3.0 VU Formale Modellierung

Gernot Salzer

Arbeitsbereich Theoretische Informatik und Logik
Institut für Computersprachen

SS 2016

Inhalt

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

Wohlgeklammerte Ausdrücke sind nicht regulär

$\{(), (()), ()(), ((())), (())(), ()()(), \ldots\}$

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

$\{\varepsilon, ab, aabb, aaabbb, aaaabbbb, aaaaabbbbb, \ldots\}$

Kontextfreie Grammatiken

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

$$
\begin{array}{lll}
\mathit{Satz} \to \mathit{HwP}\ \mathit{ZwP} & & \mathit{Art} \to \texttt{der} \mid \texttt{den} \\
\mathit{HwP} \to \mathit{Art}\ \mathit{Hw} & & \mathit{Hw} \to \texttt{Hund} \mid \texttt{Mond} \\
\mathit{ZwP} \to \mathit{Hzw}\ \mathit{HwP}\ \mathit{Zw} & & \mathit{Hzw} \to \texttt{wird} \\
 & & \mathit{Zw} \to \texttt{anbellen}
\end{array}
$$

$$
\begin{aligned}
\mathit{Satz} &\Rightarrow_p \mathit{HwP}\ \ \mathit{ZwP} \\
&\Rightarrow_p \mathit{Art}\ \ \mathit{Hw}\ \ \mathit{Hzw}\ \ \mathit{HwP}\ \ \mathit{Zw} \\
&\Rightarrow_p \texttt{der}\ \ \texttt{Hund}\ \ \texttt{wird}\ \ \mathit{Art}\ \ \mathit{Hw}\ \ \texttt{anbellen} \\
&\Rightarrow_p \texttt{der}\ \ \texttt{Hund}\ \ \texttt{wird}\ \ \texttt{den}\ \ \texttt{Mond}\ \ \texttt{anbellen}
\end{aligned}
$$

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: $A \to w$ statt $(A, w) \in P$

$A \to w_1 \mid \cdots \mid w_n$ statt $A \to w_1, \ldots, A \to w_n$

Ableitbarkeit

... in einem Schritt:

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

... in mehreren Schritten:

$u \overset{*}{\Rightarrow} v$, falls

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

Von Grammatik G generierte Sprache

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

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

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

$S \overset{*}{\Rightarrow} \mathtt{aabb}$, weil $\quad S \Rightarrow \mathtt{a}\, S\, \mathtt{b} \Rightarrow \mathtt{aa}\, S\, \mathtt{bb} \Rightarrow \mathtt{aa}\, \varepsilon\, \mathtt{bb} = \mathtt{aabb}$

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

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 $A \to w \in P$ und $x \in T^*$
(In jedem Schritt wird die linkeste Variable ersetzt.)

Rechtsableitbarkeit: $x\,A\,y \Rightarrow_R x\,w\,y$ falls $A \to w \in P$ 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 $A_1 \to w_1, \ldots, A_n \to w_n \in P$ und $x_0, \ldots, x_n \in T^*$
(In jedem Schritt werden alle Variablen ersetzt.)

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

$$\mathcal{L}(G) = \{\, w \in T^* \mid S \overset{*}{\Rightarrow} w \,\} = \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_L w \,\}$$
$$= \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_R w \,\} = \{\, w \in T^* \mid S \overset{*}{\Rightarrow}_P w \,\}$$

Inhalt

Syntax aussagenlogischer Formeln

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

$T = \{\top,\ \bot,\ \neg,\ (,\ ),\ \wedge,\ \uparrow,\ \vee,\ \downarrow,\ \equiv,\ \not\equiv,\ \supset,\ \subset\} \cup \mathcal{V}$

$$P = \{\ \begin{aligned} Fm &\to Var \mid \top \mid \bot \mid \neg Fm \mid (\ Fm\ Op\ Fm\ )\ , \\ Op &\to \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \equiv \mid \not\equiv \mid \supset \mid \subset\ , \\ Var &\to \mathtt{A} \mid \mathtt{B} \mid \mathtt{C} \mid \cdots\ \} \end{aligned}$$

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

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

Syntax aussagenlogischer Formeln

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

$T = \{\top,\ \bot,\ \neg,\ (,\ ),\ \wedge,\ \uparrow,\ \vee,\ \downarrow,\ \equiv,\ \not\equiv,\ \supset,\ \subset\} \cup \mathcal{V}$

$$P = \{\ \begin{aligned} Fm &\to Var \mid \top \mid \bot \mid \neg Fm \mid (\ Fm\ Op\ Fm\ )\ , \\ Op &\to \wedge \mid \uparrow \mid \vee \mid \downarrow \mid \equiv \mid \not\equiv \mid \supset \mid \subset\ , \\ Var &\to \mathtt{A} \mid \mathtt{B} \mid \mathtt{C} \mid \cdots\ \} \end{aligned}$$

