

Formale Systeme
WS 2011/2012

Skript zur Vorlesung

Christel Baier, Manuela Berg, Walter Nauber
Lehrstuhl für Algebraische und Logische Grundlagen der Informatik
Fakultät für Informatik
TU Dresden

Einleitung

Der erste Teil der Vorlesung behandelt Automatentheorie und formale Sprachen. Die behandelten Konzepte formaler Sprachen und der Automatentheorie bilden eine wichtige Grundlage z.B. für den Übersetzerbau, den Schaltkreisentwurf oder die Textverarbeitung. Im zweiten Teil beschäftigt sich die Vorlesung mit der Aussagenlogik.

Die Inhalte des ersten Teils der Vorlesung können zu großen Teilen in zahlreichen Lehrbüchern, die eine Einführung in die Theoretische Informatik anhand der Themenkomplexe formale Sprachen und Automatentheorie zum Ziel haben, nachgelesen werden. Eine Auswahl an Lehrbüchern, die den Inhalten der Vorlesung am nächsten kommen, ist:

1. A. Asteroth, C. Baier, *Theoretische Informatik. Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispielen*. Pearson Studium, 2002. (In der SLUB können einige Exemplare ausgeliehen werden.)
2. H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*. Prentice Hall, 1998.
3. J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 3.Auflage, 2007.
4. U. Schöning, *Theoretische Informatik kurz gefaßt*. Spektrum Akademischer Verlag, 4. Auflage, 2000.
5. T. Sudkamp, *Languages and Machines. An Introduction to the Theory of Computer Science*. Addison Wesley, 3. Auflage, 2006.
6. I. Wegener, *Theoretische Informatik*. B.G. Teubner, 1993.

Es gibt ebenfalls zahlreiche Bücher, die sich mit mathematischer Logik und deren Anwendungen in der Informatik befassen und in denen die Inhalte der Vorlesung zum Themenkomplex Aussagenlogik wiederzufinden sind. Wir erwähnen hier (in alphabetischer Reihenfolge) zwei Standardwerke, in denen Teile der Vorlesung nachgelesen werden können.

1. Steffen Hölldobler: *Logik und Logikprogrammierung* Kolleg Synchron Publishers, 3. Auflage, 2003.
2. Uwe Schöning: *Logik für Informatiker* Spektrum Verlag, 5. Auflage, 2000.

Die Vorlesung setzt Kenntnisse über die Module Algorithmen und Datenstrukturen, Programmierung, sowie Mathematik voraus, wie sie in den betreffenden Lehrveranstaltungen für Studierende der Informatik, Medieninformatik und Informationssystemtechnik an der TUD vermittelt werden.

Das griechische Alphabet. Häufig werden griechische Buchstaben zur Bezeichnung mathematischer Objekte verwendet. Um eine Vorstellung zu vermitteln, was die einzelnen Zeichen bedeuten und wie sie ausgesprochen werden, ist in Abbildung 1 das griechische Alphabet aufgelistet. Die Symbole Γ , Δ , Θ , Λ , Ξ , Σ , Υ , Φ , Ψ und Ω sind griechische Großbuchstaben. Alle anderen Symbole sind Kleinbuchstaben.

α	alpha	ι	iota	ρ	rho
β	beta	κ	kappa	σ, Σ	sigma
γ, Γ	gamma	λ, Λ	lambda	τ	tau
δ, Δ	delta	μ	mu	υ, Υ	upsilon
ϵ, ε	epsilon	ν	nu	ϕ, φ, Φ	phi
ζ	zeta	ξ, Ξ	xi	χ	chi
η	eta	\omicron	o	ψ, Ψ	psi
$\theta, \vartheta, \Theta$	theta	π, ϖ, Π	pi	ω, Ω	omega

Figure 1: Das griechische Alphabet

1. Teil: Formale Sprachen und Automatentheorie

Formale Sprachen und Automatenmodelle

Formale Sprachen und Automatenmodelle wie endliche Automaten oder Kellerautomaten haben zahlreiche Anwendungen in der Informatik. Zu den wichtigsten zählt der Compilerbau. Das Grobschema der ersten Phasen des Übersetzungsvorgangs ist in Abbildung 2 skizziert.

