

7. Syntax: Grammatiken, EBNF

Teil 2

Java-Syntax

Aspekte der Korrektheit von Programmen

EBNF für Java (14 Seiten)

Lexik:

Symbole korrekt ?

kontextfreie Syntax:

Reihenfolge der Symbole korrekt ?

```
class T1 {  
    public static main (...) {  
        { x = 2 }  
    }  
}
```

kontextabhängige Syntax:

*Symbole in die Umgebung
korrekt eingebunden?*

Semantik:

- Keine Laufzeitfehler ?
- Abarbeitung des
Programms korrekt ?

https://www2.informatik.hu-berlin.de/swt/lehre/GdP-WS-15/folien/java_syntax.html

GdP: Web-Seiten

[vorige Seite](#) - [übergeordnete Seite](#) - [nächste Seite](#) - [Inhaltsverzeichnis](#) - [Index](#)

Syntax -- kompakt

The Unicode Standard: Worldwide Character Encoding ::=

<ftp://unicode.org> .

Unicode-Zeichen ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Unicode-Ziffer ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Unicode-Buchstabe ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Zeichen ::=

$\backslash u \{ u \}$ [Hexadezimalziffer](#) [Hexadezimalziffer](#) [Hexadezimalziffer](#) [Hexadezimalziffer](#) | [Unicode-Zeichen](#) .

Eingabezeichen ::=

< [Zeichen](#), aber nicht ASCII-CR, ASCII-LF > .

Zeilenendekennzeichen ::=

ASCII-CR ASCII-LF | ASCII-CR | ASCII-LF .

Quelltext ::=

{ [Kommentar](#) | [Leerzeichen](#) | [Sprachelement](#) } .

Leerzeichen ::=

ASCII-SP | ASCII-HT | ASCII-FF | [Zeilenendekennzeichen](#) .

Sprachelement ::=

[Schlüsselwort](#) | [Identifikator](#) | [Literal](#) | [Separator](#) | [Operator](#) .

Synchronisationsanweisung ::=

synchronized ([Ausdruck](#)) [Anweisung](#) .

Ausnahmeanweisung ::=

```
try Block { catch ( Ausnahme ) Block }
( finally | catch ( Ausnahme Identifikator ) ) Block .
```

Ausnahme ::=

[Klassentyp](#) .

Argumentliste ::=

[Ausdruck](#) { , [Ausdruck](#) } .

Referenzausdruck ::=

[Referenzzugriff](#) | [Objekterzeugung](#) .

Grundausdruck ::=

[Referenzausdruck](#) | [Literal](#) .

Referenzzugriff ::=

[Bezeichner](#) | [Feldzugriff](#) | [Komponentenzugriff](#) | [Methodenaufruf](#) ([Ausdruck](#)) .

Bezeichner ::=

[[Qualifikator](#) .] [Identifikator](#) | this | super | null .

Qualifikator ::=

{ [Identifikator](#) . } [Identifikator](#) .

Feldzugriff ::=

[Referenzausdruck](#) [[Ausdruck](#)] .

Komponentenzugriff ::=

[([Referenzausdruck](#) | [Zeichenkette-Literal](#)) .] [Identifikator](#) .

