Sprachen, Grammatiken und Automaten

Eduard Mehofer

Fakultät für Informatik

Währinger Straße 29

Universität Wien

Inhalt

- **Formale Sprachen und kontextfreie Grammatiken**
  (unser Interesse gilt formalen Sprachen im Gegensatz zu natürlichen Sprachen wie der deutschen Sprache)
- **Reguläre Grammatiken und reguläre Ausdrücke**
- **endliche Automaten**
- **Chomsky Hierarchie: Beziehung Sprachen, Grammatiken, Automaten**
- **Kellerautomaten**
- **Turingmaschinen**

Literatur

- Ein Klassiker:
  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.

  wesentlich detaillierter als in Vorlesung

- viele freie Ressourcen

Anregungen zu Sprachen (1)

Problemstellungen:

- **Wie lässt sich die Syntax (Form) einer formalen Sprache (Beispiel Programmiersprache) präzise beschreiben?**
- **Wie lässt sich die Semantik (Bedeutung) einer Programmiersprache beschreiben?**
- **Problem Übersetzung eines Programms in Sprache A nach (einem semantisch äquivalenten Programm in) Sprache B**
- **Problem der Korrektheit eines Programms**

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." -** lexikalisch/syntaktisch richtig aber semantisch falsch.

6. **C++: "float I; I=''Hello"; I=3.14; I++;'' -** lexikalisch/syntaktisch richtig aber semantisch falsch.

7. **Deutsch: "Ein Hund ist ein Säugetier." -** auch semantisch korrekt.

Anregungen zu Sprachen (4)

Semantik von Programmiersprachen

- **Semantik:**
  - Laufzeitverhalten von Programmen
  - Semantikregeln beschreiben Bedeutung von Programmkonstrukten
- **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“

Wiederholung Mengen (1)

Eine **Menge** ist eine Zusammenfassung von wohlunterschiedenen Objekten. Diese Objekte werden als **Elemente** der Menge bezeichnet.

Verschiedene Arten der Mengenangabe**:**

- **aufzählendes Verfahren**:
  - Es werden die Elemente durch Beistriche getrennt genau einmal angeführt.
  - Beispiel: $A = \{1,2,3\}$

- **beschreibendes Verfahren**:
  - Es wird eine Eigenschaft angegeben, welche die Menge definiert.
  - Beispiel: $A = \{x \mid x \in \mathbb{N} \wedge 1 \leq x \leq 10\}$

Wiederholung Mengen (2)

Seien *A* und *B* Mengen.

- **Teilmenge:** $A \subseteq B \Leftrightarrow \forall x: (x \in A \Rightarrow x \in B)$
- **echte Teilmenge:** $A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$
- **Vereinigung:** $A \cup B := \{ x \mid x \in A \vee x \in B \}$ – *A vereinigt B*
- **Durchschnitt:** $A \cap B := \{ x \mid x \in A \wedge x \in B \}$ – *A geschnitten B*
- **Differenz:** $A - B := \{ x \mid x \in A \wedge x \notin B \}$ – *A minus B*
  oder $A \setminus B$ … *A ohne B*
- **Potenzmenge**:
  - Die Potenzmenge von *A* ist die Menge aller Teilmengen von *A*.
  - $\mathcal{P}(A) := \{ B \mid B \subseteq A \}$
  - Beispiel: $A = \{1,2,3\}$, $\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$.
- **kartesisches Produkt:**
  - $A_1 \times \cdots \times A_n := \{(a_1, \ldots, a_n) \mid a_1 \in A_1, \ldots, a_n \in A_n\}$
  - Das Ergebnis ist eine Menge von n-Tupel mit allen möglichen Kombinationen.

Tupel und Mengen (3)

Ein **Tupel** ist eine geordnete Liste von **n** Elementen und wird auch als **n-Tupel** bezeichnet. Wir symbolisieren n-Tupel durch runde Klammern und schreiben $(a_1, \ldots, a_n)$ – man vergleiche das kartesische Produkt.

Ein wichtiger Unterschied zwischen **Tupel** und **Menge** ist, dass bei einem Tupel die **Reihenfolge** wichtig ist, bei einer Menge hingegen nicht.

