

# Reguläre und Kontextfreie Sprachen 2

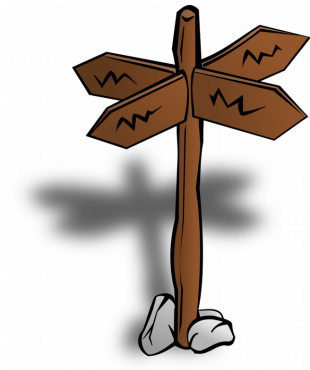
Grundzüge digitaler Systeme

Vortrag von: Gernot Salzer

# Reguläre und Kontextfreie Sprachen

---

- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit



# Grenzen regulärer Sprachen

*Was reguläre Ausdrücke und endliche Automaten beschreiben können:*

- lexikale Elemente in Programmiersprachen: Identifier, Numerale, ...
- Morphologie von Worten in natürlichen Sprachen: korrekte Fall-/Zahl-/Zeitendungen, ...
- Systeme, die nur ein endliches Gedächtnis besitzen

*Was sie nicht beschreiben können:*

- geklammerte Ausdrücke in Programmiersprachen
- geschachtelte Programmstrukturen wie  
`if ... while ... if ... endif ... endwhile ... endif`
- Schachtelung von Haupt- und Nebensätzen in natürlichen Sprachen
- Systeme mit einem (potentiell) unendlichen Speicher

**Sprache der wohlgeklammerten Ausdrücke ist nicht regulär**

$\{(), (()), ()(), ((())), (()()), ()()(), \dots\}$

**$\{a^n b^n \mid n \geq 0\}$  ist nicht regulär**

$\{\epsilon, ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, \dots\}$

# Kontextfreie Grammatiken

- Menge von Ersetzungsregeln
- 2 Arten von Symbolen: Terminale, Nonterminale
- Ausgehend vom Start-Nonterminal werden Nonterminale solange ersetzt, bis nur noch Terminale bleiben.

$Satz \rightarrow HwP \ ZwP$   
 $HwP \rightarrow Art \ Hw$   
 $ZwP \rightarrow Hzw \ HwP \ Zw$

$Art \rightarrow der \mid den$   
 $Hw \rightarrow Hund \mid Mond$   
 $Hzw \rightarrow wird$   
 $Zw \rightarrow anbell$

$Satz \Rightarrow_p HwP \ ZwP$   
 $\Rightarrow_p Art \ Hw \ Hzw \ HwP \ Zw$   
 $\Rightarrow_p der \ Hund \ wird \ Art \ Hw \ anbell$   
 $\Rightarrow_p der \ Hund \ wird \ den \ Mond \ anbell$

Generative Grammatiken gehen zurück auf *Noam Chomsky* (Linguist, Philosoph, Aktivist; \*1928).

## Kontextfreie Grammatik

... wird beschrieben durch ein 4-Tupel  $G = \langle V, T, P, S \rangle$ , wobei

- $V$  ... Menge der Nonterminalsymbole (Variablen)
- $T$  ... Menge der Terminalsymbole ( $V \cap T = \{\}$ )
- $P \subseteq V \times (V \cup T)^*$  ... Produktionen
- $S \in V$  ... Startsymbol

Schreibweise:  $A \rightarrow w$  statt  $(A, w) \in P$

$A \rightarrow w_1 \mid \dots \mid w_n$  statt  $A \rightarrow w_1, \dots, A \rightarrow w_n$

## Ableitbarkeit

... *in einem Schritt*:

$x Ay \Rightarrow x w y$ , falls  $A \rightarrow w \in P$  und  $x, y \in (V \cup T)^*$ .

... *in mehreren Schritten*:

$u \xRightarrow{*} v$ , falls

- $u = v$ , oder
- $u \Rightarrow u'$  und  $u' \xRightarrow{*} v$  für ein Wort  $u' \in (V \cup T)^*$ .

## Von Grammatik $G$ generierte Sprache

$$\mathcal{L}(G) = \{ w \in T^* \mid S \xRightarrow{*} w \}$$

Grammatiken  $G_1$  und  $G_2$  heißen äquivalent, wenn  $\mathcal{L}(G_1) = \mathcal{L}(G_2)$  gilt.

$$G = \langle \{S\}, \{a, b\}, \{S \rightarrow \varepsilon \mid a S b\}, S \rangle$$

$$S \xRightarrow{*} aabb, \text{ weil } S \Rightarrow a S b \Rightarrow aa S bb \Rightarrow aa \varepsilon bb = aabb$$

$$\mathcal{L}(G) = \{ a^n b^n \mid n \geq 0 \} = \{ \varepsilon, ab, aabb, aaabbb, \dots \}$$

## Kontextfreie Sprachen

Eine Sprache heißt kontextfrei, wenn es eine kontextfreie Grammatik gibt, die sie generiert.

# Verschiedene Ableitbarkeitsbegriffe

**Linksableitbarkeit:**  $x A y \Rightarrow_L x w y$  falls  $A \rightarrow w \in P$  und  $x \in T^*$

(In jedem Schritt wird die linkeste Variable ersetzt.)

**Rechtsableitbarkeit:**  $x A y \Rightarrow_R x w y$  falls  $A \rightarrow w \in P$  und  $y \in T^*$

(In jedem Schritt wird die rechteste Variable ersetzt.)

