3. Zeichenreihen und formale Sprachen

3.1. Definition formaler Sprachen

Def. 3.1:

Eine endliche Menge von Symbolen heißt **Zeichenvorrat** Σ; die Elemente des ZV heißen **Zeichen**: $\alpha \in \Sigma$; ein ZV Σ mit einer darauf erklärten vollständigen Ordnung < heißt **Alphabet**.

Ordnungsrelation auf Σ hat folg. Eig.:

1. $\alpha \not< \alpha$ (Irreflexivität)
2. $\alpha_1 \not< \alpha_2 \rightarrow \alpha_2 < \alpha_1$ (Nichtkommutativität)
3. $\alpha_1 < \alpha_2$ und $\alpha_2 < \alpha_3 \rightarrow \alpha_1 < \alpha_3$ (Transitivität)

Def. 3.2:

Eine Zeichenreihe (-kette) über Σ wird definiert durch folg. Eig.:

(1) Die leere ZR ε ist eine ZR über Σ.
(2) Wenn α eine ZR über Σ ist und $x \in \Sigma$, so ist $x\alpha$ eine ZR über Σ.
(3) β ist genau dann ZR über Σ, wenn sie laut (1), (2) gebildet wurde.

Bem.: ZR oft als ***Wort*** bezeichnet.

Def. 3.3:

2 ZR $\alpha = \alpha_1 \alpha_2 \ldots \alpha_n$ und $\beta = \beta_1 \beta_2 \ldots \beta_m$ heißen **gleich**, d.h. $\alpha = \beta$, g. d., w.

(1) $n = m$ und
(2) $\alpha_i = \beta_i$ für $i = 1(1)n$.

Def. 3.4:

Als **Konkatenation** $\alpha \circ \beta$ zweier ZR $\alpha, \beta \in \Sigma^*$ mit $\alpha = \alpha_1 \alpha_2 \ldots \alpha_n$ und $\beta = \beta_1 \beta_2 \ldots \beta_m$ bezeichnet man $\alpha \circ \beta = \alpha_1 \alpha_2 \ldots \alpha_n \beta_1 \beta_2 \ldots \beta_m$.

Def. 3.5:

Sei β eine ZR. Die ZR α heißt **Teil–ZR** von β g. d., w.
$\exists$ ZR w_1, w_2 mit $\beta = w_1 \alpha w_2$.
Für $w_1 = \varepsilon$ heißt α **Anfangs–ZR;** für $w_2 = \varepsilon$ heißt α **End–ZR**.

Def. 3.6:

Sei $\alpha \in \sum^*$, und $l(\alpha)$ bezeichne die Anzahl Elemente von α.
Dann heißt $l(\alpha)$ **Länge** der ZR α.

Per Def.: $l(\varepsilon) = 0$.
Klar: $\alpha, \beta \in \sum^* \Rightarrow l(\alpha \circ \beta) = l(\alpha) + l(\beta)$.

Def. 3.7:

Sei $\sum$ ein Alphabet und $\alpha \in \sum^*$; $n > 0$ sei eine natürliche Zahl.
Als n-te Potenz von α bezeichnet man die ZR

$$\alpha^n = \alpha\, \alpha^{n-1} = \alpha^{n-1}\, \alpha = \alpha \ldots \alpha.$$

Per Def.: $\alpha^0 = \varepsilon$.

Def. 3.8:

Jede Teilmenge $L \subset \sum^*$ heißt **formale Sprache**.

3.2. Syntax, Semantik, Darstellungsmöglichkeiten der Syntax

Def. 3.9:

Syntax einer Sprache L – Menge der Regeln, die festlegen, welche Elemente aus Σ^* zu L gehören.

Def. 3.10:

Semantik einer Sprache legt die Bedeutung der Zeichen(-reihen) fest.

Syntax–Festlegung weitgehend **formal** aus 2 Gründen:

(1) Struktur u. Bedeutung von Pr.-texten ***automatisch*** erkennbar und umsetzbar (Compilation).

(2) Progr.-rer kann entscheiden, ob seine Entwicklung in gültigen Sprachnotation vorliegt.

- 4 Prinzipien der Festlegung von L bzw. PS:
  1° Prinzip der Aufzählung
  2° Prinzip der Erkennung
  3° Prinzip der Berechnung
  4° Prinzip der Generierung

Zu 1°: Nur, wenn nicht zu viele Wörter !!!