Gleichheit von Tupel und Mengen:

- $(a_1, \ldots, a_n) = (b_1, \ldots, b_n)$ g.d.w. $a_1 = b_1, \ldots, a_n = b_n$
- Mengen A,B: $A = B \Leftrightarrow (A \subseteq B \wedge B \subseteq A)$ i.e. keine Ordnung verlangt

Beispiele:

- Tupel: $(1,2,3) \neq (3,2,1)$
- Menge: $\{1,2,3\} = \{3,2,1\}$

Bezeichnungen für Tupel: geordnetes Paar, Tripel, Quadrupel, Quintupel, 6-Tupel, 7-Tupel, usw.

Alphabet und Sprache

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, ..., z}	Kleinbuchstaben
{a, ..., z, 0, ..., 9}	Alphanumerische Zeichen

Eine **Zeichenkette** ist eine endliche Sequenz von Symbolen eines Alphabets.

Definition:

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

1. **ε ist eine Zeichenkette über Σ.**
   ε heißt die leere Zeichenkette oder 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 Beispiele ganz unten.

- **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

Alphabet/Zeichenkette: Definitionen (1)

Sei Σ ein beliebiges Alphabet.

- Sei $a \in \Sigma$, dann gilt
  - $a^0 = \varepsilon$
  - $a^1 = a$
  - $a^n = \underbrace{a...a}_{\text{n-mal}}$ ('a' n-fach wiederholt) für $n \geq 1$.

- Die Länge einer Zeichenkette x wird mit $|x|$ bezeichnet, wobei:

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

Alphabet/Zeichenkette: Definitionen (2)

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

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

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

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

Konkatenation von Zeichenketten

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

$$x\,y = a_1a_2... a_m\, b_1b_2... b_n.$$

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

Für die Konkatenation von Zeichenketten gilt das **Assoziativgesetz**:

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

Präfix, Suffix, Teilwort

Definition: Sei $x = a_1 a_2 \ldots a_m$ mit $a_i \in \Sigma$ für $1 \leq i \leq m$ und $m > 0$.

- Jedes Wort $x' = a_1 \ldots a_k$ $(1 \leq k \leq m)$ ist ein **Präfix** von x.
- Jedes Wort $x'' = a_k \ldots a_m$ $(1 \leq k \leq m)$ ist ein **Suffix** von x.

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

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

Dann ist y ein **Teilwort** von $w = x\,y\,z$.
Falls $y \neq w$, 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 Sprachen bei geeigneter Wahl des Alphabets Σ: C, Pascal, Fortran, Java, Deutsch, Chinesisch.

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)

Eine kontextfreie Grammatik besteht aus 4 Komponenten:

1) **N: Menge von Nonterminalsymbole oder Variable, 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 definiert.**

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

Kontextfreie Grammatiken (2)

Definition: Eine kontextfreie Grammatik (KFG) ist ein Quadrupel

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

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

Wir verlangen $N \cap \Sigma = \varnothing$ 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, α) $\in$ P. Sie werden aus Gründen der einfacheren Lesbarkeit meistens als $A \rightarrow \alpha$ geschrieben.

- (A,α_1), (A,α_2), ..., (A, α_n), wird zusammengefasst zu $A \rightarrow \alpha_1 | \alpha_2 | ... | \alpha_n$.

- 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. +++…+, n-mal mit n>0.

A) G = (N, Σ, P, S) mit :

i) N = {S}, Σ = {+} und P = {S → +S, S → +} *Rechtsrekursion*

oder

ii) N = {S}, Σ = {+} und P = {S → S+, S → +} *Linksrekursion*

B) Oder etwas komplizierter mit G = (N, Σ, P, S) wobei

N = {S, A}, Σ = {+} und P = { 1: S → +A, 2: A → +A, 3: A → ε }

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 Möglichkeiten eine Sprache zu generieren.

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 $P = \{1: S \rightarrow 0,\ 2: S \rightarrow 1,\ 3: S \rightarrow 0S,\ 4: S \rightarrow 1S\ \}$

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: Gegeben sei G = (N, Σ, P, S).

1. Seien $\alpha_1, \alpha_2, \alpha_3 \in \Gamma^*, A \in N$, dann leitet $\alpha_1 A \alpha_3$ nach $\alpha_1 \alpha_2 \alpha_3$ ab, in Zeichen $\alpha_1 A \alpha_3 \Rightarrow \alpha_1 \alpha_2 \alpha_3$, wenn $A \rightarrow \alpha_2 \in P$.

   Man sagt $\alpha_1 A \alpha_3$ leitet nach $\alpha_1 \alpha_2 \alpha_3$ direkt in einem Ableitungsschritt ab.