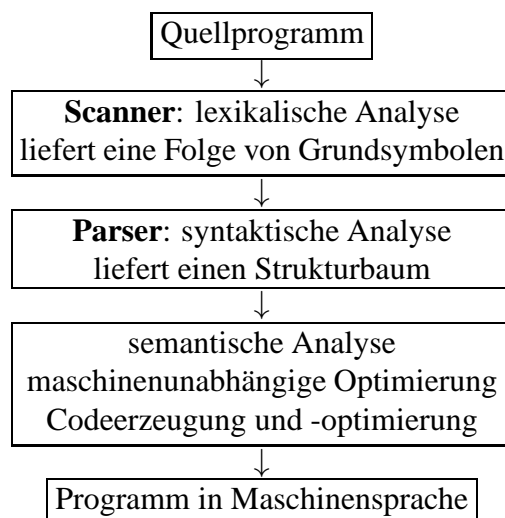


Figure 2: Grobschema des Übersetzungsvorgangs

Das Ziel der *lexikalischen Analyse* ist die Erkennung und Codierung der *Grundsymbole* sowie die Ausblendung bedeutungsloser Zeichen und Zeichenketten (z.B. Kommentare und Leerzeichen). Man unterscheidet vier Arten von Grundsymbolen:

- Schlüsselwörter: z.B. IF , THEN , BEGIN , etc.,
- Spezialsymbole: z.B. < , + , ; , etc.,
- Identifizier: vom Benutzer gewählte Namen für Variablen, Programmmodule, etc.
- Literale: vordefinierte Bezeichner für Werte gewisser Datentypen z.B. Zahlen.

Für Schlüsselwörter und Spezialembole ist keine spezielle Spezifikation notwendig, sie können explizit vorgegeben werden. Anders verhält es sich mit den Identifiern und Literalen. Diese werden durch *reguläre Sprachen* spezifiziert. Zu den unterschiedlichen Formalismen zur Darstellung regulärer Sprachen zählen Syntaxdiagramme, reguläre Ausdrücke, reguläre Grammatiken und endliche Automaten. Auf diese Formalismen werden wir in Kapitel 2 eingehen.

Das Programm geht als Folge von Grundsymbolen aus der lexikalischen Analyse hervor. In der *syntaktischen Analyse* geht es darum, die Struktur des Programms zu erkennen und eine abstrakte Baumstruktur des Quellprogramms zu erstellen. Dazu wird ein *Parser* verwendet, d.h. ein Analysealgorithmus, der die syntaktische Korrektheit des Programms (repräsentierende Folge von Grundsymbolen) bzgl. der syntaktischen Regeln, die der Programmiersprache zugrundeliegen, prüft. Die zulässige syntaktische Strukturierung von Programmen einer höheren Programmiersprache wird durch eine (deterministisch) *kontextfreie Grammatik* festgelegt. Beispielsweise könnte man die Konstrukte einer an unsere Algorithmensprache anlehnenen imperativen Sprache durch Regeln wie in Abbildung 3 angeben. Im Anschluss an die syntaktische

Analyse findet die *semantische Analyse* statt, deren Aufgaben sehr vielfältig sind (Erstellen eines Definitionsmoduls, Prüfung von Typverträglichkeit, Operatoridentifikation, etc.).

$Stmts$	\rightarrow	$Assignment \mid CondStmt \mid Loop \mid Stmts; Stmts \mid \dots$
$Assignment$	\rightarrow	$Variable := Expr$
$CondStmt$	\rightarrow	$IF BoolExpr THEN Stmts FI \mid \dots$
$Loop$	\rightarrow	$WHILE BoolExpr DO Stmts OD \mid$ $REPEAT Stmts UNTIL BoolExpr \mid \dots$
$BoolExpr$	\rightarrow	$Expr < Expr \mid Expr \leq Expr \mid \dots$ $BoolExpr \wedge BoolExpr \mid \neg BoolExpr \mid \dots$
		\vdots

Figure 3: Grammatik für Programme einer Pseudo-Programmiersprache

Die theoretischen Grundkonzepte von kontextfreien Sprachen werden wir in Kapitel 3 behandeln. Für deren Einsatz im Kontext des Compilerbaus sowie Details zur semantischen Analyse, Codegenerierung und Codeoptimierung verweisen wir auf Vorlesungen zum Thema Compilerbau.

Grammatiken versus Automaten. Ein wesentlicher Aspekt ist die Beziehung zwischen den verschiedenen Sprachtypen und Automatenmodellen (endliche Automaten, Kellerautomaten, Turingmaschinen). Während die Spezifikation einer formalen Sprache mit Hilfe einer Grammatik die Regeln festlegt, die ein Benutzer anwenden darf, um Wörter der Sprache zu bilden (z.B. den Programmcode zu formulieren), dienen die Automaten als Sprachakzeptoren. Die wesentliche Aufgabe des Automaten besteht darin, zu prüfen, ob ein gegebenes Eingabewort zu einer z.B. durch eine Grammatik gegebenen Sprache gehört. Diese Fragestellung taucht in natürlicherweise im Kontext von Übersetzern auf, da dort zu prüfen ist, ob der vom Benutzer verfasste Quellcode (das Eingabewort) den syntaktischen Regeln der betreffenden Programmiersprache (spezifiziert durch eine Grammatik oder ein Syntaxdiagramm) entspricht. Tatsächlich basiert die Funktionsweise eines Scanners auf einem *endlichen Automaten*; die eines Parsers auf einem *Kellerautomaten*.