Im Detail:
kleine
Abweichungen
zur aktuellen
Java-Version

Sehr gut für
Navigation,
Prinzip der EBNF

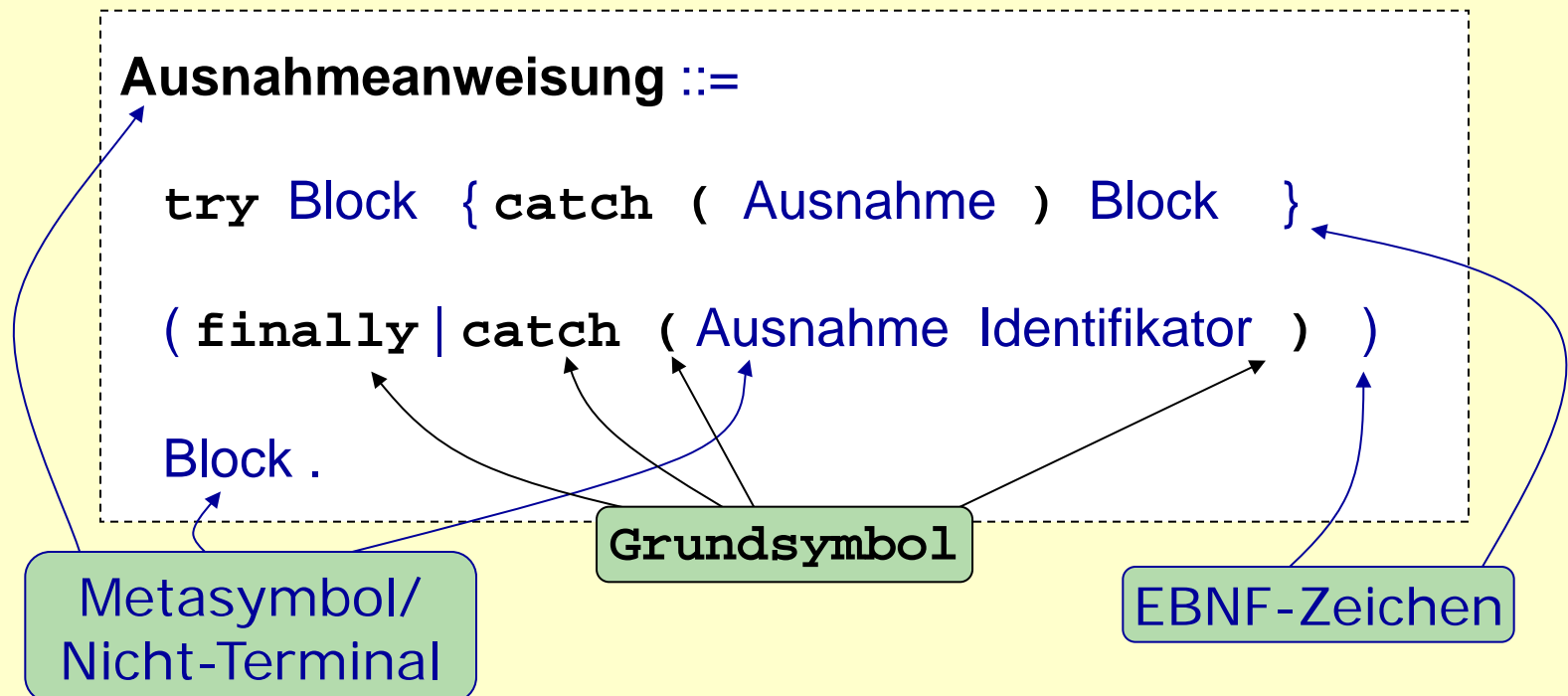
Überblick zur Java-Syntax

→ siehe GdP-Web-Seiten (14 Seiten)

https://www2.informatik.hu-berlin.de/swt/lehre/GdP-WS-15/fohlen/java_syntax.html

in EBNF angegeben

unterschiedliche Schriftsätze regeln Mehrdeutigkeiten !



Java: EBNF-Syntax → Grammatik

$$G_{\text{Java}} = [A , M , s , R]$$

Grundsymbole A :	Sprachelement (ableitbar aus ...)
Metasymbole M :	alle Metasymbole auf linker Seite einer Regel z.B.: Ausnahmeanweisung ::= ...
Satzsymbol s :	Quelltextdatei
Regeln R :	wie angegeben (als EBNF)

Regel für das Satzsymbol

Quelltextdatei ::= [Paketfestlegung] {Import} {Typdeklaration} .

Paketfestlegung ::= package Paketname ; .

Import ::= Typimport | Paket-Import .

Typimport ::= import Paketname . Identifikator ; .

Paket-Typimport ::= import Paketname .* ; .

Typdeklaration ::= Klassendeklaration | Schnittstellendeklaration .

Klassendeklaration ::=

[Abstraktionsebene] class Identifikator

[extends Klassentyp]

[implements Schnittstellentyp { , Schnittstellentyp }]

Klassenkörper .

Klassensichtbarkeit ::= public .

Abstraktionsebene ::= abstract | final .

Mit welchem Symbol
kann ein korrektes
Java-Programm
beginnen?

u.a.

Java-Grundsymbole: abgeleitet aus ‚Sprachelement‘

[vorige Seite](#) - [übergeordnete Seite](#) | [Übersicht](#) | [Suche](#) | [Feedback](#) | [Impressum](#) | [Datenschutz](#)

Syntax -- kompakt

The Unicode Standard: Worldwide Character Encoding ::=

<ftp://unicode.org> .

Unicode-Zeichen ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Unicode-Ziffer ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Unicode-Buchstabe ::=

[The Unicode Standard: Worldwide Character Encoding](#) .

Zeichen ::=

\ u { u } [Hexadezimalziffer](#) [Hexadezimalziffer](#) [Hexadezimalziffer](#) [Hexadezimalziffer](#) | [Unicode-Zeichen](#) .

Eingabezeichen ::=

< [Zeichen](#), aber nicht ASCII-CR, ASCII-LF > .

Zeilenendekennzeichen ::=

ASCII-CR | ASCII-LF | ASCII-CR | ASCII-LF .

Quelltext ::=

{ [Kommentar](#) | [Leerzeichen](#) | [Sprachelement](#) } .

Leerzeichen ::=

ASCII-SP | ASCII-HT | ASCII-FF | [Zeilenendekennzeichen](#) .

Sprachelement ::=

[Schlüsselwort](#) | [Identifikator](#) | [Literal](#) | [Separator](#) | [Operator](#) .

synchronized ([Ausdruck](#)) [Anweisung](#) .

Ausnahmeanweisung ::=

try [Block](#) { catch ([Ausnahme](#)) [Block](#) }
(finally | catch ([Ausnahme](#) [Identifikator](#))) [Block](#) .

Ausnahme ::=

[Klassentyp](#) .

Argumentliste ::=

[Ausdruck](#) { , [Ausdruck](#) } .

Referenzausdruck ::=

[Referenzzugriff](#) | [Objekterzeugung](#) .

Grundausdruck ::=

[Referenzausdruck](#) | [Literal](#) .

Referenzzugriff ::=

[Bezeichner](#) | [Feldzugriff](#) | [Komponentenzugriff](#) |
[Methodenaufruf](#) ([Ausdruck](#)) .

Bezeichner ::=

[[Qualifikator](#) .] [Identifikator](#) | this | super | null .

Qualifikator ::=

{ [Identifikator](#) . } [Identifikator](#) .

Feldzugriff ::=

[Referenzausdruck](#) [[Ausdruck](#)] .

Komponentenzugriff ::=

[([Referenzausdruck](#) | [Zeichenkette-Literal](#)) .] [Identifikator](#) .

Sprachelement

Java-Grundsymbole (1)

(s. Web-Seite)

Grundsymbole in Java: "Sprachelemente"
→ auch sie haben Syntax (lexikalische Struktur)

Sprachelement ::= Schlüsselwort | Identifikator | Literal | Separator | Operator .

