

# FORMALE SPRACHEN

## Kurzbeschreibung

Nach der Behandlung von Systemen und Sprachen zur Textersetzung werden die Chomsky-Grammatiken eingeführt, durch die die Formalen Sprachen in insgesamt vier Sprachklassen eingeordnet werden. Mit den Endlichen Automaten wird ein Maschinenmodell vorgestellt, das die einfachste Sprachklasse in der Chomsky-Hierarchie zu verarbeiten gestattet.

## Schlüsselwörter

Algorithmus, Textersetzung, Semi-Thue-System, Grammatik, Chomsky-Sprachklassen, endlicher Automat, regulärer Ausdruck

## Lernziele

1. Die Bedeutung der Textersetzung als elementarste Form der Beschreibung von Algorithmen und der Verarbeitung von Informationen wird verstanden.
2. Ein endlicher Automat zur Erkennung und Erzeugung von Wörtern einer formalen Sprache kann erstellt werden.
3. Die durch Textersetzungssysteme (Semi-Thue-Systeme) und endliche Automaten erzeugten Sprachen können in das durch die Chomsky-Sprachklassen beschriebene Spektrum der Formalen Sprachen eingeordnet werden.

## Hauptquellen

- Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1995.

## Inhaltsverzeichnis

1	SEMI-THUE-SYSTEME .....	2
1.1	Regeln und Metaregeln .....	3
1.2	Zusammenhang zu Sprache und Algorithmus .....	4
1.3	Beispiel Kaffeedosenspiel .....	6
1.4	Markov-Algorithmen .....	7
1.5	Formale Systeme .....	9
2	GRAMMATIKEN .....	10
2.1	Chomsky-Hierarchie .....	12
2.2	Backus-Naur-Form .....	15
3	ENDLICHE AUTOMATEN .....	17
3.1	Akzeptoren .....	20
3.2	Regulärer Ausdruck .....	23
4	VERZEICHNISSE .....	26
	Abkürzungen und Glossar .....	26
	Index .....	27
	Informationen und Interaktionen .....	27
	Literatur .....	27

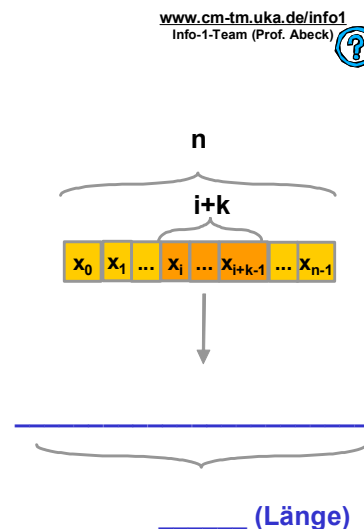
- SEMI-THUE-SYSTEME
  - Regeln und Metaregeln, Markov-Algorithmen, Formale Systeme
- GRAMMATIKEN
  - Nichtterminale und Terminale, Axiom, Chomsky-Hierarchie, Ableitungsbaum
- ENDLICHE AUTOMATEN
  - Mealy-Automat, Moore-Automat, Akzeptor, Reguläre Ausdrücke, Einordnung in die Chomsky-Hierarchie

### Information 1: FORMALE SPRACHEN

## 1 SEMI-THUE-SYSTEME

Systeme zur Textersetzung, so genannte Semi-Thue-Systeme (Interaktion 1) wurden bereits 1914 von Axel Thue (1863 - 1922, norwegischen Mathematiker und Logiker) entwickelt [Go95].

- Textersetzung mittels Semi-Thue-Systemen stellt die einfachste Form von Algorithmen dar
- Vorgehen
  - endlicher Zeichenvorrat  $\Sigma$
  - Wörter  $x = x_0 \dots x_{n-1}$ , wobei Zeichen  $x_i \in \Sigma$
  - $|x| = n$  heißt Länge eines Wortes
  - Überführung eines Wortes  $x$  in ein anderes Wort durch Ersetzen
    - von Teilwörter  $x_i \dots x_{i+k-1}$  durch andere Wörter  $y_j \dots y_{j+l-1}$ , wobei  $k, l \geq 0$  und  $i+k \leq n$
- Schreibweise:  $l_1 \dots l_n \rightarrow r_1 \dots r_m$ 
  - $l_1 \dots l_n$  heißt linke Seite  $l$
  - $r_1 \dots r_m$  heißt rechte Seite  $r$



### Interaktion 1: SEMI-THUE-SYSTEME – Prinzip

Semi-Thue-Systeme zeichnen sich nicht nur durch ihre Einfachheit, sondern auch durch ihre hohe Allgemeinheit aus. Es wird von einem endlichen Zeichenvorrat und einer abzählbaren (ggf. unendlichen) Anzahl an Regeln ausgegangen. Die einzige (einfache und allgemeine) Operation, die zugelassen wird, besteht in der Ersetzung von Teilworten: Befindet sich in einem gegebenen Wort  $x$  ein Teilwort (hier der Länge  $k$ ), das die linke Seite einer Regel ist, so kann dieses Teilwort gegen die rechte Seite in  $x$  ersetzt werden. Eine solche einfache Textersetzung soll in Interaktion 1 exemplarisch durchgeführt werden und die Länge des sich ergebenden Textes ist anzugeben.

## 1.1 Regeln und Metaregeln

In Information 2 wird ein einfaches Semi-Thue-System beschrieben, das eine Addition auf der Basis einer Strichdarstellung durchführt. Als Zeichenvorrat werden nur zwei Zeichen, der Strich und das Additionszeichen, benötigt.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck)

- Zeichenvorrat  $\Sigma = \{ |, + \}$
- Regeln
  - (1)  $+| \rightarrow |+$
  - (2)  $+ \rightarrow \varepsilon$
- Bedeutung der Regeln, also des Algorithmus
  - Addition natürlicher Zahlen, die als Folge von Strichen dargestellt sind
  - Beispiel zur Anwendung der Regeln
 
$$|||+|| \Rightarrow ||||+| \Rightarrow |||||+ \Rightarrow |||||$$
- $l \Rightarrow r$  heißt Transformation oder direkte Ableitung
  - $l \Rightarrow^+ r$  heißt:  $r$  kann aus  $l$  durch fortgesetzte Ableitung gewonnen werden
  - $l \Rightarrow^* r$  heißt:  $l \Rightarrow^+ r$  oder  $l = r$  ( $r$  kann auf  $l$  reduziert werden)

### Information 2: Beispiel eines Semi-Thue-Systems

Das Ergebnis der Addition von zwei durch ein Additionszeichen getrennte Strichzahlen wird durch Wegstreichen des Additionszeichens und Zusammenschieben der beiden Strichdarstellungen erzielt.

Wurde der Übergang von  $l$  nach  $r$  durch die Anwendung einer einzigen Regel erreicht (z.B.  $l = |||+||$  und  $r = ||||+|$ ), so wird von einer direkten Ableitung gesprochen und der Doppelpfeil  $\Rightarrow$  benutzt. Wurden eine oder mehrere Regeln angewendet, so ist das eine so genannte fortgesetzte Ableitung, was durch eine Ergänzung des Doppelpfeils um ein hochgestelltes Additionszeichen, also  $\Rightarrow^+$ , kenntlich gemacht wird. Falls auch der Fall eingeschlossen sein soll, dass keine Regel angewendet werden kann, d.h.  $l = r$  ist, so wird das Additionszeichen durch einen Stern ergänzt ( $\Rightarrow^*$ ).

In dem in Information 2 beschriebenen Ableitungsbeispiel wurde die Regel (1) solange angewendet, bis die linke Seite dieser Regel nicht mehr im Text auftrat und damit die Regel auch nicht mehr genutzt werden konnte. Wie durch die in Interaktion 2 angegebenen Metaregeln festgelegt wird, hätte man auch mit der Anwendung der Regel (2) beginnen können (was zu dem gleichen Endresultat führt).