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

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

Reelle Numerale

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

$V = \{\mathit{Real}, \mathit{Scale}, \mathit{Sign}, \mathit{Digits}, \mathit{Digit}\}$
$T = \{0, \ldots, 9, ., \mathtt{E}, +, -\}$

und P folgende Produktionen sind:

$$
\begin{aligned}
\mathit{Real} &\to \mathit{Digit}\ \mathit{Digits}\ .\ \mathit{Digits}\ \mathit{Scale} \\
\mathit{Scale} &\to \varepsilon \mid \mathtt{E}\ \mathit{Sign}\ \mathit{Digit}\ \mathit{Digits} \\
\mathit{Sign} &\to \varepsilon \mid + \mid - \\
\mathit{Digits} &\to \varepsilon \mid \mathit{Digit}\ \mathit{Digits} \\
\mathit{Digit} &\to 0 \mid \cdots \mid 9
\end{aligned}
$$

Optionalität:
$A \to \varepsilon \mid B$ (A steht für optionales B)

Wiederholung:
$A \to \varepsilon \mid B\,A$ (A steht für wiederholtes B, auch 0 Mal)
$A \to B \mid B\,A$ (A steht für wiederholtes B, mind. 1 Mal)

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 $w_1, w_2 \in WKA$

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

Beispiel einer Ableitung:

$$\begin{aligned} W &\Rightarrow_P W\, W \\ &\Rightarrow_P (W)(W) \\ &\Rightarrow_P ()(W\, W) \\ &\Rightarrow_P ()((W)(W)) \\ &\Rightarrow_P ()(()()) \end{aligned}$$

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

Induktive Definition vs. kontextfreie Grammatik

Induktive Definition für $\mathcal{M}$	kontextfreie Grammatik für $\mathcal{M}$
Mengen $\mathcal{M}$, $\mathcal{M}_0$, A_1, B_1, ...	Var. $\langle\mathcal{M}\rangle$, $\langle\mathcal{M}_0\rangle$, $\langle A_1\rangle$, $\langle B_1\rangle$, ...
$\mathcal{M}$ ist die kleinste Menge, sodass:	$\mathcal{M} = \mathcal{L}(\langle \ldots, P, \langle\mathcal{M}\rangle\rangle)$, wobei P folgende Produktionen sind:
$\mathcal{M}_0 \subseteq \mathcal{M}$	$\langle\mathcal{M}\rangle \to \langle\mathcal{M}_0\rangle$
$f(x_1, \ldots, x_m) \in \mathcal{M}$, falls $x_1 \in A_1, \ldots, x_m \in A_m$	$\langle\mathcal{M}\rangle \to f(\langle A_1\rangle, \ldots, \langle A_m\rangle)$
$g(x_1, \ldots, x_n) \in \mathcal{M}$, falls $x_1 \in B_1, \ldots, x_n \in B_n$	$\langle\mathcal{M}\rangle \to g(\langle B_1\rangle, \ldots, \langle B_n\rangle)$
$\cdots \in \mathcal{M}$, falls ...	$\langle\mathcal{M}\rangle \to \ldots$
$h(x_1, x_2) \in \mathcal{M}$, falls $x_1, x_2 \in \mathcal{M}$	$\langle\mathcal{M}\rangle \to h(\langle\mathcal{M}\rangle, \langle\mathcal{M}\rangle)$
$h(x, x) \in \mathcal{M}$, falls $x \in \mathcal{M}$	keine Entsprechung, nicht kontextfrei

Inhalt

Mehrdeutigkeit

Eine Grammatik heißt mehrdeutig, wenn es ein Wort mit mehreren Linksableitungen gibt.
Eine kontextfreie Sprache heißt (inhärent) mehrdeutig, wenn alle kontextfreien Grammatiken für sie mehrdeutig sind.

If-then-else Anweisungen

$G_1 = \langle \{A\}, \{\texttt{if}, \texttt{then}, \texttt{else}, \texttt{expr}, \texttt{others}\}, P_1, A \rangle$

mit $P_1 = \{A \to \texttt{if expr then } A \mid \texttt{if expr then } A \texttt{ else } A \mid \texttt{others}\}$.