Grundbegriffe für formale Sprachen

Bevor wir auf das formale Konzept von Grammatiken eingehen, fassen wir einige Notationen für Sprachen und Wörter zusammen, die teilweise bereits in der Mathematik-Vorlesung eingeführt wurden, und im Verlauf der Vorlesung häufig benutzt werden.

Alphabet. Ein *Alphabet* bezeichnet eine endliche nicht-leere Menge. Die Elemente eines Alphabets werden häufig *Zeichen* oder *Symbole* genannt. Alphabete bezeichnen wir meist mit

griechischen Großbuchstaben wie Σ (“Sigma”) oder Γ (“Gamma”). Wir schreiben $|\Sigma|$ für die Anzahl an Elementen in Σ .

Wort. Ein *endliches Wort* über Σ , kurz *Wort* genannt, ist eine endliche (eventuell leere) Folge von Zeichen in Σ , also von der Form $a_1 a_2 \dots a_k$, wobei $k \in \mathbb{N}$ und $\{a_1, \dots, a_k\} \subseteq \Sigma$. Für den Sonderfall $k = 0$ erhält man das *leere Wort*, welches mit dem griechischen Buchstaben ε (“epsilon”) bezeichnet wird.

Die *Länge* eines Wortes w ist die Anzahl an Zeichen, die in ihm vorkommen, und wird mit $|w|$ bezeichnet. Das leere Wort hat also die Länge 0 (d.h., $|\varepsilon| = 0$). Allgemein gilt $|a_1 a_2 \dots a_k| = k$.

Sind $w = a_1 \dots a_k$ und $u = b_1 \dots b_m$ Wörter über Σ , so bezeichnet $w \circ u$, oder kurz wu , das Wort $a_1 \dots a_k b_1 \dots b_m$, welches sich durch das Hintereinanderhängen von w und u ergibt. Man spricht auch von der *Konkatenation* von w und u .

Seien $w = a_1 \dots a_k$ und $u = b_1 \dots b_m$ Wörter über Σ .

- w wird *Präfix* von u genannt, falls es ein Wort v mit $u = wv$ gibt.
- w heißt *Suffix* von u , falls es ein Wort v mit $u = vw$ gibt.
- w heißt *Teilwort* von u , falls es Wörter v und x mit $u = vwx$ gibt.

Der Fall $v = \varepsilon$ (bzw. $x = \varepsilon$) ist dabei zugelassen. Also ist jedes Wort sowohl Präfix als auch Suffix (und auch Teilwort) von sich selbst. Ebenso ist ε zugleich Präfix und Suffix jedes Wortes. Weiter ist klar, dass jedes Präfix und jedes Suffix von u zugleich ein Teilwort von u ist. Ein Wort w mehrfach als Teilwort von u vorkommen, z.B. hat $w = 01$ drei Vorkommen in $u = 1101110101$. Wir sagen, dass w ein *echtes* Präfix von u ist, falls $w \neq u$ und w Präfix von u ist. In Analogie hierzu ist der Begriff “echtes Suffix” definiert. Ein Wort w wird echtes Teilwort von u genannt, falls w ein Teilwort von u ist und $w \neq u$.

Die Menge aller Wörter über einem Alphabet. Σ^* bezeichnet die Menge aller Wörter über Σ , einschliesslich dem leeren Wort ε . Die Bezeichnung Σ^+ wird für die Menge aller nichtleeren Wörter über Σ verwendet. Also:

$$\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$$

Ist $k \in \mathbb{N}$, so schreiben wir Σ^k für die Menge aller Wörter $w \in \Sigma^*$ der Länge k . Also $\Sigma^0 = \{\varepsilon\}$ und $\Sigma^k = \{a_1 \dots a_k : a_i \in \Sigma, i = 1, \dots, k\}$ für $k \geq 1$. Offenbar gilt:

$$\Sigma^* = \bigcup_{k \geq 0} \Sigma^k \quad \text{und} \quad \Sigma^+ = \bigcup_{k \geq 1} \Sigma^k$$

Mit Σ ist auch jede der Mengen Σ^k endlich. Genauer gilt: ist $|\Sigma| = n$, so ist $|\Sigma^k| = n^k$. Die Mengen Σ^* und Σ^+ sind jedoch unendlich, da es zu jedem $k \in \mathbb{N}$, $k \geq 1$, ein Wort der Länge k in Σ^+ (und Σ^*) gibt. (Beachte, dass Alphabete als nicht-leer vorausgesetzt werden.)

Sprache über einem Alphabet. Eine *formale Sprache*, kurz *Sprache* genannt, über einem Alphabet Σ ist eine beliebige Teilmenge L von Σ^* .