**Parallelableitbarkeit:**  $x_0 A_1 x_1 \cdots A_n x_n \Rightarrow_P x_0 w_1 x_1 \cdots w_n x_n$

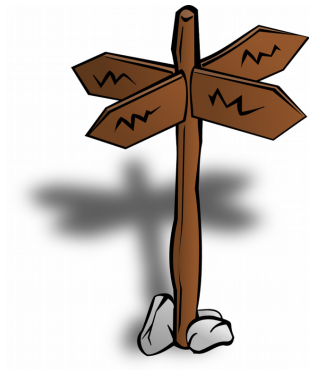
falls  $A_1 \rightarrow w_1, \dots, A_n \rightarrow w_n \in P$  und  $x_0, \dots, x_n \in T^*$

(In jedem Schritt werden alle Variablen ersetzt.)

- $\Rightarrow_L^*$ ,  $\Rightarrow_R^*$  und  $\Rightarrow_P^*$  sind eingeschränkte Ableitungsrelationen:  
Manche Wörter  $w \in (V \cup T)^*$  sind mit  $\Rightarrow^*$  herleitbar ( $S \Rightarrow^* w$ ),  
aber nicht mit diesen Relationen.
- Sie können aber jedes Wort der Sprache ableiten. Für alle  $w \in T^*$  gilt:  
 $S \Rightarrow^* w$  gdw.  $S \Rightarrow_L^* w$  gdw.  $S \Rightarrow_R^* w$  gdw.  $S \Rightarrow_P^* w$

$$\begin{aligned}\mathcal{L}(G) &= \{ w \in T^* \mid S \Rightarrow^* w \} = \{ w \in T^* \mid S \Rightarrow_L^* w \} \\ &= \{ w \in T^* \mid S \Rightarrow_R^* w \} = \{ w \in T^* \mid S \Rightarrow_P^* w \}\end{aligned}$$

- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit





# Kontextfreie Grammatiken - Beispiele

## Syntax aussagenlogischer Formeln

$G = \langle \{Fm, Op, Var\}, T, P, Fm \rangle$ , wobei

$T = \{ \top, \perp, \neg, (, ), \wedge, \uparrow, \vee, \downarrow, \Leftrightarrow, \nLeftrightarrow, \Rightarrow, \Leftarrow \} \cup \mathcal{V}$

$P = \{ Fm \rightarrow Var \mid \top \mid \perp \mid \neg Fm \mid ( Fm Op Fm ) ,$   
 $Op \rightarrow \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \Leftrightarrow \mid \nLeftrightarrow \mid \Rightarrow \mid \Leftarrow ,$   
 $Var \rightarrow A \mid B \mid C \mid \dots \}$

$((A \uparrow B) \uparrow (A \uparrow B))$  ist eine aussagenlogische Formel, weil:

$Fm \Rightarrow_L (Fm Op Fm)$	$\Rightarrow_L ((Fm Op Fm) Op Fm)$
$\Rightarrow_L ((Var Op Fm) Op Fm)$	$\Rightarrow_L ((A Op Fm) Op Fm)$
$\Rightarrow_L ((A \uparrow Fm) Op Fm)$	$\Rightarrow_L ((A \uparrow Var) Op Fm)$
$\Rightarrow_L ((A \uparrow B) Op Fm)$	$\Rightarrow_L ((A \uparrow B) \uparrow Fm)$
$\Rightarrow_L ((A \uparrow B) \uparrow (Fm Op Fm))$	$\Rightarrow_L ((A \uparrow B) \uparrow (Var Op Fm))$
$\Rightarrow_L ((A \uparrow B) \uparrow (A Op Fm))$	$\Rightarrow_L ((A \uparrow B) \uparrow (A \uparrow Fm))$
$\Rightarrow_L ((A \uparrow B) \uparrow (A \uparrow Var))$	$\Rightarrow_L ((A \uparrow B) \uparrow (A \uparrow B))$

# Kontextfreie Grammatiken - Beispiele

## Syntax aussagenlogischer Formeln

$G = \langle \{Fm, Op, Var\}, T, P, Fm \rangle$ , wobei

$T = \{ \top, \perp, \neg, (, ), \wedge, \uparrow, \vee, \downarrow, \Leftrightarrow, \nLeftrightarrow, \Rightarrow, \Leftarrow \} \cup \mathcal{V}$

$P = \{ \begin{array}{l} Fm \rightarrow Var \mid \top \mid \perp \mid \neg Fm \mid ( Fm Op Fm ) , \\ Op \rightarrow \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \Leftrightarrow \mid \nLeftrightarrow \mid \Rightarrow \mid \Leftarrow , \\ Var \rightarrow A \mid B \mid C \mid \dots \end{array} \}$

$((A \uparrow B) \uparrow (A \uparrow B))$  ist eine aussagenlogische Formel, weil:

$$\begin{aligned} Fm &\Rightarrow_P ( Fm Op Fm ) \\ &\Rightarrow_P ( ( Fm Op Fm ) \uparrow ( Fm Op Fm ) ) \\ &\Rightarrow_P ( ( Var \uparrow Var ) \uparrow ( Var \uparrow Var ) ) \\ &\Rightarrow_P ( ( A \uparrow B ) \uparrow ( A \uparrow B ) ) \end{aligned}$$

# Kontextfreie Grammatiken - Beispiele

## Reelle Numerale

$G = \langle V, T, P, Real \rangle$ , wobei

$V = \{ Real, Scale, Sign, Digits, Digit \}$

$T = \{ 0, \dots, 9, ., E, +, - \}$

und  $P$  folgende Produktionen sind:

$Real \rightarrow Digit Digits . Digits Scale$

$Scale \rightarrow \varepsilon \mid E Sign Digit Digits$

$Sign \rightarrow \varepsilon \mid + \mid -$

$Digits \rightarrow \varepsilon \mid Digit Digits$

$Digit \rightarrow 0 \mid \dots \mid 9$

## Optionalität:

$A \rightarrow \varepsilon \mid B$  ( $A$  steht für optionales  $B$ )

## Wiederholung:

$A \rightarrow \varepsilon \mid B A$  ( $A$  steht für wiederholtes  $B$ , auch 0 Mal)

$A \rightarrow B \mid B A$  ( $A$  steht für wiederholtes  $B$ , mind. 1 Mal)

# Kontextfreie Grammatiken - Beispiele

## Wohlgeformte Klammerausdrücke

$WKA$  ist die kleinste Menge, sodass

- $\varepsilon \in WKA$
- $(w) \in WKA$ , wenn  $w \in WKA$
- $w_1 w_2 \in WKA$ , wenn  $w_1, w_2 \in WKA$

$$G = \langle \{W\}, \{ (, ) \}, \{ W \rightarrow \varepsilon \mid (W) \mid W W \}, W \rangle$$

Beispiel einer Ableitung:  $W \Rightarrow_P W W$   
 $\Rightarrow_P (W)(W)$   
 $\Rightarrow_P ()(W W)$   
 $\Rightarrow_P ()((W)(W))$   
 $\Rightarrow_P ()((())())$

$$\begin{aligned} \mathcal{L}(G) &= \{ \varepsilon, (), (()), ()(), ((())), (()()), ()(), ()()(), \dots \} \\ &= WKA \end{aligned}$$

# Induktive Definition vs. kontextfreie Grammatik

## Induktive Definition für $\mathcal{M}$

Mengen  $\mathcal{M}, \mathcal{M}_0, \mathcal{A}_1, \mathcal{B}_1, \dots$

$\mathcal{M}$  ist die kleinste Menge, sodass:

$$\mathcal{M}_0 \subseteq \mathcal{M}$$

$$f(x_1, \dots, x_m) \in \mathcal{M}, \text{ falls } x_1 \in \mathcal{A}_1, \dots, x_m \in \mathcal{A}_m$$

$$g(x_1, \dots, x_n) \in \mathcal{M}, \text{ falls } x_1 \in \mathcal{B}_1, \dots, x_n \in \mathcal{B}_n$$

$$\dots \in \mathcal{M}, \text{ falls } \dots$$

$$h(x_1, x_2) \in \mathcal{M}, \text{ falls } x_1, x_2 \in \mathcal{M}$$

$$h(\mathbf{x}, \mathbf{x}) \in \mathcal{M}, \text{ falls } \mathbf{x} \in \mathcal{M}$$

## kontextfreie Grammatik für $\mathcal{M}$

Nonterminale  $M, M_0, A_1, B_1, \dots$

$$\mathcal{M} = \mathcal{L}(\langle \dots, P, M \rangle), \text{ wobei } P:$$

$$M \rightarrow M_0$$

$$M \rightarrow f(A_1, \dots, A_m)$$

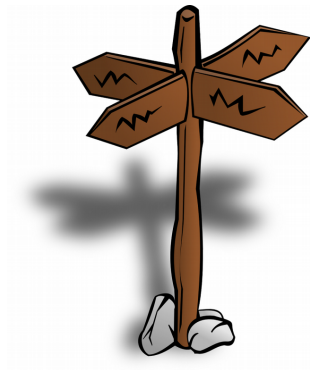
$$M \rightarrow g(B_1, \dots, B_n)$$

$$M \rightarrow \dots$$

$$M \rightarrow h(M, M)$$

keine Entsprechung, nicht kontextfrei

- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit



# Backus-Naur-Form (BNF)

Synonym für kontextfreie Produktionen

Historische Notation:

- Nonterminale in spitzen Klammern
- Terminale unter Anführungszeichen
- $::=$  als Trennsymbol

Durch diese Konvention entfällt die Notwendigkeit, die Mengen der Nonterminale ( $V$ ) und der Terminale ( $T$ ) anzugeben.

```
 $\langle Real \rangle ::= \langle Digit \rangle \langle Digits \rangle " ." \langle Digits \rangle \langle Scale \rangle$   
 $\langle Digit \rangle ::= "0" \mid \dots \mid "9"$   
 $\langle Digits \rangle ::= \varepsilon \mid \langle Digit \rangle \langle Digits \rangle$   
 $\langle Scale \rangle ::= \varepsilon \mid "E" \langle Sign \rangle \langle Digit \rangle \langle Digits \rangle$   
 $\langle Sign \rangle ::= \varepsilon \mid "+" \mid "-"$ 
```

# Erweiterte Backus-Naur-Form (EBNF)

„Regular Right Part Grammar“ (RRPG)

- erlaubt reguläre Ausdrücke auf der rechten Seite von Produktionen
- Bessere Lesbarkeit durch Vermeidung von Rekursionen und Leerwörtern
- Kompaktere Spezifikationen
- keine echte Erweiterung: EBNF-Notationen lassen sich eliminieren

*Elimination der EBNF-Notation:*

$A \rightarrow w_1 (w_2) w_3$  entspricht  $A \rightarrow w_1 B w_3$   
 $B \rightarrow w_2$