- Durch folgende drei Metaregeln wird die Anwendung der Regeln festgelegt:

(M1) wenn  $a...b \rightarrow c...d$  anwendbar ist, ersetze das Teilwort  $a...b$  von  $x$  durch  $c...d$

(M2) wenn  $a...b$  mehrfach vorkommt oder mehrere Regeln anwendbar sind, so wähle das Teilwort bzw. die Regel beliebig

(M3) wiederhole die Anwendung von Regeln beliebig oft

Zu dem angegebenen Wort  $III+II+I$  ist zu zeigen, dass im Beispiel-Semi-Thue-System beide Formen des Nichtdeterminismus aus Metaregel (M2) vorkommen

$III+II+I \Rightarrow$  \_\_\_\_\_

oder  $\Rightarrow$  \_\_\_\_\_

oder  $\Rightarrow$  \_\_\_\_\_

oder  $\Rightarrow$  \_\_\_\_\_

oder  $\Rightarrow$  \_\_\_\_\_

## Interaktion 2: Metaregeln eines Semi-Thue-Systems

Die erste Metaregel (M1) legt das grundsätzliche Prinzip der Textersetzung fest, das am Anfang dieses Abschnitts vorgestellt wurde.

Die zweite Metaregel (M2) betrifft Fälle, in denen eine oder mehrere Regeln an mehreren Stellen zur Anwendung kommen können. Aus diesem Grund weisen Semi-Thue-Systeme ein nichtdeterministisches Verhalten auf.

Die dritte Meta-Regel (M3) realisiert eine Schleife, durch die eine fortlaufende Regel-Anwendung gewährleistet ist, bis keine passende Regel mehr gefunden werden kann.

In der in Information 2 beschriebenen Beispiel-Ableitung entstand ein Nichtdeterminismus dadurch, dass Regel (1) und Regel (2) beide angewendet werden konnten. In der in Interaktion 2 zu ergänzenden Ableitung ergibt sich der Nichtdeterminismus nicht nur aufgrund der Anwendungsmöglichkeiten mehrerer Regeln, sondern zudem aufgrund mehrerer möglicher Anwendungsstellen einer Regel.

Neben der Eigenschaft des Nichtdeterminismus, der eine Aussage zum Verhalten eines Algorithmus trifft, ist die Eigenschaft der Nichtdeterminiertheit zu unterscheiden, die das Ergebnis eines Algorithmus zu einer Eingabe betrifft [Sa04].

## 1.2 Zusammenhang zu Sprache und Algorithmus

Ein Semi-Thue-System ist somit definiert durch die dem System zugrunde liegende Regelmenge und die für alle Systeme geltenden Metaregeln. Anstelle der Bezeichnung Semi-Thue-System ist auch die Bezeichnung Textersetzungssystem (*String Replacement System*, *String Rewrite System*) üblich.



- Eine Menge  $T = \{p \rightarrow q\}$  von Regeln zusammen mit den Metaregeln M1, M2 und M3 heißt ein Semi-Thue-System

Beispiel-Semi-Thue-System  $T_B$ 

$$T_B = \{ \underline{\hspace{10em}} \}$$

- Die Menge aller Texte  $r$ , die aus dem Text  $l$  abgeleitet werden können, heißt die Formale Sprache
  - Schreibweise  $L_l = L(T, l)$

$$L(T_B, |||+||) =$$

$$\{ \underline{\hspace{10em}} \}$$

- $T^{-1} = \{q \rightarrow p\}$ 
  - inverses Semi-Thue-System, das dadurch entsteht, dass alle Pfeilrichtungen umgekehrt werden

$$T_B^{-1} = \{ \underline{\hspace{10em}} \}$$

$$L(T_B^{-1}, ||||) =$$

$$\{ \underline{\hspace{10em}} \}$$

### Interaktion 3: Semi-Thue-System und Formale Sprache

Ein Semi-Thue-System ist ein Beispiel eines Formalen Systems. Unmittelbar verknüpft mit den Formalen Systemen sind die Formalen Sprachen, wie anhand des Beispiel-Semi-Thue-Systems in Interaktion 3 verdeutlicht wird. Das Semi-Thue-System erzeugt aus einem vorgegebenen Text weitere Texte. Diese Textmenge ist die durch das Semi-Thue-System erzeugte formale Sprache.

Es lässt sich zu jedem Semi-Thue-System  $T$  sein Inverses  $T^{-1}$  bilden, indem einfach die linken und rechten Seiten der Regeln vertauscht werden.  $T^{-1}$  ist dann selbst wieder ein Semi-Thue-System.

- Ein Semi-Thue-System stellt die Grundform eines Algorithmus dar
  - Ersetzungsregeln sind die Operationen, die im Prinzip beliebig oft auf eine Eingabe angewendet werden
  - Terminierung des Algorithmus, sobald keine Ersetzungsregel mehr anwendbar ist
  - Semi-Thue-Systeme sind (potenziell) nicht-terminierende Algorithmen
- Semi-Thue-Systeme sind im Allgemeinen nicht-deterministische Algorithmen
  - siehe Meta-Regel (M2)
  - ein nicht-deterministischer Algorithmus kann in Abhängigkeit der (nicht-deterministisch) gewählten Operationen terminieren oder nicht terminieren

### Information 3: Semi-Thue-System und Algorithmus

Semi-Thue-Systeme stellen die einfachste Form eines Algorithmus dar. Die Regeln entsprechen hierbei den Operationen und die Meta-Regeln legen die Ablaufstruktur fest.


Ein Algorithmus terminiert nicht, wenn er für eine Eingabe eine unendliche Folge von Operationen erzeugt. Eine solche Folge kann zustande kommen, da die Ablaufstruktur eine im Prinzip beliebige, ggf. auch unendliche Anzahl von Ausführungen derselben Operation ermöglicht.

Semi-Thue-Systeme können so konstruiert werden, dass für eine gewisse Eingabe nie eine Ableitungssituation erreicht wird, die keine weitere Regelanwendung mehr zulässt. In diesem Fall stellt das Semi-Thue-System mit dieser Eingabe einen nichtterminierenden Algorithmus dar. Er liefert für diese Eingabe dann ein undefiniertes Ergebnis.

Der Sachverhalt der Nichtterminierung wird noch dadurch verschärft, dass Semi-Thue-Systeme auch aufgrund der zweiten Meta-Regel nichtdeterministisch sind. Dadurch kann es möglich sein, dass für ein und dieselbe Eingabe eine Terminierung erfolgt oder nicht erfolgt. In diesem Falle liefert das System ein nichteindeutiges Ergebnis.

### 1.3 Beispiel Kaffeedosenspiel

In Interaktion 4 wird das durch Regeln mechanisch ausführbare so genannte Kaffeedosenspiel in Form eines Semi-Thue-Systems formuliert. Gegenstand des Spiels ist eine Dose mit schwarzen und weißen Bohnen. Diese Bohnen lassen sich durch einen 2-elementigen Zeichenvorrat  $\Sigma$  beschreiben. Die Umsetzung der zwei Spielregeln in insgesamt vier Semi-Thue-Regeln erfolgt dann auf der Grundlage dieses Zeichenvorrats.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck) 

- Gegeben: Eine Dose mit beliebig angeordneten weißen und schwarzen Bohnen Formulierung als Semi-Thue-System
- Spielregeln Zeichenvorrat  $\Sigma =$  \_\_\_\_\_  
(Eingabe ist der initiale Doseninhalt)
  - Es sind fortgesetzt blind zwei Bohnen aus der Dose zu nehmen Spielregeln als Semi-Thue-Regeln
  - (1) Falls die Bohnen die gleiche Farbe haben, so ist eine schwarze Bohne in \_\_\_\_\_ die Dose zurück zu legen
  - (2) Falls die Bohnen verschiedene Farben haben, so ist nur die weiße Bohne zurück zu legen \_\_\_\_\_

Es sind zwei Spielverlauf-Beispiele bei einem Doseninhalt von vier weißen und drei schwarzen Bohnen anzugeben

---



---

#### Interaktion 4: Kaffeedosenspiel als Semi-Thue-System

Es ergeben sich für eine Eingabe – das ist bei diesem Spiel der am Anfang des Spiels bestehende Doseninhalt – ganz unterschiedliche Spielverläufe, wie am Beispiel einer Eingabe von vier weißen und drei schwarzen Bohnen in Interaktion 4 gezeigt werden soll. Die Spielverläufe gleichen sich allerdings sowohl in der Anzahl der Ableitungen (entspricht der Anzahl der Spielrunden) und im Ergebnis (der Farbe der letzten Bohne in der Dose), wie in der folgenden Interaktion 5 näher auszuführen ist.