Man beachte, dass alle Sprachen abzählbar sind. Ist nämlich Σ ein Alphabet (also eine endliche nicht-leere Menge), so ist Σ^* als abzählbare Vereinigung der endlichen Mengen Σ^k , $k \geq 0$, abzählbar. Somit ist jede Teilmenge von Σ^* , also jede Sprache L über Σ , ebenfalls abzählbar.

Zur Erinnerung: eine Menge X heißt *abzählbar*, falls $X = \emptyset$ oder es eine surjektive Abbildung $h: \mathbb{N} \rightarrow X$ gibt. Ist $h: \mathbb{N} \rightarrow X$ surjektiv, so werden durch die unendliche Folge $h(0), h(1), h(2), \dots$ alle Elemente aus X aufgezählt (eventuell manche Elemente mehrfach). Abzählbarkeit von X ist gleichbedeutend damit, dass X entweder endlich ist oder es eine bijektive Abbildung $f: X \rightarrow \mathbb{N}$ gibt. Aus den Mathematik-Vorlesungen ist bekannt, dass alle Teilmengen einer abzählbaren Menge abzählbar sind und abzählbare Vereinigungen abzählbarer Mengen wieder abzählbar sind. Beispiele für abzählbare Mengen sind alle endlichen Mengen, die Menge \mathbb{N} der natürlichen Zahlen, aber auch die Menge \mathbb{Q} der rationalen Zahlen. Beispiele für *überabzählbare* Mengen, d.h., Mengen, die nicht abzählbar sind, sind die Menge \mathbb{R} der reellen Zahlen, die Potenzmenge $2^{\mathbb{N}}$ von \mathbb{N} oder auch $\mathbb{N}^{\mathbb{N}}$. Hier sowie im Folgenden bezeichnet 2^X die Potenzmenge von X .

Operatoren für Sprachen. Zu den üblichen Verknüpfungsoperatoren für Sprachen L, L_1, L_2 über Σ zählen Vereinigung $L_1 \cup L_2$, Durchschnitt $L_1 \cap L_2$, Komplement \bar{L} sowie Konkatenation $L_1 \circ L_2$ und Kleeneabschluss L^* . Die zuletzt genannten Sprachen sind wie folgt definiert. Seien L, L_1, L_2 Sprachen über Σ .

- Das *Komplement* von L ist durch $\bar{L} \stackrel{\text{def}}{=} \Sigma^* \setminus L$ gegeben.
- *Konkatenation*: $L_1 \circ L_2$, oder kurz $L_1 L_2$, bezeichnet die Menge aller Wörter $x_1 x_2$, welche sich durch Konkatenation eines Worts $x_1 \in L_1$ und eines Worts $x_2 \in L_2$ ergeben:

$$L_1 \circ L_2 = L_1 L_2 \stackrel{\text{def}}{=} \{x_1 x_2 : x_1 \in L_1, x_2 \in L_2\}$$

- Die Sprachen L^n für $n \in \mathbb{N}$ sind wie folgt definiert:

$$L^0 \stackrel{\text{def}}{=} \{\varepsilon\}, \quad L^{n+1} \stackrel{\text{def}}{=} L \circ L^n$$

Für $n \geq 1$ gilt also $w \in L^n$ genau dann, wenn w die Form $w = x_1 x_2 \dots x_n$ mit $\{x_1, \dots, x_n\} \subseteq L$ hat.

- Der *Kleeneabschluss* von L ist durch

$$L^* \stackrel{\text{def}}{=} \bigcup_{n \geq 0} L^n$$

definiert. Man spricht auch manchmal von dem *Sternoperator*. Die Elemente von L^* sind also genau diejenigen Wörter, die sich durch Aneinanderhängen von einer beliebigen endlichen Anzahl (eventuell 0) von Wörtern aus L ergeben. Der *Plusoperator* ist durch

$$L^+ \stackrel{\text{def}}{=} \bigcup_{n \geq 1} L^n$$

definiert. Beachte, dass stets $\varepsilon \in L^*$ gilt. Dagegen gilt $\varepsilon \in L^+$ nur dann, wenn $\varepsilon \in L$ gilt. Daher gilt $L^+ = L^*$, falls $\varepsilon \in L$, und $L^+ = L^* \setminus \{\varepsilon\}$, falls $\varepsilon \notin L$.

Beispielsweise kann man die Menge von natürlichen Dezimalzahlen ohne führende Nullen durch die Sprache

$$L_{\text{Dezimalzahl}} = \{0\} \cup \{1, 2, \dots, 9\} \circ \{0, 1, 2, \dots, 9\}^*$$

über dem Alphabet $\Sigma = \{0, 1, 2, \dots, 9\}$ charakterisieren.

