VL und Übung diese Woche getauscht

diese Woche Di, Mi, Do VL

nächste Woche Di, Mi, Do, Fr Termine

1x VL
2x UE ↑ VL

Kap. 4 Syntax und Semantik von Programmiersprachen

↙

Menge der grammatischen Regeln für eine formale Sprache

↳ beschreiben wie man aus Symbolen der Sprache syntaktisch (= grammatisch) korrekte Sätze bildet

Symbole in Java:

Schlüsselwörter: if while return int double ...

Zeichen: a b ... z A B ... Z 0 1 ... 9 ä ö, α β γ

unicode Zeichen

Sonderzeichen: / % ( ) { } && || ++ --

einzelne Zeichen

Durch Aneinanderreihung solcher Zeichen wird ein Satz (hier = Java Programm) gebildet. Syntaxregeln legen fest, wann das syntaktisch korrekt ist.

verschiedene Darstellungsmöglichkeiten

- Backus Naur Form (bzw. erweiterte BNF)
- Syntaxgraph (Syntaxdiagramme)

Beide bieten Darstellungsmöglichkeiten für

- Option
- Alternative
- (begrenzte) Wiederholung
- Aneinanderreihung (Konkatenation)

4.1 Backus Naur Form

Man verwendet Metazeichen für die Beschreibung der Konstruktionsmöglichkeiten

[...] für Option

| ... | ... | für Alternativen

{ ... } für Wiederholung

Konkatenation durch Aneinanderreihung

Symbole der Sprache heißen Terminalsymbole

Daneben kann man noch nicht erklärte Begriffe verwenden, die Nonterminalsymbole. Gekennzeichnet durch $\langle \ldots \rangle$. Diese werden durch $::=$ definiert

Beispiel: Identifier in Java

$\langle$ identifier $\rangle ::= \langle$ Unicode-Buchstabe $\rangle \{$

$\langle$ Unicode-Buchstabe $\rangle \mid \langle$ Unicode-Ziffer $\rangle \}$

$\langle$ Unicode-Buchstabe $\rangle ::= A \mid B \mid C \mid \ldots \mid a \mid b \mid \ldots \mid \alpha \mid \beta \mid \ldots \}$ _

↑ ↑ ↑ Terminalsymbole

$\langle$ Unicode-Ziffer $\rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

Nach diesen Regeln korrekte Identifier sind:

a1 a_very_long_identifier αβγ

nicht korrekt

1a x-11 a*

Neben der Syntax gibt es die Semantik

↑ ↑

syntaktische Regeln legt Bedeutung der korrekt gebildeten Ausdrücke

fest, meist durch zusätzliche Regeln außerhalb des Syntax

Bsp bei Identifier:

- Groß- und Kleinschreibung wird an jeder Position unterschieden

  abba, ABBA, aBBa, abBa ...

  sind paarweise verschiedene Identifier

- Schlüsselwörter können keine Identifier sein

Beispiel 4.2 if-statement in Java

<if-statement> ::= if (<Bedingung>) <Anweisung>

[else <Anweisung>]

<Bedingung> ::=

<Anweisung> ::=

siehe Java Spezifikation

Ein anderes Beispiel: Integer-Literale

↑

Zeichenkette die Zahl vom Typ int oder long beschreibt

<integerliteral> ::= <Zahlenwert> [<Typsuffix>]

<Zahlenwert> ::= <Dezimalwert> | <Oktalwert> | <Hexadezimalwert>

<Typsuffix> ::= l | L

<Dezimalwert> ::= 0 | <erste Dezimalziffer> { <Dezimalziffer> }

<erste Dezimalziffer> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Oktalwert> ::= <Oktalpräfix> <Oktalziffer> { <Oktalziffer> }

<Oktalpräfix> ::= 0

<Oktalziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<Hexadezimalwert> ::= <Hexadezimalpräfix> <Hexadezimalziffer>
{ <Hexadezimalziffern> }

<Hexadezimalpräfix> ::= 0x | 0X

<Hexadezimalziffer> ::= 0 | 1 | 2 | ... | 9 | a | A | b | B | c | C | d | D |
e | E | f | F

↑
syntaktische Regeln Bsp. 0x Da Da Cafe

zur Semantik regelt Bedeutung ↗

- Präfix 0 bedeutet Oktaldarstellung
  d.h. zur Basis 8 mit Ziffern 0 1 2 ... 7
  Literal 045 ≙ Zahlenwert $4 \cdot 8^1 + 5 \cdot 8^0 = \underline{\underline{37}}$
  dezimal

- Präfix 0x oder 0X bedeutet Hexadezimaldarstellung
  d.h. zur Basis 16 mit Ziffern 0 .... 9 a A b B .... f F
  Zahlenwerte der Ziffern: 0 ... 9 10 11 ... 15
  Literal 0x Da Da Cafe ≙ $13 \cdot 16^7 + 10 \cdot 16^6 + 13 \cdot 16^5 + 10 \cdot 16^4$

16^7 16^0 $+ 12 \cdot 16^3 + 10 \cdot 16^2 + 15 \cdot 16^1 + 14 \cdot 16^0$

- Typsuffix l oder L bezieht sich auf den Java Typ long zur Darstellung ganzer Zahlen

  standard für int Zahlen: 32 bit zur Darstellung im sog. Zweierkomplement

  Werte von -2^{31} bis $2^{31}-1$

  standard für long Zahlen 64 bit im Zweierkomplement

  Werte von -2^{63} bis $2^{63}-1$

- Liegt Zahlenwert eines Literals außerhalb des durch 32 bzw 64 bit festgelegten Bereichs, so soll beim Compilieren (laut Spezifikation) eine Fehlermeldung kommen