Zu 2°: Erkennungsproblem: Sei $\alpha \in \Sigma^*$, so ist $\alpha \in$ L zu prüfen.
➔ Prüfung mittels Algorithmus

Zu 3°: Menge von Ausdrücken, die Wörter aus L erzeugen.
➔ Klasse der **regulären Sprachen**

Zu 4°: Verallgemeinerung von 3°, indem mittels Regeln Wörter aus L definiert, generiert werden;
Hilfsmittel dazu ➔ **generative Grammatik**

Def. 3.11:

G = (∑, N, P, s) heißt **generative Grammatik**, wenn gilt:

∑ – Alphabet der ***Terminalsymbole***;

N – Alphabet der ***Nicht–Terminalsymbole***;

P – Menge von ***Produktionen*** (synt. Regeln);

s – ein ***Startsymbol*** (Startpunkt) aus N.

Beispiel: Definition einer Grammatik, die es ermöglicht, Sätze und Satzfolgen einer Sprache „Berichte aus dem öffentlichen Leben“ zu erzeugen.

s : < Satzfolge>

Regeln mit Nicht–Terminalsymbolen:

Folg. Vereinbarungen ➔
Ableitungszeichen ::= (Definitionszeichen)
Nichttermin. Wortsymbole <…> (Winkelklammern)
Mehrere rechte Seiten, Trennung mit | (Alternativzeichen)

{ als **Backus–Naur–Form** (BNF–Notation) bezeichnet }

Nichtterminale Regeln:

<Satzfolge> ::= <Satz> | <Satz> <Satzfolge>

<Satz> ::= <Subjektteil> <Prädikatteil>

<Subjektteil> ::= <Substantiv> | <Substantiv mit Attribut>

<Substantiv mit Attribut> ::= <Substantiv> <substantives Attr. mit Genitiv>

<Prädikatteil> ::= <Prädikat> <Objekt>

<Prädikat> ::= <Verb>

<Objekt> ::= <Artikel> <zus.-ges. Subst.> | <Artikel> <Adjektiv> <zus.-ges. Subst.>

<zus.-ges. Subst.> ::= <1. Subst.-hälfte> <2. Subst.-hälfte>

Terminale Regeln:

<Substantiv> ::= Der_Vertreter | Der_Abgeordnete | Der_Vorsitzende

<substantives Attr. mit Genitiv> ::= der_Regierung | der_Opposition | der_Gewerkschaft

<Verb> ::= beklagte_ | verurteilte_ | lobte_

<Artikel> ::= die_

<Adjektiv> ::= permanente_ | übertriebene_ | vernachlässigbare_

<1. Subst.-hälfte> ::= Führungs | Aktions | Wachstums

<2. Subst.-hälfte> ::= struktur. | potenz. | phase.

Erklärung zu P:

$P = \{ (w_1, w_2) : w_1, w_2 \in (\sum \cup N)^* \}$

diese Produktionen definieren sogen. Ableitungen ➔

a) direkte Ableitung:
sei $\alpha \in \sum \cup N$; wenn $\alpha = x\, w_1\, y$, so gilt β als direkte Ableitung aus α, wenn eine Regel (w_1, w_2) ex. mit $\beta = x\, w_2\, y$
schreiben $\alpha \Rightarrow \beta$

b) allg. Ableitung:
aus α wird β abgeleitet, wenn es eine Folge von Wörtern gibt, die direkt voneinander abgeleitet von α zu β führen:
$\alpha \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = \beta$ (schreiben $\alpha \rightarrow \beta$).

- Eine durch G definierte formale Sprache ist definiert als
  $L(G) = \{ x \in \sum^* : s \rightarrow x \}$; dabei ist s das Startsymbol aus G.

  In Abhängigkeit von Regeltypen versch. Grammatiken bzw. Grammatiktypen (CHOMSKY–Hierarchie).

  Bem.: Eine durch Grammatik erzeugte Sprache heißt
  Regel–Sprache.

- Bem.:
  Für Def. höherer PS meist spez. Klasse generat. Grammatiken – die **kontextfreien** G. (Typ–2–Grammatik nach Chomsky)

  Entspr. Sprache – CF–Sprache (context free)

  Was heißt **Kontext**?

  Neben Syntax- und Semantikdef. noch wichtig die Def. der
  Kontextbedingungen.