- Das Kaffeedosenspiel terminiert, wobei immer eine Bohne in der Dose bleibt

Begründung:

---

---

- Das Ergebnis ist unabhängig vom Spielverlauf

Begründung:

---

---

### Interaktion 5: Eigenschaften des Kaffeedosenspiels

Die Terminierungseigenschaft des Kaffeedosenspiels kann man sich dadurch klarmachen, dass die stetige Abnahme der Anzahl der Bohnen in der Dose gewährleistet ist. Die Eigenschaft, dass das Ergebnis unabhängig vom Spielverlauf ist, lässt sich über eine Invariante erklären, die nach einer beliebigen Regelanwendung gültig bleibt.

## 1.4 Markov-Algorithmen

Die Eigenschaft des Nichtdeterminismus ist meist nicht wünschenswert. Die Forderung nach deterministischen Semi-Thue-Systemen führt zu den Markov-Algorithmen. Andrei Markov (russischer Mathematiker) setzte dabei nicht auf den Arbeiten von Thue auf, sondern entwickelte diese von ihm selbst als normale Algorithmen bezeichneten Textersetzungssysteme parallel zu den Semi-Thue-Systemen Anfang der 50er Jahre.

- Ein (gesteuerter) Markov-Algorithmus ist ein deterministisches Semi-Thue-System, das zur Beschreibung von Textersetzungen dient
- Bestandteile eines Markov-Algorithmus
  - endlich viele Regeln
  - Einführung einer so genannten haltenden Regel  $x \rightarrow .y$
  - Metaregeln:
    - (MM1) Wähle in jedem Schritt die erste anwendbare Regel.  
Falls sie auf mehrere Teilwörter anwendbar ist, wende sie auf das am weitesten links stehende Teilwort an
    - (MM2) Wende Regeln solange an, bis eine haltende Regel angewandt wurde, oder bis keine Regel mehr anwendbar ist
- Es werden zusätzliche Zeichen ( $\alpha, \beta, \chi, \dots$ ) verwendet, die weder im Eingabetext noch im Ergebnis vorkommen

#### **Information 4: Markov-Algorithmen**

Der Determinismus beim Markov-Algorithmus wird dadurch erreicht, dass die Regel-Anwendung gemäß der Reihenfolge der Aufschreibung erfolgt. Diese erste anwendbare Regel ist auf das am weitesten links stehende Teilwort anzuwenden.

Außerdem werden zwei verschiedene Formen von Endbedingungen vorgesehen. Neben der bereits von den Semi-Thue-Systemen bekannten Endbedingung, dass keine anwendbare Regel mehr existiert, wird außerdem eine spezielle haltende Regel eingeführt, die durch den Punkt nach dem Pfeil kenntlich gemacht ist.

Die zusätzlichen Zeichen, die in etwa den Hilfsvariablen zur Aufnahme von Zwischenergebnissen in der normalen Programmierung entsprechen, werden auch als "Schiffchen" bezeichnet. Die Assoziation ist dabei, dass die Hilfszeichen durch das Wort hindurch manövrieren und dadurch Kontextinformation zu speichern gestatten.

Die Arbeitsweise von Markov-Algorithmen soll anhand eines einfachen Beispiels verdeutlicht werden.

- Regeln
  - (1)  $\alpha L \rightarrow L\alpha$
  - (2)  $\alpha 0 \rightarrow 0\alpha$
  - (3)  $\alpha \rightarrow \beta$
  - (4)  $L\beta \rightarrow \beta 0$
  - (5)  $0\beta \rightarrow L$
  - (6)  $\beta \rightarrow L$
  - (7)  $\varepsilon \rightarrow \alpha$

- Ablauf und Ergebnis des Algorithmus für das Wort LOLL

LOLL  $\Rightarrow$   $\alpha$ LOLL  $\Rightarrow$   $L\alpha$ OLL  $\Rightarrow$   $L0\alpha$ LL  $\Rightarrow$   $L0L\alpha$ L

$\Rightarrow$   $L0LL\alpha$   $\Rightarrow$   $L0LL\beta$   $\Rightarrow$   $L0L\beta 0$   $\Rightarrow$   $L0\beta 00$   $\Rightarrow$   $LL00$

### Information 5: Beispiel eines Markov-Algorithmus

Der Beispiel-Algorithmus besteht aus sieben Ersetzungsregeln. Die Nummerierung (die hier nur zur Veranschaulichung angegeben ist und fehlen kann) macht die Priorisierung deutlich (d.h., zuerst wird versucht, Regel (1) anzuwenden, dann Regel (2) usw.).

Das Ziel ist, eine Wirkungsweise (Semantik) mit den Regeln zu verbinden, die ja ansonsten nur ein rein syntaktisches Spiel darstellen.

Die Wirkungsweise kann an dem in Information 4 angegebenen Beispiel nachvollzogen werden.

## 1.5 Formale Systeme

Semi-Thue-Systeme und deren deterministische Abwandlung in Form der Markov-Algorithmus sind Beispiele für formale Systeme. Formale Systeme beschreiben die allgemeinste Form von Relationen zwischen Gegenständen, die mit algorithmischen Methoden modelliert und realisiert werden können, wie in Information 6 näher ausgeführt wird.

- $(U, \Rightarrow)$  ist ein formales System, wenn
  - $U$  eine endliche oder abzählbar unendliche Menge ist
  - $\Rightarrow \subseteq U \times U$  eine Relation ist,
  - es einen Algorithmus gibt, der jedes  $r \in U$  mit  $l \Rightarrow r, l \in U$ , in endlich vielen Schritten aus  $l$  berechnet
    - $r$  heißt dann aus  $l$  (effektiv) berechenbar
- Bereits behandelte Beispiele formaler Systeme
  - $U$  ist eine Menge von Wörtern
  - $\Rightarrow$  Ableitungsrelation eines Semi-Thue-Systems oder eines Markov-Algorithmus
- Ein anderes Beispiel
  - $U$   $Z \times Z$  ( $Z$ : ganze Zahlen)
  - $\Rightarrow$  Addition  $(m, n) \Rightarrow (m+n, 0)$

### Information 6: Formale Systeme

Zwei Elemente  $l$  und  $r$  aus der Menge  $U$  stehen genau dann in Relation, wenn es einen Algorithmus gibt, der  $l$  als Eingabe und  $r$  als Ausgabe hat. In diesem Falle gilt, dass  $r$  aus  $l$  effektiv berechenbar ist. Offensichtlich wird deshalb durch jedes Semi-Thue-System und jeden Markov-Algorithmus ein formales System definiert.

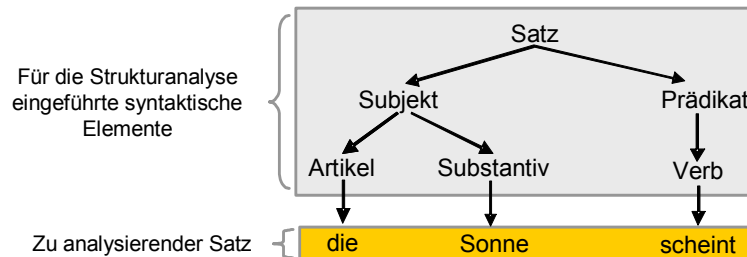
Es bestehen aber auch noch ganz andere Konstruktionsmöglichkeiten, wie das Beispiel einer Relation, die auf der Addition von zwei ganzen Zahlen basiert. Es sei angemerkt, dass in diesem Beispiel die ganzen Zahlen  $Z$  nicht gegen eine beliebige andere Zahlenmenge ausgetauscht werden darf.

## 2 GRAMMATIKEN

Die Linguisten haben in den 50er Jahren mithilfe von Semi-Thue-Systemen den Aufbau und die Struktur von natürlichen Sprachen analysiert und in Form von so genannten Chomsky-Grammatiken formal beschrieben. An erster Stelle ist hier Noam Chomsky (amerikanischer Linguist) zu nennen, nach dem auch die für diese Arbeiten erstellten Semi-Thue-Systeme, die Chomsky-Grammatiken, ernannt wurden [Go95].



