma = \{0,1\}$ und $P = \{1\colon S \rightarrow 0,\ 2\colon S \rightarrow 1,\ 3\colon S \rightarrow 0S,\ 4\colon S \rightarrow 1S\ \}$

Die Rekursionen 3 und 4 erzeugen beliebige Länge, die Regeln 1 und 2 leiten terminal ab und garantieren eine Länge größer Null.

Ableitung $S \overset{4}{\Rightarrow} 1S \overset{4}{\Rightarrow} 11S \overset{3}{\Rightarrow} 110S \overset{4}{\Rightarrow} 1101S \overset{3}{\Rightarrow} 11010S \overset{2}{\Rightarrow} 110101$

110101 ist ein Satz der von *G* erzeugten Sprache.

Ableitungen

Definition: Sei $G = (N, \Sigma, P, S)$

1. Seien $\alpha, \beta \in \Gamma^*$.
   $\alpha \Rightarrow \beta$ genau dann, genau dann wenn: $\alpha = \alpha_1 A \alpha_3$ und $\beta = \alpha_1 \alpha_2 \alpha_3$ mit $A \to \alpha_2 \in P$ und $\alpha_1, \alpha_2, \alpha_3 \in \Gamma^*$, $A \in N$.

   Man sagt α leitet ß direkt ab oder ß lässt sich in einem Ableitungsschritt aus α ableiten.

2. Eine Folge $(\alpha_i)^n_{i=0}$ mit $\forall i, 0 \leq i < n : \alpha_i \Rightarrow \alpha_{i+1}$ heißt Ableitung der Länge n von α_0 nach α_n $(n \geq 0)$.

Notationen:

Zeichen	Schritte	Zeichen	Schritte
$\Rightarrow$	1	$\overset{n}{\Rightarrow}$	$n \geq 1$
$\overset{+}{\Rightarrow}$	≥ 1	$\overset{*}{\Rightarrow}$	≥ 0

Erzeugte Sprache

Definition:
Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik.

1. Ein Wort $\alpha \in \Gamma^*$ mit $S \overset{*}{\Rightarrow} \alpha$ heißt **Satzform** von G. Eine Satzform α mit $\alpha \in \Sigma^*$ heißt **Satz** von G.

2. Die von G **erzeugte Sprache** ist die Menge aller Sätze von G:

$$L(G) := \{w \mid w \in \Sigma^* \wedge S \overset{+}{\Rightarrow} w\}.$$

3. L(G) ist eine **kontextfreie Sprache** oder **Chomsky-Typ-2 Sprache.**

Definition:
Seien G_1 und G_2 Grammatiken. G_1 und G_2 heißen **äquivalent** genau dann, wenn $L(G_1) = L(G_2)$.

Bemerkung: Zu jeder Grammatik gibt es unendlich viele äquivalente Grammatiken.

Ableitungsbäume

Definition: Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik (KFG). Ein Ableitungsbaum $T = (V, E, v_0)$ mit Knotenmenge V, Kantenmenge E, und Wurzel v_0, ist wie folgt definiert:

- Die Wurzel v_0 ist mit S markiert.
- Jeder innere Knoten ist mit einer Variablen aus N markiert.
- Jedes Blatt ist mit einem Symbol aus $\Sigma \cup \{\varepsilon\}$ markiert.
- Wenn ein innerer Knoten mit A markiert ist und die Kindknoten mit $A_1,\ldots,A_k$ markiert sind, so ist

