Universität Trier

Fachbereich IV - Informatik

Christoph Meinel:

Automatentheorie und Formale Sprachen

Vorlesungsskript SS 2002
September 2001

Inhaltsverzeichnis

0 Vorbemerkungen **2**
0.1 Wörter, Alphabete, Sprachen . . . 2
0.2 Graphen und Bäume . . . 3
0.3 Relationen . . . 5
0.4 Mengen . . . 6
0.5 Komplexitätsfunktionen . . . 7

1 Grammatiken **8**
1.1 Beispiele . . . 8
1.2 Definitionen . . . 9
1.3 Chomsky - Hierarchie . . . 12
1.4 Wortprobleme . . . 14
1.5 Syntaxbäume . . . 16
1.6 Backus-Naur-Form . . . 19

2 Reguläre Sprachen **21**
2.1 Deterministische und Nichtdeterministische endliche Automaten . . . 21
2.2 Reguläre Ausdrücke . . . 26
2.3 Äquivalenzrelationen und Minimalautomaten . . . 29
2.4 Periodizitätseigenschaften . . . 34
2.5 Abschlußeigenschaften . . . 35
2.6 Algorithmische Eigenschaften . . . 36

3 Kontextfreie Sprachen **38**
3.1 Normalformen . . . 38
3.2 Pumping Lemma . . . 40
3.3 Abschlußeigenschaften . . . 43
3.4 Algorithmische Eigenschaften . . . 44
3.5 Pushdown Automaten . . . 47
3.6 Deterministisch kontextfreie Sprachen . . . 52

4 Kontextsensitive und Typ 0–Sprachen **55**

0 Vorbemerkungen

Literaturhinweis: Uwe Schöning: Theoretische Informatik – kurz gefasst. BI, 1992.

0.1 Wörter, Alphabete, Sprachen

- $I\!N = \{0, 1, 2, \ldots\}$: natürliche Zahlen
- Σ: endliche Menge (Alphabet) von Symbolen (Buchstaben)
- w: endliche Folge von Buchstaben (Wort)
- $|w|$: Zahl der Buchstaben von w (Länge des Wortes)
- $\Box$: Wort, das aus der Buchstabenfolge der Länge 0 besteht (sog. *leeres Wort*)
  $\Rightarrow |\Box| = 0$
- Präfix: beliebiges Anfangsstück eines Wortes
- Suffix: beliebiges Endstück eines Wortes
- Konkatenation: Bildung eines Wortes durch Hintereinanderschreiben zweier Wörter
- L (formale) Sprache: Menge von Wörtern über einem Alphabet
- Für endliches Alphabet Σ:

$$\begin{array}{lcl} \Sigma^0 & := & \{\Box\} \\ \Sigma^{i+1} & := & \{w \mid w = w'a \text{ mit } w' \in \Sigma^i,\, a \in \Sigma\} \quad (i \geq 0) \\ \Sigma^* & := & \bigcup_{i \geq 0} \Sigma^i \\ \Sigma^+ & := & \bigcup_{i > 0} \Sigma^i = \Sigma^* - \{\Box\} \end{array}$$

- Wir bezeichnen Konkatenation durch $\circ$:

  $(\Sigma^*, \circ)$ ist ein Monoid (Halbgruppe mit neutralem Element), d.h. es gilt:

$$\begin{array}{rcl} u, v \in \Sigma^* & \Rightarrow & u \circ v \in \Sigma^* \\ u, v, w \in \Sigma^* & \Rightarrow & (u \circ v) \circ w = u \circ (v \circ w) \\ u \in \Sigma^* & \Rightarrow & u \circ \Box = \Box \circ u = u \end{array}$$

- Da Konkatenation assoziativ ist, können wir w^n definieren als:

$$\begin{array}{rcl} w^n & = & \underbrace{w \ldots w}_{n-mal} \quad (n \geq 1) \\ w^0 & = & \Box \end{array}$$

- Es gilt

$$\begin{array}{rcl} |uv| & = & |u| + |v| \\ |\Box| & = & 0 \\ |w^n| & = & n \cdot |w| \end{array}$$

- $L \subseteq \Sigma^*$ heißt (formale) Sprache über Σ.