Neben den üblichen Gesetzen der Mengenalgebra (z. B. Kommutativität und Assoziativität für Vereinigung und Durchschnitt) gelten z. B. folgende Gesetze für die Konkatenation. Dabei sind L_1, L_2, K beliebige Sprachen.

$$\begin{aligned} (L_1 \cup L_2)K &= L_1K \cup L_2K & \{\varepsilon\}K &= K\{\varepsilon\} = K \\ K(L_1 \cup L_2) &= KL_1 \cup KL_2 & \emptyset K &= K\emptyset = \emptyset \end{aligned}$$

Weiter gilt für den Kleeneabschluss und Plusoperator $K^* = (K \setminus \{\varepsilon\})^* = K^+ \cup \{\varepsilon\}$ und $K^+ = KK^* = K^*K$.

Abgeschlossenheit unter Verknüpfungsoperatoren. Sei \mathfrak{K} eine Klasse von Sprachen. Die Klasse \mathfrak{K} ist unter der Vereinigung abgeschlossen, falls für je zwei Sprachen L_1 und L_2 gilt:

$$\text{Falls } L_1 \in \mathfrak{K} \text{ und } L_2 \in \mathfrak{K}, \text{ so ist auch } L_1 \cup L_2 \in \mathfrak{K}.$$

Dabei kann man o. E. annehmen, dass L_1 und L_2 dasselbe Alphabet zugrundeliegt. Sind $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$, dann betrachten wir das Alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ und fassen L_1, L_2 und $L_1 \cup L_2$ als Sprachen über dem Alphabet Σ auf. Entsprechende Bedeutung haben die Begriffe Abgeschlossenheit unter Durchschnitt, Komplement, Konkatenation oder Kleeneabschluss.

1 Grammatiken

Grammatiken geben Regeln vor, nach denen Wörter einer Sprache generiert werden können. Die Grundidee ist ein *Termersetzungsprozess*, bei dem einzelne Zeichen oder Zeichenketten – gebildet aus Terminalzeichen und Hilfsvariablen (Nichtterminalen) – durch andere Wörter ersetzt werden dürfen bis ein Wort entsteht, das ausschließlich Terminalzeichen enthält. Die Terminalzeichen sind die Symbole der Sprache. Beispielsweise wird man die Symbole "“+”, "“–”, 0, ..., 9 als Terminalzeichen verwenden, um ganzzahlige Dezimalzahlen zu spezifizieren.

Beispiel 1.1 (Spezifikation für "Identifer"). Bevor wir die formale Definition einer Grammatik angeben, erläutern wir die Grundidee an einem einfachen Beispiel. Viele Programmiersprachen lassen als Identifer genau diejenigen Wörter zu, die aus einer beliebig langen Zeichenkette, gebildet aus Buchstaben a, b, \dots, z sowie den Ziffern $0, 1, \dots, 9$ bestehen und mit einem Buchstaben beginnen.

Jeder *Identifer* besteht aus einem *Buchstaben*, gefolgt von beliebig vielen *Buchstaben* oder *Ziffern*.

Mit den Operatoren \circ (Konkatenation), Vereinigung und dem Kleeneabschluss können wir die Identifer durch die Sprache

$$L_{Idf} = \mathfrak{B} \circ (\mathfrak{B} \cup \mathfrak{Z})^*$$

charakterisieren. Dabei ist $\mathfrak{B} = \{a, b, \dots, z\}$ die Menge (endliche Sprache) der Buchstaben und $\mathfrak{Z} = \{0, 1, \dots, 9\}$. (Wir nehmen hier an, dass der Begriff "Buchstabe" im Sinn von "Kleinbuchstabe" verwendet wird.) Alternativ können wir L_{Idf} durch die in Abbildung 4 angegebenen *Regeln* darstellen.

Idf	\rightarrow	BA
A	\rightarrow	$\varepsilon \mid BA \mid ZA$
B	\rightarrow	$a \mid b \mid c \mid \dots \mid z$
Z	\rightarrow	$0 \mid 1 \mid 2 \mid \dots \mid 9$

Figure 4: Grammatik G_{Idf}

Intuitiv steht Idf für eine Variable, die über alle Identifer quantifiziert, während das Symbol A alle Zeichenketten repräsentiert, die aus Buchstaben und Ziffern gebildet werden (einschließlich der Zeichenkette der Länge 0, die wir durch das leere Wort ε repräsentieren). Die Symbole B und Z stehen intuitiv für alle Buchstaben bzw. Ziffern. Die erste Regel $Idf \rightarrow BA$ besagt, dass wir einen Identifer erhalten, indem wir zuerst einen Buchstaben wählen und an diesen ein aus dem Symbol A hergeleitetes Wort hängen. Die zweite Zeile ist eine Kurzschreibweise für die drei Regeln $A \rightarrow \varepsilon$, $A \rightarrow BA$ und $A \rightarrow ZA$. Diese drei Regeln erlauben es, das Symbol A durch eines der drei Wörter ε , BA oder ZA zu ersetzen. Dieser Vorgang kann beliebig oft wiederholt werden bis schliesslich alle generierten B 's und Z 's durch konkrete Buchstaben bzw. Ziffern ersetzt werden. ■