$$A \to A_1 \ldots A_k \in P.$$

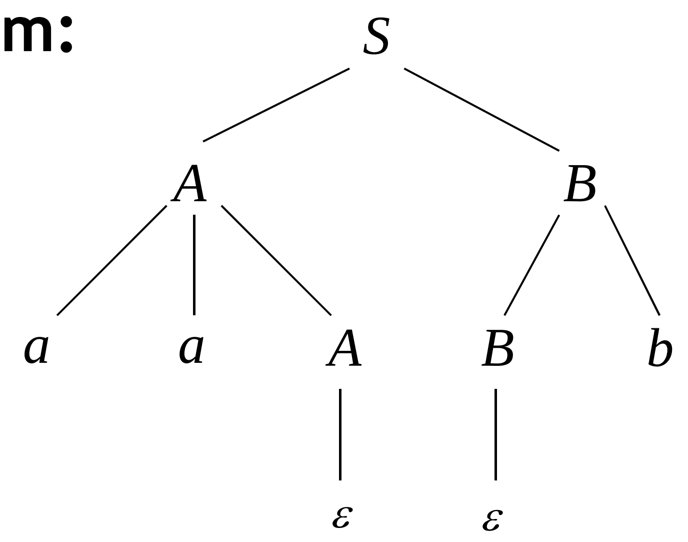
Das Wort w, das mit einer Ableitung $S \overset{*}{\Rightarrow} w$ erzeugt worden ist, kann man konstruieren, indem man alle Blätter des zugehörigen Ableitungsbaumes von links nach rechts durchläuft.

Beispiel: Ableitung und Ableitungsbaum

- **G = ({S,A,B}, {a,b}, P, S) mit**
  P = { S → AB, A → aaA | ε, B → Bb | ε }.
- **Ableitung:**

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaABb \Rightarrow aaBb \Rightarrow aab$$

- **Ableitungsbaum:**



- Wie lautet L(G)? L(G) = { $a^{2m}b^{n}$ | m,n≥ 0 }

Beispiel

$G = (N, \Sigma, P, S)$ mit $N = \{S, Z\}$, $\Sigma = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$ und
$P = \{ 0: Z \to 0,\ 1: Z \to 1,\ \ldots,\ 9: Z \to 9,\ 10: S \to Z,\ 11: S \to SZ \}$

Aus G lassen sich alle vorzeichenlosen, ganzen Zahlen beliebiger Länge erzeugen, z.B. 1995:

$$S \overset{11}{\Rightarrow} SZ \overset{5}{\Rightarrow} S5 \overset{11}{\Rightarrow} SZ5 \overset{9}{\Rightarrow} S95 \overset{11}{\Rightarrow} SZ95 \overset{9}{\Rightarrow} S995 \overset{10}{\Rightarrow} Z995 \overset{1}{\Rightarrow} 1995$$

Beispiel

$G = (N, \Sigma, P, S)$ mit $N = \{S, B, Z\}$, $\Sigma = \{a, b, \ldots, z, 0, 1, \ldots, 9\}$ und
$P = \{\ 1: B \to a,\ \ 2: B \to b,\ \ldots,\ 26: B \to z,$
$27: Z \to 0,\ 28: Z \to 1,\ \ldots,\ 36: Z \to 9,$
$37: S \to B,\ 38: S \to SB,\ 39: S \to SZ \}$

Aus G lassen sich alle Bezeichner (Identifier) beliebiger Länge erzeugen, z.B. a oder a1c2:

$$S \overset{37}{\Rightarrow} B \overset{1}{\Rightarrow} a$$

$$S \overset{39}{\Rightarrow} SZ \overset{29}{\Rightarrow} S2 \overset{38}{\Rightarrow} SB2 \overset{3}{\Rightarrow} Sc2 \overset{39}{\Rightarrow} SZc2 \overset{28}{\Rightarrow} S1c2 \overset{37}{\Rightarrow} B1c2 \overset{1}{\Rightarrow} a1c2$$

Kontrollwort einer Ableitung

Definition: Sei G eine Grammatik und $(\alpha_i)^n_{i=0}$ eine Ableitung in G. Weiters sei $A \to \gamma \in P$ die Regel mit der Nummer p_i, deren Anwendung den Übergang $\alpha_i \stackrel{p_i}{\Rightarrow} \alpha_{i+1}$ bewirkt. Dann heißt $\pi = (p_0,\ldots,p_n)$ das Kontrollwort der Ableitung .