Beispiele $\emptyset$, Σ^*, $\{w_1 \dots w_n \mid w_i \in \Sigma, w_i = w_{n-i+1}$ für alle $1 \leq i \leq n\}$

- Sei $L, L' \subseteq \Sigma^*$. Definiere

$$\begin{aligned} L \cup L' &:= \{w \mid w \in L \text{ oder } w \in L'\} \\ L \cap L' &:= \{w \mid w \in L \text{ und } w \in L'\} \\ \overline{L} &:= \Sigma^* - L \\ LL' &:= \{uv \mid u \in L \text{ und } v \in L'\} \end{aligned}$$

- Sei $\mathcal{K}$ eine Klasse von Sprachen:

  Definition: $\mathcal{K}$ heißt abgeschlossen gegenüber Vereinigung (Durchschnitt, Komplement, Produkt), wenn aus $L, L' \in \mathcal{K}$ folgt, dass auch $L \cup L' \in \mathcal{K}$ ($L \cap L' \in \mathcal{K}$, $\overline{L} \in \mathcal{K}$, $LL' \in \mathcal{K}$).

- Es gilt

$$\begin{aligned} L \cup L' &= \overline{\overline{L} \cap \overline{L'}} \\ L \cap L' &= \overline{\overline{L} \cup \overline{L'}} \end{aligned}$$

- Sei $L \subseteq \Sigma^*$. Wir definieren

$$\begin{aligned} L^0 &:= \{\Box\} \\ L^{n+1} &:= L^n L \quad (n \geq 0) \end{aligned}$$

  Insbesondere gilt:

$$\begin{aligned} L^1 &= L \\ L^i L^j &= L^{i+j} \\ (L^i)^j &= L^{ij} \\ L^* &= \bigcup\nolimits_{n \geq 0} L^n \\ L^+ &= \bigcup\nolimits_{n > 0} L^n \end{aligned}$$

Beispiele $\{a^{2n} \mid n \geq 1\} = \{a^2\}^+$
$\{a^{7n+3} \mid n \geq 1\} = \{a^7\}^* \{a^3\}$

0.2 Graphen und Bäume

- $G = (V, E)$ heißt „gerichteter Graph", falls V eine Menge von „Knoten" (i.A. endlich) und $E \subseteq V \times V$ eine Menge von „Kanten" ist.

- p heißt „Weg" in G, falls p eine Folge von Knoten $v_1, \dots, v_k$ ist mit $(v_i, v_{i+1}) \in E$ für alle $1 \leq i \leq k$.