G_1 ist mehrdeutig, da z.B. das Wort

$w = \texttt{if expr then if expr then others else others}$

zwei Linksableitungen besitzt:

$$A \Rightarrow_L \begin{cases} \texttt{if expr then } A \\ \texttt{if expr then } A \texttt{ else } A \end{cases}$$
$$\Rightarrow_L \texttt{if expr then if expr then } A \texttt{ else } A \overset{*}{\Rightarrow}_L w$$

If-then-else Anweisungen (Fortsetzung)

Die Sprache $\mathcal{L}(G_1)$ ist nicht mehrdeutig, da es eine zu G_1 äquivalente, aber eindeutige Grammatik G_2 gibt:

$$G_2 = \langle \{A, T, TE\}, \{\texttt{if}, \texttt{then}, \texttt{else}, \texttt{expr}, \texttt{others}\}, P_2, A \rangle, \text{ wobei}$$

$$\begin{aligned} P_2 = \{ \quad A &\to T \mid TE \\ T &\to \texttt{if expr then } A \mid \texttt{if expr then } TE \texttt{ else } T \\ TE &\to \texttt{if expr then } TE \texttt{ else } TE \mid \texttt{others} \quad \} \end{aligned}$$

In G_2 besitzt w nur eine einzige Linksableitung:

$$\begin{aligned} A &\Rightarrow_L T \\ &\Rightarrow_L \texttt{if expr then } A \\ &\Rightarrow_L \texttt{if expr then } TE \\ &\Rightarrow_L \texttt{if expr then if expr then } TE \texttt{ else } TE \\ &\Rightarrow_L \texttt{if expr then if expr then others else } TE \\ &\Rightarrow_L \texttt{if expr then if expr then others else others} = w \end{aligned}$$

In mehrdeutigen Grammatiken gibt es Wörter mit mehreren Interpretationen.
$\Rightarrow$ Problem für die weitere Verarbeitung

Problem von G_1: Die Zugehörigkeit der `else`-Zweige zu den `if`s ist nicht eindeutig festgelegt.

G_2: Jeder `else`-Zweig gehört zum letzten unvollständigen `if`.

Inhärent mehrdeutige Sprachen

Sei $L = L_1 \cup L_2$, wobei

$$L_1 = \{ \mathrm{a}^m \mathrm{b}^m \mathrm{c}^n \mid m, n \geq 0 \}$$
$$L_2 = \{ \mathrm{a}^m \mathrm{b}^n \mathrm{c}^n \mid m, n \geq 0 \}$$

L ist kontextfrei:

$$S \to S_1 \mid S_2$$

$$S_1 \to S_1\,\mathrm{c} \mid A \qquad S_2 \to \mathrm{a}\,S_2 \mid B$$
$$A \to \mathrm{a}\,A\,\mathrm{b} \mid \varepsilon \qquad B \to \mathrm{b}\,B\,\mathrm{c} \mid \varepsilon$$

Die Grammatik ist mehrdeutig: $\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n$ besitzt zwei Linksableitungen.

Man kann zeigen, dass alle Grammatiken für L mehrdeutig sind, d.h., L ist inhärent mehrdeutig.

Inhalt

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.

$$
\begin{aligned}
\langle \mathit{Real} \rangle &::= \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \, \texttt{"."} \, \langle \mathit{Digits} \rangle \, \langle \mathit{Scale} \rangle \\
\langle \mathit{Digit} \rangle &::= \texttt{"0"} \mid \cdots \mid \texttt{"9"} \\
\langle \mathit{Digits} \rangle &::= \varepsilon \mid \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \\
\langle \mathit{Scale} \rangle &::= \varepsilon \mid \texttt{"E"} \, \langle \mathit{Sign} \rangle \, \langle \mathit{Digit} \rangle \, \langle \mathit{Digits} \rangle \\
\langle \mathit{Sign} \rangle &::= \varepsilon \mid \texttt{"+"} \mid \texttt{"-"}
\end{aligned}
$$

Erweiterte Backus-Naur-Form (EBNF)

„Regular Right Part Grammar“ (RRPG)

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

Elimination der EBNF-Notation:

$A \to w_1\,(w_2)\,w_3$ entspricht $A \to w_1\,B\,w_3$, $B \to w_2$

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

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

Listen von Bezeichnern

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

Ohne EBNF:
$$\begin{aligned}\mathit{IdList} &\rightarrow \mathit{Id}\;B\\ B &\rightarrow \varepsilon \mid \texttt{","}\;\mathit{Id}\;B\end{aligned}$$

oder $\mathit{IdList} \rightarrow \mathit{Id} \mid \mathit{Id}\;\texttt{","}\;\mathit{IdList}$

