# 1 Formale Sprachen

Vorbetrachtungen:

1. Zeichenvorrat = endliche Menge von eindeutig unterscheidbaren Zeichen oder Symbolen. Bsp.:

$$T = \{0, 1\}, \qquad T = \{a, b, c\}, \qquad T = \{Sommer, Winter\}$$

2. Alphabet: Zeichenvorrat mit einer Ordnungsrelation.

$$A = \{T, <\}$$

   wobei $x < y$ heißt, dass $x$ vor $y$ kommt.

Im folgenden keine Unterscheidung zwischen Zeichenvorrat und Alphabet vorgenommen.

**Definition 1 (Konkatenation)** *Aneinanderreihung von Zeichen eines Alphabetes. Operationsymbol* $\circ$.

Beispiel: $T = \{a, b, c\}$. Konkatenationen sind $a \circ b$ oder $b \circ a \circ c$ usw. Das Konkatenationssymbol meist weggelassen, d.h. $bac$ statt $b \circ a \circ c$.

**Definition 2 (Wort oder Satz)** *Jede endliche Zeichenfolge über einem Alphabet* $T$ *heißt Wort oder auch Satz über* $T$.

**Definition 3 (Leerwort)** *Das Wort, das kein einziges Zeichen enthält. Symbol* $\varepsilon$. *Sei* $w$ *ein Wort über* $T$. *Es gilt*

$$w \circ \varepsilon = \varepsilon \circ w = w$$

Wichtige Bezeichnungen:

1. Die Menge aller Wörter mit $n \geq 1$ Zeichen über $T$ wird als $T^+$ bezeichnet. Bsp. Sei $T = \{0, 1\}$ dann ist $T^+ = \{0, 1, 00, 01, 10, 11, 000, \dots\}$

2. Die Menge aller Wörter mit $n \geq 0$ Zeichen über $T$ wird als $T^*$ bezeichnet. Bsp. Sei $T = \{0, 1\}$ dann ist $T^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$. Es ist also

$$T^* = T^+ \cup \{\varepsilon\}$$

**Definition 4 (Sprache)** *Eine Sprache $L$ über einem Alphabet $T$ ist eine Teilmenge von $T^*$ : $L \subseteq T^*$. die Elemente der Sprache heißen Sätze (bzw Wörter) der Sprache.*

Spezialfälle:

1. $L = \emptyset$ (die leere Sprache)
2. $L = \{\varepsilon\}$ (die Sprache, die nur aus dem leeren Satz besteht)
3. $L = T^*$ (die Allsprache)

Die Menge der zulässigen oder sog. wohlgeformten Sätze einer Sprache kann defiiert werden durch

1. Aufzählen (endliche Sprache)
2. Angabe einer Bildungsvorschrift = Grammatik (bei unendlichen Sprachen)
3. Eigenschaftspezifikation: $L = \{w : w \in T^* \wedge P(w)\}$ wobei $P(w)$ eine Eigenschaft ausdrückt, die für alle wohlgeformten Sätze erfüllt sein muss.

Beispiele einer (endlichen) Sprache über $T = \{0, 1\}$:

- $L = \{0, 1\}$
- $L = \{101, 1001, 10001\}$
- usw.

## 1.1 Grammatik

Wir nennen $V = T \cup N$ das Vokabular (der Sprache). Dieses enthält also sowohl die Terminal- wie auch die Nichtterminalsymbole.

**Definition 5 (Grammatik)** *Eine Grammatik $G$ ist ein Quadrupel $G = (T, N, P, S)$ wobei*

$T =$ Alphabet der Terminalsymbole (aus einem endlichen Zeichenvorrat). Bsp. $T = \{a, b, c\}$

$N =$ Nichtterminalsymbole (syntaktische Kategorien), auch Variable genannt. Schreibweise: Einschluss in spitze Klammern, Bsp. <A>, <Satz>, usw.

$P =$ Menge von Ersetzungsregeln (Produktionsregeln) der Form $u \to v$ mit $u \in V^*NV^*$ und $v \in (T \cup N)$.

$S =$ Startsymbol (Startsymbol) mit $S \in N$.

Formal ist eine Produktionsregel eine Relation $u \Rightarrow v$ auf $V^* \times V^*$. Jede Regel besteht aus dem

1. Kopf, der der syntaktischen Kategorie der linken Seite entspricht, d.h. eine Folge von Symbolen $\in V$
2. Metasymbol $\to$
3. Rumpf, bestehend aus 0 oder mehr syntaktischen Kategorien und/oder Terminalsymbolen auf der rechten Seite.