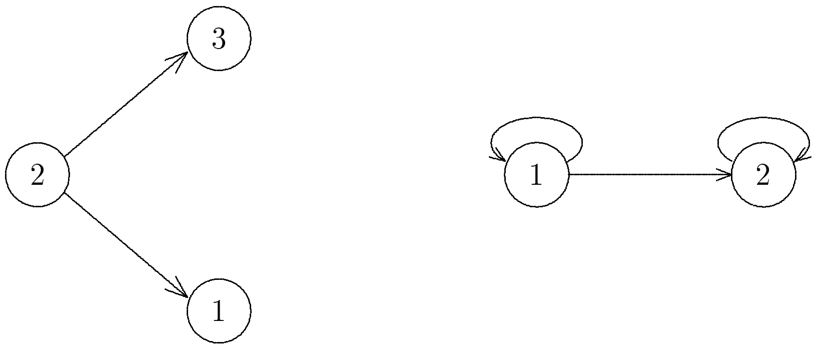
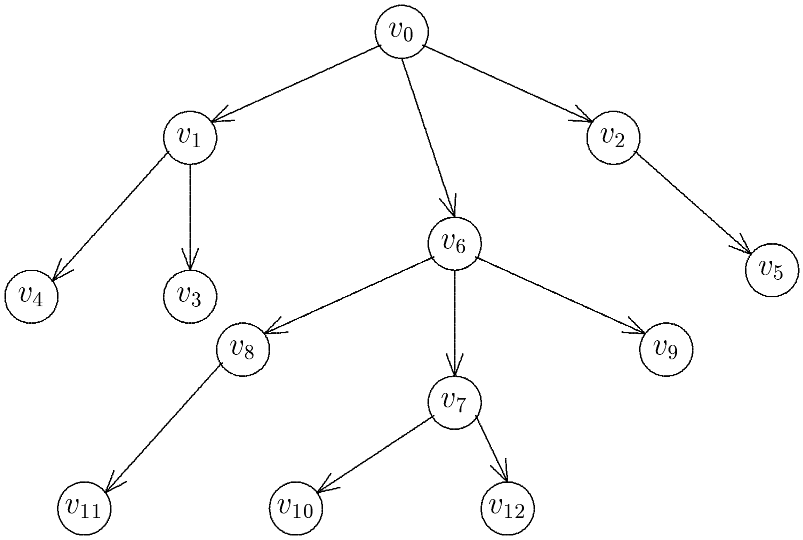


Abbildung 1: Beispiele für Graphen

- Die „Länge“ von p ist die Zahl der Kanten in p.
- Ein „Zyklus“ ist ein Weg $p = (v_1, \ldots, v_k)$ mit $v_1 = v_k$
- $G = (V, E)$ heißt „ungerichteter Graph“, falls mit $(v, w) \in E$ stets auch $(w, v) \in E$ ist.
- $T = (V, E)$ heißt „Baum“, falls gilt:

  T ist ein gerichteter Graph ohne Zyklen, in dem jeder Knoten v von einem ausgezeichneten Knoten v_0 (der „Wurzel“) aus auf genau einem Weg erreichbar ist.



- Ist $v_1, \ldots, v_k$ ein Weg in einem Baum T, so heißt
  - v_{i-1} „Vater“ von v_i (Vorgänger)
  - v_i „Sohn“ von v_{i-1} (Nachfolger)
  - v_k „Blatt“, falls v_k keine Söhne besitzt

0.3 Relationen

Sei A eine Menge.
R heißt (binäre) Relation (über A), falls $R \subseteq A \times A$.

R ist <u>reflexiv</u>	–	aRa für alle $a \in A$
R ist <u>irreflexiv</u>	–	aRa für kein $a \in A$
R ist <u>transitiv</u>	–	aRb und bRc implizieren aRc
R ist <u>symmetrisch</u>	–	aRb impliziert bRa
R ist <u>antisymmetrisch</u>	–	für keine $a, b \in R$ gilt aRb und bRa
R ist <u>Äquivalenzrelation</u>	–	R ist reflexiv, symmetrisch und transitiv.

- Äquivalenzrelationen R partitionieren eine Menge in paarweise disjunkte Äquivalenzklassen („Klasseneinteilung“).
  Index(R) := Anzahl der Äquivalenzklassen.

- Sei P eine Eigenschaft von Relationen, R eine Relation über A. Der <u>P-Abschluß</u> von R ist die kleinste Relation R' mit: R' hat die Eigenschaft P und $R \subseteq R'$.

- Seien R, S Relationen über A.
  $RS = \{(x, y) \mid \text{es existiert } z \text{ mit } xRz \text{ und } zSy\}$

- Sei $R \subseteq \Sigma^* \times \Sigma^*$. Wir definieren

$$\begin{aligned} R^0 &:= \{(w, w) \mid w \in \Sigma^*\} \\ R^{n+1} &:= R^n R \\ R^* &:= \textstyle\bigcup_{n \geq 0} R^n \\ R^+ &:= \textstyle\bigcup_{n > 0} R^n \end{aligned}$$

  <u>Beispiel</u>
  xR^*y, falls $x = y$ oder
  es existiert eine Folge $z_1 \ldots z_n$ mit xRz_1, z_1Rz_2, …, z_nRy.