Beispiel:

Wir beziehen uns auf die vorhin definierte Grammatik zur Erzeugung von Bezeichnern.

$$S \stackrel{*}{\Rightarrow} a1c2, \text{ mit}$$

$$\pi = (39, 29, 38, 3, 39, 28, 37, 1)$$

Um das Kontrollwort eindeutig zu machen und die Wahlmöglichkeiten bei Ableitungen auszuschließen, gibt es Links- bzw. Rechtsableitung.

Linksableitungen und Rechtsableitungen

- Wenn bei jedem Ableitungsschritt immer die am weitesten links stehende Variable ersetzt wird, sprechen wir von einer Linksableitung. Analog wird bei einer Rechtsableitung immer die am weitesten rechts stehende Variable ersetzt, wie die folgende Definition besagt.

Def.: Sei G = (N,Σ,P,S) und gelte $\alpha \Rightarrow \beta$ mit $\alpha=\alpha_1 A\alpha_3$, $\beta=\alpha_1\alpha_2\alpha_3$, und $A\rightarrow\alpha_2 \in P$.

1. $\alpha \Rightarrow \beta$ heißt ein Linksableitungsschritt, in Zeichen $\alpha \underset{L}{\Rightarrow} \beta$, gdw $\alpha_1 \in \Sigma^*$.
2. $\alpha \Rightarrow \beta$ heißt ein Rechtsableitungsschritt, in Zeichen $\alpha \underset{R}{\Rightarrow} \beta$, gdw $\alpha_3 \in \Sigma^*$.
3. Eine Ableitung heißt Linksableitung, wenn jeder Schritt ein Linksableitungsschritt ist.
4. Eine Ableitung heißt Rechtsableitung, wenn jeder Schritt ein Rechtsableitungsschritt ist.

Mehrdeutigkeit einer Grammatik

Definition:

- Eine KFG heißt eindeutig, wenn für jeden Satz $w \in L(G)$ mit beliebigen Ableitungen von w aus S gilt: die mit den Ableitungen verbundenen Ableitungsbäume sind identisch.

  Anderenfalls heißt eine KFG mehrdeutig.

Bemerkungen:

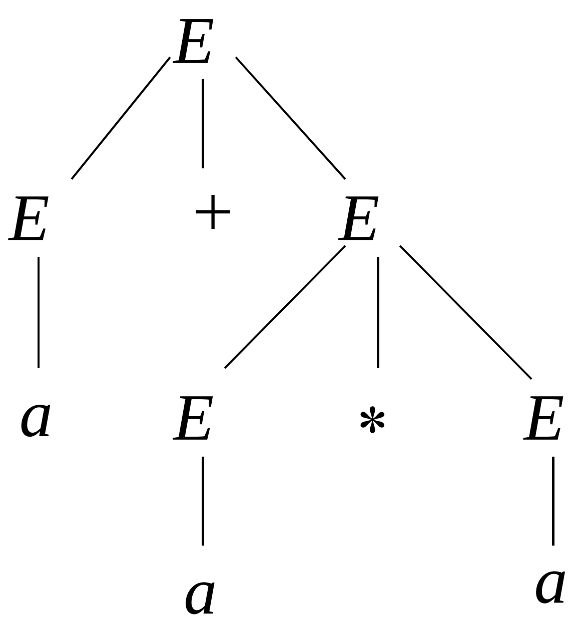
- Ist G mehrdeutig, dann gibt es immer ein $w \in L(G)$ und zwei Ableitungen von w, die unterschiedliche Ableitungsbäume erzeugen.