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

Notationen:

Zeichen	Schritt	Zeichen	Schritte
$\Rightarrow$	1 (ohne Regelnr.)	$\overset{*}{\Rightarrow}$	≥ 0
$\overset{p}{\Rightarrow}$	1 (mit Regel p)	$\overset{+}{\Rightarrow}$	≥ 1

Erzeugte Sprache

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

1. Die von G **erzeugte Sprache** ist die Menge aller Worte mit:

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

2. Ein Wort $\alpha \in \Gamma^*$ mit $S \overset{*}{\Rightarrow} \alpha$ wird auch als **Satzform** von G bezeichnet. Eine Satzform α mit $\alpha \in \Sigma^*$ wird als **Satz** von G bezeichnet.

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, Σ, 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 $\mathbb{v}_1 \dots \mathbb{v}_k$ markiert sind, so ist

$$A \to \mathbb{v}_1 \dots \mathbb{v}_k \in P$$

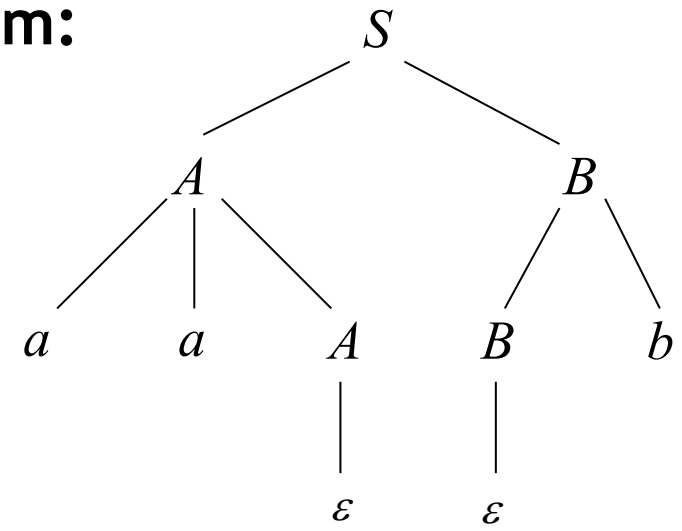
Das Wort w, das mit einer Ableitung $S \overset{*}{\Rightarrow} w$ erzeugt worden ist, kann man konstruieren, indem man alle Blätter des zugehörigen Ableitungsbaumes von links nach rechts durchläuft.

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, Σ, P, S) mit N = {S, Z}, Σ = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 } und
P = { 0: Z → 0, 1: Z → 1, ..., 9: Z → 9, 10: S → Z, 11: S → 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, Σ, P, S) mit N = {S, B, Z}, Σ = {a, b ,..., z, 0, 1,..., 9} und
P = { 1: B → a, 2: B → b, ..., 26: B → z,
27: Z → 0, 28: Z → 1, ..., 36: Z → 9,
37: S → B, 38: S → SB, 39: S → 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: Gegeben sei eine KFG G = (N, Σ, P, S) und eine Ableitung $S \overset{p_0}{\Rightarrow} \alpha_0 \overset{p_1}{\Rightarrow} \ldots \overset{p_n}{\Rightarrow} \alpha_n$, wobei p_i die Regelnummern bezeichnen. Dann heißt $\pi = (p_0, p_1, \ldots, p_n)$ das Kontrollwort der Ableitung .

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

$$S \overset{*}{\Rightarrow} a1c2, \text{ mit}$$
$$\pi = (39, 29, 38, 3, 39, 28, 37, 1)$$

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 $G = (N,\Sigma,P,S)$ und gelte $\alpha \Rightarrow \beta$ mit $\alpha=\alpha_1 A \alpha_3$, $\beta=\alpha_1\alpha_2\alpha_3$, und $A \to \alpha_2 \in P$.

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 G 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.