$A \rightarrow w_1 \{w_2\} w_3$  entspricht  $A \rightarrow w_1 B w_3$   
 $B \rightarrow \varepsilon \mid w_2 B$

$A \rightarrow w_1 [w_2] w_3$  entspricht  $A \rightarrow w_1 B w_3$   
 $B \rightarrow \varepsilon \mid w_2$



## Listen von Bezeichnern

EBNF:  $IdList \rightarrow Id \{ ", " Id \}$

Ohne EBNF:  $IdList \rightarrow Id B$   
 $B \rightarrow \varepsilon \mid ", " Id B$

oder  $IdList \rightarrow Id \mid Id ", " IdList$

## Skalierungsfaktor der reellen Numerale

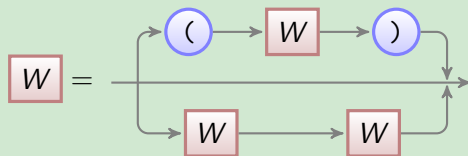
EBNF:  $Scale \rightarrow "E" [ "+" \mid "-" ] Digit \{ Digit \}$

Ohne EBNF:  $Scale \rightarrow "E" B_1 Digit B_2$   
 $B_1 \rightarrow \varepsilon \mid "+" \mid "-"$   
 $B_2 \rightarrow \varepsilon \mid Digit B_2$

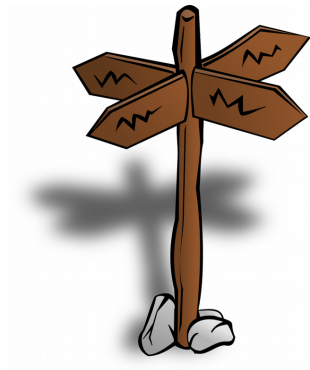
# Syntaxdiagramme

Mit *rekursiven* Diagrammen können beliebige kontextfreie Grammatiken in EBNF dargestellt werden.

## Wohlgeklammerte Ausdrücke


$$\begin{array}{l} W \rightarrow ( W ) \\ | \epsilon \\ | W W \end{array}$$

- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit



# Style Guide für Grammatiken

---

Wiederholungsoperator statt Rekursion (wenn möglich).

Schlecht:  $A \rightarrow a \mid A A$  oder  $A \rightarrow a \mid a A$

Besser:  $a \{ a \}$  oder  $a +$  oder  $\dots$  (je nach gewählter Notation)

Optionsoperator statt Leerwort.

Schlecht:  $\varepsilon \mid A$

Besser:  $[A]$  oder  $A?$  oder  $\dots$  (je nach gewählter Notation)

Modularisierung statt Duplizierung.

Schlecht:  $Identifier \rightarrow ("a" \mid \dots \mid "z") \{ "a" \mid \dots \mid "z" \mid "0" \mid \dots \mid "9" \}$

Besser:  $Identifier \rightarrow Letter \{ Letter \mid Digit \}$   
 $Letter \rightarrow "a" \mid \dots \mid "z"$   
 $Digit \rightarrow "0" \mid \dots \mid "9"$

Verwendung sprechender Namen.

Schlecht:  $X \rightarrow Y \{ Y \mid Z \}$   
 $Y \rightarrow "a" \mid \dots \mid "z"$   
 $Z \rightarrow "0" \mid \dots \mid "9"$

Besser: Siehe oben.

Klare Unterscheidung zwischen Terminalen, Nonterminalen und Metanotationen.

Schlecht: `\begin{tabular}[[Position]]{Spalte{Spalte}}`

Besser: `"\begin{tabular}" [ "[" Position "]" ] "{" Spalte { Spalte } "}"`  
`"\begin{tabular}" [ "["  $\langle Position \rangle$  "]" ] "{"  $\langle Spalte \rangle$  {  $\langle Spalte \rangle$  } "}"`

Inhaltliche Strukturierung statt Minimierung der Nonterminale und Regeln.

Schlecht:

`"\begin{tabular}" [ "[" ("t" | "b") "]" ] "{" ("l" | "c" | "r") { "l" | "c" | "r" } "}"`

Besser: `... → "\begin{tabular}" [ Position ] Spalten`

`Position → "[b]" | "[t]"`

`Spalten → "{" Spalte { Spalte } "}"`

`Spalte → "l" | "c" | "r"`

# Beispiel: Tabellen in LaTeX

TeX ... Textsatzsystem von Donald E. Knuth

LaTeX ... TeX-Makros für „logisches Markup“ von Leslie Lamport

```
\begin{tabular}[t]{lc}  
Eintrag 11 & Eintrag 12 \\  
Eintrag 21 &   
  \begin{tabular}{rr}  
    Eintrag 22 & ist selber \\  
    eine      & & Tabelle.  
  \end{tabular} \\  
Eintrag 31 & Eintrag 32  
\end{tabular}
```

Eintrag 11	Eintrag 12
Eintrag 21	Eintrag 22 ist selber eine Tabelle.
Eintrag 31	Eintrag 32

Gesucht: Kontextfreie Grammatik  $G$  in EBNF für die Sprache der LaTeX-Tabellen

# Beispiel: Tabellen in LaTeX

$G = \langle V, T, P, \text{Tabelle} \rangle$ , wobei:

$P = \{ \text{Tabelle} \rightarrow "\backslash\text{begin}\{\text{tabular}\}" [Position] \text{Spalten}$   
 $\text{Zeilen}$   
 $"\backslash\text{end}\{\text{tabular}\}" ,$