Beispiel Grammatik: Sei $T = \{a, b\}$, $N = \{< W >, < A >, < B >\}$. Startsymbol $S =< W >$. Produktionsregeln:

$$\begin{aligned} P &= \{< W > \to < A > c, \quad < A > \to a < B >, && (1) \\ &< A > \to < B >< B > b, \quad < B > \to b, \quad < B > \to ab\} && (2) \end{aligned}$$

Andere Schreibweise (Zusammenfassung mehrerer Regeln mit dem Symbol | ("oder")):

$$P = \{< W > \to < A > c, \quad < A > \to a < B > \mid < B >< B > b, \quad < B > \to b|ab\} \tag{3}$$

Aus dem Startsymbol $< W >$ kann man durch Ableitung Zeichenketten erzeugen.

**Naiv**: Ableitung ist die Erzeugung neuer Zeichenketten durch Anwendung der Ersetzungsregeln. Ziel: Beginnend mit dem Startsymbol durch wiederholte Anwendung der Ersetzungsregeln Zeichenketten erzeugen, die nur aus Terminalsymbolen bestehen. Die dabei aus dem Startsymbol entstehenden Zwischenketten $u \in V^*$ heißen **Satzformen**. Für diese gilt also

$$S \Rightarrow^* u$$

wobei $\Rightarrow^*$für die $n-$ fache Anwendung ($n \geq 0$) der Ersetzungsregeln steht. Satzformen sind also allgemein Ketten, die sowohl Terminal- als auch Nichtterminalsymbole enthalten können. Eine Satzform

$$u \in T^*$$

heißt Satz (oder Wort).

**Definition 6 (Direkte Ableitung und Ableitung)** *Eine Kette $v$ ist die direkte Ableitung einer Kette $u$ wenn $u = xyz$ und $v = xy'z$ wobei $x, y, z$ irgendwelche Ketten aus $(T \cup N)^*$ sind und $y \to y' \in P(G)$ gilt, d.h. die Regel zur Grammatik $G$ gehört. Schreibweise: $u \Rightarrow v$. Ableitung ist eine Folge von direkten Ableitungen.*

Wir schreiben $u \Rightarrow^+ v$ für $u \Rightarrow u_1 \Rightarrow u_2 ... \Rightarrow v$ ($u$ kann in $v$ abgeleitet werden) und $u \Rightarrow^* v$ ($u$ ist entweder $v$ oder kann in $v$ abgeleitet werden).

**Definition 7 (Formale Sprache)** *Sei $G$ eine Grammatik. Dann ist die Sprache, die durch diese Grammatik generiert wird durch die Menge*

$$L(G) = \{w \in T^* | < S > \underset{G}{\Rightarrow}^* w\}$$

*gegeben, wobei $\Rightarrow^*$ für die Folge von $n$ Ableitungen ($n$-fach mit $n \geq 0$) steht.*

Bsp.: Für die Regeln in (3) ergeben sich etwa folgende Ableitungen (spitze Klammern weggelassen, es steht $\Rightarrow^+$ für die mindestens einfache Ableitung):

- $S \Rightarrow^+ abc$: $S \Rightarrow Ac \Rightarrow aBc \Rightarrow abc$
- $S \Rightarrow^+ ababb$: $S \Rightarrow Ac \Rightarrow BBc \Rightarrow abBbc \Rightarrow ababb$
- Insgesamt definiert unsere Beispielgrammatik die Sprache aus den 6 Worten: $L(G) = \{abc, aabc, bbbc, babbc, abbbc, ababbc\}$

**Definition 8 (Rekursive Grammatiken)** *Enthalten Ableitungen der Form $A \Rightarrow^+ u_1 A u_2$ mit $u_1, u_2 \in (T \cup N)^*$. Man unterscheidet linksrekursive Regeln ($A \to u|Av$), rechtsrekursive Regeln ($A \to u|vA$) und zentralrekursive Regeln ($A \to u|uAv$).*

## 1.2 Die Chomsky Hierarchie

Nach N. Chomsky (1959) können die Grammatiken (und damit die von ihnen erzeugten Sprachen) in eine vierstufige Hierarchie geordnet werden. Es sei wieder das Vokabular durch $V = T \cup N$ gegeben. Eine Grammatik $G = (T, N, P, S)$ heißt vom

**Typ0** oder *unbeschränkt*, falls keinerlei Einschränkungen an die Regeln gemacht werden, d.h. $xyz \to u$ mit $x, z, u \in V^*$ und $y \in N$.