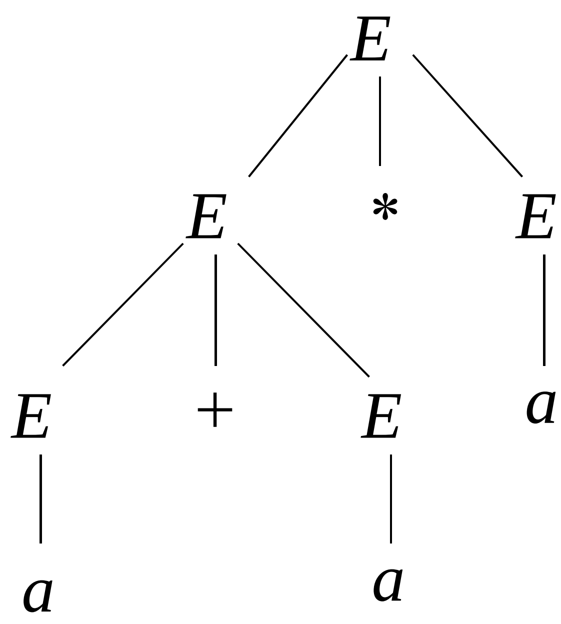
- In der Praxis von Programmiersprachen werden mehrdeutige Grammatiken nicht benutzt, da sie zu Problemen bei der Analyse führen.

- Eine kontextfreie Sprache L heißt inhärent mehrdeutig gdw. alle KFG für L mehrdeutig sind.

Beispiel: Arithmetischer Ausdruck (1)

- G = ({E}, {+,∗,(,),a}, P, E } mit
  P = { E → E + E | E ∗ E | (E) | a }
- Für Wort a+a∗a existieren 2 Ableitungsbäume.

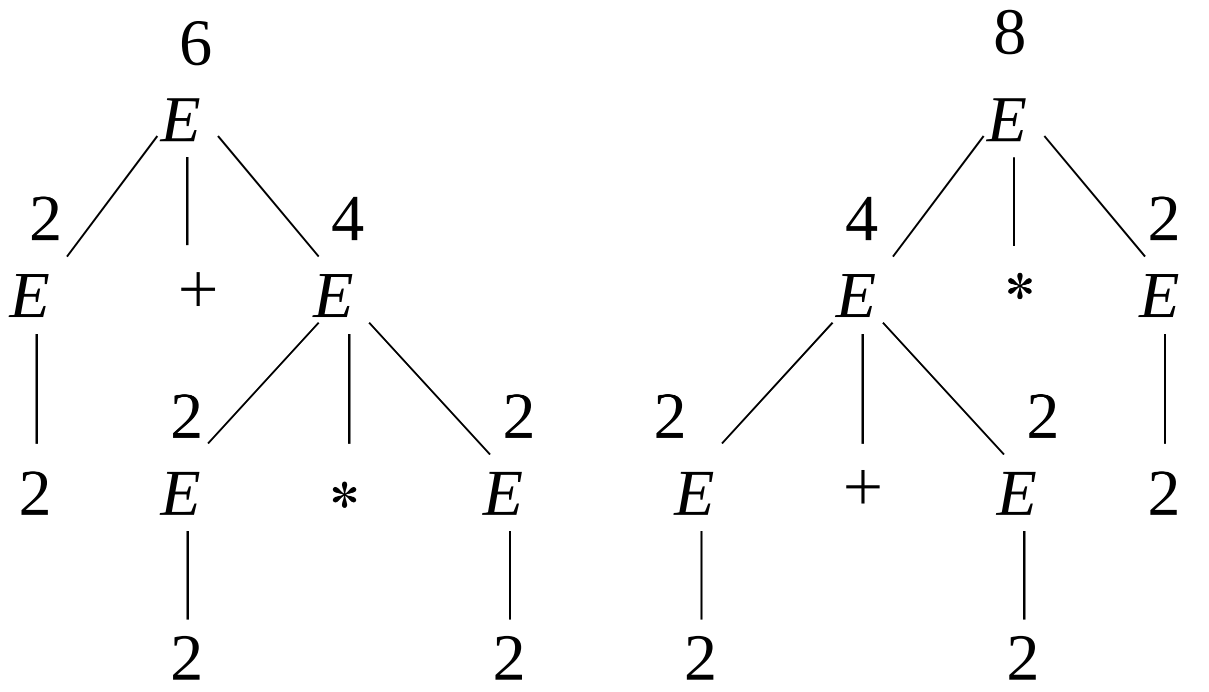




Beispiel: Arithmetischer Ausdruck (2)

- Setzen wir a=2 und berechnen den Ausdruck.