Schlüsselwort ::= abstract | boolean | break | byte | case | cast | catch | char | class | const | continue | default | do | double | else | extends | final | finally | float | for | future | generic | goto | if | implements | import | inner | instanceof | int | interface | long | native | new | null | operator | outer | package | private | protected | public | rest | return | short | static | super | switch | synchronized | this | throw | throws | transient | try | var | void | while .

Identifikator ::= Unicode-Buchstabe { Unicode-Buchstabe | Unicode-Ziffer | _ | \$ } .

z.B.: x1, Matrix, M_1

Java-Grundsymbbole (2)

25

1.1 e-2

Konstanten

Literal ::=

Zahl-Literal | Gleitkommazahl-Literal |
Wahrheitswert-Literal | Zeichen-Literal |
Zeichenkette-Literal | Klassen-Literal .

Operator ::=

Inkrementoperator | Dekrementoperator |
Vorzeichenoperator | Negationsoperator |
Multiplikationsoperator | Additionsoperator |
Schiebeoperator | Ordnungsoperator |
Vergleichsoperator |
? ... : | && | || | & | | | ^ | Zuweisungsoperator .

Separator ::=

(|) | { | } | [|] | ; | , | . .

Inkrementoperator ::=

++ .

...

Zuweisungsoperator ::=

= | += | *= | -= | /= | %= | &= | |= | ^= | <<= | >>= .

**Kann es auch ein andere
Grammatik für Java geben?**

Äquivalenz von Grammatiken

Zwei Grammatiken G_1 und G_2
sind **äquivalent**, falls

$$L_{G_1} = L_{G_2}$$

(die erzeugten Sprachen sind identisch)

→ **"die"** Java-Grammatik ?

Unterschiedliche Varianten für Java-Grammatiken:

- Metasymbole umbenennen (z.B. deutsch → englisch)
- Statt Metasymbol in die Regel direkt die Alternativen eintragen (z.B. Inkrementoperator, Separator ...)
- u.v.a.

Zu einem Programm wird es im allg. immer mehrere Ableitungen geben:

Ist das ein Problem – z.B. für einen Compiler?

expr → term exprrest
→ term ϵ
→ factor termrest
→ factor
→ a

expr → term exprrest
→ factor termrest exprrest
→ a termrest exprrest
→ a exprrest
→ a

Syntaxbäume:

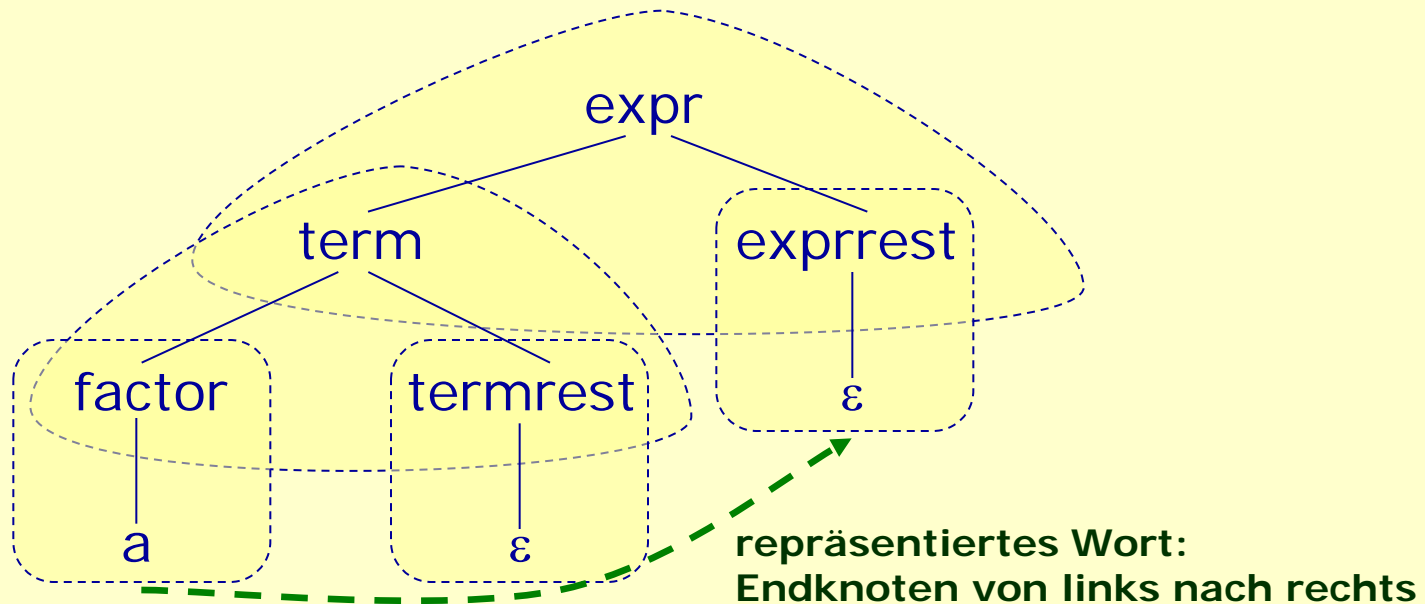
alternative Repräsentation der syntaktischen Struktur

Repräsentation der syntaktischen Syntax:

- Ableitungen mit Regeln
- Syntaxbäume

expr → term exprrest
→ term ϵ
→ factor termrest
→ factor
→ a

Syntaxbäume



entspricht in G_1 den Ableitungen:

expr \rightarrow term exprrest
 \rightarrow term ϵ
 \rightarrow factor termrest
 \rightarrow factor
 \rightarrow a