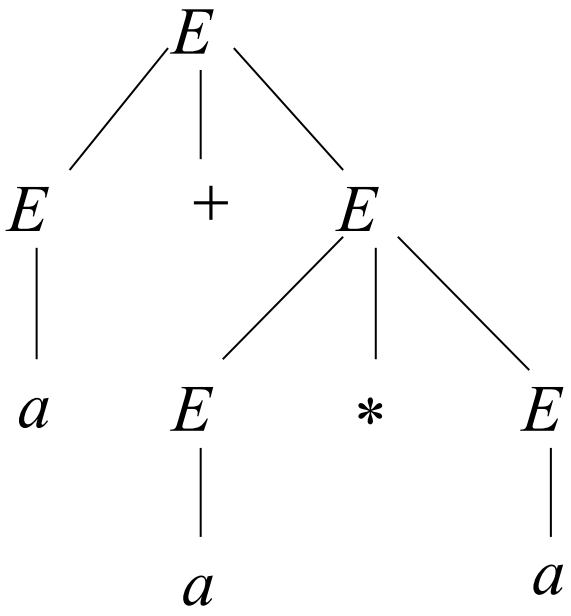
Bemerkungen:

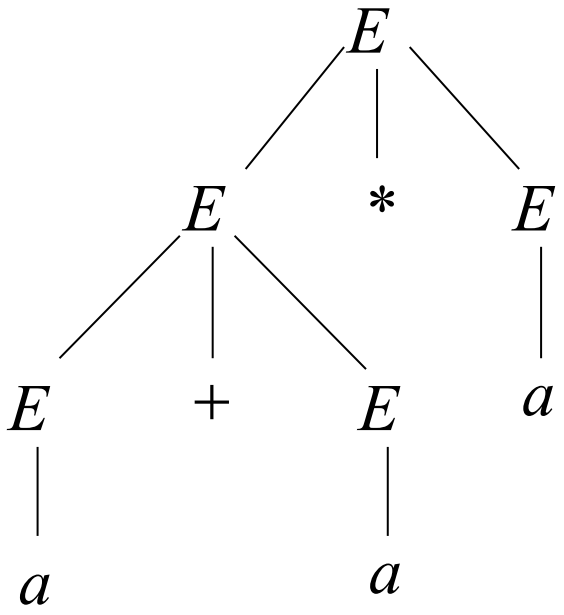
- Ist G mehrdeutig, dann gibt es mind. ein $w \in L(G)$ und zwei Ableitungen von w, die unterschiedliche Ableitungsbäume erzeugen.

- Mehrdeutige Grammatiken bei Programmiersprachen führen zu Problemen bei der Analyse.

Beispiel: Arithmetischer Ausdruck (1)

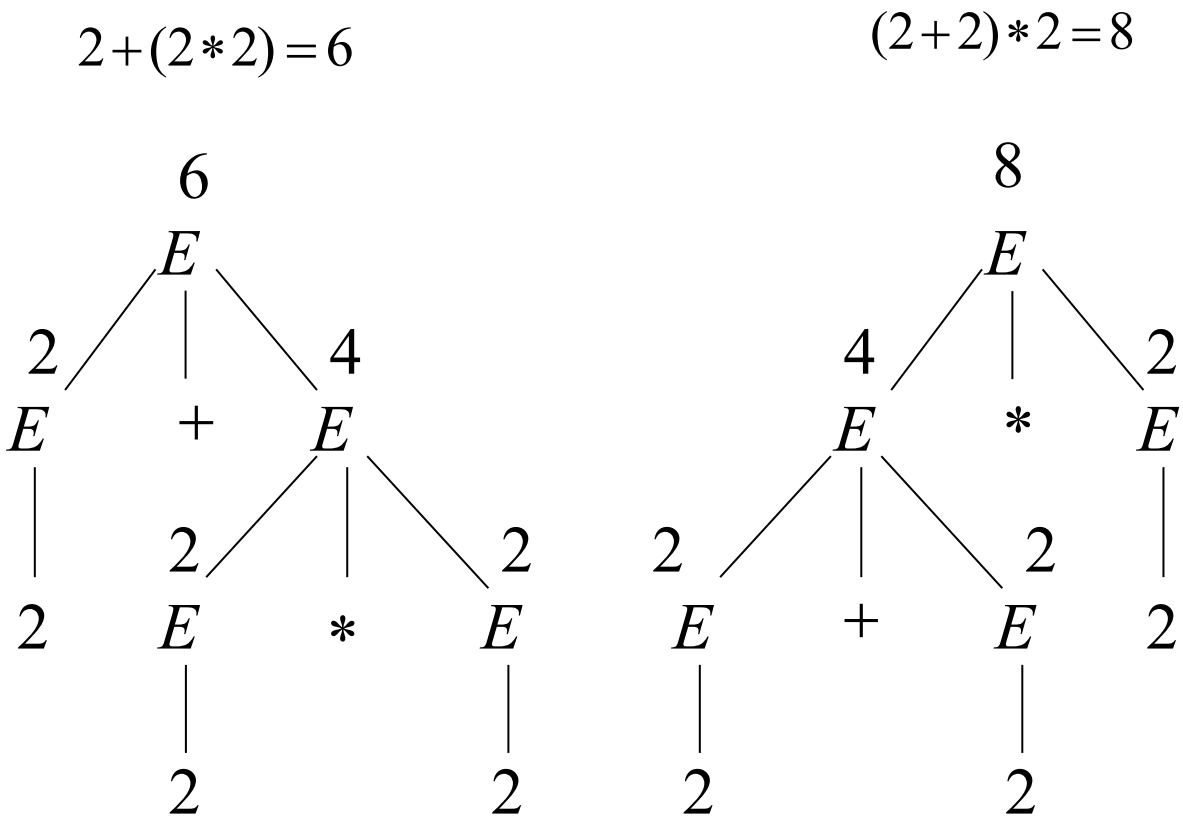
- G = ({E}, {+,*,(,),a}, P, E } mit
  P = { E → E + E | E * E | (E) | a }
- Für Wort a+a*a existieren 2 Ableitungsbäume.