**Typ1** oder *kontextsensitiv*, wenn in jeder Regel die linke Seite nicht länger (d.h. aus mehr Zeichen besteht) als die rechte., d.h. $u \rightarrow v$ mit $u, v \in V$ und $|u| \leq |v|$. Anders formuliert: Bei Vorhandensein eines Kontextes darf ein Nichterminalsymbol auf der linken Seite einer Ableitung nur unter Beibehaltung dieses Kontextes ersetzt werden. Formal: Eine Regel $y \rightarrow y'$ $(y' \neq \varepsilon)$ darf in einem Kontext $x \ldots z$ nur so verwendet werden: $xyz \rightarrow xy'z$.

**Typ2** oder kontextfrei, wenn für Für die Regeln aus $P$ jetzt noch restriktiver gefordert wird, dass auf der linken Seite genau ein Nichtterminal stehen muss:

$$\forall\, u \rightarrow v \in P \quad \text{gilt: } u \in N, \quad v \in V^*$$

**Typ3** oder *regulär*, wenn einchränkend zu Typ 2 auf der rechten Seite eine beliebige Kette von Terminalsymbolen gefolgt (bei rechtslinearen Grammatiken) von höchstens einem Nichtterminalsymbol, d..h. die Regeln haben die Form

$$\forall\, u \rightarrow v \in P \quad \text{gilt: } u \in N, \quad v = w | w < A >$$

wobei $w \in T^*$ und $< A > \in N$. Bei linkslinearen Grammatiken steht das NTS auf der linken Seite, d..h. die Regeln haben die Form

$$\forall\, u \rightarrow v \in P \quad \text{gilt: } u \in N, \quad v = w | < A > w$$

Eine Sprache $L\,(G)$ heißt vom Chomsky Typ $i$ falls die Grammatik $G$ vom Typ $i$ ist. Alle gängigen Programmiersprachen sind kontextfreie Sprachen.

**Theorem 9 (Chomsky Hierarchie)** *Wenn $L_i$ die Menge aller Sprachen bezeichnet, die von Grammatiken des Typs i erzeugt werden, dann gilt $L_0 \supset L_1 \supset L_2 \supset L_3$.*

- Die Ableitungen können für kontextfreie Sprachen durch **Syntax- oder Ableitungsbäume** oder durch die **Syntaxdiagramme in der Backus-Naur-Form** anschaulich gemacht werden. Syntaxbäume: partiell geordnet, d.h. Ordnung der Kindknoten eines jeden Knotens von li. nach rechts festgelegt. Blätter tragen die Terminalsymbole, die inneren Knoten sind mit Nichtterminalsymbolen belegt. Wurzelknoten trägt das Startsymbol. Hat ein Knoten $< A >$ Nachfolger $x_1 x_2 ... x_n$ ,dann gibt es eine Regel $< A > \rightarrow x_1 x_2 ... x_n$.

**Beispiel**: Grammatik für **Bezeichner** (Name einer Variablen in einer Programmiersprache):

Grammatik $G = \{T, N, P, S\}$ mit

$$\begin{aligned} T &= \{0 \ldots 9, a \ldots z, A \ldots Z, \_\} \\ N &= \{< B >, < D >, < Z >, < N >\} \end{aligned}$$

(Mnemonische Notation: $B$=Buchstabe, $D$ =Digit, $Z$=Zeichenkette, $N$= Name)

Produktionsregeln (Ersetzungsregeln):

$$\begin{aligned} < \ & N > \rightarrow < B > \mid < B >< Z > \\ < \ & Z > \rightarrow < B > \mid < B >< Z > \mid < D > \mid < D >< Z > \\ < \ & B > \rightarrow a|b| \ldots |A| \ldots |Z|\_ \\ < \ & D > \rightarrow 0|1| \ldots |9 \end{aligned}$$

## 1.3 Backus-Naur-Form

Eine etwas andere Notation zur Darstellung der Regeln einer Grammatik.