Satz
Es gilt

1. R^* ist kleinste reflexive und transitive Relation mit $R \subseteq R^*$ (reflexiver und transitive Hülle)

2. R^+ ist kleinste transitive Relation mit $R \subseteq R^+$ (transitive Hülle)

<u>Beweis:</u>
(1) $R \subseteq R^*$ wegen $R = R^1 \subseteq \bigcup_{i \geq 0} R^i = R^*$
R^* ist reflexiv wegen $R^0 \subseteq R^*$
R^* ist transitiv wegen xR^*y, yR^*z
$\Rightarrow$ es existieren i, j mit xR^iy und yR^jz
$\Rightarrow xR^{i+j}z$
$\Rightarrow xR^*z$
noch zu zeigen: $[R'$ reflexiv, transitiv und $R \subseteq R'] \Rightarrow R^* \subseteq R'$
Sei R' reflexiv ($\Rightarrow R^0 \subseteq R'$), transitiv und $R \subseteq R'$ ($\Rightarrow R = R^1 \subseteq R'$)
Dann gilt: $R^2, R^3, \ldots \subseteq R'$, da R' transitiv ist, und folglich $R^* \subseteq R'$
(2) analog. ∎

0.4 Mengen

Wir können Mengen beschreiben durch:

- Aufzählung
- Bildungsgesetz (definierende Eigenschaften)

Seien A,B Mengen.

$A \subseteq B$	falls	$x \in A \Rightarrow x \in B$
$A = B$	falls	$x \in A \Rightarrow x \in B$ und $x \in B \Rightarrow x \in A$
$A \cup B$	$=$	$\{x \mid x \in A$ oder $x \in B\}$ (Vereinigung)
$A \cap B$	$=$	$\{x \mid x \in A$ und $x \in B\}$ (Durchschnitt)
$A - B$	$=$	$\{x \mid x \in A$ und $x \notin B\}$ (Differenz)
$A \times B$	$=$	$\{(a, b) \mid a \in A$ und $b \in B\}$ (kartesisches Produkt)
2^A	$=$	$\{A' \mid A' \subseteq A\}$ (Potenzmenge)
A endlich	:	$\#A :=$ Anzahl der Elemente von A

Definition

A, B heißen <u>gleichmächtig</u>	$=_{def}$	es existiert eine Bijektion $\alpha : A \to B$
A heißt <u>abzählbar unendlich</u>	$=_{def}$	A ist gleichmächtig mit $I\!N$
A heißt <u>abzählbar</u>	$=_{def}$	A ist endlich oder abzählbar unendlich

Bemerkung
Es gilt

1. A abzählbar $\iff$ es existiert eine Surjektion $\alpha : I\!N \to A$.
   Es gilt dann $A = \{\alpha(0), \alpha(1), \ldots\}$ („Abzählung" von A).
2. Für jedes endliche Alphabet Σ gilt, dass Σ^* abzählbar ist. Damit ist auch jede Sprache $L \subseteq \Sigma^*$ abzählbar.
3. 2^{Σ^*}: Menge aller Sprachen über Σ (Potenzmenge von Σ^*).
4. Seien A, B endlich. Dann gilt
$$\begin{aligned} \#(A \times B) &= \#A \cdot \#B \\ \#(A \cup B) &\leq \#A + \#B \\ \#2^A &= 2^{\#A} \end{aligned}$$
5. Seien A, B abzählbar unendlich. Dann gilt
   - $A \times B$ ist abzählbar unendlich
   - $A \cup B$ ist abzählbar unendlich
   - $A \cap B$ ist abzählbar

aber: 2^A ist nicht abzählbar.