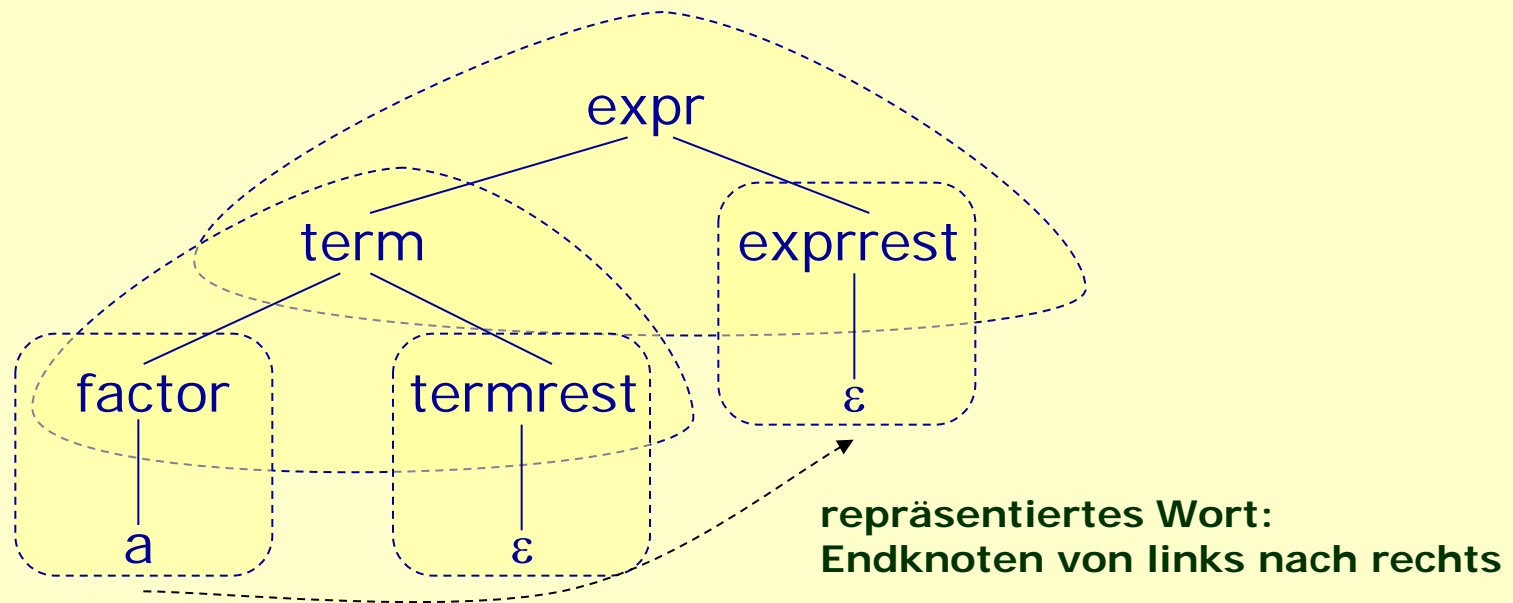
Rechtsableitung

und

expr \rightarrow term exprrest
 \rightarrow factor termrest exprrest
 \rightarrow a termrest exprrest
 \rightarrow a exprrest
 \rightarrow a

Linksableitung

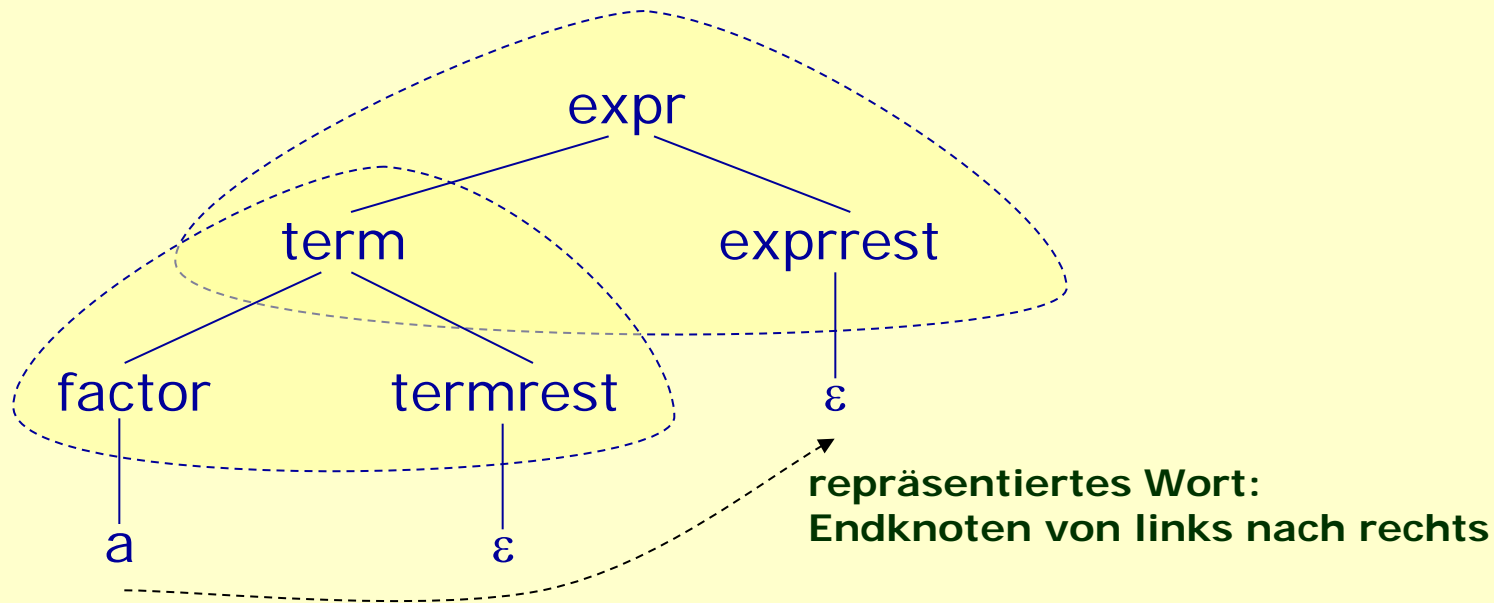
Syntaxbaum: Regelnanwendungen entwickeln Baum von oben nach unten (1)