- Grammatiken sind Ausprägungen von Semi-Thue-Systemen
  - Regeln heißen Produktionen
  - Ziel ist die Strukturanalyse und Beschreibung von Sprachen



- Beispiele für Produktionen

---



---

### Interaktion 6: GRAMMATIKEN - Spezielle Semi-Thue-Systeme

Das Beispiel in Interaktion 6 zeigt eine Analyse eines einfachen Satzes. Das Ergebnis ist ein Ableitungsbaum, der aus der Anwendung von gewissen Grammatikregeln hervorgeht. Die Regeln werden in den Grammatiken als Produktionen bezeichnet.

Das Ziel ist die Analyse und die Erzeugung (Produktion) von Sprachen. Den Ausgangspunkt der Analyse bilden die Bestandteile, aus denen ein Satz der (natürlichen) Sprache aufgebaut sein kann. Beispiele für solche Bestandteile, die als syntaktische Elemente bezeichnet werden, sind Satz, Subjekt, Prädikat, Artikel, Substantiv, Verb.

Aus dem obigen Ableitungsbaum sollen in Interaktion 6 einige Produktionen exemplarisch formuliert werden.

- In einer Grammatik werden zwei Arten von Zeichen unterschieden:
  1. Zeichenvorrat  $N$  der Nichtterminale
  2. Zeichenvorrat  $\Sigma$  der Terminale (Einzelzeichen der Sprache)
- $V = N \cup \Sigma$  heißt das Vokabular der Grammatik
- Ein ausgezeichnetes Element  $A$  aus  $N$  heißt das Axiom
  - bildet das Startsymbol
- Produktionen sind Regeln  $l \rightarrow r$  mit Zeichenreihen  $l, r \in V^*$

#### Information 7: Bestandteile einer Grammatik

Diese syntaktischen Elemente bilden in einer Grammatik den Zeichenvorrat der so genannten Nichtterminale (siehe Information 7). Dieser wird ergänzt um den Vorrat mit Zeichen, die konkret in den Sätzen der analysierten Sprache auftreten können. Da diese Zeichen nicht weiter in andere Zeichen überführt werden können, heißen sie Terminalzeichen. Nichtterminale und Terminale bilden gemeinsam das Vokabular der Grammatik. Ein ausgezeichnetes Nichtterminal, das so genannte Axiom bildet das Startsymbol.

Eine Produktion besteht aus einer linken Seite  $l$  und einer rechten Seite  $r$ . Ihnen liegt das allgemeine Prinzip der Textersetzung zugrunde.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck)



- Eine Grammatik  $G = (\Sigma, N, P, A)$  mit Zeichenvorrat  $\Sigma$ , Nichtterminalen  $N$ , Produktionenmenge  $P$  und Axiom  $A$  heißt eine Chomsky-Grammatik

Beispiel-Grammatik  $G_B$

$\Sigma =$  \_\_\_\_\_

$N =$  \_\_\_\_\_

$A =$  \_\_\_\_\_

- Eine Zeichenreihe  $x \in V^*$  heißt Satzform (oder Phrase), wenn sie durch endlich viele Anwendungen von Produktionen aus dem Axiom  $A$  abgeleitet werden kann

Beispiele für Satzformen

\_\_\_\_\_

\_\_\_\_\_

- $L(G)$  heißt die Sprache der Grammatik und besteht aus den terminalen Phrasen, die mittels der Produktionen von  $G$  erzeugt werden können

$L(G_B) = \{ \text{_____} \}$

### Interaktion 7: Grammatik und Sprache

In Interaktion 6 wurden die Produktionen der in Interaktion 7 als  $G_B$  bezeichneten Grammatik bereits teilweise erfasst, weshalb jetzt noch zur Vervollständigung der 4-Tupel-Beschreibung die Zeichenvorräte und das Axiom anzugeben sind.

Satzformen, die Nichtterminale beinhalten – also keine terminalen Phrasen darstellen – sind gewissermaßen die Zwischenergebnisse einer vom Axiom startenden Ableitung. Die in  $G_B$  auftretenden Satzformen und die von dieser Grammatik erzeugte Sprache sind zu ermitteln.

## 2.1 Chomsky-Hierarchie

Welche Sprache durch eine Grammatik erzeugt werden kann, hängt davon ab, welche Anforderungen an den Aufbau der Produktionen gestellt wird. Je nach Anforderung werden gemäß einer Chomsky-Hierarchie die in Information 8 aufgeführten Grammatiktypen unterscheiden.

- Chomsky-0-Grammatik (CH-0)
  - allgemeiner Produktionstyp:  $l \rightarrow r$  wobei  $l, r \in V^*$  beliebig
  - insbesondere auch  $\varepsilon$ -Produktionen ( $r = \varepsilon$ )
  
- Chomsky-1-Grammatik (CH-1)
  - längenbeschränkt:  $l \rightarrow r$  wobei  $l, r \in V^*$ ,  $1 \leq |l| \leq |r|$
  - kontextsensitiv:  $uAv \rightarrow urv$  wobei  $A \in N$ ,  $u, v \in V^*$   
 $r \in V^+$ , d.h.  $r \neq \varepsilon$
  
- Chomsky-2-Grammatik (CH-2)
  - kontextfrei:  $A \rightarrow r$  wobei  $A \in N$ ,  $r \in V^*$
  
- Chomsky-3-Grammatik (CH-3)
  - linkslinear:  $A \rightarrow Bx$  oder  $A \rightarrow x$  wobei  $A, B \in N$ ,  $x \in \Sigma$
  - rechtslinear:  $A \rightarrow xB$  oder  $A \rightarrow x$  wobei  $A, B \in N$ ,  $x \in \Sigma$

### Information 8: Chomsky-Grammatiktypen

Die Chomsky-Grammatiken werden in Abhängigkeit der erlaubten Typen von Produktionen in insgesamt vier Typen (Typ 0 bis Typ 3, CH-0 bis CH-3) eingeteilt. Dabei bilden die vier Typen eine Hierarchie in der Form, dass eine mittels einer CH-i-Grammatik erzeugte Sprache auch mittels einer CH-i-1-Grammatik erzeugt werden kann aber nicht zwingend umgekehrt. D.h., CH-3-Sprachen sind in CH-2-Sprachen enthalten, CH-2-Sprachen in CH-1-Sprachen und CH-1-Sprachen in CH-0-Sprachen.

Bei CH-0 ist das leere Wort  $\varepsilon$  auf beiden Seiten einer Regel erlaubt, d.h.

- $\varepsilon$ -Produktionen (rechte Seite  $r = \varepsilon$ , also  $l \rightarrow \varepsilon$ )
- Produktionen, die aus dem leeren Wort  $\varepsilon$  etwas produzieren, also  $\varepsilon \rightarrow r$ .

Man kann jede z.B. durch Markov-Algorithmen berechenbare Menge als Sprache einer CH-0-Grammatik erhalten, was bedeutet, dass CH-0 genauso mächtig ist wie Markov-Algorithmen (oder auch wie die nichtdeterministischen Semi-Thue-Systeme).

Für CH-1 gibt es zwei Festlegungen, die die gleiche Einschränkung bedeuten. CH-1-Grammatiken heißen wegen der zweiten Art von Produktions-Restriktion auch kontextsensitive Grammatiken, wobei die „Kontexte“ hier die beiden Zeichenreihen  $u$  und  $v$  darstellen.

Die eine Richtung, nämlich dass kontextsensitive Produktionen zugleich längenbeschränkt sind, ist offensichtlich, weil  $r \neq \varepsilon$  gefordert ist.

CH-2-Grammatiken heißen kontextfrei, weil die Produktionen ohne Vorhandensein eines Kontextes angewendet werden können.

Wie die Definition in Information 8 zeigt, gilt  $r \in V^*$ , was  $r = \varepsilon$  einschließt. Eine kontextfreie Grammatik, die auf  $\varepsilon$ -Produktionen und Produktionen der Form  $V \rightarrow V$  verzichtet (also auf linker und rechter Seite jeweils ein Nichtterminalzeichen) heißt anständig (*proper*).

Die Menge der kontextfreien Sprachen ist erheblich weniger umfangreich als die Menge der kontextsensitiven Sprachen. Allerdings kann man mit kontextfreien Grammatiken sehr viel leichter umgehen, weshalb alle gängigen Programmiersprachen, wie z.B. Java oder C, auf der Basis von CH-2-Grammatiken beschrieben sind.

Noch eingeschränkter sind die Möglichkeiten, die die CH-3-Grammatiken bieten, die auch als reguläre Grammatiken bezeichnet werden. Produktionen, die auf der rechten Seite nur ein terminales Zeichen aufweisen, heißen terminierend.