  Bsp.:
  Sei folg. synt. Regel festgelegt
  <ergibtanw> ::= <variable> := <ausdruck>

  Es könnte die Kontextbedingung gelten, daß
  Typ von <variable> muß gleich Typ von <ausdruck> sein.

  Anderes Bsp.:
  Jede im Programm benutzte Variable muß vereinbart sein.

- **<u>Darstellungsmöglichkeiten:</u>**

  BNF, Syntaxdiagramme, EBNF;

  <u>a) BNF:</u>
  - es werden die syntaktische Regeln dargestellt;

  - in den Regeln wenige Metasymbole:
    ::= trennt linke und rechte Seite einer synt. Regel
    < > zum Einschluß von Nicht–Terminalsymbolen
    | zur Darstellung von Alternativen

  - Terminalsymbole sind entweder einzelne Zeichen oder groß-geschriebene oder unterstrichene oder fettgeschriebene ZR

■ Bsp.:

<programm>	::=	<kopf> <block> <ende>
<block>	::=	<vereinbar_teil> <anweis_teil>
<ende>	::=	.
<anweis_teil>	::=	BEGIN <anweisungen> END
<anweisungen>	::=	<anweisung> | <anweisungen>; <anweisung>
<anweisung>	::=	<ergibtanw> | <verbundanw> | <leeranw> | <sprunganw>
<ergibtanw>	::=	<variable> := <ausdruck> | <fktname> := <ausdruck>

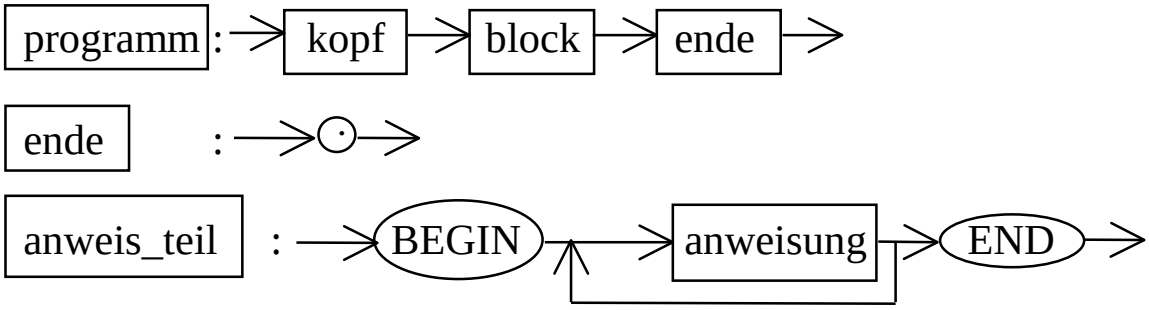
<u>b) Syntaxdiagramme:</u>

- Syntaxdiagramm – knotenmarkierter gerichteter Graph

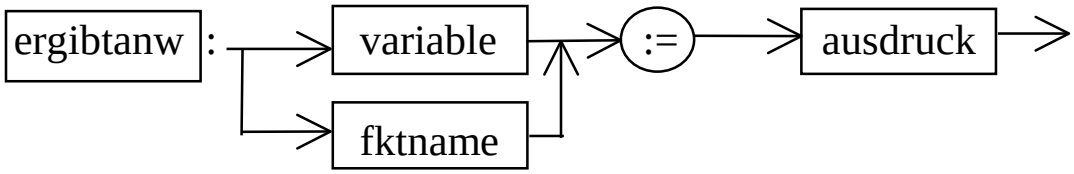
  – **<u>1</u>** Bezeichnung und **<u>2</u>** ausgezeichnete Knoten: einen Eingangs- und einen Ausgangsknoten;

- 2 Arten von Knoten ➔
  - Rechtecke mit Nicht–Terminalsymbolen als Markierung;
  - Ovale / Kreise mit Terminalsymbolen als Markierung.

- **<u>Jeder Weg</u>** – eine zulässige syntaktische Struktur.



usw.