entspricht in G_1 der Ableitung:

```
expr → term exprrest  
→ term ε  
→ factor termrest  
→ factor ε  
→ a
```


Syntaxbaum: Regelnanwendungen entwickeln Baum von oben nach unten (2)



entspricht in G_1 der Ableitung:

Identischer Baum
- nur in anderer Reihenfolge
aufgebaut.

```
expr → term exprrest
      → factor termrest exprrest
      → a termrest exprrest
      → a exprrest
      → a
```

Syntaxbäume: Aufgabe

- Repräsentieren syntaktische Struktur von Programmen
(lesbarer und aussagefähiger als Ableitungen)
- Abstraktion von konkret gewählter Ableitungsfolge
- Eindeutigkeit von Grammatiken definiert
- Sind Ergebnis der Syntaxanalyse von Compilern
(interne Repräsentation von Programmen zur weiteren Analyse und Optimierung)

Eindeutigkeit von Grammatiken

Def.:

Eine Grammatik G heißt **eindeutig**:

Für jedes Wort x aus L_G führen alle Ableitungen von x zum selben Syntaxbaum.

m. a. W.: Unterschiedliche Ableitungen desselben Wortes dürfen keine unterschiedliche syntaktische Struktur des Wortes definieren!

Beispiel:

G_1 ist eindeutig (ohne Beweis),

aber ein Beispiel „a“ gegeben:

mehrere Ableitungen von $a \rightarrow$ derselbe Syntaxbaum

Eine zu G_1 äquivalente Grammatik G'

G' mit den Regeln:

$$\begin{array}{l} E = E + E \mid \\ E * E \mid \\ a \quad \quad \mid \\ (E) . \end{array}$$

- Erzeugte Sprachen gleich: $a, a+a, a*a, (a+a)*a \dots$
- G' kürzer (lesbarer / übersichtlicher).

G' bevorzugen anstelle
von G_1 ?

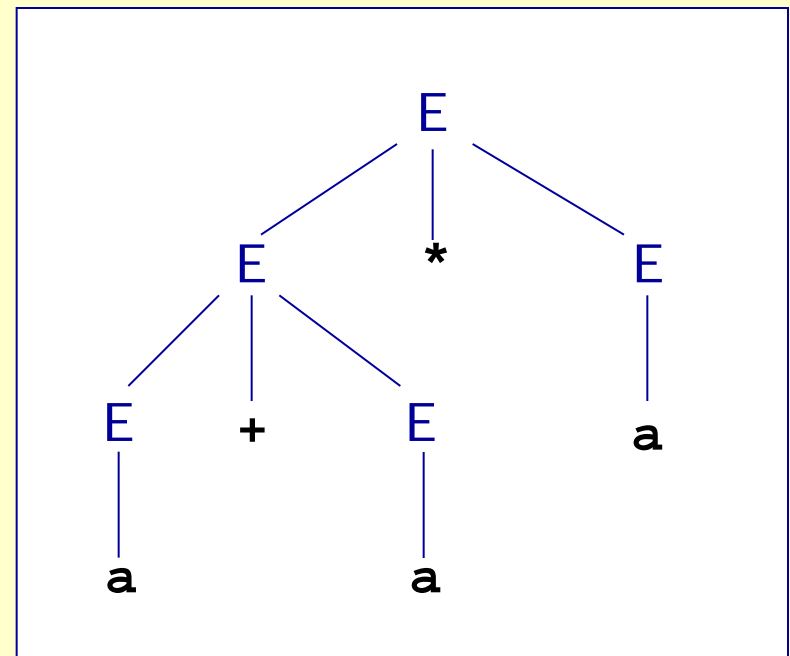
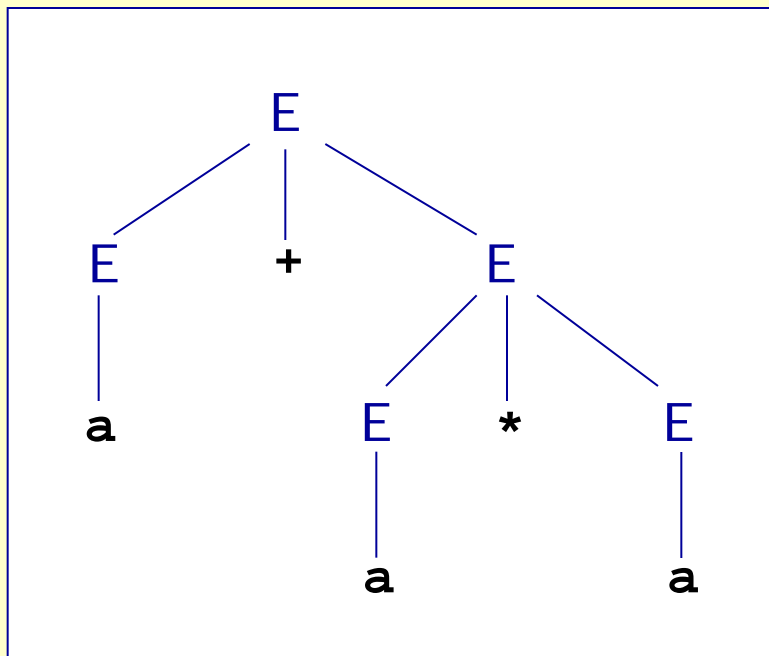
$a + a * a$ ist Element der Sprache

... und damit G' nicht eindeutig!