$Position \rightarrow "[b]" \mid "[t]" ,$

$Spalten \rightarrow "\{" Spalte \{ Spalte \} "\} " ,$

$Spalte \rightarrow "l" \mid "c" \mid "r" ,$

$Zeilen \rightarrow Zeile \{ "\backslash" Zeile \} ,$

$Zeile \rightarrow Eintrag \{ "&" Eintrag \} ,$

$Eintrag \rightarrow \{ Tabelle \mid Zeichen \} ,$

$Zeichen \rightarrow "0" \mid \dots \mid "9" \mid "a" \mid \dots \mid "Z" \mid "." \mid "\square" \}$

$V = \{ \text{Tabelle}, Position, Spalten, Spalte, Zeilen, Zeile, Eintrag, Zeichen \}$

$T = \{ "0", \dots, "9", "a", \dots, "z", "A", \dots, "Z",$   
 $".", "\square", "\{", "\}", "[", "]", "&", "\backslash" \}$

Nur eine Rekursion für Schachtelung der Tabellen notwendig!



- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit



# N. Wirth: Programming in Modula-2

## Appendix 1

### The Syntax of Modula-2

```
1  ident = letter (letter | digit).
2  number = integer | real.
3  integer = digit {digit} | octalDigit {octalDigit} ("B"|"C") |
4    digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [ScaleFactor].
6  ScaleFactor = "E" | "+" | "-" {digit} digit {digit}.
7  hexDigit = digit | "A"|"B"|"C"|"D"|"E"|"F".
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
10 string = "" (character) "" | "" (character) ""'.
11 qualident = ident ("." ident).
12 ConstantDeclaration = ident "=" ConstExpression.
13 ConstExpression = SimpleConstExpr [relation SimpleConstExpr].
14 relation = "<" | "<=" | ">" | ">=" | "=" | "<=" | ">=" | IN.
15 SimpleConstExpr = ["+"|"-"] ConstTerm (AddOperator ConstTerm).
16 AddOperator = "+" | "-" | OR.
17 ConstTerm = ConstFactor (MulOperator ConstFactor).
18 MulOperator = "*" | "/" | DIV | MOD | AND | "&".
19 ConstFactor = qualident | number | string | set |
20   "(" ConstExpression ")" | NOT ConstFactor.
21 set = [qualident] "(" [element "(" ConstExpression)"] ".
22 element = ConstExpression | "." ConstExpression].
23 TypeDeclaration = ident "=" type.
24 type = SimpleType | ArrayType | RecordType | SetType |
25   PointerType | ProcedureType.
26 SimpleType = qualident | enumeration | SubrangeType.
27 enumeration = "(" identList ")".
28 identList = ident ("," ident).
29 SubrangeType = "[" ConstExpression "," ConstExpression "]" type.
30 ArrayType = ARRAY SimpleType ("," SimpleType) OF type.
31 RecordType = RECORD FieldListSequence END.
32 FieldListSequence = FieldList (";" FieldList).
33 FieldList = identList ":" type |
34   CASE [ident ":"] qualident OF variant ("|" variant)
35   [ELSE FieldListSequence] END.
36 variant = CaseLabelList ":" FieldListSequence.
37 CaseLabelList = CaseLabels ("," CaseLabels).
38 CaseLabels = ConstExpression ["." ConstExpression].
39 SetType = SET OF SimpleType.
40 PointerType = POINTER TO type.
41 ProcedureType = PROCEDURE [FormalTypeList].
42 FormalTypeList = "(" ["[VAR] FormalType" |
43   "(" [VAR] FormalType)"] ")" ":" FormalType.
44 VariableDeclaration = identList ":" type.
```

158 A1. The Syntax of Modula-2

```
45 designator = qualident ("." ident | "[" ExpList "]" | "("").
46 ExpList = expression ("," expression).
47 expression = SimpleExpression [relation SimpleExpression].
48 SimpleExpression = ["+"|"-"] term (AddOperator term).
49 term = factor (MulOperator factor).
50 factor = number | string | set | designator [ActualParameters] |
51   "(" expression ")" | NOT factor.
52 ActualParameters = "(" [ExpList] ")" ".
53 statement = [assignment | ProcedureCall |
54   IFStatement | CaseStatement | WhileStatement |
55   RepeatStatement | LoopStatement | ForStatement |
56   WithStatement | EXIT | RETURN [expression] ].
57 assignment = designator "=" expression.
58 ProcedureCall = designator [ActualParameters].
59 StatementSequence = statement (";" statement).
60 IFStatement = IF expression THEN StatementSequence
61   [ELSEIF expression THEN StatementSequence]
62   [ELSE StatementSequence] END.
63 CaseStatement = CASE expression OF case ("|" case)
64   [ELSE StatementSequence] END.
65 case = CaseLabelList ":" StatementSequence.
66 WhileStatement = WHILE expression DO StatementSequence END.
67 RepeatStatement = REPEAT StatementSequence UNTIL expression.
68 ForStatement = FOR ident "=" expression TO expression
69   [BY ConstExpression] DO StatementSequence END.
70 LoopStatement = LOOP StatementSequence END.
71 WithStatement = WITH designator DO StatementSequence END.
72 ProcedureDeclaration = ProcedureHeading ":" block ident.
73 ProcedureHeading = PROCEDURE ident [FormalParameters].
74 block = (declaration) (BEGIN StatementSequence) END.
75 declaration = CONST (ConstantDeclaration ";" ) |
76   TYPE (TypeDeclaration ";" ) |
77   VAR (VariableDeclaration ";" ) |
78   ProcedureDeclaration ";" | ModuleDeclaration ";".
79 FormalParameters =
80   "(" [FPSection (";" FPSection)] ")" [";" qualident].
81 FPSection = [VAR] identList ":" FormalType.
82 FormalType = [ARRAY OF] qualident.
83 ModuleDeclaration =
84   MODULE ident [priority] ":" {import} [export] block ident.
85 priority = "(" ConstExpression ")".
86 export = EXPORT [QUALIFIED] identList ":".
87 import = [FROM ident] IMPORT identList ":".
88 DefinitionModule = DEFINITION MODULE ident ":" {import}
89   [export] (definition) END ident ":".
90 definition = CONST (ConstantDeclaration ";" ) |
91   TYPE (ident ["=" type] ";" ) |
92   VAR (VariableDeclaration ";" ) |
93   ProcedureHeading ":".
94 ProgramModule =
95   MODULE ident [priority] ":" {import} block ident ":".
```

