BNF: Backus-Naur Form
(auch: Backus Normal Form)

- **BNF ist eine Notationsform für kontextfreie Grammatiken**
  - häufig für Eingabesprachen eingesetzt
  - leichter lesbar
  - keine Erweiterungen zu kontextfreien Grammatiken

- **Elemente von BNF:**
  - Nonterminale: gekennzeichnet durch spitze Klammern <…>
  - Terminale: Symbole außerhalb spitzer Klammern
  - Symbol ´::=´: bedeutet ´ist definiert als´, i.e. ident zu →
  - Symbol ´|´ : bedeutet ´oder´

- **Rekursion zentral für BNF**

Beispiele BNF:
- <while> ::= while **(** <expression> **)** <statement>
- <if statement> ::= if <condition> then <statement> else <statement>

EBNF: Erweiterte Backus-Naur Form

- **Programmiersprachen werden häufig in EBNF beschrieben**
  - notationelle Erweiterungen zu kontextfreien Grammatiken: Notation für Wiederholung und Optionalität
  - EBNF nach ISO/IEC 14977:1996(E) standardisiert
  - viele leicht unterschiedliche Varianten im Gebrauch

- **Elemente von EBNF:**
  - Terminale: gekennzeichnet durch Anführungszeichen ´"´
  - Nonterminale: Symbole außerhalb Anführungszeichen ´"´
  - Symbol ´=´: bedeutet ´ist definiert als´, i.e. ident zu $\rightarrow$
  - Symbol ´|´ : bedeutet ´oder´
  - Symbole ´[ ]´: umgeben optionale Symbole
  - Symbole ´{ }´: umgeben sich wiederholende Symbole (0-mal oder öfter), mit Wiederholungen kann man Rekursionen ersetzen

Beispiele EBNF:

- parameter-list = parameter-declaration { "," parameter-declaration }
- if-statement = "if" condition "then" statement ["else" statement]

EBNF-ähnliche Notationsformen (1)

- **häufig werden EBNF-ähnliche Notationsformen verwendet**
  - in diesem Fall werden die verwendeten Notationsformen einleitend beschrieben
  - BNF/EBNF wurde definiert bevor kursiv, fett-gedruckt oder unterschiedliche Fonts zur Verfügung standen

- **Beispiel ISO-C Standard, Auszug aus Kap. 6.1 Notation:**

> In the syntax notation used in this clause, syntactic categories (nonterminals) are indicated by *italic type*, and literal words and character set members (terminals) by **`bold type`**. A colon (:) following a nonterminal introduces its definition. Alternative definitions are listed on separate lines, except when prefaced by the words "one of". An optional symbol is indicated by the subscript "opt", so that
>
> { $expression_{opt}$ }
>
> indicates an optional expression enclosed in braces.

EBNF-ähnliche Notationsformen (2)

- **Beispiel ISO-C Syntax:**

(6.8.3) *expression-statement:*
expression$_{opt}$ **;**

(6.8.4) *selection-statement:*
if **(** *expression* **)** *statement*
if **(** *expression* **)** *statement* **else** *statement*
switch **(** *expression* **)** *statement*