Skalierungsfaktor der reellen Numerale

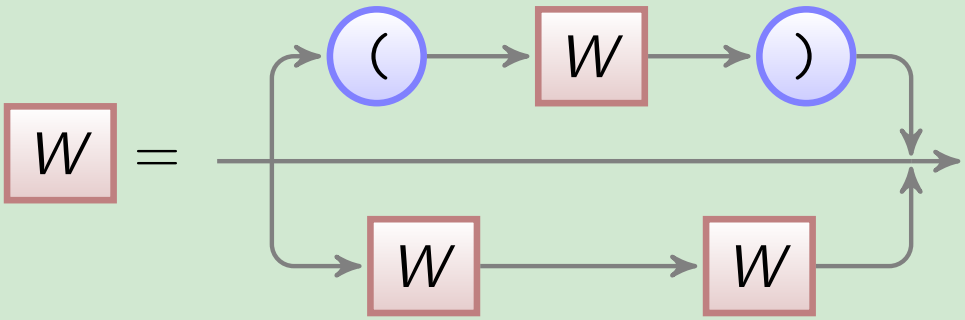
EBNF: $\mathit{Scale} \rightarrow \texttt{"E"}\,[\texttt{"+"} \mid \texttt{"-"}]\,\mathit{Digit}\,\{\mathit{Digit}\}$

Ohne EBNF:
$$\begin{aligned}\mathit{Scale} &\rightarrow \texttt{"E"}\;B_1\;\mathit{Digit}\;B_2\\ B_1 &\rightarrow \varepsilon \mid \texttt{"+"} \mid \texttt{"-"}\\ B_2 &\rightarrow \varepsilon \mid \mathit{Digit}\;B_2\end{aligned}$$

Syntaxdiagramme

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

Wohlgeklammerte Ausdrücke



$$W \to (\; W \;)$$
$$\mid \; \varepsilon$$
$$\mid \; W \; W$$

Inhalt

Style Guide für Grammatiken

Wiederholungsoperator statt Rekursion (wenn möglich).

Schlecht: $A \to a \mid A\,A$ oder $A \to a \mid a\,A$

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

Optionsoperator statt Leerwort.

Schlecht: $\varepsilon \mid A$

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

Modularisierung statt Duplizierung.

Schlecht: $\mathit{Identifier} \rightarrow (\texttt{"a"} \mid \cdots \mid \texttt{"z"})\,\{\,\texttt{"a"} \mid \cdots \mid \texttt{"z"} \mid \texttt{"0"} \mid \cdots \mid \texttt{"9"}\,\}$

Besser:
$$\begin{aligned} \mathit{Identifier} &\rightarrow \mathit{Letter}\,\{\,\mathit{Letter} \mid \mathit{Digit}\,\} \\ \mathit{Letter} &\rightarrow \texttt{"a"} \mid \cdots \mid \texttt{"z"} \\ \mathit{Digit} &\rightarrow \texttt{"0"} \mid \cdots \mid \texttt{"9"} \end{aligned}$$

Verwendung sprechender Namen.

Schlecht:
$$\begin{aligned} X &\rightarrow Y\,\{\,Y \mid Z\,\} \\ Y &\rightarrow \texttt{"a"} \mid \cdots \mid \texttt{"z"} \\ Z &\rightarrow \texttt{"0"} \mid \cdots \mid \texttt{"9"} \end{aligned}$$

Besser: Siehe oben.

Klare Unterscheidung zwischen Terminalen, Nonterminalen und Metanotationen.

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

Besser: `"\begin{tabular}"` [`"["` *Position* `"]"`] `"{"` *Spalte* { *Spalte* } `"}"`

`"\begin{tabular}"` [`"["` ⟨*Position*⟩ `"]"`] `"{"` ⟨*Spalte*⟩ { ⟨*Spalte*⟩ } `"}"`

Inhaltliche Strukturierung statt Minimierung der Nonterminale und Regeln.

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

Besser:

$$\begin{aligned} \ldots &\to \texttt{"\textbackslash begin\{tabular\}"}\ [\mathit{Position}]\ \mathit{Spalten} \\ \mathit{Position} &\to \texttt{"[b]"} \mid \texttt{"[t]"} \\ \mathit{Spalten} &\to \texttt{"\{"}\ \mathit{Spalte}\ \{\mathit{Spalte}\}\ \texttt{"\}"} \\ \mathit{Spalte} &\to \texttt{"l"} \mid \texttt{"c"} \mid \texttt{"r"} \end{aligned}$$