## N. Wirth: Programming in Modula-2

## Appendix 1

## The Syntax of Modula-2

```

1  ident = letter (letter | digit),
2
3  number = integer | real,
4
5  integer = digit {digit} | octalDigit {octalDigit} |
6  digit (hexDigit) "h",
7
8  real = digit (digit) "." digit [ScaleFactor],
9  ScaleFactor = "E" ["+"|"-"] digit (digit),
10
11 hexDigit = digit | "A"|"B"|"C"|"D"|"E"|"F",
12
13 digit = octalDigit | "8"|"9",
14
15 octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7",
16
17 string = "" (character) "" | " (character) "
18
19 qualified = ident (".", ident),
20
21 ConstantDeclaration = ident "=" ConstExpr
22
23 ConstExpr = SimpleConstExpr [relation]
24
25 relation = "<" | ">" | "<=" | ">=" | "==" | "!="
26
27 SimpleConstExpr = ["+"|"-"] ConstTerm (A
28
29 AddOperator = "+" | "-" | "0",
30
31 ConstTerm = ConstFactor (MulOperator Const
32
33 MulOperator = "*" | "/" | DIV | MOD | AND
34
35 ConstFactor = qualified | number | string |
36
37 "(" ConstExpression ")" | NOT ConstFact
38
39 set = [qualified] {"[" element "]" element
40
41 element = ConstExpression ["." ConstExpres
42
43 TypeDeclaration = ident "=" type,
44
45 type = SimpleType | ArrayType | RecordType | SetType |
46
47 PointerType | ProcedureType,
48
49 SimpleType = qualified | enumeration | SubrangeType,
50
51 enumeration = "(" (Identifier ")"
52
53 Identifier = ident (".", ident),
54
55 SubrangeType = "[" ConstExpression "," ConstExpression "]"
56
57 ArrayType = ARRAY SimpleType ("[" SimpleType] OF type,
58
59 RecordType = RECORD FieldListSequence END,
60
61 FieldListSequence = FieldList {"," FieldList},
62
63 FieldList = [Identifier ":" type |
64
65 CASE [ident ":"] qualifiedOf variant ("[" variant]
66
67 [ELSE FieldListSequence] END),
68
69 variant = CaseLabelList ":" FieldListSequence,
70
71 CaseLabelList = CaseLabels ("[" CaseLabels],
72
73 CaseLabels = ConstExpression ["." ConstExpression],
74
75 SetType = SET OF SimpleType,
76
77 PointerType = POINTER TO type,
78
79 ProcedureType = PROCEDURE [FormalTypeList],
80
81 FormalTypeList = "(" ["[" [VAR] FormalType
82
83 {"[" [VAR] FormalType"] "}" ["." qualified],
84
85 VariableDeclaration = Identifier ":" type,

```

```

1  ident = letter {letter | digit}.
2  number = integer | real.
3  integer = digit {digit} | octalDigit {octalDigit} ("B"|"C") |
4    digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [ScaleFactor].
6  ScaleFactor = "E" ["+"|"-" ] digit {digit}.
7  hexDigit = digit | "A"|"B"|"C"|"D"|"E"|"F".
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".

```

158 A1. The Syntax of Modula-2

```

72 ProcedureDeclaration = ProcedureHeading ":" block ident.
73
74 ProcedureHeading = PROCEDURE ident [FormalParameters].
75 block = (declaration) [BEGIN StatementSequence] END.
76 declaration = CONST (ConstantDeclaration ";" )
77 TYPE (TypeDeclaration ";" )
78 VAR (VariableDeclaration ";" )
79 ProcedureDeclaration ":" | ModuleDeclaration ";".
80 FormalParameters = "(" [FSection ";" FSection] ")" [";" qualident].
81 FSection = [VAR] IdentList ";" FormalType.
82 FormalType = [ARRAY OF] qualident.
83 ModuleDeclaration =
84 MODULE ident [priority] ";" (import) [export] block ident.
85 priority = "(" [ConstExpression] ")".
86 export = EXPORT [QUALIFIED] IdentList ";".
87 import = [FROM] ident IMPORT IdentList ";".
88 DefinitionModule = DEFINITION MODULE ident ";" (import)
89 [export] [definition] END ident ";".
90 definition = CONST (ConstantDeclaration ";" )
91 TYPE (ident ("=" type) ";" )
92 VAR (VariableDeclaration ";" )
93 ProcedureHeading ";" .
94
95 ProgramModule =
96 MODULE ident [priority] ";" (import) block ident ";".