  ↳ ACHTUNG, geschieht in der Regel nicht

Beispiele für integer Literale

13	039	0	f17	000	0xfL	0-3
✓	✗	✓	✗	✓	✓	✗

Beachte: Vorzeichen gehört nicht zum Integer-Literal
- ist einstelliger Operator, der auf ein Literal angewendet wird

4.2 erweiterte BNF

$\{ \ldots \}_{\ell}^{k}$ mindestens ℓ mal, höchstens k mal wiederholen

$\{ \ldots \}_{\ell}^{*}$ mindestens ℓ mal, sonst beliebig oft

Beispiele

$\langle \text{integer literal} \rangle ::= \langle \text{Zahlenwert} \rangle \{ \langle \text{Typsuffix} \rangle \}_{0}^{1}$

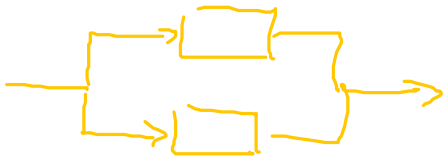
$\langle \text{Oktalwert} \rangle ::= \langle \text{Oktalpräfix} \rangle \{ \langle \text{Oktalziffer} \rangle \}_{1}^{*}$

4.3 Syntaxgraphen

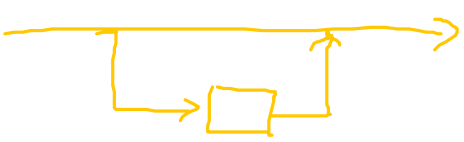
Nonterminalsymbol

Terminalsymbol

Konkatenation



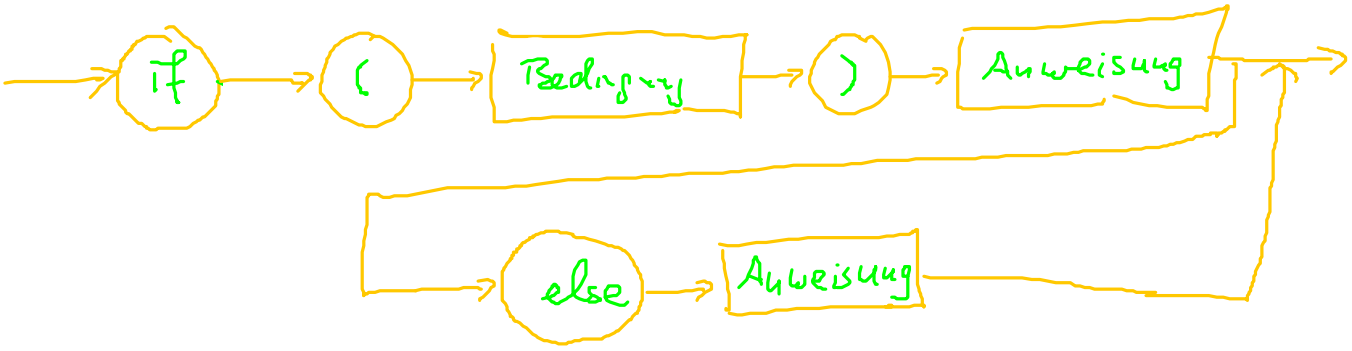
Alternative



Option

Wiederholung

Beispiel: if statement



Jeder Weg von Anfang bis Ende in Pfeilrichtung

formt ein korrektes Wort der Grammatik

Eine Anwendung außerhalb von Java

Korrekte Klammerausdrücke

{ { } { } { { } { } } } ← korrekt oder nicht ?

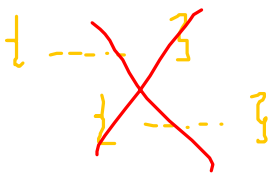
↑

zu jeder Klammer { gehört

eine Klammer }

das dazwischen heißt Block

Blocke dürfen sich nicht echt überlappen



Suchen: Grammatik (= Syntaxregeln) die genau die

korrekt geformten Klammerausdrücke beschreibt

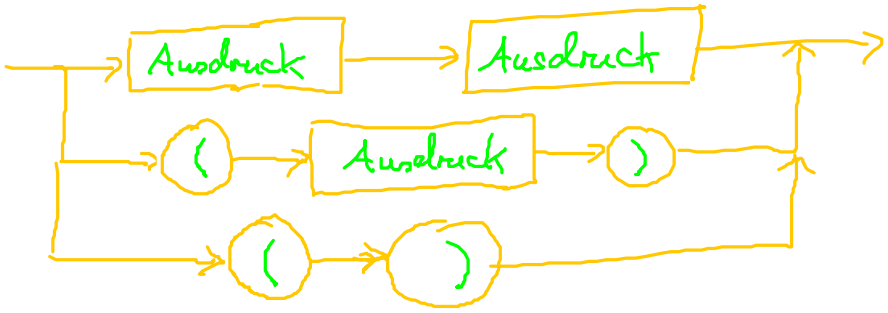
- kleinster korrekter (≠ leer) ( )
- wie kann man aus korrekten Klammerausdrücken

größere machen ?

- A korrekt ⇒ (A) korrekt
- A, B korrekt ⇒ A B korrekt

(), (()) ⇒ ()(())

festhalten als Syntaxregeln



Frage: beschreibt diese Grammatik alle korrekten Klammerausdrücke ?