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

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

$$
\begin{aligned}
P = \{\ \mathit{Tabelle} &\to \texttt{"\textbackslash begin\{tabular\}"}\ [\,\mathit{Position}\,]\ \mathit{Spalten} \\
&\quad\ \mathit{Zeilen} \\
&\quad\ \texttt{"\textbackslash end\{tabular\}"}\ , \\
\mathit{Position} &\to \texttt{"[b]"} \mid \texttt{"[t]"}\ , \\
\mathit{Spalten} &\to \texttt{"\{"}\ \mathit{Spalte}\ \{\,\mathit{Spalte}\,\}\ \texttt{"\}"}\ , \\
\mathit{Spalte} &\to \texttt{"l"} \mid \texttt{"c"} \mid \texttt{"r"}\ , \\
\mathit{Zeilen} &\to \mathit{Zeile}\ \{\ \texttt{"\textbackslash\textbackslash"}\ \mathit{Zeile}\ \}\ , \\
\mathit{Zeile} &\to \mathit{Eintrag}\ \{\ \texttt{"\&"}\ \mathit{Eintrag}\ \}\ , \\
\mathit{Eintrag} &\to \{\,\mathit{Tabelle} \mid \mathit{Zeichen}\,\}\ , \\
\mathit{Zeichen} &\to \texttt{"0"} \mid \cdots \mid \texttt{"9"} \mid \texttt{"a"} \mid \cdots \mid \texttt{"Z"} \mid \texttt{"."} \mid \texttt{"␣"}\ \}
\end{aligned}
$$

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

$T = \{\,\texttt{"0"}, \ldots, \texttt{"9"}, \texttt{"a"}, \ldots, \texttt{"z"}, \texttt{"A"}, \ldots, \texttt{"Z"},$
$\texttt{"."}, \texttt{"␣"}, \texttt{"\{"}, \texttt{"\}"}, \texttt{"["}, \texttt{"]"}, \texttt{"\&"}, \texttt{"\textbackslash"}\,\}$

Nur eine Rekursion für Schachtelung der Tabellen notwendig!