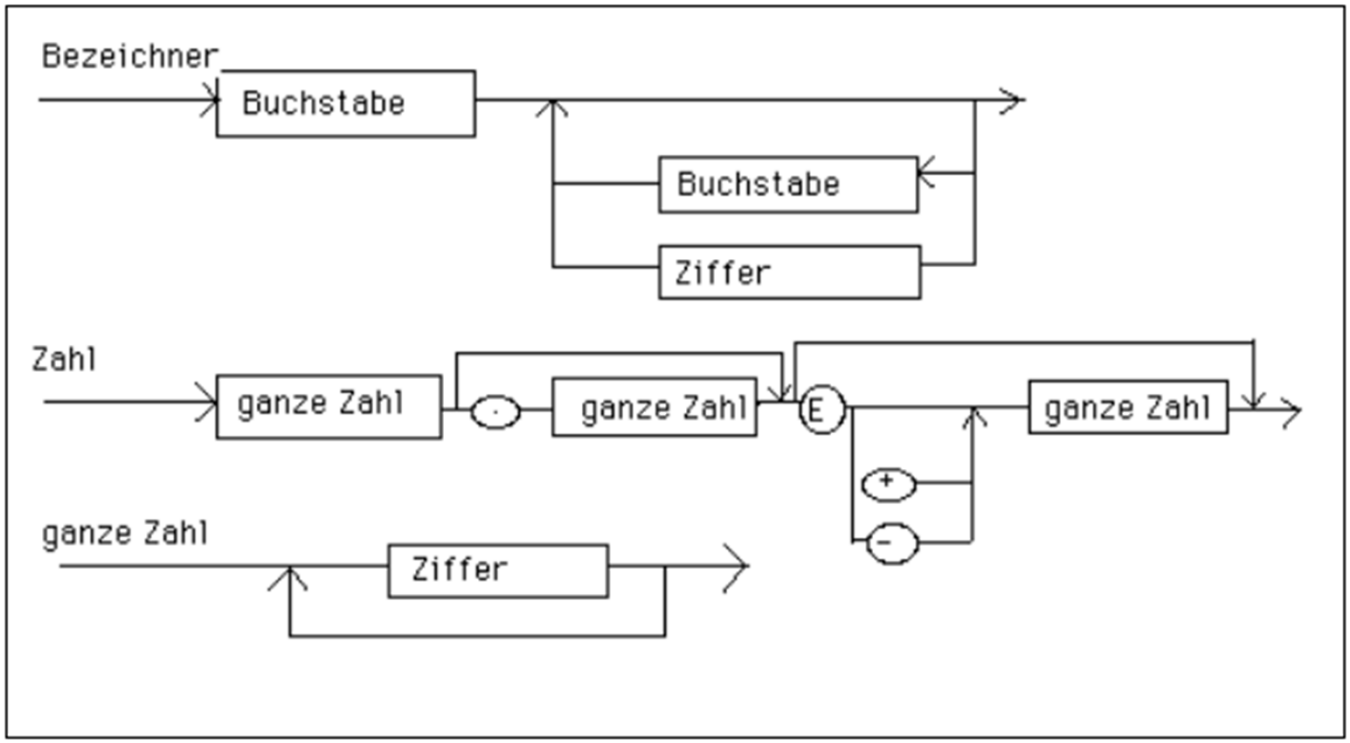
(6.8.5) *iteration-statement:*
while **(** *expression* **)** *statement*
do *statement* **while** **(** *expression* **)** **;**
for **(** *expression*$_{opt}$ **;** *expression*$_{opt}$ **;** *expression*$_{opt}$ **)** *statement*
for **(** *declaration* *expression*$_{opt}$ **;** *expression*$_{opt}$ **)** *statement*

(6.8.6) *jump-statement:*
goto *identifier* **;**
continue **;**
break **;**
return *expression*$_{opt}$ **;**

Syntaxdiagramme

EBNF kann als Syntaxdiagramm graphisch dargestellt werde.

Beispiel:



Sprachbeschreibungen allgemein

Lexikalische Ebene:

- Beschreibung mittels regulärer Ausdrücke.

Syntaktische Ebene:

- Beschreibung mit Grammatiken.
- Linguist Noam Chomsky definierte eine 4-stufige Hierarchie von Klassen von Grammatiken (1956,1959).
- Zur Beschreibung von Programmiersprachen sind vor allem kontextfreie Grammatiken von Bedeutung.

Semantische Ebene:

- Beschreibung in Prosa.
- axiomatische Semantik.
- wp-Semantik.

ISO C Standard – Beispiel

Syntax und Semantik bei Programmiersprachen.

Auszug aus ISO-C Standard.

Syntax und Semantik einer while-Schleife.

Syntax

iteration-statement :

while (*expression*) *statement*

Constraints

The controlling expression of an iteration statement shall have scalar type.

Semantics

- An iteration statement causes a statement called the loop body to be executed repeatedly until the controlling expression compares equal to 0.
- The evaluation of the controlling expression takes place before each execution of the loop body.

ISO C Standard – Beispiel

Nehmen sie bitte den ISO C Standard zur Hand (s. Moodle):

- Sehen sie sich das **while**-Statement von der vorhergehenden Folie im Standard genauer an.

- Gehen sie zu *Annex A: Language syntax summary* and betrachten die Produktionsregeln für *expression*:
  - Beachten sie die Hierarchie von Nonterminal-Symbolen, wodurch die Priorität von Operatoren ausgedrückt wird.
  - Beachten sie die Linksrekursionen, welche die Linksassoziatitvität der Operatoren widerspiegelt.

- Anmerkung am Rande:
  Die im Standard angegebene KFG ist mehrdeutig. Ursache sind die Produktionsregeln für das **if**-Statement, auch bekannt als *dangling-else* Problem (wir wollen aber in dieser VO nicht genauer darauf eingehen).