- Beispielgrammatik  $G_A = (\Sigma, N, P, A)$ 
  - $\Sigma = \{\text{bez}, +, *, (, )\}$
  - $N = \{A, T, F\}$
  - $P = \left\{ \begin{array}{l} A \rightarrow T \mid A + T, \\ T \rightarrow F \mid T * F, \\ F \rightarrow \text{bez} \mid (A) \end{array} \right\}$
  - A ist das Axiom

- Beispiel einer Ableitung einer Formel aus dem Axiom A

Ableitungsbaum zu  
 $(\text{bez} + \text{bez}) * \text{bez} + \text{bez}$

$$\begin{aligned} A &\Rightarrow A + T \Rightarrow T + T \Rightarrow T * F + T \\ &\Rightarrow F * F + T \Rightarrow (A) * F + T \\ &\Rightarrow (A + T) * F + T \Rightarrow (T + T) * F + T \\ &\Rightarrow (F + T) * F + T \Rightarrow (\text{bez} + T) * F + T \\ &\Rightarrow (\text{bez} + F) * F + T \Rightarrow (\text{bez} + \text{bez}) * F + T \\ &\Rightarrow (\text{bez} + \text{bez}) * \text{bez} + T \\ &\Rightarrow (\text{bez} + \text{bez}) * \text{bez} + F \\ &\Rightarrow (\text{bez} + \text{bez}) * \text{bez} + \text{bez} \end{aligned}$$


### Interaktion 8: Beispiel einer kontextfreien Grammatik

Die in Interaktion 8 angegebene Grammatik  $G_A$  beschreibt den Aufbau arithmetischer Ausdrücke mit den Operatoren + und \*.

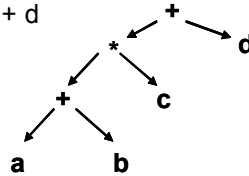
Das terminale Zeichen **bez** (Bezeichner) steht für beliebige einfache Operanden (Variablen, Konstante). Die Zeichen + und \* repräsentieren die bekannten Operatoren. Die Klammerzeichen ( und ) dienen zur Beeinflussung der Reihenfolge der Operator-Auswertung.

An einem einfachen Beispiel-Ausdruck  $(\text{bez} + \text{bez}) * \text{bez} + \text{bez}$  soll gezeigt werden, wie sich die Produktionen anwenden lassen. Die Ableitungsfolge soll in Interaktion 8 graphisch durch einen so genannten Ableitungsbaum beschrieben werden, wodurch sich die Anwendung der Produktionen sehr viel anschaulicher darstellen lässt.

Der Übergang vom Ableitungsbaum zum bereits kennen gelernten Kantorowitsch-Baum wird durch eine Eliminierung der Nichtterminale (d.h. eine Beschränkung ausschließlich auf die Terminalzeichen) erreicht.

- Ein Ableitungsbaum eines arithmetischen Ausdrucks, der auf das Wesentliche beschränkt ist, heißt Kantorowitsch-Baum
- Kantorowitsch-Baum zum vorhergehenden Beispiel

$$A \Rightarrow^* (a + b) * c + d$$



- Ableitung:  $A \Rightarrow A + A \Rightarrow A * A + A \Rightarrow (A) * A + A \Rightarrow (A + A) * A + A$   
 $\Rightarrow (\text{bez} + A) * A + A \Rightarrow (\text{bez} + \text{bez}) * A + A \Rightarrow (\text{bez} + \text{bez}) * \text{bez} + A$   
 $\Rightarrow (\text{bez} + \text{bez}) * \text{bez} + \text{bez}$
- Wie lautet die dazugehörige (vereinfachte) Grammatik?

---

### Interaktion 9: Grammatik und Kantorowitsch-Baum

Das Prinzip besteht darin, anstelle des Nichtterminals das Operatorzeichen an der Stelle im Ableitungsbaum vorzusehen. Da es zu jedem Operator (hier + und \*) immer nur genau eine Produktion gibt, in der der Operator vorkommt, ergibt sich die Produktion von selbst.

Im Kantorowitsch-Baum fehlen offensichtlich die Klammersymbole. Diese werden nicht explizit benötigt, weil sich die notwendige Klammersetzung sich aus der Struktur des Baumes und den Vorrangregeln der Operatoren ergibt.

## 2.2 Backus-Naur-Form

Zur Beschreibung von höheren Programmiersprachen wurde eine praxistaugliche Notation für kontextfreie Grammatiken gesucht. Dieses Ziel wurde von John Backus 1959 mit einer Notation verfolgt, die von Peter Naur das erste Mal zur Beschreibung von Algol 60 genutzt wurde und daher heute als Backus-Naur-Form (BNF) bezeichnet wird.

Die BNF hat eine hohe praktische Bedeutung, da sie auch heute noch zur Beschreibung der Syntax von Programmiersprachen dient. In der Kurseinheit PROGRAMMIERGRUNDLAGEN [C&M-PG] wird eine entsprechend erweiterte Form der BNF zur Beschreibung der Java-Syntax genutzt [Mö03].

- Kontextfreie Sprachen werden zur Beschreibung der Syntax von Programmiersprachen verwendet
- John Backus gab hierzu folgende Notation an (wurde von Peter Naur erstmals zur Beschreibung von Algol 60 eingesetzt):
  - Nichtterminale werden gewöhnlich durch Wörter in natürlicher Sprache wiedergegeben und in eckigen Klammern geschrieben
    - zur einfacheren rechnergestützten Verarbeitung der BNF wird auf die eckigen Klammern verzichtet und stattdessen werden die Terminale in Apostroph ".." gesetzt
  - statt  $\rightarrow$  benutzte Backus das Symbol  $::=$ 
    - heute wird statt  $::=$  häufig nur das Gleichheitszeichen = verwendet
  - Der senkrechte Strich | trennt rechte Seiten zur gleichen linken Seite von Produktionen
    - diese Konvention ist erhalten geblieben

### Information 9: Backus-Naur-Form (BNF) und Anpassungen

Wie in Information 9 ausgeführt wird, wurde die Notation zur besseren rechnergestützten Verarbeitung im Laufe der Zeit weiterentwickelt. Der wesentliche Unterschied in der Aufschreibung betrifft die Konvention, wie Terminale und Nichtterminale unterschieden werden. In der weiterentwickelten BNF-Notation werden nicht die Nichtterminale (für die Backus eckige Klammern vorgeschlagen hat), sondern die Terminale (durch Apostroph-Zeichen) gekennzeichnet.



- Die Beispiel-Grammatik mit

$$P = \{ \begin{array}{l} A \rightarrow T \mid A + T, \\ T \rightarrow F \mid T * F, \\ F \rightarrow \text{bez} \mid (A) \end{array} \}$$

in (angepasster) BNF-Notation:

Ausdruck = Term | Ausdruck "+" Term.

---



---

### Interaktion 10: BNF-Beispiel

In Interaktion 10 ist die BNF-Notation zur Aufschreibung der Produktionen der Beispiel-Grammatik zu nutzen.

Aus der Nutzung der BNF zur Beschreibung von Programmiersprachen ergab sich die Anforderung, gewisse Metazeichen einzuführen, durch die die Grammatik kompakter und verständlicher dargestellt werden konnte.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck)

- Die BNF wurde um so genannte Metazeichen zu einer EBNF-Notation erweitert
  - es ist zu beachten, dass diese Metazeichen nicht verbindlich festgelegt sind und daher unterschiedliche Notationen existieren
- Die folgende Notation wird hier verwendet
  - = trennt linke und rechte Regelseite
  - . schließt Regel ab
  - | trennt Alternativen  
Beispiel:  $x | y$  beschreibt:  $x, y$
  - () klammert Alternativen  
Beispiel:  $(x | y) z$  beschreibt:  $xz, yz$
  - [] wahlweises Vorkommen  
Beispiel:  $[x] y$  beschreibt:  $xy, y$
  - { } 0-maliges bis n-maliges Vorkommen  
Beispiel:  $\{x\} y$  beschreibt:  $y, xy, xxy, \dots$

#### **Information 10: Erweiterte Backus-Naur-Form (EBNF)**

Es bestehen verschiedene Ausprägungen von EBNFs, die die Erweiterungen unter Verwendung unterschiedlicher Metazeichen beschreiben. In Information 10 sind die in der Kurseinheit PROGRAMMIERGRUNDLAGEN [C&M-PG] ausführlich behandelten Metazeichen aufgeführt.