Inhalt

N. Wirth: Programming in Modula-2

Appendix 1

The Syntax of Modula–2

```
ident = letter {letter | digit}.
number = integer | real.
integer = digit {digit} | octalDigit {octalDigit} ("B"|"C")|
   digit {hexDigit} "H".
real = digit {digit} "." {digit} [ScaleFactor].
ScaleFactor = "E" ["+"|"-"] digit {digit}.
hexDigit = digit |"A"|"B"|"C"|"D"|"E"|"F".
digit = octalDigit | "8"|"9".
octalDigit = "0"|"1"|"2"|"3"|"4"|"5"|"6"|"7".
string = "'" {character} "'" | '"' {character} '"' .
qualident = ident {"." ident}.
ConstantDeclaration = ident "=" ConstExpression.
ConstExpression = SimpleConstExpr [relation SimpleConstExpr].
relation = "=" | "#" | "<>" | "<" | "<=" | ">" | ">=" | IN .
SimpleConstExpr = ["+"|"-"] ConstTerm {AddOperator ConstTerm}.
AddOperator = "+" | "-" | OR .
ConstTerm = ConstFactor {MulOperator ConstFactor}.
MulOperator = "*" | "/" | DIV | MOD | AND | "&" .
ConstFactor = qualident | number | string | set |
   "(" ConstExpression ")" | NOT ConstFactor.
set = [qualident] "{" [element {"," element}] "}".
element = ConstExpression [".." ConstExpression].
TypeDeclaration = ident "=" type.
type = SimpleType | ArrayType | RecordType | SetType |
   PointerType | ProcedureType.
SimpleType = qualident | enumeration | SubrangeType.
enumeration = "(" IdentList ")".
IdentList = ident {"," ident}.
SubrangeType = "[" ConstExpression ".." ConstExpression "]".
ArrayType = ARRAY SimpleType {"," SimpleType} OF type.
RecordType = RECORD FieldListSequence END.
FieldListSequence = FieldList {";" FieldList}.
FieldList = [IdentList ":" type |
   CASE [ident ":"] qualident OF variant {"|" variant}
   [ELSE FieldListSequence] END].
variant = CaseLabelList ":" FieldListSequence.
CaseLabelList = CaseLabels {"," CaseLabels}.
CaseLabels = ConstExpression [".." ConstExpression].
SetType = SET OF SimpleType.
PointerType = POINTER TO type.
ProcedureType = PROCEDURE [FormalTypeList].
FormalTypeList = "(" [[VAR] FormalType
   {"," [VAR] FormalType}] ")" [":" qualident].
VariableDeclaration = IdentList ":" type.
```



```
designator = qualident {"." ident | "[" ExpList "]" | "↑"}.
ExpList = expression {"," expression}.
expression = SimpleExpression [relation SimpleExpression].
SimpleExpression = ["+"|"-"] term {AddOperator term}.
term = factor {MulOperator factor}.
factor = number | string | set | designator [ActualParameters] |
   "(" expression ")" | NOT factor.
ActualParameters = "(" [ExpList] ")" .
statement = [assignment | ProcedureCall |
   IfStatement | CaseStatement | WhileStatement |
   RepeatStatement | LoopStatement | ForStatement |
   WithStatement | EXIT | RETURN [expression] ].
assignment = designator ":=" expression.
ProcedureCall = designator [ActualParameters].
StatementSequence = statement {";" statement}.
IfStatement = IF expression THEN StatementSequence
   {ELSIF expression THEN StatementSequence}
   [ELSE StatementSequence] END.
CaseStatement = CASE expression OF case {"|" case}
   [ELSE StatementSequence] END.
case = CaseLabelList ":" StatementSequence.
WhileStatement = WHILE expression DO StatementSequence END.
RepeatStatement = REPEAT StatementSequence UNTIL expression.
ForStatement = FOR ident ":=" expression TO expression
   [BY ConstExpression] DO StatementSequence END.
LoopStatement = LOOP StatementSequence END.
WithStatement = WITH designator DO StatementSequence END .
ProcedureDeclaration = ProcedureHeading ";" block ident.
ProcedureHeading = PROCEDURE ident [FormalParameters].
block = {declaration} [BEGIN StatementSequence] END.
declaration = CONST {ConstantDeclaration ";"} |
   TYPE {TypeDeclaration ";"} |
   VAR {VariableDeclaration ";"} |
   ProcedureDeclaration ";" | ModuleDeclaration ";".
FormalParameters =
   "(" [FPSection {";" FPSection}] ")" [":" qualident].
FPSection = [VAR] IdentList ":" FormalType.
FormalType = [ARRAY OF] qualident.
ModuleDeclaration =
   MODULE ident [priority] ";" {import} [export] block ident.
priority = "[" ConstExpression "]".
export = EXPORT [QUALIFIED] IdentList ";".
import = [FROM ident] IMPORT IdentList ";".
DefinitionModule = DEFINITION MODULE ident ";" {import}
   [export] {definition} END ident ".".
definition = CONST {ConstantDeclaration ";"} |
   TYPE {ident ["=" type] ";"} |
   VAR {VariableDeclaration ";"} |
   ProcedureHeading ";" .
ProgramModule =
   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  number = integer | real.
3  integer = digit {digit} | octalDigit {
4     digit {hexDigit} "H".
5  real = digit {digit} "." {digit} [Scale
6  ScaleFactor = "E" ["+"|"-"] digit {d
7  hexDigit = digit |"A"|"B"|"C"|"D"|
8  digit = octalDigit | "8"|"9".
9  octalDigit = "0"|"1"|"2"|"3"|"4"|"
10 string = "'" {character} "'" | '"' {cha
11 qualident = ident {"." ident}.
12 ConstantDeclaration = ident "=" Co
13 ConstExpression = SimpleConstExp
14 relation = "=" | "#" | "<>" | "<" | "
15 SimpleConstExpr = ["+"|"-"] Cons
16 AddOperator = "+" | "-" | OR .
17 ConstTerm = ConstFactor {MulOpe
18 MulOperator = "*" | "/" | DIV | M
19 ConstFactor = qualident | number
20    "(" ConstExpression ")" | NOT
21 set = [qualident] "{" [element {","
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 "]".
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}] ")" [":" qualident].
44 VariableDeclaration = IdentList ":" 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".
```

```
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 "." .
```

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:
`()` *[]]* *[*`default` *ElementValue]*

J. Gosling et al.: JAVA Language Specification (6.2.2012)

CHAPTER 18

THIS chapter presents a grammar fo

The grammar presented piecemeal in t
for exposition, but it is not well suited a
in this chapter is the basis for the refe
LL(1) grammar, though in many cases

The grammar below uses the followin

- *[x]* denotes zero or one occurrences
- *{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

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.

AnnotationTypeElementDeclarations AnnotationTypeElementDeclaration

AnnotationTypeElementDeclaration:
{Modifier} AnnotationTypeElementRest

AnnotationTypeElementRest:
Type Identifier AnnotationMethodOrConstantRest ;
ClassDeclaration
InterfaceDeclaration
EnumDeclaration
AnnotationTypeDeclaration

AnnotationMethodOrConstantRest:
AnnotationMethodRest
ConstantDeclaratorsRest

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

J. Gosling et al.: JAVA Language Specification (6.2.2012)

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 occurrence
- *x | y* means one of either x or y.