Beweis: Sei $A = \{\alpha(0), \alpha(1), \ldots\}$ eine Abzählung von A.
Annahme: 2^A ist abzählbar
$\Rightarrow$ es existiert eine Abzählung β von 2^A, d.h. $2^A = \{\beta(0), \beta(1), \ldots\}$.
Betrachte $D = \{\alpha(j) \mid \alpha(j) \notin \beta(j)\}$ („Diagonalmenge“).
Wegen $D \subseteq A$ gilt $D \in 2^A$
$\Rightarrow$ es existiert ein n mit $D = \beta(n)$, aber

$$\begin{array}{ll} & \alpha(n) \in D \\ \iff & \alpha(n) \notin \beta(n) \\ \iff & \alpha(n) \notin D \qquad \text{Widerspruch!} \end{array}$$

■

0.5 Komplexitätsfunktionen

Wir sind lediglich an asymptotischen Beschreibungen von Komplexitätsfunktionen interessiert.

Sei $f : \mathbb{N} \longrightarrow \mathbb{N}$. Dann definieren wir

$$O(f(n)) = \{g : \mathbb{N} \longrightarrow \mathbb{N} \mid \text{es gibt Konstanten } c, n_0 \text{ mit } \; g(n) \leq c \cdot f(n) \text{ für alle } n \geq n_0\}$$

Beispiel: $2n^7 + 16n^2 + log(n) = O(n^7)$
(Eigentlich müßte es heißen: $\ldots \in O(n^7)$, aber die oben angegebene Schreibweise ist allgemein üblich.)

1 Grammatiken

1.1 Beispiele

1. Beispiel einer Sprache:
   Es sei $\Sigma = \{(,),+,-,*,/,a\}$.
   Wir betrachten

   EXPR = „Sprache der korrekt geklammerten arithmetischen Ausdrücke“

   Dabei soll a ein Platzhalter für Konstanten oder Variablen sein.

   $$\begin{array}{rcl} (a-a)/a+a*(a+(a+a))+a & \in & \text{EXPR} \\ (((a))) & \in & \text{EXPR} \\ (\,)a) & \notin & \text{EXPR} \end{array}$$

   EXPR ist ein unendliches Objekt. Wir suchen nach einer endlichen Beschreibung:

   - Grammatiken (Sammlung von „Produktionsregeln“)
   - Automaten (Objekt zur Erkennung von Sprachen)