G' nicht eindeutig:

$a + a * a$

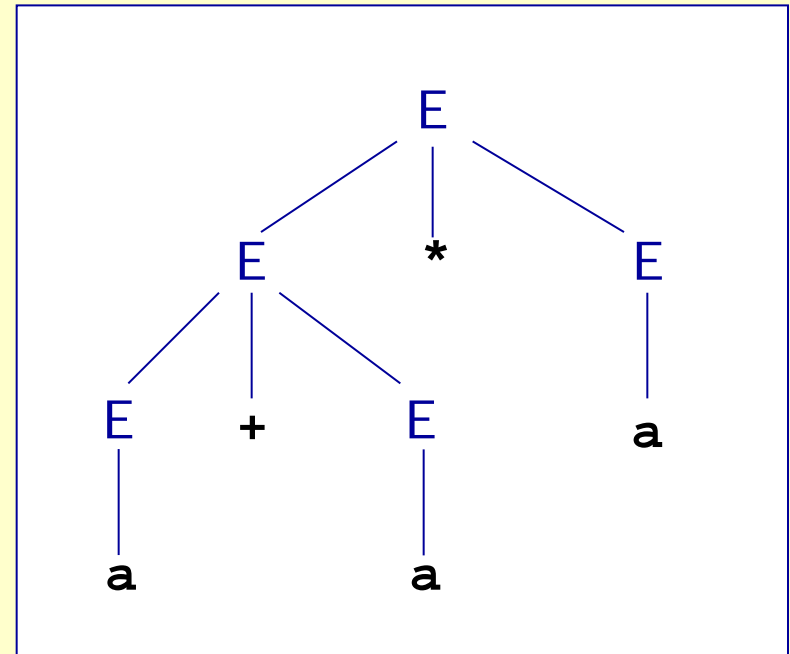
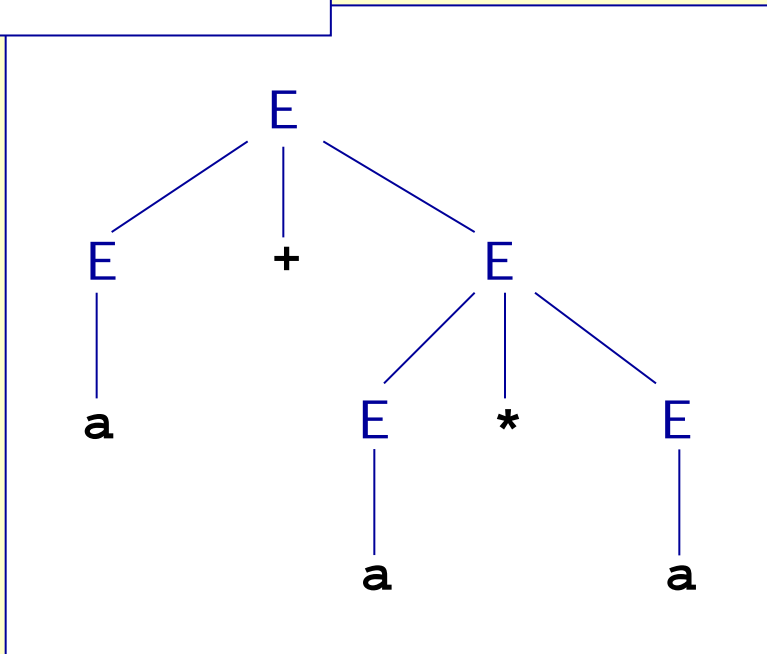
hat zwei Ableitungen mit unterschiedlichen Syntaxbäumen



G' nicht eindeutig:

$E = E + E \mid$
 $E * E \mid$
 $a \mid$
 $(E) .$

$a + a * a$
hat zwei Ableitungen mit
unterschiedlichen Syntaxbäumen



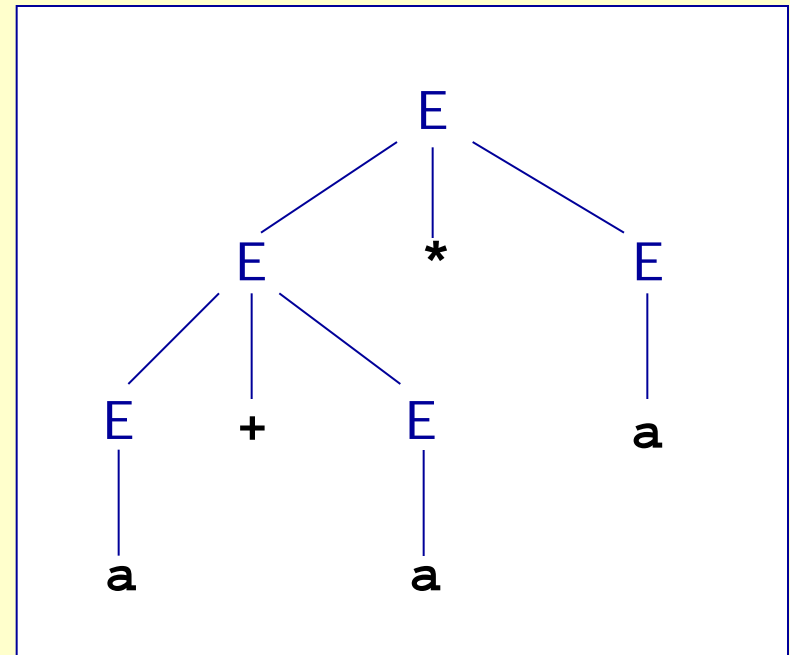
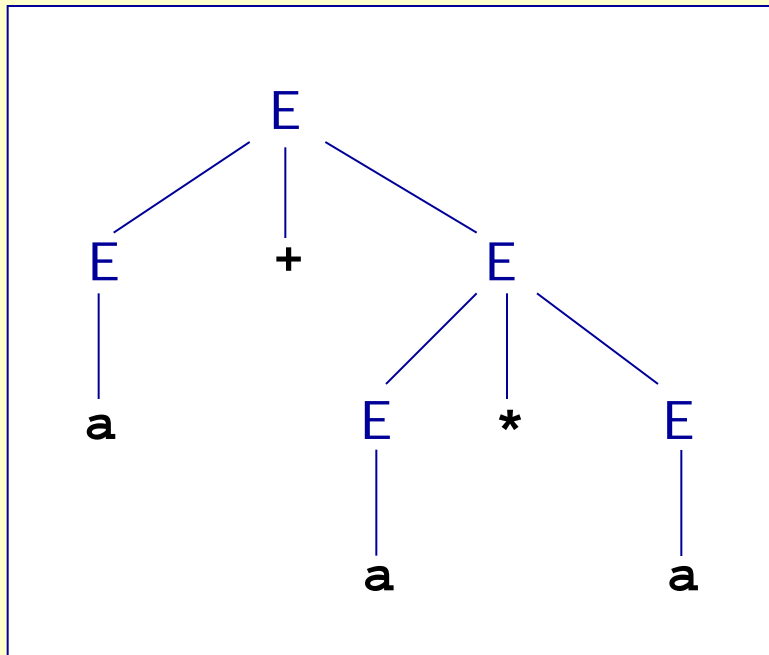
$E \rightarrow E + E \rightarrow E + E * E \rightarrow a + E * E \rightarrow a + a * E \rightarrow a + a * a$

$E \rightarrow E * E \rightarrow E + E * E \rightarrow a + E * E \rightarrow a + a * E \rightarrow a + a * a$

G' nicht eindeutig:

$a + a * a$

hat zwei Ableitungen mit unterschiedlichen Syntaxbäumen



Ist Nicht-Eindeutigkeit ein Problem?

JA: Semantik ist unterschiedlich (erst „+“ oder erst „*“)

Grammatik von Java ist eindeutig

... bis auf eine Ausnahme

Grammatiken zu Sprachen finden

Bisherige Aufgabe

Gegebene Grammatik definiert Sprache

Grammatik $G \rightarrow$ erzeugte Sprache L_G

Anwendung: EBNF für Java

→ alle syntaktisch korrekten Java-Programme

Umgekehrte Aufgabe

Anwendung: InformatikerIn möchte eine (Fach-)Sprache L einführen:

1. Schritt: kontextfreie Grammatik für L definieren

Für eine vorgegebene Sprache wird eine kf. Grammatik gesucht.

Problem:

Existiert überhaupt eine kontextfreie Grammatik ?

Formale Sprachen

(Formale) **Sprache** L über einem Alphabet $A =_{def}$

$$L \subseteq A^*$$

(Definition unabhängig vom Grammatikbegriff!)

→ Programme sind Folgen von Grundsymbolen

L heißt **kontextfreie Sprache** $=_{def}$

Formale Sprache L lässt sich durch eine kontextfreie Grammatik G beschreiben (d.h. $L = L_G$)

Beispiele: formale Sprachen

formale Sprachen über $A_2 = \{a, b, c\}$

- $L_1 = \{a, b, c\} \subseteq A_2^*$
- $L_2 = \{a^n \mid n > 0\} = \{a, aa, aaa, aaaa, \dots\}$
- $L_3 = \{a^n b^n \mid n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \dots\}$
- $L_4 = \{a^n b^n c^n \mid n > 0\} = \{abc, aabbcc, aaabbbccc, \dots\}$

Sind $L_1 \dots L_4$ auch kontextfreie Sprachen?

- $G_1 = [\{a, b, c\}, \{s\}, s, \dots]$
- $G_2 = [\{a\}, \{s\}, s, \dots]$
- $G_3 = [\{a, b\}, \{s\}, s, \dots]$
- $G_4 = [\{a, b, c\}, \{s\}, s, \dots]$

Beispiele: kontextfreie Grammatiken für die Sprachen

formale Sprachen über $A_2 = \{a, b, c\}$

- $L_1 = \{a, b, c\}$
- $L_2 = \{a^n \mid n > 0\} = \{a, aa, aaa, aaaa, \dots\}$
- $L_3 = \{a^n b^n \mid n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \dots\}$
- $L_4 = \{a^n b^n c^n \mid n > 0\} = \{abc, aabbcc, aaabbbccc, \dots\}$

Sind $L_1 \dots L_4$ auch kontextfreie Sprachen?