$2+2*2=6$ $\quad\quad$ $2+2*2=8$



Beispiel: Arithmetischer Ausdruck (3)

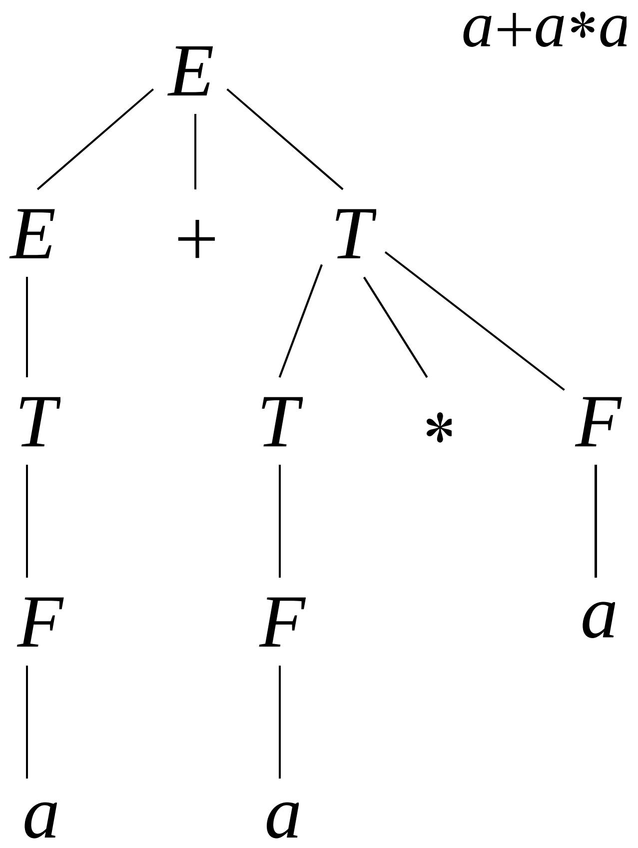
- **Um diese Mehrdeutigkeit zu beheben und der Multiplikation eine höhere Priorität als der Addition zu geben, formulieren wir die Grammatik folgendermaßen um:**
  G = ({E,T,F}, {+,∗,(,),a}, P, E) mit

$$P = \{1: E \to E + T, 2: E \to T, 3: T \to T * F, \\ 4: T \to F, 5: F \to (E), 6: F \to a\}$$

Die Ableitung für a+a∗a ist nun eindeutig und liefert das korrekte Resultat.

Linksableitung:

$$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \\ \Rightarrow a + T * F \Rightarrow a + F * F \Rightarrow a + a * F \\ \Rightarrow a + a * a$$



Beispiel: Arithmetischer Ausdruck (4)

Beispiel: Zwei Ableitungen des Satzes a ∗ (a + a)

$$E \overset{2}{\Rightarrow} T \overset{3}{\Rightarrow} T * F \overset{4}{\Rightarrow} F * F \overset{6}{\Rightarrow} a * F \overset{5}{\Rightarrow} a * (E) \overset{1}{\Rightarrow} a * (E + T) \overset{2}{\Rightarrow}$$
$$a * (T + T) \overset{4}{\Rightarrow} a * (F + T) \overset{6}{\Rightarrow} a * (a + T) \overset{4}{\Rightarrow} a * (a + F) \overset{6}{\Rightarrow} a * (a + a)$$

Dies ist eine **Linksableitung**: bei jedem Schritt wird das am weitesten *links* stehende Nichtterminalsymbol ersetzt.