Eine Grammatik besteht aus einer Menge von Variablen, auch Nichtterminale genannt, die repräsentativ für herleitbare Wörter stehen. Weiter gibt es eine Menge von Terminalsymbolen. Dies sind die Zeichen des Alphabets Σ , über dem wir eine Sprache definieren möchten. Die Regeln (auch Produktionen genannt) sind Paare (x, y) , üblicherweise in Pfeilnotation $x \rightarrow y$ geschrieben, wobei x und y Wörter gebildet aus Terminalzeichen und Variablen sind. Die einzige Forderung ist, dass das Wort x auf der linken Seite mindestens ein Nichtterminal enthält.

Definition 1.2 (Grammatik). Eine Grammatik ist ein Tupel $G = (V, \Sigma, \mathcal{P}, S)$ bestehend aus

- einer endlichen Menge V von *Variablen* (auch *Nichtterminale* genannt),
- einem Alphabet Σ mit $V \cap \Sigma = \emptyset$,
- einem *Startsymbol* $S \in V$,
- einer endlichen Menge \mathcal{P} bestehend aus Paaren (x, y) von Wörtern $x, y \in (V \cup \Sigma)^*$ mit $x \notin \Sigma^*$, d.h., x enthält wenigstens eine Variable.

Σ wird auch *Terminalalphabet* genannt. Die Elemente von Σ werden als *Terminalsymbole* oder *Terminalzeichen*, oder auch *Terminale*, bezeichnet. Die Elemente von \mathcal{P} werden *Produktionen* oder *Regeln* genannt. Man nennt \mathcal{P} daher auch *Produktionssystem* oder *Regelsystem*. Regeln der Form $x \rightarrow \varepsilon$ werden ε -Regeln genannt. Regeln der Form $A \rightarrow B$, wobei A und B Nichtterminale sind, werden auch Kettenregeln genannt. ■

Das Produktionssystem wird üblicherweise nicht als Menge von Wortpaaren angegeben. Stattdessen wird eine Pfeilnotation verwendet. Üblich ist die Schreibweise $x \rightarrow y$ anstelle von $(x, y) \in \mathcal{P}$. Entsprechend schreibt man $x \not\rightarrow y$, falls $(x, y) \notin \mathcal{P}$. Ferner ist

$$x \rightarrow y_1 \mid y_2 \mid \dots \mid y_n$$

eine Kurzschreibweise für die Regeln $x \rightarrow y_1, x \rightarrow y_2, \dots, x \rightarrow y_n$.¹ Wir werden im Folgenden häufig auf die explizite Angabe der Komponenten V, Σ und S einer Grammatik verzichten, sondern nur das zugehörige Produktionssystem angeben. Wir verwenden Großbuchstaben S, A, B, \dots für die Nichtterminale und Kleinbuchstaben a, b, c, \dots am Anfang des (gewöhnlichen) Alphabets für die Terminalzeichen. Werden keine weiteren Angaben gemacht, dann ist S das Startsymbol. Für Wörter über $V \cup \Sigma$ verwenden wir meist Kleinbuchstaben wie u, v, w, x, y, z am Ende des Alphabets.

¹Die Backus-Naur-Form, kurz BNF, ist eine weit verbreitete syntaktische Notation, um Produktionssysteme für Grammatiken anzugeben. Neben der oben erläuterten Notation $x \rightarrow y_1 \mid y_2 \mid \dots \mid y_n$ stellt die BNF noch weitere Kurzschreibweisen zur Verfügung. Eckige Klammern stehen für Optionen, geschweifte Klammern stehen für beliebig viele Wiederholungen, d.h.,

$$\begin{aligned} x \rightarrow u[v]w & \text{ steht für } x \rightarrow uw \mid uvw \\ x \rightarrow u\{v\}w & \text{ steht für } x \rightarrow uBw, \quad B \rightarrow \varepsilon \mid v \mid vB, \end{aligned}$$

wobei B eine neue Variable ist. Ferner ist in der BNF-Notation die Verwendung des Symbols “ $::=$ ” anstelle des Pfeils “ \rightarrow ” gebräuchlich.