- $G_1 = [\{a, b, c\}, \{s\}, s, \{(s, a), (s, b), (s, c)\}]$
- $G_2 = [\{a\}, \{s\}, s, \{(s, a), (s, as)\}]$
- $G_3 = [\{a, b\}, \{s\}, s, \{(s,), (s, asb)\}]$
- G_4 : es ex. keine kontextfreie Grammatik mit $L_{G_4} = L_4$

Historie:

Untersuchung zum Aufbau *natürlicher* Sprachen
→ erster (formaler) Grammatik-Begriff (Chomsky 1959)

Satzsymbol

Metasymbole
= Fachbegriffe
der Grammatik

SATZ	:	SUBJEKT PRAEDIKAT OBJEKT .
SUBJEKT	:	ARTIKEL SUBSTANTIV .
OBJEKT	:	ARTIKEL SUBSTANTIV .
PRAEDIKAT	:	VERB .
SUBSTANTIV	:	Hund Brieftraeger .
VERB	:	beisst .
ARTIKEL	:	den der .

Regeln der
Grammatik

Grundsymbole
= Worte der Sprache

SATZ → SUBJEKT PRAEDIKAT OBJEKT → ...
... → der Hund beisst den Brieftraeger

aber auch:

SATZ →* den Brieftraeger beisst der Hund
SATZ →* den Hund beisst der Brieftraeger

Problem: Reichhaltigkeit der Grammatik einer natürlichen Sprache
(Konjugation, Deklination u. v. a.)

→ erweiterte Grammatikbegriffe nötig

Grammatiken in Disziplinen der Informatik



Nachfolgend

**Zusatzmaterial:
am 2. 11.2015 in der Vorlesung
behandelt
(zur Information)**

Allgemeiner Grammatik-Begriff

Beispiele: kontextfreie Grammatiken für die Sprachen

formale Sprachen über $A_2 = \{a, b, c\}$

- $L_1 = \{a, b, c\}$
- $L_2 = \{a^n \mid n > 0\} = \{a, aa, aaa, aaaa, \dots\}$
- $L_3 = \{a^n b^n \mid n \geq 0\} = \{\varepsilon, ab, aabb, aaabbb, \dots\}$
- $L_4 = \{a^n b^n c^n \mid n > 0\} = \{abc, aabbcc, aaabbbccc, \dots\}$

Sind $L_1 \dots L_4$ auch kontextfreie Sprachen?

- $G_1 = [\{a, b, c\}, \{s\}, s, \{(s, a), (s, b), (s, c)\}]$
- $G_2 = [\{a\}, \{s\}, s, \{(s, a), (s, as)\}]$
- $G_3 = [\{a, b\}, \{s\}, s, \{(s,), (s, asb)\}]$
- G_4 : es ex. keine kontextfreie Grammatik mit $L_{G_4} = L_4$

ABER: Es gibt allgemeine Grammatik für L_4

(Allgemeine) Grammatik

$G = [A, M, s, R]$ heißt **Grammatik**, falls

- A, M, s wie bisher (bei kf. Grammatiken)
- $R \subseteq (A \cup M)^+ \times (A \cup M)^*$ Menge von Regeln (l, r) , wobei l mindestens ein Metasymbol enthält

$(A \cup M)^+$ beliebige nicht-leere Symbolfolgen

Kontextfreie Grammatik:
Kontextfreie Regeln

$$\begin{aligned} X_2 &= \mathbf{b c} . \\ X_1 &= X_1 \mathbf{c} . \end{aligned}$$

Metasymbole können beliebig ersetzt werden

Allgemeine Grammatik:
Kontextabhängige Regeln

$$\begin{aligned} X_2 X_1 &= \mathbf{b} X_2 \mathbf{c} . \\ \mathbf{c} X_1 &= X_1 \mathbf{c} . \end{aligned}$$

Metasymbole können nur im Kontext ersetzt werden

(Allgemeine) Grammatik

$G = [A, M, s, R]$ heißt **Grammatik**, falls

- A, M, s wie bisher (bei kf. Grammatiken)
- $R \subseteq (A \cup M)^+ \times (A \cup M)^*$ Menge von Regeln (l, r) , wobei l mindestens ein Metasymbol enthält

$(A \cup M)^+$ beliebige nicht-leere Symbolfolgen

Kontextfreie Grammatik: Spezialfall einer allgemeinen Grammatik

Allg. Grammatiken leistungsfähiger als kf. Grammatiken

→ auch kontextabhängige Syntax beschreibbar!

Also auch: benutzte Variablen müssen vorher vereinbart sein

aber: allg. Regeln schwer zu lesen

Praxis: Beschreibung von Programmiersprachen

- Kontextfreie Syntax:
kontextfreie Grammatik (EBNF)
- Kontextabhängige Syntax:
verbal (umgangssprachlich)
z. B.: "*Jede benutzte Variable muss vereinbart werden.*"

Beispiel für allgemeine Grammatik

$$L_4 = \{a^n b^n c^n \mid n > 0\}$$

Mögliche Regeln einer
(allg.) Grammatik, die
 L_4 erzeugt:

**kontextfrei: Ersetzung
eines Metasymbols
unabh. von Umgebung**

$$S = a S X_1 . \quad (r1)$$

$$S = a X_2 . \quad (r2)$$

$$X_2 X_1 = b X_2 c . \quad (r3)$$

$$c X_1 = X_1 c . \quad (r4)$$

$$X_2 = b c . \quad (r5)$$

r1, r2, r5: kontextfreie Regeln

r3, r4: kontextabhängige Regeln

Beispielableitung:

$$s \rightarrow a s X_1 \quad (r1)$$

$$\rightarrow a a X_2 X_1 \quad (r2)$$

$$\rightarrow a a b X_2 c \quad (r3)$$

$$\rightarrow a a b b c c \quad (r5)$$