2. Beispiel einer Grammatik:

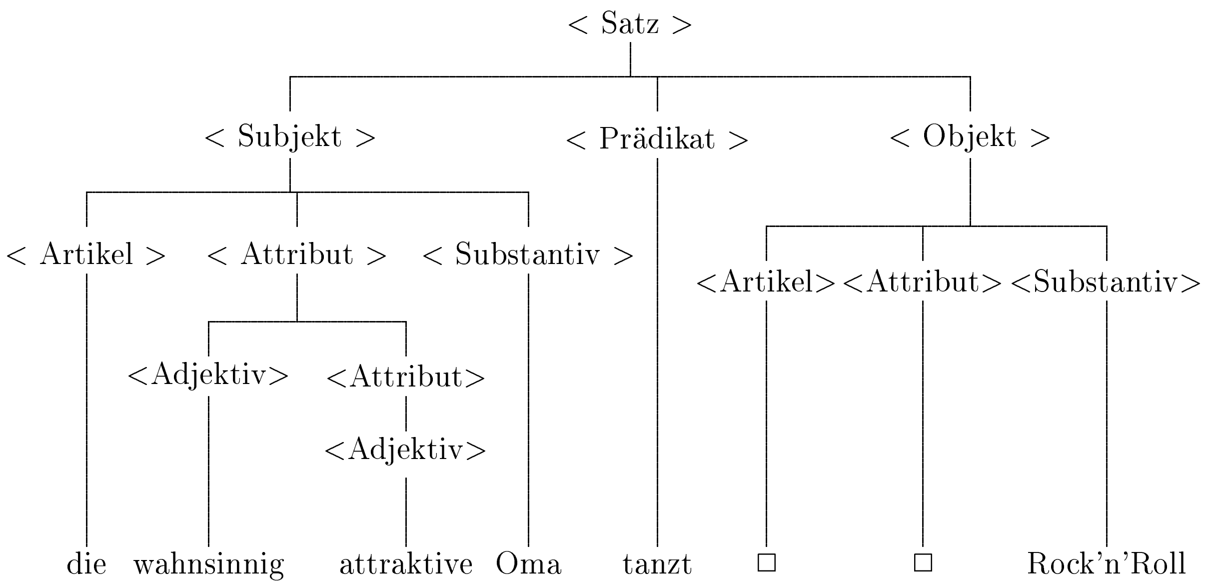
   $$\begin{array}{lcl} <\text{Satz}> & \longrightarrow & <\text{Subjekt}><\text{Prädikat}><\text{Objekt}> \\ <\text{Subjekt}> & \longrightarrow & <\text{Artikel}><\text{Attribut}><\text{Substantiv}> \\ <\text{Artikel}> & \longrightarrow & \Box \\ <\text{Artikel}> & \longrightarrow & \text{der} \\ <\text{Artikel}> & \longrightarrow & \text{die} \\ <\text{Artikel}> & \longrightarrow & \text{das} \\ <\text{Attribut}> & \longrightarrow & \Box \\ <\text{Attribut}> & \longrightarrow & <\text{Adjektiv}> \\ <\text{Attribut}> & \longrightarrow & <\text{Adjektiv}><\text{Attribut}> \\ <\text{Adjektiv}> & \longrightarrow & \text{debile} \\ <\text{Adjektiv}> & \longrightarrow & \text{wahnsinnig} \\ <\text{Adjektiv}> & \longrightarrow & \text{attraktiv} \\ <\text{Substantiv}> & \longrightarrow & \text{Oma} \\ <\text{Substantiv}> & \longrightarrow & \text{Rock 'n' Roll} \\ <\text{Prädikat}> & \longrightarrow & \text{tanzt} \\ <\text{Objekt}> & \longrightarrow & <\text{Artikel}><\text{Attribut}><\text{Substantiv}> \end{array}$$

   Wir kennzeichnen Platzhalter (Variablen) durch spitze Klammern.

   So können wir den folgenden Satz ableiten:

   „die wahnsinnig attraktive Oma tanzt Rock ’n’ Roll“

Illustration der Ableitung mit Hilfe eines Syntaxbaumes:



Wir können mit der endlichen Grammatik eine unendliche Sprache beschreiben:

“die wahnsinnig debile debile ... debile attraktive Oma tanzt Rock 'n' Roll“

1.2 Definitionen

Wir arbeiten mit
- Startsymbol
- Variablen (Platzhalter)
- eigentlichen Symbolen (Terminalsymbole)
- Produktionsregeln: linke Seite $\longrightarrow$ rechte Seite

Definition
Eine Grammatik ist ein 4–Tupel $G = (V, \Sigma, P, S)$ mit:

- V ist eine endliche Menge von Variablen.
- Σ ist eine endliche Menge, das Terminalalphabet, wobei $V \cap \Sigma = \emptyset$.
- P ist eine endliche Menge von Regeln oder Produktionen
  $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$
- $S \in V$ ist das Startsymbol

Bezeichnungen Sei $G = (V, \Sigma, P, S)$.