Identifier:
IDENTIFIER

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , Qualifiec

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

EnumConstants:
EnumConstant
EnumConstants , EnumConstant

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

EnumBodyDeclarations:
; *{ClassBodyDeclaration}*

AnnotationTypeBody:
{ *[AnnotationTypeElementDeclarations]* }

AnnotationTypeElementDeclarations:

QualifiedIdentifier:
Identifier { . Identifier }

QualifiedIdentifierList:
QualifiedIdentifier { , QualifiedIdentifier }

J. Gosling et al.: JAVA Language Specification (6.2.2012)

CHAPTER 18

Syntax

EnumConstants:
EnumConstant
EnumConstants , EnumConstant

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

EnumBodyDeclarations:
; {ClassBodyDeclaration}

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

EnumConstants:
EnumConstant
EnumConstants , EnumConstant

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

EnumBodyDeclarations:

eclarations] }

ations:
laration
larations AnnotationTypeElementDeclaration

ation:
lementRest

ethodOrConstantRest ;

Rest:

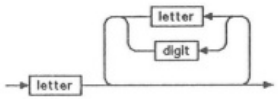
( ) *[]] [*`default` *ElementValue]*

N. Wirth: Programming in Modula-2

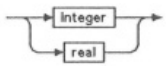
Appendix 4

Syntax Diagrams

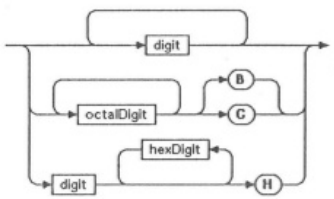
ident



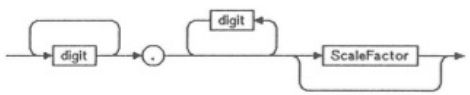
number



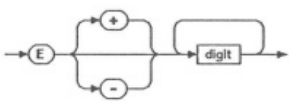
integer