Beispiel: Arithmetischer Ausdruck (2)

- Setzen wir a=2 und berechnen den Ausdruck.



Beispiel: Arithmetischer Ausdruck (3)

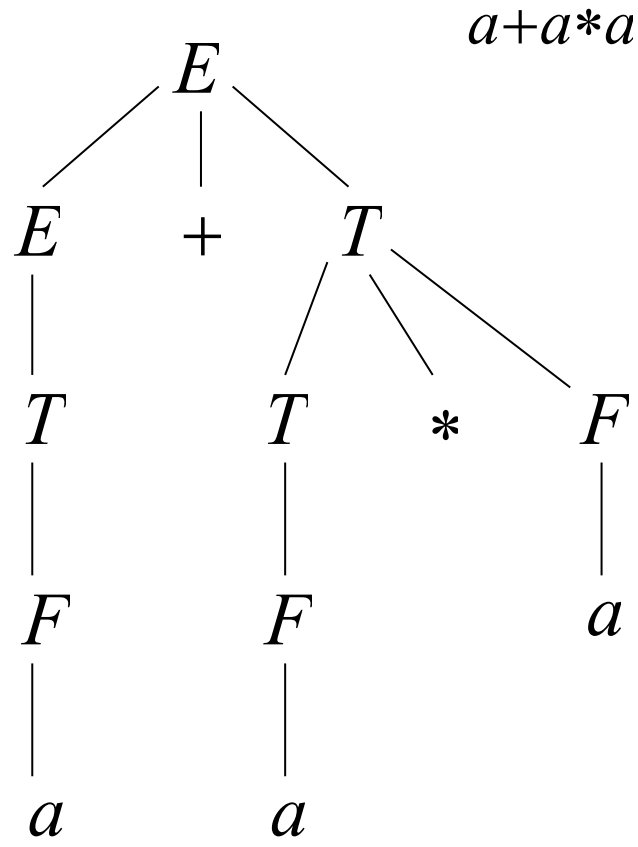
- Um diese Mehrdeutigkeit zu beheben und der Multiplikation eine höhere Priorität als der Addition zu geben, formulieren wir die Grammatik folgendermaßen um:
  G = ({E,T,F}, {+,*,(,),a}, P, E) mit

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

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.