Reguläre Grammatiken

Einführende Anmerkungen zu reguläre Grammatiken:

- Reguläre Grammatiken haben gegenüber kontextfreien Grammatiken Einschränkungen.

- Die Einschränkungen bewirken, dass reguläre Grammatiken nicht so mächtig sind wie kontextfreie Grammatiken; "mächtig" bedeutet, dass es Sprachen gibt, die mit kontextfreien Grammatiken beschrieben werden können, nicht aber mit regulären Grammatiken.

- Mit regulären Grammatiken wird häufig die lexikalische Ebene von Programmiersprachen beschrieben; dafür sind sie ausreichend mächtig.

- Reguläre Grammatiken werden ebenfalls mit einem Quadrupel (N,Σ,P,S) definiert. Alles über kontextfreie Grammatiken Gesagte, gilt auch für reguläre Grammatiken, mit Ausnahme der **Produktionsregeln**.

Reguläre Grammatik - Formal

Definition:

Sei G = (N, Σ, P, S) eine KFG, in der die Regelmenge P wie folgt eingeschränkt ist:

1. Alle Regeln in P haben die Form (a) oder die Form (b):

   (a) $A \to xB$, mit $x \in \Sigma^*$ und $B \in N$.

   (b) $A \to x$, mit $x \in \Sigma^*$.

   In diesem Falle heißt G eine rechtslineare Grammatik.

2. Alle Regeln in P haben die Form (a) oder die Form (b):

   (a) $A \to Bx$, mit $x \in \Sigma^*$ und $B \in N$.

   (b) $A \to x$, mit $x \in \Sigma^*$.

   In diesem Falle heißt G eine linkslineare Grammatik.

3. Eine Grammatik heißt **regulär** oder **Chomsky-Typ 3 Grammatik**, wenn sie entweder rechtslinear oder linkslinear ist.

Reguläre Grammatiken: Beispiele

Die Grammatiken G_1, G_2, G_3 erzeugen die Menge der Binärworte über dem Alphabet {0,1}:

- G_1 : P={S $\rightarrow$ 0 | 1 | 0S | 1S} ist rechtslinear
- G_2 : P={S $\rightarrow$ 0 | 1 | S0 | S1} ist linkslinear
- G_3 : P={S $\rightarrow$ Z | ZS, Z $\rightarrow$ 0 | 1} ist **nicht** regulär, sondern kontextfrei

Es gibt auch Sprachen, für die es keine reguläre Grammatik gibt:

- Es gibt **keine** reguläre Grammatik, die die Sprache $L = \{a^n b^n \mid n \geq 1\}$ erzeugt (wird erzeugt von kontextfreier Grammatik mit Regeln P={S $\rightarrow$ aSb | ab}).

Reguläre Grammatik: Beispiel Identifier

Betrachten wir folgende Sprache gegeben durch

$$L = \{ w \mid w = bX, X \in \{b, z\}^* \}$$

wobei ´b´ für einen beliebigen Buchstaben und ´z´ für eine beliebige Ziffer steht, i.e. ein Identifier der mit einem Buchstaben beginnen muss gefolgt von einer beliebigen Folge von Buchstaben und Ziffern.

Die reguläre (rechtslineare) Grammatik G ist gegeben durch:

$G = (\{S, A\}, \{b, z\}, P, S)$ mit
$P = \{ S \rightarrow bA, A \rightarrow bA \mid zA \mid \varepsilon \}$

Das Wort **bzzb** läßt sich wie folgt in G ableiten:

$$S \Rightarrow bA \Rightarrow bzA \Rightarrow bzzA \Rightarrow bzzbA \Rightarrow bzzb$$

Ableitungen in einer rechtslinearen Grammatik

Sei G = (N, Σ, P, S) rechtslinear. Alle Ableitungen in G haben folgende Struktur (für linkslineare Grammatik analog):

- **Fall 1:** Es gibt eine Regel $S \to x$ mit $x \in \Sigma^*$. Dann ist $x \in L(G)$ und $S \Rightarrow x$ die zugehörige Ableitung.