```

## CHAPTER 18

### Syntax

THIS chapter presents a grammar for the Java programming language. The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *x / y* means one of either *x* or *y*.

*Identifier:*  
*IDENTIFIER*

*QualifiedIdentifier:*  
*Identifier* [ *.* *Identifier* ]

*QualifiedIdentifierList:*  
*QualifiedIdentifier* [ *,* *QualifiedIdentifier* ]

*EnumBody:*  
{ *[EnumConstants]* [ *.* ] *[EnumBodyDeclarations]* }

*EnumConstants:*  
*EnumConstant*  
*EnumConstants* , *EnumConstant*

*EnumConstant:*  
*[Annotations]* *Identifier* *[Arguments]* *[ClassBody]*

*EnumBodyDeclarations:*  
*:* *[ClassBodyDeclaration]*

*AnnotationTypeBody:*  
{ *[AnnotationTypeElementDeclarations]* }

*AnnotationTypeElementDeclarations:*  
*AnnotationTypeElementDeclaration*  
*AnnotationTypeElementDeclarations* *AnnotationTypeElementDeclaration*

*AnnotationTypeElementDeclaration:*  
*[Modifier]* *AnnotationTypeElementRest*

*AnnotationTypeElementRest:*  
*Type* *Identifier* *AnnotationMethodOrConstantRest* *:*  
*ClassDeclaration*  
*InterfaceDeclaration*  
*EnumDeclaration*  
*AnnotationTypeDeclaration*

*AnnotationMethodOrConstantRest:*  
*AnnotationMethodRest*  
*ConstantDeclaratorsRest*

*AnnotationMethodRest:*  
( ) [ *[1]* ] *[default ElementValue]*

## CHAPTER 18

*EnumBody:*  
{ [*EnumConstants*] [ . ] [*EnumBodyDeclarations*] }

*EnumConstants:*  
*EnumConstant*

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *x | y* means one of either *x* or *y*.

*AnnotationTypeElementDeclaration:*  
*[Modifier]* *AnnotationTypeElementRest*

*AnnotationTypeElementRest:*  
*Type* *Identifier* *AnnotationMethodOrConstantRest* ;  
*ClassDeclaration*  
*InterfaceDeclaration*  
*EnumDeclaration*  
*AnnotationTypeDeclaration*

*AnnotationMethodOrConstantRest:*  
*AnnotationMethodRest*  
*ConstantDeclaratorsRest*

*AnnotationMethodRest:*  
( ) [ *{} [ default *ElementValue* ]* ]

THIS chapter presents a grammar for the Java programming language. The grammar presented piecemeal in the preceding chapters (§§ 13.1–13.5) for exposition, but it is not well suited as a basis for a parser. The grammar in this chapter is the basis for the reference implementation. Note that the LL(1) grammar, though in many cases it minimizes the necessary

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *x | y* means one of either *x* or *y*.

*Identifier:*  
*IDENTIFIER*

*QualifiedIdentifier:*  
*Identifier* [ . *Identifier* ]

*QualifiedIdentifierList:*  
*QualifiedIdentifier* [ , *QualifiedIdentifier* ]

## CHAPTER 18

### Syntax

THIS chapter presents a grammar for the Java programming language. The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions

- $[x]$  denotes zero or one occurrences of  $x$ .
- $\{x\}$  denotes zero or more occurrences of  $x$ .
- $x/y$  means one of either  $x$  or  $y$ .

*Identifier:*  
*IDENTIFIER*

*QualifiedIdentifier:*  
*Identifier*  $\{ \cdot \textit{Identifier} \}$

*QualifiedIdentifierList:*  
*QualifiedIdentifier*  $\{ , \textit{QualifiedIdentifier} \}$

*EnumBody:*  
 $\{ [\textit{EnumConstants}] [ \cdot ] [\textit{EnumBodyDeclarations}] \}$

*EnumConstants:*  
*EnumConstant*  
*EnumConstants*  $, \textit{EnumConstant}$

*EnumConstant:*  
 $[\textit{Annotations}] \textit{Identifier} [\textit{Arguments}] [\textit{ClassBody}]$

*EnumBodyDeclarations:*  
 $\cdot [\textit{ClassBodyDeclaration}]$

*AnnotationTypeBody:*  
 $\{ [\textit{AnnotationTypeElement}] [\textit{AnnotationTypeElementList}] \}$

*QualifiedIdentifier:*  
*Identifier*  $\{ \cdot \textit{Identifier} \}$

*QualifiedIdentifierList:*  
*QualifiedIdentifier*  $\{ , \textit{QualifiedIdentifier} \}$

$( ) [\{ \}] [\textit{default} \textit{ElementValue}]$

## Syntax

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

```
AnnotationTypeBody:
    {
        declarations:
        declaration
        declarations AnnotationTypeElementDeclaration
        declaration:
        elementRest
        methodOrConstantRest :
    }

classBody]

rest:
    value]
```

*EnumConstants:*  
*EnumConstant*  
*EnumConstants* , *EnumConstant*

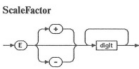
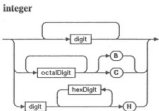
*EnumConstant:*  
*[Annotations] Identifier [Arguments] [ClassBody]*