### **3 ENDLICHE AUTOMATEN**

Ein endlicher Automat ist ein Beispiel einer (abstrakten) Maschine. Es besteht ein enger Zusammenhang zwischen Grammatiken (bzw. den von diesem erzeugten formalen Sprachen) und bestimmten Arten von Maschinen [Go95].

- Ein endlicher Automat ist eine Art von (abstrakter) Maschine
- Es besteht ein enger Zusammenhang zwischen den durch Grammatiken erzeugten Sprachen und den Maschinen
  - Maschine soll zu einer als Eingabe anliegenden Zeichenreihe feststellen, ob diese Zeichenreihe zur Sprache gehört oder nicht
- Zu jedem Chomsky-Grammatiktyp lässt sich ein diese Sprachklasse bearbeitbare Maschinentyp angeben
  - CH-0 Turing-Maschine
  - CH-1 Linear beschränkter Automat
  - CH-2 Kellerautomat
  - CH-3 Endlicher Automat
- Eine vertiefte Behandlung des Zusammenhangs zwischen Sprachen und Maschinen ist Teil der Berechenbarkeitstheorie


### **Information 11: ENDLICHE AUTOMATEN - Einordnung**

Den Zusammenhang zwischen den Chomsky-Grammatik und den in der Informatik unterschiedenen Maschinentypen zeigt Information 11. Der in diesem Kapitel näher behandelte Endliche Automat ist dabei der "schwächste" Maschinentyp, dessen Mächtigkeit sich allerdings durch gewisse Verallgemeinerungen erhöhen lässt. Viele Systeme, die zu einer Eingabe eine definierte Ausgabe erzeugen, lassen sich mithilfe von Automaten analysieren, beschreiben und realisieren.

- Ein endlicher Automat wird in der Informatik zur Analyse, Beschreibung und Realisierung von Systemen eingesetzt
- Bestandteile und Arbeitsweise eines endlichen Automaten
  - endliche Menge  $Q$  von Zuständen mit einem Anfangszustand  $q_0 \in Q$
  - Zeichenvorrat  $\Sigma$
  - Lesen eines Zeichens  $a \in \Sigma$  führt zu einem Zustandsübergang vom aktuellen Zustand  $q \in Q$  in einen neuen Zustand  $q' \in Q$ 
    - Notation:  $qa \rightarrow q'$
    - bei gegebenem  $q$  bestimmt  $a$  den Nachfolgerzustand bzw. bei gegebenem  $a$  hängt die Wirkung  $q'$  vom bisherigen Zustand  $q$  ab
    - der Zustand lässt sich als ein (endliches) Gedächtnis über die Vorgeschichte und die bisher eingegebenen Zeichen auffassen

### **Information 12: Einsatz eines endlichen Automaten**

Das Automatenmodell kann dabei an verschiedenen Stellen erweitert werden, wodurch der Einsatzbereich dieses Modells erheblich vergrößert wird. Zum Automatenmodell gehören eine Zustandsmenge  $Q$ , ein Zeichenvorrat  $\Sigma$  und eine Menge von Zustandsübergängen.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck) 

- Zustände  $Q = \{q_0, q_1, \dots, q_n\}$  des endlichen Automaten lassen sich als Ecken eines Graphen auffassen



- Zustandsübergänge  $q_i a \rightarrow q_j$  mit  $a \in \Sigma$  entsprechen markierte gerichtete Kanten



- Ein im endlichen Automaten erreichter Zustand  $q_k$  ist durch den Anfangszustand  $q_0$  und die bisher eingegebene Zeichenreihe  $x = x_1 \dots x_i$  bestimmt

Beschreibung als markierter Graph

- Graph-Notation  
 $q_0 \Rightarrow^+ q_k$  bzw.  $q_0 \Rightarrow^* q_k$

### Interaktion 11: Endliche Automaten und Graphen

Es besteht eine enge Verbindung zwischen Automaten und Graphen. Ein endlicher Automat kann als gerichteter Graph mit den Zuständen  $Q$  als Eckenmenge und den Zustandsübergängen als markierte Kanten angesehen werden (siehe Interaktion 11). Falls mittels einer Zeichenreihe  $x = x_1 \dots x_i$  ein Übergang vom Ausgangszustand  $q_0$  in einen Zustand  $q_k$  vollzogen werden kann, so beschreibt diese Zeichenreihe den Weg von  $q_0$  nach  $q_k$  im Graphen. Zu beachten ist, dass die durchwanderten Zustände dabei nicht zwingend verschieden sein müssen. Entsprechend zu übernehmen ist die eingeführte Graph-Notation  $q_0 \Rightarrow^+ q$  bzw.  $q_0 \Rightarrow^* q$ , falls auch das leere Eingabewort  $x = \varepsilon$  zugelassen ist.

Die Automaten-Arten unterscheiden sich darin, auf welche Weise sie Ausgaben erzeugen.

- Mealy-Automat
  - Erzeugung einer Ausgabe bei jedem Zustandsübergang
  - Ausgabe ist ein Wort  $t = t_0 \dots t_{n-1}$  über einem Ausgabezeichenvorrat  $T$
  - Markieren der Kanten mit  $a / t$
- Moore-Automat
  - Erzeugung einer Ausgabe bei Erreichen eines Zustands
  - Ausgabe ist ein Wort  $t \in T^n$
- Akzeptor
  - häufigster Spezialfall eines Moore-Automaten
  - Ausgabe nicht bei allen Zuständen
  - Zustände  $F \subseteq Q$ , bei denen eine Ausgabe erfolgt, heißen Endzustände
  - Ausgegebenes Wort  $t \in T^n$  hängt vom erreichten Endzustand  $q \in F$  ab

### Information 13: Arten von Automaten

Gemäß diesem Kriterium werden in Information 13 drei Arten von Automaten unterschieden:

#### (1) Mealy-Automat

In der Notation  $a / t$  bedeutet  $a$  das Zeichen, durch das der Übergang erfolgt (Übergangszeichen) und  $t = t_0 \dots t_{n-1}$  das beim Übergang erzeugte Ausgabewort.

#### (2) Moore-Automat

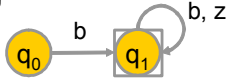
Welche Ausgabe erfolgt, hängt nicht vom Zustandsübergang, sondern vom Zustand ab. Das bedeutet, dass beim Moore-Automaten die Ausgabe unabhängig davon ist, über welchen Übergang ein Zustand erreicht wurde.

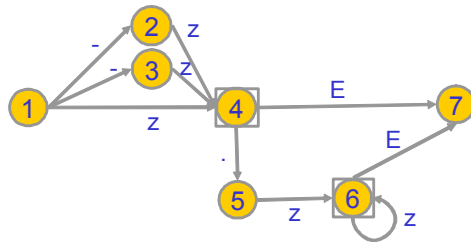
#### (3) Akzeptor

Der Akzeptor ist die in der Informatik am häufigsten zum Einsatz kommende Automaten-Art. Das Neue am Akzeptor ist die Auszeichnung von gewissen Zuständen, bei denen der Automat eine Ausgabe erzeugt. Diese Zustände heißen Endzustände.

## 3.1 Akzeptoren

Endliche Automaten werden in der Informatik u.a. im Zusammenhang mit der Übersetzung von höheren Programmiersprachen (Übersetzerbau) genutzt. Warum die endlichen Automaten bzw. die Akzeptoren hier so wichtig sind, verdeutlichen die in Interaktion 12 angegebenen zwei Beispiele.

- Erkennen von Bezeichnern einer üblichen Programmiersprache
  - Aufbau eines Bezeichners: erstes Zeichen ist ein Buchstabe gefolgt von beliebigen Buchstaben oder Ziffern
    - b: beliebiger Buchstabe aus  $\{a, \dots, z, A, \dots, Z\}$
    - z: beliebige Ziffer aus  $\{0, \dots, 9\}$
  - Übergangsgraph des Bezeichner-Akzeptors: 
- Erkennen einer Gleitkommazahl einer üblichen Programmiersprache
  - Beispiele: 3 67.34 5E2 2.58E-15
  - Zeichenvorrat  $\{z, E, ., +, -\}$ , wobei z eine beliebige Ziffer ist
  - Übergangsgraph des Gleitkommazahl-Akzeptors (zu vervollständigen)



### Interaktion 12: Beispiele für Akzeptoren

Das erste Beispiel beschreibt einen Akzeptor, durch den Bezeichner, wie sie in einer üblichen höheren Programmiersprache verwendet werden, erkannt werden.