Metasymbole sind ::= anstelle von $\rightarrow$und | für die Alternative wie gehabt.

| **Sprachelement** | **Bedeutung** |
|---|---|
| $< A >$ | Nichtterminales Symbol |
| $< A >::= \ldots$ | Produktionsregel |
| $A_1 A_2 A_3 \ldots A_n$ | Sequenz |
| $A_1 \mid A_2 \mid A_3 \mid \ldots \mid A_n$ | Alternative |

Erweiterte BNF (EBNF): Metasymbole wie in BNF dazu $[\ldots]$ für die Option und $\{\ldots\}$ für die Wiederholung. Wird unterschiedlich gebraucht. Hier:

| **Sprachelement** | **Bedeutung** |
|---|---|
| $[A]$ | $\varepsilon \mid A$ (Option, d.h. $A$ erscheint keinmal oder einmal) |
| $\{A\}$ | $A \mid AA \mid AAA \mid \ldots$ (Wiederholung, $A$ erscheint mindestens einmal) |
| $[\{A\}]$ | $\varepsilon \mid A \mid AA \mid AAA \mid \ldots$ ($A$ erscheint $n$ mal mit $n \geq 0$) |
| $(A \mid B \mid ...)$ | $A \mid B \mid ...$ ein Element der Klammer steht |

Damit schreibt sich die Grammatik zur Erzeugung eines Namens nach obigem Beispiel als

$$< N >::=< B > [\{(< B > \mid < Z >)\}]$$

Runde Klammern dienen der zusätzlichen Strukturierung der Ausdrücke, so bedeutet der Ausdruck

$$\{(< B > | < Z >)\} = \varepsilon | (< B > | < Z >) | (< B > | < Z >) (< B > | < Z >) \dots$$

dass also jede dadurch erzeugte Zeichenkette aus einer Folge (aus $n \geq 0$ Elementen) von Buchstaben und Zahlen in beliebiger Mischung besteht.

## 1.4 Reguläre Ausdrücke

Eine etwas andere Notation zur Darstellung der Regeln einer regulären (nach Chomsky) Grammatik. Metasymbole sind

$$+, \quad *, \quad |, \quad (\,), \quad [\,]$$

| **Sprachelement** | **Bedeutung** |
|---|---|
| Worte aus $w \in T^*$ | Stehen für sich selbst |
| $A_1 \| A_2 \| A_3 \| \dots \| A_n$ | Alternative |
| $[w]$ mit $w \in T^*$ | Alternative für Einzelzeichen: Genau ein Zeichen aus der Zeichenfolge $w$ steht. |
| $A*$ | entspricht $\varepsilon \| A \| AA \| AAA \| \dots$ ($A$ steht $n$ mal mit $n \geq 0$) |
| $A+$ | entspricht $A \| AA \| AAA \| \dots$ ($A$ erscheint $n$ mal mit $n > 0$) |

Beispielsweise ist die Menge $M$ der Zeichenfolgen, die durch

$$[abc]+$$

generiert werden, gegeben als

$$M = \{a, b, c, aa, ab, ac, ba, \dots, caaabca, \dots\}$$

und die Vorschrift für die Bildung eines Namens für einen Bezeichner lautet

$$[a - zA - Z\,][0 - 9a - zA - Z\_\,]*$$

wobei das $-$ Zeichen in $\alpha - \beta$ für alle Zeichen des Alphabetes steht, die zwischen $\alpha$ und $\beta$ liegen (Bereichsangabe).

## 1.5 Syntax der while Programme

Wir verwenden nun die BNF Notation, um die Syntax der *while* Programme (s. Kap. Berechenbarkeit) darzustellen. Beispiel sei das folgende *while* Programm für die

**Multiplikation**: Eingabe: x1, x2, Ausgabe: x0 (Vorbelegung x0 = 0)
while (x1 !=0)
{ x3 = x2;
while (x3 != 0)
{ x0 = x0 + 1;
x3 = x3 - 1;
}
x1 = x1 - 1;
}

**Analyse allgemeines while Programm:**

1. Lexikalisch: Variablennamen bestehen aus einem Buchstaben $x$ oder $y$ oder $z$ gefolgt von einer natürlichen Zahl (einschließlich der 0).

2. Syntaktisch (Die Syntax ist hier etwas allgemeiner als die im Kapitel Berechenbarkeit vorgestellte, speziell sind auch einfache Zuweisungen <Variable>=<Variable> möglich)

| | | |
|---|---|---|
| <Programm> | ::= | <Zuweisung>\|<Programm><Programm> |
| | \| | while(<Test>){<Programm>} |
| <Zuweisung> | ::= | <Variable>=<Konstante>;\|<Variable>=<Variable>; |
| | \| | <Variable>=<Variable>-<Konstante>; |
| | \| | <Variable>=<Variable>+<Konstante>; |
| <Test> | ::= | <Variable>!=0 |

und $< Konstante >$ bezeichnet eine beliebige natürliche Zahl.