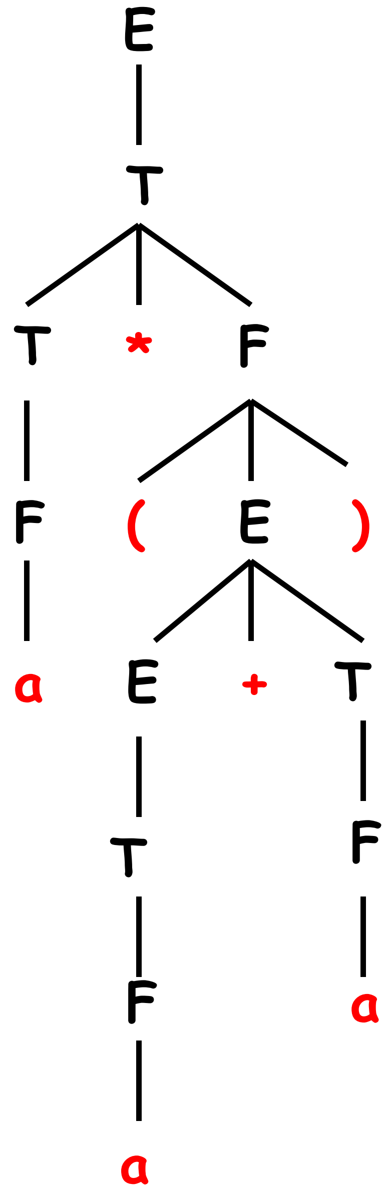
$$E \overset{2}{\Rightarrow} T \overset{3}{\Rightarrow} T * F \overset{5}{\Rightarrow} T * (E) \overset{1}{\Rightarrow} T * (E + T) \overset{4}{\Rightarrow} T * (E + F) \overset{6}{\Rightarrow}$$
$$T * (E + a) \overset{2}{\Rightarrow} T * (F + a) \overset{6}{\Rightarrow} T * (a + a) \overset{4}{\Rightarrow} F * (a + a) \overset{6}{\Rightarrow} a * (a + a)$$

Dies ist eine **Rechtsableitung**: bei jedem Schritt wird das am weitesten *rechts* stehende Nichtterminalsymbol ersetzt.

Das Kontrollwort einer Links (Rechts)ableitung heißt Links (Rechts)kontrollwort (LKW bzw. RKW). Analog: Links (Rechts)satzform.

Beispiel: Arithmetischer Ausdruck (5)

Ableitungsbaum für $E \overset{*}{\Rightarrow} a * (a + a)$



Beispiel Palindrom:

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

$N = \{ S \}$, $\Sigma = \{ a, b \}$

$P = \{ S \to aSa \mid bSb \mid a \mid b \mid \varepsilon \}$

erzeugt die Sprache der Palindrome.

(z.B. otto, pop, madam)

Backus-Naur Form

Programmiersprachen werden häufig in Backus-Naur-Form (BNF) beschrieben (leichter lesbar).

In BNF wird jedes Nichtterminalsymbol in spitze Klammern <...> eingeschlossen, anstelle von '→' wird das Symbol '::=' verwendet und Terminalsymbole werden fett gedruckt

Beispiel BNF: <while> ::= **while (** <expression> **)** <statement>

Weitere BNF Notationen:.

- **[...] für optional:**
  z.B. <if-stmt> ::= **if** <condition> **then** <statement> [**else** <statement>]

- **... für Wiederholungen:**
  z.B. <identifier> ::= <letter>...

ISO C Standard - Beispiel

Syntax und Semantik bei Programmiersprachen.

Auszug aus ISO-C Standard.

Syntax und Semantik einer while-Schleife.

Syntax

iteration-statement **:**

while (*expression* **)** *statement*

Constraints

The controlling expression of an iteration statement shall have scalar type.

Semantics

- **An iteration statement causes a statement called the loop body to be executed repeatedly until the controlling expression compares equal to 0.**
- **The evaluation of the controlling expression takes place before each execution of the loop body.**

Sprachbeschreibungen allgemein

Lexikalische Ebene:

- Beschreibung mittels regulärer Ausdrücke.

Syntaktische Ebene:

- Beschreibung mit Grammatiken.
- Linguist Noam Chomsky definierte eine 4-stufige Hierarchie von Klassen von Grammatiken (1956,1959).
- Zur Beschreibung von Programmiersprachen sind vor allem kontextfreie Grammatiken von Bedeutung.

Semantische Ebene:

- Beschreibung in Prosa.
- axiomatische Semantik.
- wp-Semantik.