- Wir schreiben $x \longrightarrow y$, falls $(x, y) \in P$.
- Seien $u, v \in (V \cup \Sigma)^*$.

$$\begin{array}{lll} u \Rightarrow_G v & \text{falls} & u = xyz,\ v = xy'z \\ & \text{mit} & y \in (V \cup \Sigma)^+, x, y', z \in (V \cup \Sigma)^* \text{ und } y \longrightarrow y' \in P \end{array}$$

- Die Folge $(w_0, \ldots, w_n)$, $w_i \in (V \cup \Sigma)^*$, $w_0 = S$, $w_n \in \Sigma^*$ heißt <u>Ableitung von w_n in G</u>, falls $w_0 \;\Rightarrow_G\; w_1 \;\Rightarrow_G\; \ldots \;\Rightarrow_G\; w_n$.
- $L(G) = \{w \in \Sigma^* \mid S \;\Rightarrow_G^*\; w\}$,
  wobei $\Rightarrow_G^*$ die reflexive und transitive Hülle von $\Rightarrow_G$ bezeichnet.

<u>Beispiele</u>

1. Sei $G = (\{E, T, F\}, \{(,), +, -, *, /, a\}, P, E)$ mit

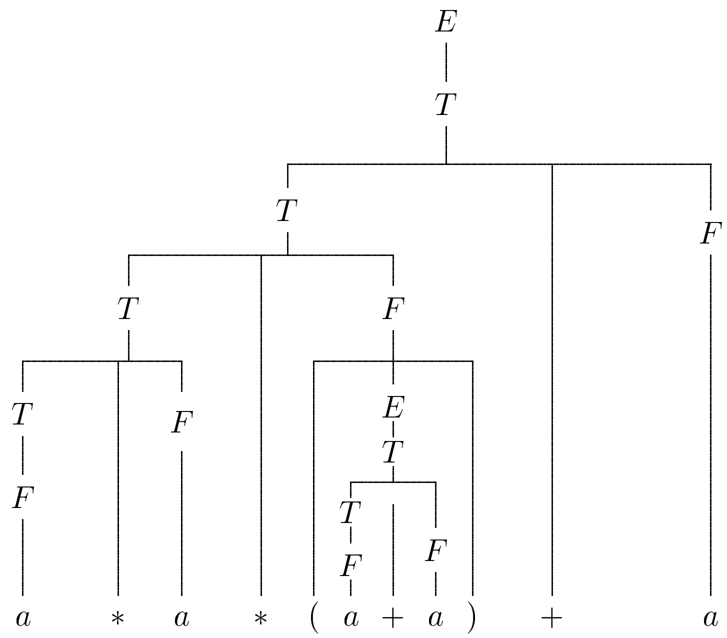
$$
\begin{array}{lccl}
P = \{ & E & \longrightarrow & T \\
 & T & \longrightarrow & T + F \\
 & T & \longrightarrow & T - F \\
 & T & \longrightarrow & T * F \\
 & T & \longrightarrow & T \;/\; F \\
 & T & \longrightarrow & F \\
 & F & \longrightarrow & a \\
 & F & \longrightarrow & (E)\ \}
\end{array}
$$

Diese Grammatik erzeugt korrekt geklammerte arithmetische Ausdrücke:

z.B.: $a * a * (a + a) + a \;\in\; L(G)$, da:

$$
\begin{array}{lll}
E & \Longrightarrow_G & T + F \\
 & \Longrightarrow_G & T + a \\
 & \Longrightarrow_G & T * F + a \\
 & \Longrightarrow_G & T * (E) + a \\
 & \Longrightarrow_G & T * (T) + a \\
 & \Longrightarrow_G & T * (T + F) + a \\
 & \Longrightarrow_G & T * (T + a) + a \\
 & \Longrightarrow_G & T * (F + a) + a \\
 & \Longrightarrow_G & T * (a + a) + a \\
 & \Longrightarrow_G & T * F * (a + a) + a \\
 & \Longrightarrow_G & T * a * (a + a) + a \\
 & \Longrightarrow_G & F * a * (a + a) + a \\
 & \Longrightarrow_G & a * a * (a + a) + a \quad \in L(G)
\end{array}
$$

Wir haben immer die am weitesten rechts stehende Variable ersetzt („Rechtsableitung“).