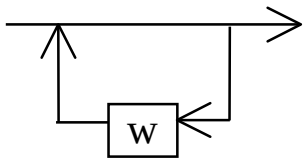
<u>c) EBNF:</u>

- alle Festlegungen der BNF mit 2 Ausnahmen:

  (1) Zeichen ::= ersetzt durch Zeichen =
  (2) Terminalsymbole durch Zeichen " eingeschlossen

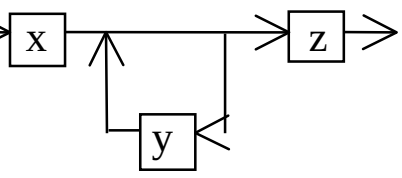
- zusätzlich hinzu:
  1) ***Wiederholungsklammer*** { } zur Kennzeichnung der Wdh.; auch 0-malige;

     {w} bedeutet: ε, w, w w, w w w, …

     {w}: 

     {Nicht–Terminalsymbol}

     x {y} z bedeutet: x z, x y z, x y y z, …
     Konkatenation von x, {y} und z;

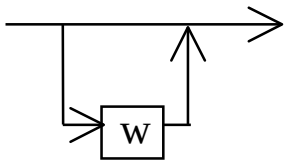
     x {y} z: 

     {Nicht–Terminalsymbole}

2) ***Optionsklammer*** [ ] zur Kennzeichnung von optional (wahlweise) angebbaren syntaktischen Strukturen;

[w] bedeutet: ε, w ! Max. 1 Wiederholung !!!

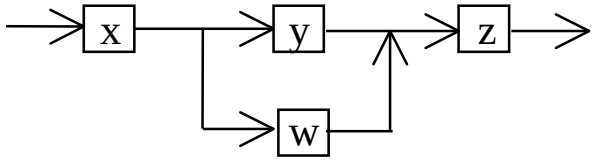
[w]:



3) ***Gruppenklammer*** ( ) zur Bildung von Teilausdrücken auf rechten Seiten von synt. Regeln;

x (y|w) z bedeutet: x y z | x w z

x (y|w) z:



Bem.:

Diese Metasymbole oft zum Alphabet.
Darum Terminalsymbole in Anführungsstrichen.
Gehören nicht zum Symbol bzw. Zeichen.

3.3. Spezielle Probleme der Arbeit mit Zeichenreihen

- ZR als Datenobjekt: (name, typ, wert)

  ➔ neu einzuführende Operation: Test auf Gleichheit
  oder Ungleichheit

  ➔ **lexikografische Ordnung**

Def. 3.12:

Seien α, β ZR über $\sum$ mit $l(\alpha) = a$ und $l(\beta) = b$. Dann gilt
$\alpha <\cdot\cdot \beta$ g.d., w. gilt
Fall 1 ($a = b$): $\exists$ i mit $\alpha_i <\cdot \beta_i$ und für alle $j < i$ gilt $\alpha_j = \beta_j$.
Fall 2 ($a < b$): $a < b$ und für $\forall\ j \leq a$ gilt $\alpha_j = \beta_j$.
{Wort α ist Teil- oder Anfangswort von β.}

HA – Beweis: $<\cdot\cdot$ ist eine Ordnungsrelation.

- **Kodierung und Dekodierung:**

geg.: Alphabete A und B sowie injektive Abb. $c: A \rightarrow B$
$\alpha = a_1 a_2 \ldots a_n$ mit $a_n \in A$ bzw. $\alpha \in A^*$.
Per Def. sei nun
(*) $c(\alpha) = c(a_1)\, c(a_2) \ldots c(a_n) \Rightarrow c(\alpha) \in B^*$.

Def. 3.13:

Eine injektive Abb. $c: A \rightarrow B$ mit der Eigenschaft (*) heißt ***Kodierung***, die Abb. c^{-1} ***Dekodierung***.

Def. 3.13':

Seien A und B endliche Alphabete. Jede injektive Abbildung
$c: A \rightarrow B^* \setminus \{\varepsilon\}$
heißt Kodierung (Kode). Die Abb. c^{-1} heißt Dekodierung.
Die Wörter c(a) für $a \in A$ heißen ***Kodewörter***.
Gilt $B = \{0, 1\}$, so heißt c ein ***binärer Kode***.