- **Fall 2:** Es wird eine Folge von $n \geq 2$ Ableitungsschritten durchgeführt. Jede Satzform mit Ausnahme der letzten enthält **genau ein Nichtterminalsymbol als letztes Zeichen**, d.h.

$$S \Rightarrow x_0A_1 \Rightarrow x_0x_1A_2 \Rightarrow \ \ldots \Rightarrow x_0x_1 \ldots x_{n-1}A_n \Rightarrow x_0x_1 \ldots x_{n-1}x_n.$$

wobei

- $S \to x_0A_1 \in P$
- $A_i \to x_iA_{i+1} \in P$ **für alle** i, $1 \leq i \leq n$, **und** $A_n \to x_n \in P$
- $A_i \in N$, $1 \leq i \leq n$
- $x_i \in \Sigma^*$, $0 \leq i \leq n$

Reguläre Ausdrücke

Regulärer Ausdruck:
- eine weitere Möglichkeit Sprachen zu definieren
- gleichmächtig wie reguläre Grammatiken
- definieren reguläre Sprachen

Vorteil regulärer Ausdrücke:
- deklarative Beschreibungsweise, die leicht nachvollziehbar ist
- Eingabesprache von vielen Tools wie Editoren, awk, grep, Scanner Generatoren, etc.

Beispiele: intuitiv verständlich?
- $\mathbf{1}.\,ab^*, \mathbf{2}.\,(ab)^*, \mathbf{3}.\,ab^*|ba^*, \mathbf{4}.\,(F|f)red$
  (1. a gefolgt von b's, 2. beliebige Sequenz abab…, 3. a gefolgt von b's oder b gefolgt von a's, 4. Fred oder fred)

Operationen auf Sprachen

Zur Definition von regulären Ausdrücken benötigen wir 3 Operationen auf Sprachen: **Vereinigung, Konkatenation, Stern.**

1. **Vereinigung** zweier Sprachen $L_1 \cup L_2$ ist die Menge aller Worte in L_1 oder L_2.
   Beispiel: L_1={001,10,111}, L_2={ε,001}, $L_1 \cup L_2$={ε,10,001,111}

2. **Konkatenation** zweier Sprachen L_1 und L_2, geschrieben $L_1.L_2$ oder L_1L_2, ist wie folgt definiert:

   $$L_1L_2 := \{x_1x_2 \mid x_1 \in L_1 , x_2 \in L_2 \}$$

   Die Zeichenketten in L_1L_2 werden gebildet, indem man an eine aus L_1 gewählte Zeichenkette eine Zeichenkette aus L_2 anhängt und das für alle möglichen Kombinationen.

   Beispiel: L_1 = {10,1}, L_2 = {011,11,ε}, L_1L_2 = {10011, 1011, 10, 111, 1}

3. **Stern oder Hülle oder Kleene Hülle** einer Sprache L bezeichnet als L^*: Vereinigung über alle Konkatenationen

(1) $\boldsymbol{L^0 = \{\, \varepsilon \,\}}$ (unabhängig von L), $\boldsymbol{L^1 = L}$

(2) $\boldsymbol{L^n = L^{n-1}\, L}$ für alle $n \geq 2$ (n-fache Konkatenation von L)

(3) $\boldsymbol{L^* = \bigcup_{i=0}^{\infty} L^i}$, Stern, Hülle, Kleene Hülle

(4) $\boldsymbol{L^+ = \bigcup_{i=1}^{\infty} L^i}$, positive Hülle

Beispiel: L = {0,11}.

$L^0 = \{\varepsilon\}$, $L^1 = \{0,11\}$, $L^2 = LL = \{00,011,110,1111\}$

$L^* = \{\varepsilon, 0, 11, 00, 011, 110, 1111, \ldots\}$

Definition: Reguläre Ausdrücke

Rekursive Definition von regulären Ausdrücken über Alphabet Σ mit den entsprechenden repräsentierten Mengen:

(1) $\emptyset$ ist ein regulärer Ausdruck und bezeichnet die leere Sprache $\emptyset$, i.e.: $L(\emptyset)=\emptyset$.
(2) ε ist ein regulärer Ausdruck und bezeichnet die Sprache $\{\varepsilon\}$, i.e. $L(\varepsilon)=\{\varepsilon\}$.
(3) Für jedes $a \in \Sigma$ gilt: **a** ist ein regulärer Ausdruck und bezeichnet die Sprache $\{a\}$, i.e. $L(\mathbf{a})=\{a\}$.