Beispiel 1.3 (Die Grammatik G_{Idf}). Zunächst machen wir uns die formale Definition einer Grammatik am Beispiel der Regeln für Identifier klar, siehe Abbildung 4 auf Seite 8. Die Variablenmenge ist $V = \{S, A, B, Z\}$, wobei $S = Idf$. Das Startsymbol ist $S = Idf$. Das Terminalalphabet ist die Menge der (Klein-)Buchstaben und Ziffern, also $\Sigma = \{a, b, \dots, z, 0, 1, \dots, 9\}$. Die oben angegebenen Regeln stehen für das Produktionssystem

$$\mathcal{P} = \left\{ \begin{array}{l} (S, BA), (A, \varepsilon), (A, BA), (A, ZA) \\ (B, a), (B, b), \dots, (B, z) \\ (Z, 0), (Z, 1), \dots, (Z, 9) \end{array} \right\}$$

In Pfeilnotation entspricht dies den Regeln $S \rightarrow BA$, $A \rightarrow \varepsilon \mid BA \mid ZA$ und $B \rightarrow a \mid b \mid \dots \mid z$ sowie $Z \rightarrow 0 \mid 1 \mid \dots \mid 9$. ■

Das Anwenden einer Regel bedeutet, dass man in einem bereits hergeleiteten Wort das Teilwort x auf der linken Seite einer Regel $x \rightarrow y$ durch das Wort y auf der rechten Seite ersetzt. Dies wird durch eine Relation \Rightarrow bestehend aus Wortpaaren über dem Alphabet $V \cup \Sigma$ formalisiert. Üblich ist die Schreibweise $u \Rightarrow v$ anstelle von $(u, v) \in \Rightarrow$.

Definition 1.4 (Her-/Ableitungsrelationen \Rightarrow und \Rightarrow^*). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine Grammatik. Die Relation \Rightarrow ist wie folgt definiert. Seien u und v zwei Wörter über dem Alphabet $V \cup \Sigma$. Dann gilt $u \Rightarrow v$ genau dann, wenn es Wörter $x, y, w, z \in (V \cup \Sigma)^*$ gibt, so dass

$$u = wxz, \quad v = wyz, \quad x \rightarrow y.$$

Die *Herleitungsrelation* von G (auch *Ableitungsrelation* genannt) \Rightarrow^* bezeichnet die reflexive, transitive Hülle von \Rightarrow . Diese ist wie folgt definiert. Sind $u, v \in (V \cup \Sigma)^*$, so gilt

$$u \Rightarrow^* v$$

genau dann, wenn es eine Folge u_0, u_1, \dots, u_n von Wörtern $u_i \in (V \cup \Sigma)^*$ gibt, so dass

$$u_0 = u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_{n-1} \Rightarrow u_n = v.$$

Man spricht auch von einer Herleitung (oder Ableitung) von v aus u . Die *Länge* einer Herleitung ist durch die Anzahl an Regelanwendungen gegeben. Obige Herleitung hat also die Länge n . Der Spezialfall $n = 0$ ist zugelassen. Es gilt also $u \Rightarrow^* u$ für alle Wörter $u \in (V \cup \Sigma)^*$. Falls $S \Rightarrow^* v$ (wobei S das Startsymbol ist), so sagen wir, dass v in G herleitbar ist. ■

Es gilt also $u \Rightarrow v$ genau dann, wenn v aus u durch Anwendung genau einer Regel hervorgeht, d.h., wenn es eine Regel $x \rightarrow y$ gibt, so dass v aus u entsteht, indem ein Vorkommen von x in u durch das Wort y ersetzt wird. Weiter gilt $u \Rightarrow^* v$ genau dann, wenn sich v aus u durch Anwenden von beliebig vielen (null oder mehreren) Regeln aus u ergeben kann. Das Symbol \Rightarrow bezeichnet also die ‘‘Ein-Schritt-Herleitungsrelation’’, während \Rightarrow^* für die ‘‘Mehrschritt-Herleitungsrelation’’ steht.

Zur Verdeutlichung, welche Grammatik zugrundeliegt, indizieren wir manchmal die Symbole \Rightarrow und \Rightarrow^* mit der betreffenden Grammatik, d.h., wir schreiben an manchen Stellen auch \Rightarrow_G statt \Rightarrow und \Rightarrow_G^* statt \Rightarrow^* . Vereinzelt verwenden wir auch die Schreibweise $x \rightarrow_G y$ statt $x \rightarrow y$, um zu verdeutlichen, dass es sich hierbei um eine Regel der Grammatik G handelt.

Beispielsweise hat die Grammatik G_{Idf} für “Identifier” in Abbildung 4 auf Seite 8 folgende Herleitung des Worts b46.

$$\begin{aligned} Idf &\Rightarrow BA \Rightarrow BZA \Rightarrow BZZA \\ &\Rightarrow BZZ \Rightarrow bZZ \Rightarrow b4Z \Rightarrow b46 \end{aligned}$$

Daher gilt $Idf \Rightarrow^* b46$.