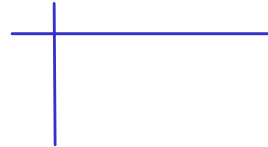
Buchstabe ist dabei ein Zeichen aus der Menge  $\{a, \dots, z, A, \dots, Z\}$  und Ziffer ist ein Zeichen aus der Menge  $\{0, \dots, 9\}$ . Es werden Platzhalter für Buchstaben in Form eines b und für Ziffern in Form eines z eingeführt. Der Übergangsgraph eines Akzeptors, der eine beliebige Zeichenfolge daraufhin überprüft, ob diese ein Bezeichner ist, besteht aus zwei Zuständen  $q_0$  und  $q_1$ , wobei  $q_1$  der Endzustand ist.

Diese Art von Akzeptor wird es in jedem Übersetzerprogramm (Compiler) einer Programmiersprache geben. Es besteht folgende zusätzliche Anforderung

- Bezeichner sind nicht nur als solche zu erkennen, sondern sind außerdem in so genannte Symboltabellen abzuspeichern.
- Der Akzeptor ist somit zugleich ein Mealy-Automat, der bekanntlich die Möglichkeit bietet, die eingegebenen Zeichen zu verarbeiten, in diesem Falle in eine Tabelle einzutragen.

Das zweite Beispiel (Gleitkommazahl-Akzeptor) zeigt, dass gemäß diesem Prinzip in einfacher und übersichtlicher Form komplexere Strukturen beschrieben werden können. Der Akzeptor verarbeitet eine beliebig aufgebaute Gleitkommazahl, wie diese üblicherweise in höheren Programmiersprachen auftreten. Der Anteil des Akzeptors, der den Exponententeil (nach dem Exponentsymbol E) folgt, ist in Interaktion 12 zu ergänzen.

- Ein Automat heißt ein vollständiger Akzeptor, wenn die Übergangsmatrix vollständig um Übergänge in einen Fehlerzustand ergänzt ist
  - Es ist die vollständige Übergangstabelle für den Bezeichner-Akzeptor anzugeben
- Ein endlicher Akzeptor lässt sich als Quintupel  $(\Sigma, Q, q_0, F, P)$  auffassen
  - $\Sigma$  : Zeichenvorrat
  - $Q$ : nichtleere endliche Zustandsmenge
  - $q_0$ : Anfangszustand aus  $Q$
  - $F$ : nichtleere Menge von Endzuständen aus  $Q$
  - $P$ : Übergänge  $q_a \rightarrow q'$  mit  $q, q' \in Q, a \in \Sigma$
- Sprache, die der Akzeptor akzeptiert:  
 $L(A) = \{x \mid x \in \Sigma^*, q_0x \Rightarrow^* q_e, q_e \in F\}$



### Interaktion 13: Festlegungen zum Akzeptor

Ein vollständiger Akzeptor ist dadurch ausgezeichnet, dass er die Fehlerfälle explizit und vollständig in der Übergangsmatrix behandelt. In Interaktion 13 wird nach der vollständigen Übergangsmatrix des Bezeichner-Akzeptors gefragt. Die wesentliche Idee, die dem vollständigen Akzeptor zugrunde liegt, besteht in der Einführung eines Fehlerzustandes und die Ergänzung sämtlicher Fehlerübergänge in diesen Zustand.

Formal kann ein endlicher Akzeptor als ein aus fünf Elementen aufgebautes System, also als Quintupel angesehen werden. Drei der fünf Elemente sind Mengen, und zwar eine Menge von (1) Zeichen, (2) Zuständen und (3) Übergängen, die wie in Grammatiken als Produktionen bezeichnet werden.

Die Endlichkeits-Eigenschaft des Akzeptors bezieht sich auf die (als endlich geforderte) Zustandsmenge.

Die zwei noch nicht erwähnten Elemente des Quintupels hängen auch unmittelbar mit der Zustandsmenge zusammen. Aus der Zustandsmenge wird ein mit  $q_0 \in Q$  bezeichneter Anfangszustand (auch Startzustand genannt) und eine Menge von Endzuständen  $F \subseteq Q$  festgelegt.

Die Sprache, die einem Akzeptor zugeordnet werden kann, hängt eng mit dem Anfangs- und Endzustand zusammen, da sie aus allen Zeichenreihen besteht, die einen Weg vom Anfangszustand zu einem Endzustand beschreiben. Formal lässt sich die Sprache wie in Interaktion 13 angegeben definieren.

Im Folgenden wird eine Eigenschaft der Übergänge bzw. Produktionen eines Automaten näher betrachtet. Die Übergänge lassen sich auffassen als eine Abbildung  $\delta: Q \times \Sigma \rightarrow Q$ .

- Bei den bisherigen Beispielen
  - die Übergänge  $q_a \rightarrow q'$  stellen eine Abbildung  $\delta: Q \times \Sigma \rightarrow Q$  dar
  - ein endlicher Automat, der die Eigenschaft besitzt, dass die Übergänge zusammen genommen eine (Übergangs-) Funktion bilden, heißt deterministisch
- Im allgemeinen Fall wird der Nichtdeterminismus zugelassen
  - die Übergänge definieren ein durch P endlich erzeugtes Semi-Thue-System mit Eingabezeichenvorrat  $\Sigma$  und syntaktischen Hilfszeichen  $q \in Q$
  - die Auffassung eines endlichen Automaten als Semi-Thue-System stellt eine Beziehung zu den regulären Chomsky-Grammatiken her

#### Information 14: Determinismus und Nichtdeterminismus

Die hier interessierende Frage ist, ob zu einem Paar (Zustand, Zeichen) nur ein oder etwa mehrere Folgezustände bestehen.

Der nichtdeterministische Fall ist der allgemeine Fall, der unmittelbar zu den Semi-Thue-Systemen führt. Bei der Auffassung eines endlichen Automaten als Semi-Thue-System spielen die Zustände die Rolle der syntaktischen Hilfszeichen. Zustandsinformation und syntaktische Hilfszeichen sind beides Formen von Kontextinformation, durch die der Berechnungsvorgang „gesteuert“ wird.

Es kann ein Übergang zwischen endlichen Automaten und Semi-Thue-Systemen konstruktiv hergestellt werden.

## 3.2 Regulärer Ausdruck

Neben den endlichen Automaten und den Chomsky-3-Grammatiken existiert eine dritte Beschreibungsform, die diese Klasse von Sprachen auszudrücken gestattet: die so genannten Regulären Ausdrücke.

- Ein regulärer Ausdruck R über einem Zeichenvorrat C ist induktiv definiert durch:
  - (1)  $c$  ist ein regulärer Ausdruck für jedes  $c \in C$
  - (2) Ist R ein regulärer Ausdruck, dann auch  $(R)^*$
  - (3) Sind R, S reguläre Ausdrücke, so sind auch  $(RS)$  und  $(R + S)$  reguläre Ausdrücke
- Beispiele
  - Bezeichner:  $b(b+z)^*$
  - Ganze Zahlen:  $zz^*$
  - Dezimalbrüche:  $zz^* \cdot / \cdot zz^*$

#### Information 15: Regulärer Ausdruck

Wie Information 15 verdeutlicht, sind die regulären Ausdrücke induktiv definiert, was heißt, dass man mit einem elementaren regulären Ausdruck beginnt und nachfolgend aus einem oder mehreren bereits bestehenden regulären Ausdrücken einen neuen regulären Ausdruck konstruiert.

Jede der oben angegebenen Konstruktionsvorschriften erweitert die Menge von Wörtern  $M(R)$ , die mit dem regulären Ausdruck  $R$  verknüpft ist.

Der Vorteil der regulären Ausdrücke besteht darin, dass sie eine kompakte und trotzdem gut lesbare Beschreibung der Sprache ermöglichen, wie die Beispiele in Information 15 zeigen.