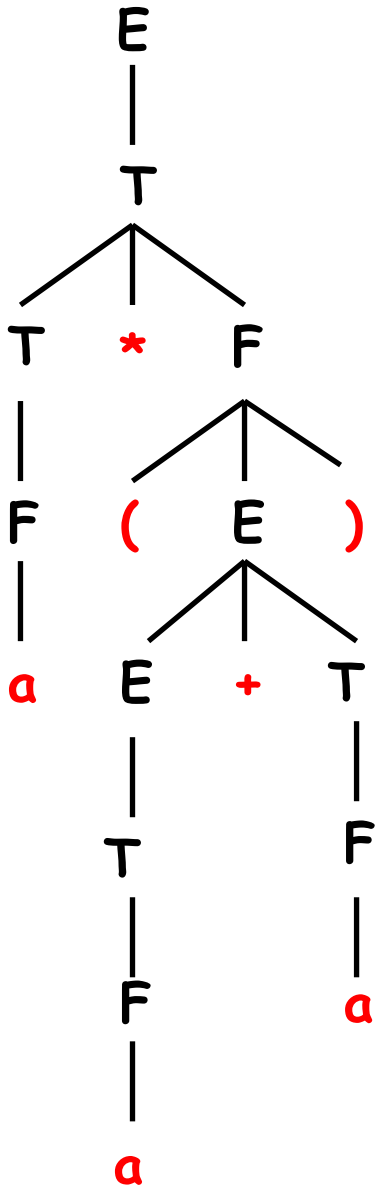
$$E \overset{2}{\Rightarrow} T \overset{3}{\Rightarrow} T * F \overset{5}{\Rightarrow} T * (E) \overset{1}{\Rightarrow} T * (E + T) \overset{4}{\Rightarrow} T * (E + F) \overset{6}{\Rightarrow}$$
$$T * (E + a) \overset{2}{\Rightarrow} T * (F + a) \overset{6}{\Rightarrow} T * (a + a) \overset{4}{\Rightarrow}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

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.

Beispiel: Arithmetischer Ausdruck (5)

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

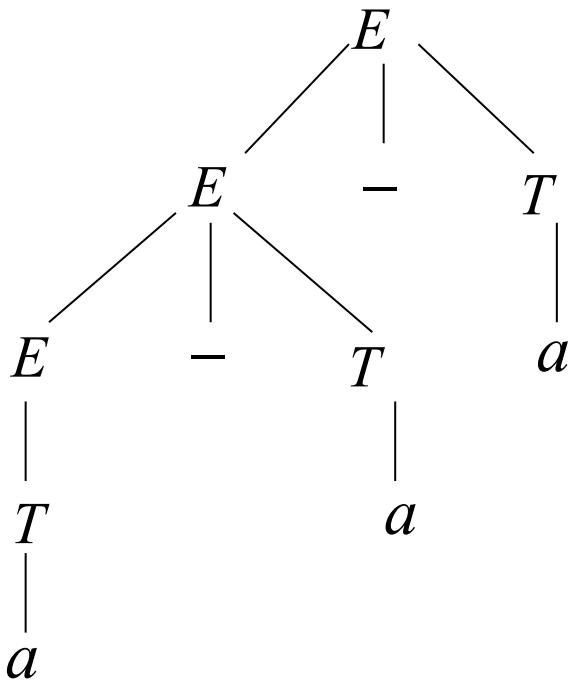


Operatoren: Assoziativität

1. **Eine Linksrekursion bewirkt Linksassoziativität.**
2. **Eine Rechtsrekursion bewirkt Rechtsassoziativität.**

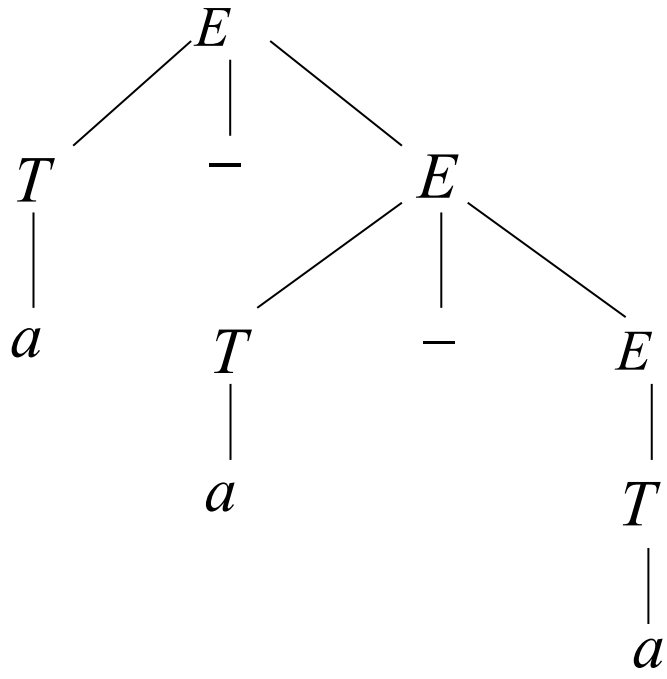
Beispiel:

$$P = \{E \to E - T \mid T, T \to a\}$$



$$linksassoziativ : a = 2, Ergebnis = -2$$

$$P = \{E \to T - E \mid T, T \to a\}$$



$$rechtsassoziativ : a = 2, Ergebnis = 2$$