Definition 1.5 (Erzeugte Sprache $\mathcal{L}(G)$). Sei $G = (V, \Sigma, \mathcal{P}, S)$ eine Grammatik. Die durch G erzeugte Sprache

$$\mathcal{L}(G) \stackrel{\text{def}}{=} \{ w \in \Sigma^* : S \Rightarrow^* w \}$$

besteht aus allen Wörtern $w \in \Sigma^*$, die sich aus dem Startsymbol S ableiten lassen. $\mathcal{L}(G)$ wird häufig auch die von G *definierte* oder *generierte* Sprache genannt. ■

Wir betrachten die Grammatik G_{Idf} in Abbildung 4 auf Seite 8. Die erzeugte Sprache $\mathcal{L}(G_{Idf})$ ist – wie erwartet – die Sprache bestehend aus allen Zeichenketten, die mit einem Buchstaben beginnen und dann mit einem beliebig langen (eventuell leeren) Wort – gebildet aus Buchstaben und Ziffern – enden. Aufgrund der einfachen Struktur der Grammatik G_{Idf} kann diese Aussage zwar als offensichtlich angesehen werden. Dennoch wollen wir exemplarisch am Beispiel einer vereinfachten Version G von G_{Idf} erläutern, wie der Nachweis, dass die erzeugte Sprache einer Grammatik mit einer gegebenen Sprache L übereinstimmt, erbracht werden kann. Grammatik G verwendet nur zwei Terminalsymbole b (anstelle aller Buchstaben) und c (anstelle aller Ziffern) und die Nichtterminalen S (statt Idf), A und B (wie in G_{Idf}) und C (statt Z). Die Regeln von G sind wie folgt:

$$\begin{array}{ll} S &\rightarrow BA & B &\rightarrow b \\ A &\rightarrow \varepsilon \mid BA \mid CA & C &\rightarrow c \end{array}$$

Die Behauptung ist nun, dass $\mathcal{L}(G) = \{ bx : x \in \{b, c\}^* \}$.

“ \supseteq ” Zu zeigen ist, dass jedes nicht-leere Wort w über dem Terminalalphabet Σ , dessen erstes Zeichen ein Symbol b ist, eine Herleitung in G besitzt. Hierzu genügt es eine Herleitung von w in G anzugeben. Sei $w = bx$, wobei $x = d_1 \dots d_k$ mit $d_i \in \{b, c\}$. Für $i \in \{1, \dots, k\}$ sei

$$D_i = \begin{cases} B & : \text{ falls } d_i = b \\ C & : \text{ falls } d_i = c \end{cases}$$

Durch Anwenden der Regeln $S \rightarrow BA$ und $A \rightarrow D_i$ für $i = 1, 2, \dots, k$ und schliesslich $A \rightarrow \varepsilon$ sowie die Regeln $B \rightarrow b$ und $D_i \rightarrow d_i$ für $i = 1, 2, \dots, k$ erhalten wir folgende Herleitung von w in G :

$$\begin{aligned} S &\Rightarrow BA \Rightarrow BD_1A \Rightarrow BD_1D_2A \Rightarrow \dots \Rightarrow BD_1D_2\dots D_kA \\ &\Rightarrow BD_1D_2\dots D_k \Rightarrow bD_1D_2\dots D_k \Rightarrow \dots \Rightarrow bd_1D_2D_3\dots D_k \\ &\Rightarrow bd_1d_2D_3\dots D_k \Rightarrow \dots \Rightarrow bd_1d_2\dots d_k = w \end{aligned}$$

“ \subseteq ” Zu zeigen ist, dass alle aus G herleitbaren Wörter über dem Terminalalphabet $\{b, c\}$ nicht-leer sind und mit dem Symbol b beginnen. Der Beweis hierfür kann erbracht werden, indem man folgende Eigenschaft für alle in G herleitbaren Wörter über $\{A, B, C\} \cup \{b, c\}$ nachweist.

Aus $S \Rightarrow^* y \in (\{A, B, C\} \cup \{b, c\})^*$ folgt $y = uv_1 \dots v_{k-1} d_k$, wobei $k \geq 1$ und

- (i) $u \in \{B\} \cup \{b\}$
- (ii) $v_1, \dots, v_{k-1} \in \{B, C\} \cup \{b, c\}$
- (iii) $v_k \in \{A, B, C\} \cup \{b, c\}$.

Diese Aussage kann durch Induktion nach der Länge einer Herleitung von y verifiziert werden. Ist nun $y \in \mathcal{L}(G)$, so gilt $S \Rightarrow^* y \in \{b, c\}^*$ und somit $u = b$ und $v_1, \dots, v_{k-1}, v_k \in \{b, c\}$.