	Variante 1	Variante 2	Variante 3 …
0 0 0 0	0	0	–
0 0 0 L	1	1	–
0 0 L 0	2	2	–
0 0 L L	3	3	0
0 L 0 0	4	4	1
0 L 0 L	5	–	2
0 L L 0	6	–	3
0 L L L	7	–	4
L 0 0 0	8	–	5
L 0 0 L	9	–	6
L 0 L 0	–	–	7
L 0 L L	–	5	8
L L 0 0	–	6	9
L L 0 L	–	7	–
L L L 0	–	8	–
L L L L	–	9	–

BCD–K. Aiken–K. 3–Exzess–K. (Stibitz–Kode)

gebräuchlicher Kode:

Binärwörter der Länge 8 ➔ 256 Zeichen abbildbar

Bsp.: ASCII – American Standard Code for Information Interchange,

EBCDIC – Extended Binary–Coded Decimal Interchange Code

■ Kryptografie:

Def. 3.14:

Als ***Hamming–Abstand*** h(x, y) zweier Binärwörter x, y ∈ {0, 1}* bezeichnet man die Anzahl Stellen, an denen sich x und y unterscheiden.

Def. 3.15:

Der ***Hamming–Abstand*** H(c) eines n–stelligen Binärkodes c ist der minimale H.–Abstand in der Menge der Kodewörter, d.h.,

$$H(c) = \min_{x, y \in \{\text{Kodewörter}\}} h(x, y).$$

Satz:

a) Wenn der H.–Abstand eines Kodes größer als 1 ist, dann ist eine einfache Störung erkennbar.
b) Wenn der H.–Abstand eines Kodes größer als 2 ist, dann ist eine einfache Störung korrigierbar.

{Beweis erfolgt vom Gegenteil}

Gray–Kode: z.B. für Kodierung über 3–stellige Binärwörter

0	000	4	110
1	001	5	111
2	011	6	101
3	010	7	100

■ Problem der Konvertierung:

geg.: ZR über $\sum_B = \{0, 1, \ldots, B–1\}$ für ein $B \in N$ mit $B > 1$, wobei ein Positionssystem zugrunde liegt.

ges.: ***Konvertierung*** natürlicher Zahlen in eine ZR über $\sum_B$ und umgekehrt.

Darstellungssatz:

Sei $B \in N$ mit $B > 1$. Jede natürliche Zahl $x > 0$ ist eindeutig in folgender Form darstellbar:

$$x = \sum_{i=1}^{n} z_i \cdot B^{n-i} \quad \text{mit} \quad n \geq 1,\ z_i \in N_0,\ 0 \leq z_i < B,\ i = 1(1)n,\ z_1 \geq 1.$$

➔ Dualsystem „nicht schlecht“ bezgl. Aufwand; sehr einfache techn. Realisierung;

Algorithmen zur Konvertierung:

1° $x_B \rightarrow x_{10}$:

geg.: $x_B = z_1 z_2 \ldots z_n$ mit $z_i \in \sum_B$ für i = 1(1)n;
ges.: $x_{10} = \underline{z}_1 \ldots \underline{z}_m$ mit $\underline{z}_k \in \sum_{10}$ für k = 1(1)m;

∃ folg. Algorithmus:

1. $b_1 \leftarrow z_1$. {Anfangswert; b_1 entspricht dem dezimalem Gegenwert von z_1}
2. Berechne für k = 2 …n: $b_k \leftarrow z_k + B \cdot b_{k-1}$.
3. Als Lösung x_{10} gib b_n zurück.

2° $x_{10} \rightarrow x_2$:

geg.: x_{10} als ZR aus $\sum_{10}$*
ges.: x_2 als zugehörige Dualzahl

Prinzip:

x_{10} ➔ $x_2 = z_1 \cdot 2^{n-1} + z_2 \cdot 2^{n-2} + ... + z_{n-1} \cdot 2^1 + z_n \cdot 2^0$
mit $z_i \in \{0, 1\} = \sum_2$
Ist x_{10} gerade ➔ $z_n = 0$, andernfalls $z_n = 1$;
hieraus $(x_2 - z_n)$ und durch 2 dividiert ergibt z_{n-1}; usw.;
STOP bei „linke Seite 0 ∨ 1“.

⇨ folg. Algorithmus:

1. $u_0 \leftarrow x_{10}$; k ← 0. {Anfangswerte}
2. Wenn $u_k = 0$, dann k ← k–1 und gib Lösung x_2 als $a_k\, a_{k-1} \ldots a_0$ zurück. STOP.
3. Wenn u_k gerade, dann $a_k \leftarrow 0$, sonst $a_k \leftarrow 1$.
4. $u_{k+1} \leftarrow (u_k - a_k) / 2$.
5. k ← k+1. Weiter bei 2.