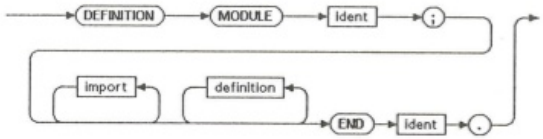
real



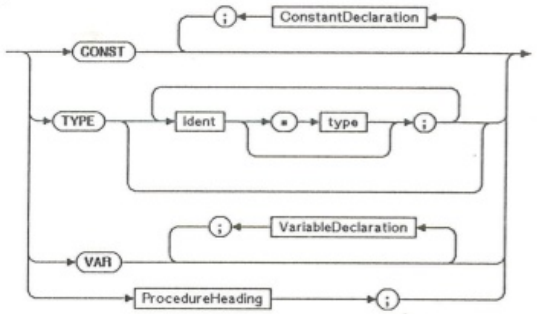
ScaleFactor



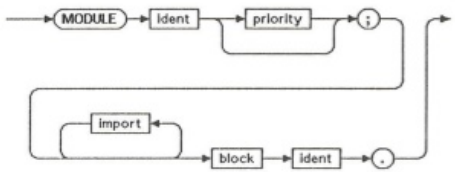
DefinitionModule



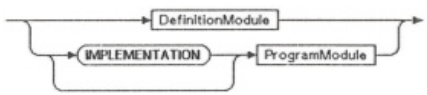
definition



ProgramModule



CompilationUnit

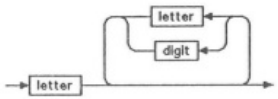


N. Wirth: Programming in Modula-2

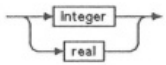
Appendix 4

Syntax Diagrams

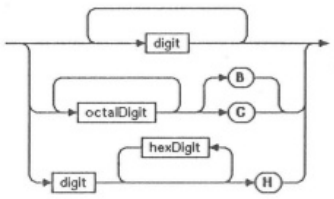
ident



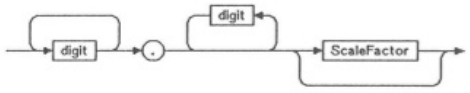
number



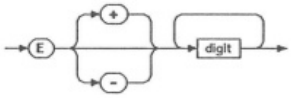
integer



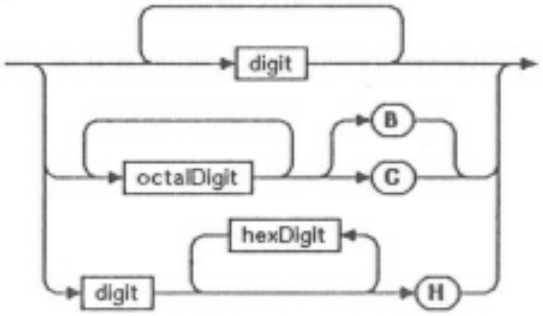
real



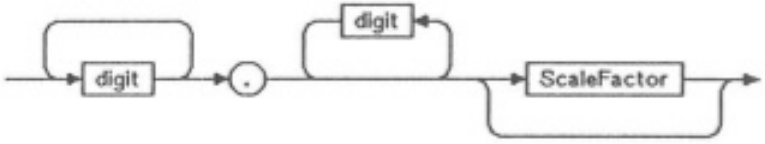
ScaleFactor



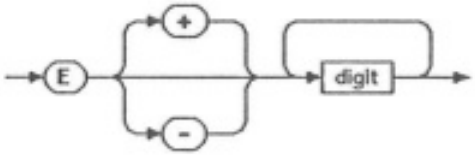
integer



real



ScaleFactor

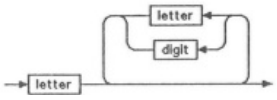


N. Wirth: Programming in Modula-2

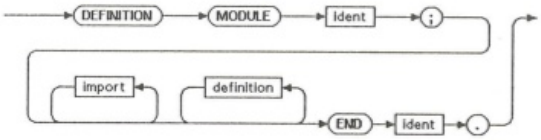
Appendix 4

Syntax Diagrams

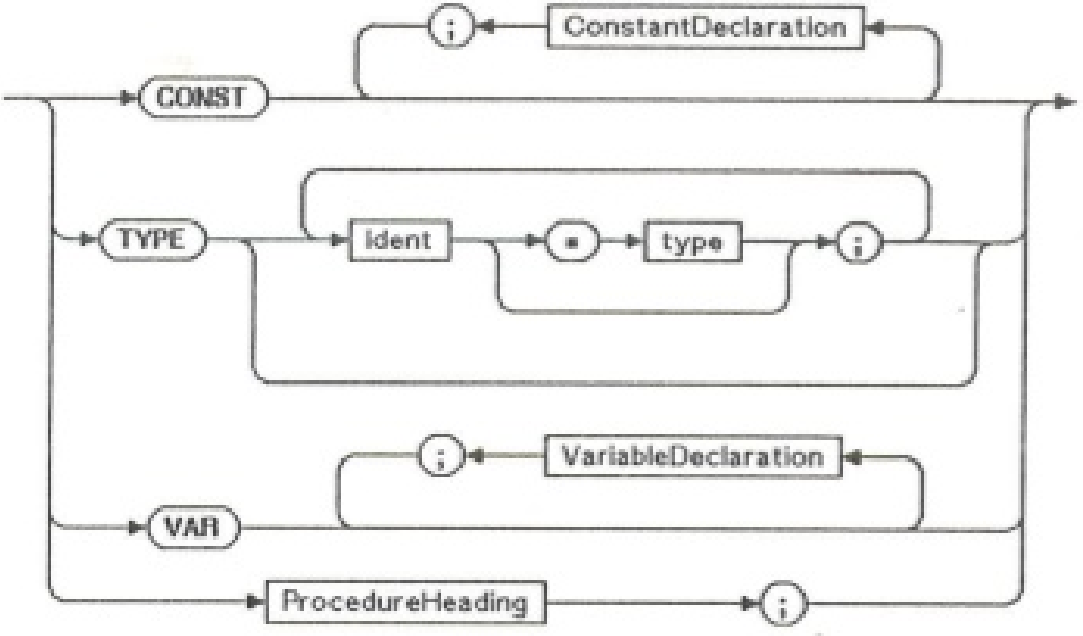
ident

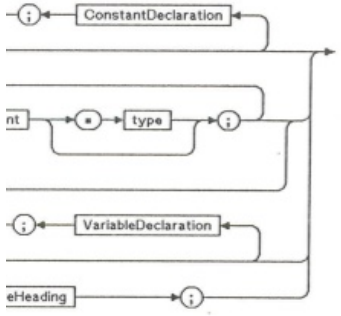


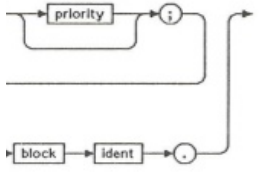
DefinitionModule

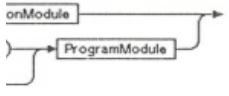


definition









Inhalt

Jenseits der Kontextfreiheit

Formale Sprachen

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

Formale Sprachen

$\{ ww \mid w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \}$ 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 aastriiche.

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