[www.cm-tm.uka.de/info1](http://www.cm-tm.uka.de/info1)  
Info-1-Team (Prof. Abeck)

- Die im Zusammenhang mit regulären Ausdrücken definierten Operationen erfüllen verschiedene Gesetze
- Eine Menge  $L^*$  ist genau dann Sprache einer regulären Grammatik  $G$ , wenn  $L$  durch einen regulären Ausdruck beschrieben wird
- Durch die folgenden Schritte [1] bis [4] ist ein regulärer Ausdruck  $R$  in einen endlichen Akzeptor überführbar

- [1] Füge zu Beginn von  $R$  sowie nach jedem terminalen Zeichen eine Zahl ein
- [2] Einzelnes Zeichen bedeutet Zustandsübergang
- [3] Vereinigung führt zu Zustandsübergängen der zu Beginn gegebenen Zustände in alle durch Einzelzeichen erreichbaren Folgezustände
- [4] Kleenescher Stern  $S^*$  führt zu Zustandsübergängen aus sämtlichen Endzuständen von  $S$  in die aus den Anfangszuständen von  $S$  mit einem Zeichen erreichbaren Zustände

### **Information 16: Eigenschaften regulärer Ausdrücke**

Reguläre Ausdrücke besitzen die interessante Eigenschaft, dass mit ihnen „gerechnet“ werden kann. Die Grundlage hierfür wird durch entsprechende Rechenregeln geschaffen, wie sie vom Rechnen mit Zahlen her bekannt sind. Es ist zu beachten, dass die Gesetze jeweils nur für gewisse Operationen gelten. So gilt z.B. das Kommutativgesetz für die Vereinigung, nicht aber für die Konkatenation.

Der in Information 16 angegebene Satz besagt, dass reguläre Ausdrücke die gleiche Mächtigkeit wie reguläre Grammatiken (und damit auch wie endliche Automaten) haben.

In den Beweisschritten, die einen regulären Ausdruck in eine reguläre Grammatik überführen, ist ein konstruktives Vorgehen enthalten, das gewissermaßen eine „Bauanleitung“ für den endlichen Akzeptor darstellt, der die durch den regulären Ausdruck beschriebene Sprache akzeptiert. In Information 16 ist das konstruktive Vorgehen skizziert.

Das konstruktive Vorgehen lässt erkennen, dass sich die  $CH_3$ -Sprachklasse einfach und effizient durch einen Rechner verarbeiten lässt. Diese positive Eigenschaft trifft auch für die  $CH_2$ -Sprachklasse zu. Sie gilt allerdings nicht mehr für  $CH_1$  und  $CH_0$ .

Eine vertiefte Behandlung von Formalen Sprachen und deren vielfältigen Einsatzformen innerhalb der Informatik (z.B. im Übersetzerbau) erfolgt in verschiedenen weiterführenden Veranstaltungen im Rahmen des Informatik-Studiums.

## 4 VERZEICHNISSE

### Abkürzungen und Glossar

<b>Abkürzung oder Begriff</b>	<b>Langbezeichnung und/oder Begriffserklärung</b>
Axiom	Bezeichnung des in einer Grammatik ausgezeichneten Nichtterminals, das die Wurzel jedes Ableitungsbaumes bildet. Synonymer Begriff: Startsymbol
BNF	Backus-Naur-Form Notation für kontextfreie Grammatiken (CH2) zur Beschreibung von höheren Programmiersprachen.
Chomsky-Grammatik	Spezielles Semi-Thue-System, durch das der Aufbau und die Struktur von natürlichen und künstlichen Sprachen analysiert und formal beschrieben werden kann.
Chomsky-Hierarchie	Sprachklassen, die aus Bildungsgesetzen von Grammatik-Produktionen entstehen und eine hierarchische Anordnung der formalen Sprachen bilden.
Endlicher Automat	Beispiel einer (abstrakten) Maschine, durch die formale Sprachen der CH3-Sprachklasse behandelt werden können.
Formales System	Die allgemeinste Form der Beschreibung von Relationen zwischen Gegenständen, die mit algorithmischen Methoden modelliert und realisiert werden können.
Metaregel	Regel, die festlegt, in welcher Reihenfolge und an welcher Stelle eine Menge von (Textersetzungs-) Regeln anzuwenden sind.
Nichtdeterminiertheit	Eigenschaft des Algorithmus-Ergebnisses, das für eine Eingabe unterschiedlich sein kann (d.h., das Ergebnis ist nicht vorhersagbar).
Nichtdeterminismus	Eigenschaft des Algorithmus-Verhaltens, das für eine Eingabe unterschiedlich sein (d.h., das Verhalten ist nicht vorhersagbar).
Nichtterminierung	Eigenschaft eines Algorithmus, der für eine Eingabe endlos läuft und somit keine (d.h., eine undefinierte) Ausgabe liefert.
Reguläre Ausdrücke	In der Praxis häufig genutzte Form zur Beschreibung einer zur CH3-Sprachklasse gehörenden regulären Sprache.
Semi-Thue-System	System, das einen aus einem vorgegebenen endlichen Zeichenvorrat gebildeten Eingabetext mittels (endlicher oder abzählbar unendlicher) Textersetzungsregeln in einen Ausgabe Text überführt. Semi-Thue-Systeme arbeiten nichtdeterministisch. Synonymer Begriff: Textersetzungssystem Englischer Begriff: <i>String Replacement System, String Rewrite System</i>

## Index

Axiom	12	Metaregeln	3
Backus-Naur-Form	15	Nichtdeterminiertheit	4
Chomsky-Grammatiken	10	Nichtdeterminismus	4
Chomsky-Hierarchie	12	Nichtterminierung	6
endlicher Automat	17	Semi-Thue-Systeme	2
Formale Systeme	9		

## Informationen und Interaktionen

Information 1: FORMALE SPRACHEN	2
Information 2: Beispiel eines Semi-Thue-Systems	3
Information 3: Semi-Thue-System und Algorithmus	5
Information 4: Markov-Algorithmen	8
Information 5: Beispiel eines Markov-Algorithmus	9
Information 6: Formale Systeme	10
Information 7: Bestandteile einer Grammatik	11
Information 8: Chomsky-Grammatiktypen	13
Information 9: Backus-Naur-Form (BNF) und Anpassungen	16
Information 10: Erweiterte Backus-Naur-Form (EBNF)	17
Information 11: ENDLICHE AUTOMATEN - Einordnung	18
Information 12: Einsatz eines endlichen Automaten	18
Information 13: Arten von Automaten	20
Information 14: Determinismus und Nichtdeterminismus	23
Information 15: Regulärer Ausdruck	23
Information 16: Eigenschaften regulärer Ausdrücke	24
Interaktion 1: SEMI-THUE-SYSTEME – Prinzip	2
Interaktion 2: Metaregeln eines Semi-Thue-Systems	4
Interaktion 3: Semi-Thue-System und Formale Sprache	5
Interaktion 4: Kaffeedosenspiel als Semi-Thue-System	6
Interaktion 5: Eigenschaften des Kaffeedosenspiels	7
Interaktion 6: GRAMMATIKEN - Spezielle Semi-Thue-Systeme	11
Interaktion 7: Grammatik und Sprache	12
Interaktion 8: Beispiel einer kontextfreien Grammatik	14
Interaktion 9: Grammatik und Kantorowitsch-Baum	15
Interaktion 10: BNF-Beispiel	16
Interaktion 11: Endliche Automaten und Graphen	19
Interaktion 12: Beispiele für Akzeptoren	21
Interaktion 13: Festlegungen zum Akzeptor	22

## Literatur

- [C&M-PG] Cooperation&Management: PROGRAMMIERGRUNDLAGEN, Kursdokument zur Vorlesung "INFORMATIK I", <http://www.cm-tm.uka.de/info1>, Universität Karlsruhe (TH), C&M (Prof. Abeck).
- [Go95] Gerhard Goos: Vorlesungen über Informatik, Band 1: Grundlagen und funktionales Programmieren, Springer Verlag 1995.

- [Mö03] Hanspeter Mössenböck: Sprechen Sie Java? – Eine Einführung in das systematische Programmieren, dpunkt.verlag 2003.
- [Sa04] Gunter Saake, Kai-Uwe Sattler: Algorithmen und Datenstrukturen – Eine Einführung mit Java, dpunkt.verlag, 2004.