(4) Seien α, β reguläre Ausdrücke mit den dazugehörenden Mengen $L(\alpha)$ und $L(\beta)$, dann gilt:

(a) $\alpha \mid \beta$ ist ein regulärer Ausdruck und bezeichnet $L(\alpha) \cup L(\beta)$ (Vereinigung)
(früher übliche Schreibweise: $\alpha + \beta$)
(b) $\alpha\,\beta$ ist ein regulärer Ausdruck und bezeichnet $L(\alpha)\,L(\beta)$ (Konkatenation)

(c) α^* ist ein regulärer Ausdruck und bezeichnet $(\,L(\alpha)\,)^*$ (Stern)

(d) (α) ist ein regulärer Ausdruck und bezeichnet $L(\alpha)$. (Klammern)

Reguläre Ausdrücke:

Priorität der Operatoren:

1. **Stern** *(höchste Priorität)*
2. **Konkatenation**
3. **Vereinigung**

Klammern können verwendet werden, um gewünschte Prioritäten festzulegen; redundante Klammern können gesetzt werden.

Beispiel: **01* | 1**

- Frage 1: (01)* oder 0(1*) ? Stern stärker $\Rightarrow$ 0(1*)
- Frage 2: 0 (1* | 1) oder (01*) | 1 ? Konkatenation stärker $\Rightarrow$ (01*)|1
- daher: steht für: (**0** (**1***)) | **1**
  i.e. (**0** gefolgt von beliebig vielen **1er)** oder (eine einzelne **1)**
- Anm: $\sum = \{a, b, c, d, e, f\}$ mit reg. Ausdr. $a\ b\ c\ d\ e \mid f$. Klammerung?

Das vereinfachende hochgestellte **+** ist erlaubt:

- Der Ausdruck α^+ steht für $\alpha\alpha^*$.

Beispiele für reguläre Ausdrücke und Sprachen:

$$\Sigma = \{0, 1\}$$

regulärer Ausdruck	bezeichnete Sprache
0	$\{0\}$
$(0 \mid 1)^*$	$\{0,1\}^*$ bzw. $\{ w \mid w \in \Sigma^* \}$
$0(0\mid1)^*1$	$\{0\}\ \{0,1\}^*\ \{1\}$ bzw. $\{ 0w1 \mid w \in \Sigma^* \}$
$(0\mid1)^*011$	$\{ w011 \mid w \in \Sigma^* \}$
0^*1^*	$\{ 0^i1^j \mid i, j \geq 0 \}$
00^*11^*	$\{ 0^i1^j \mid i, j \geq 1 \}$
0^+1^+	$\{ 0^i1^j \mid i, j \geq 1 \}$

Beispiel: $r = a^*(a|b)$

- Null oder mehrere **a** abgeschlossen mit **a** oder **b**
- $r = a^*(a|b) = a^*a|a^*b = a^+|a^*b$

Beispiel: $r = (a|b)^*(a|bb)$

- $(a|b)^*$ – beliebige Zeichenketten von a oder b inkl. ε
- $(a|bb)$ – entweder a oder bb
- L(r) ist daher die Menge aller Zeichenketten von {a,b} inkl. ε abgeschlossen mit a oder bb.

$L(r) = \{a, bb, aa, abb, ba, bbb, ...\}$

Beispiel: $r = (aa)^*(bb)^*b$

- $L(r) = \{ a^{2m}b^{2n}\, b \mid m \geq 0, n \geq 0 \} =$
  $= \{ a^{2m}b^{2n+1} \mid m \geq 0, n \geq 0 \}$

Beispiel:

Gegeben sei der reguläre Ausdruck 0(10)*. Die dadurch definierte Sprache wird von folgender rechtslinearen Grammatik erzeugt:

G = ({S,A}, {0,1}, P, S) mit

P = { S → 0A , A → 10A | ε }

bzw. von folgender linkslinearen Grammatik:

G = ({S}, {0,1}, P, S) mit

P = { S → S10 | 0 }

Regulärer Ausdruck für Suche

Beispiel Unix-Command grep / egrep / fgrep:

- erlaubt Dateien nach Strings zu durchsuchen, die mittels erweiterter regulärer Ausdrücke angegeben werden.

Beispiele zur Veranschaulichung (kein Prüfungsstoff):

- **grep 'abc' file :**
  - Zeigt alle Zeilen, die "abc" enthalten
- **grep 'a.c' file :**
  - alle Zeilen, die "axc" enthalten, wobei x ein beliebiges Zeichen ist
- **grep –n 'my_function *(' my_code.c :**
  - alle Zeilen, wo **my_function** aufgerufen wird (oder deklariert wird)
- **grep 'a\[[^]]*\]=' file :**
  - alle Vorkommen der Form "a[...]=", wobei ... eine beliebig lange Zeichenkette ist, die kein] enthält (also Zuweisungen an ein Element des Arrays a)
- **grep 'a\[[^]]*\] *=' file :**
  - wie vorher, mit beliebig vielen Spaces zwischen "]="