*EnumBodyDeclarations:*  
*; {ClassBodyDeclaration}*

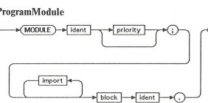
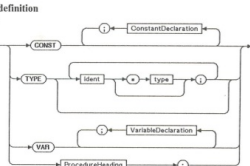
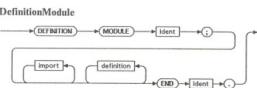
# N. Wirth: Programming in Modula-2

## Appendix 4

### Syntax Diagrams



180 A4. Syntax Diagrams





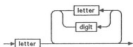
# N. Wirth: Programming in Modula-2

180 A4. Syntax Diagrams

## Appendix 4

### Syntax Diagrams

ident



number



integer



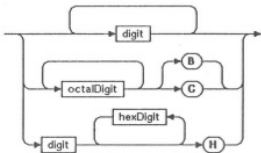
real



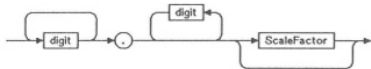
ScaleFactor



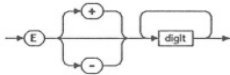
integer



real



ScaleFactor



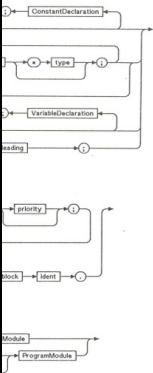
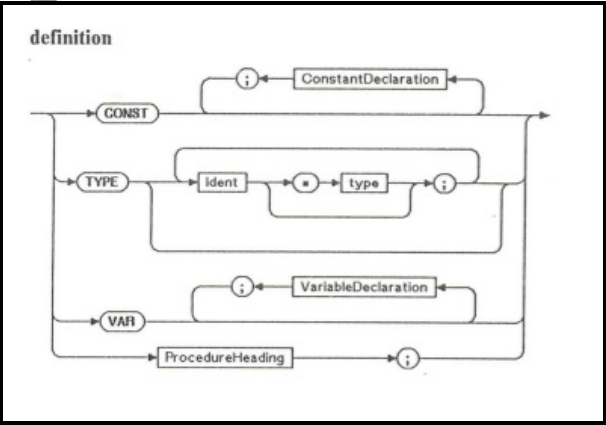
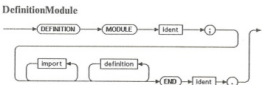
# N. Wirth: Programming in Modula-2

Appendix 4

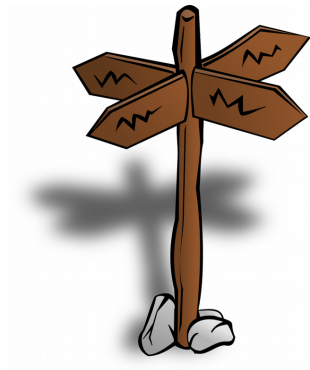
Syntax Diagrams



180 A4. Syntax Diagrams



- Reguläre Sprachen
- Kontextfreie Grammatiken
  - Motivation & Definition
  - Beispiele
  - Andere Notationen
  - Style Guide für Grammatiken
  - Grammatiken in freier Wildbahn
  - Jenseits der Kontextfreiheit



# Jenseits der Kontextfreiheit

---

## Formale Sprachen

$\{ a^n b^n c^n \mid n \geq 0 \}$  ist nicht kontextfrei.

## Formale Sprachen

$\{ ww \mid w \in \{a, b, c\}^* \}$  ist nicht kontextfrei.

## Programmiersprachen

Typen- und Deklarationsbedingungen nicht oder nur schwer kontextfrei darstellbar.

## Natürliche Sprachen (Schweizer Dialekt)

Mer d'chind em Hans es huus  
haend wele laa hälfe aastriche.

Wir wollten die Kinder dem Hans  
helfen lassen das Haus anzustreichen.

- Reguläre Sprache besitzen die gleich Ausdrucksstärke wie DEAs und NEAs.
- Gibt es auch einen passenden Automaten für Kontextfreie Sprachen?  
⇨ Ja, nichtdeterministische Kellerautomaten (pushdown automata)
- Kellerautomat = endlicher Automat + Kellerspeicher (Stack)
  - Kellerspeicher ermöglicht unendlich viele “Zustände”
  - Automat sieht nur das oberste Symbol im Keller (und Zustandsübergänge hängen davon ab)
  - Bei einem Übergang können neue Symbole oben auf den Keller geschrieben werden oder das oberste Symbol kann gelöscht werden.

# Andere Arten von Grammatiken

---

- Wir haben nur kontextfreie Grammatiken (Typ-2 Grammatik) betrachtet
- Es gibt ausdrucksstärkere Grammatiken:
  - uneingeschränkte formale Grammatik (Typ-0)
  - kontextsensitive Grammatik (Typ-1)
- ... und einfachere Arten von Grammatiken:
  - reguläre Grammatiken (Typ-3) definieren die regulären Sprachen und besitzen damit dieselbe Ausdrucksstärke wie endliche Automaten und reguläre Ausdrücke.

- Grammatik definieren formale Sprachen
- Kontextfreie Grammatiken
  - ausdrucksstärker als reguläre Ausdrücke und endliche Automaten
  - darstellbar als rekursive Syntaxdiagramme
- Style Guide für Grammatiken
  - EBNF kombiniert reguläre Ausdrücke und kontextfreie Produktionen