Achtung: Der Ableitungsprozeß ist nicht eindeutig („nichtdeterministisch“), denn $\Rightarrow_G$ ist im allgemeinen keine Funktion, sondern eine Relation:

- Zu einem x können (endlich) viele x' existieren mit $x \Rightarrow_G x'$
- Zu einem x' können (endlich) viele x existieren mit $x \Rightarrow_G x'$

2. Sei $G = (V, \Sigma, P, S)$ mit

$$\begin{array}{lcl} V & = & \{S, B, C\} \\ \Sigma & = & \{a, b, c\} \\ P & = & \{\ S \longrightarrow aSBC, \quad S \longrightarrow aBC, \quad CB \longrightarrow BC, \\ & & \quad aB \longrightarrow ab, \quad bB \longrightarrow bb, \quad bC \longrightarrow bc, \quad cC \longrightarrow cc\ \} \end{array}$$

Beispiel einer Ableitung:

$$\begin{array}{lll} S & \Longrightarrow_G & aSBC \Longrightarrow aaSBCBC \Longrightarrow aaaBCBCBC \\ & \Longrightarrow_G & aaaBBCCBC \Longrightarrow aaaBBCBCC \Longrightarrow aaaBBBCCC \\ & \Longrightarrow_G & aaabBBCCC \Longrightarrow aaabbBCCC \Longrightarrow aaabbbCCC \\ & \Longrightarrow_G & aaabbbcCC \Longrightarrow aaabbbccC \Longrightarrow aaabbbccc = a^3b^3c^3 \end{array}$$

Satz
$L(G) = \{a^n b^n c^n \mid n \geq 1\}$

Beweis:
$\supseteq$: analog wie im Beispiel.

$\subseteq$:

1. Alle Regeln halten „Balance“ zwischen (a, A), (b, B) und (c, C)
$\Longrightarrow$ für alle $w \in L(G)$ gilt: Anzahl der a’s = Anzahl der b’s = Anzahl der c’s.

2. Sei $w \in L(G)$. Wir zeigen, dass die a's in w vor den b's und diese vor den c's kommen müssen:
   a's können nur durch die Regeln (1) und (2) erzeugt werden und stehen damit jeweils ganz links, also können a's nur am Wortanfang vorkommen.
   b's und c's: Terminalworte können nur durch die Regeln (4) – (7) aufgebaut werden. Diese Regeln sind so aufgebaut, dass a's vor b's kommen und b's vor c's. ■

Bemerkung
Wir geben in Zukunft nur noch die Regeln einer Grammatik an. Dabei bedeuten große Buchstaben oder Wörter in spitzen Klammern jeweils Variablen. Terminalsymbole werden durch kleine Buchstaben gekennzeichnet. Ist aus dem Zusammenhang klar, welche Grammatik G benutzt wird, so verwenden wir $\Longrightarrow$ anstelle von $\Longrightarrow_G$.

1.3 Chomsky - Hierarchie

Einteilung der Grammatiken in Typen 0 – 3 nach Chomsky:

Definition

- Jede Grammatik ist vom Typ 0 („Phrasenstrukturgrammatik").
- Eine Grammatik ist vom Typ 1 („kontextsensitiv"), falls für alle Regeln $w_1 \longrightarrow w_2 \in P$ gilt: $|w_1| \leq |w_2|$
- Eine Typ 1–Grammatik ist vom Typ 2 („kontextfrei"), falls für alle Regeln $w_1 \longrightarrow w_2 \in P$ gilt: $w_1 \in V$
- Eine Typ 2–Grammatik ist vom Typ 3 („regulär"), falls für alle Regeln $w_1 \longrightarrow w_2 \in P$ gilt: $w_2 \in \Sigma \cup \Sigma V$
- $L \subseteq \Sigma^*$ heißt vom Typ 0 (Typ 1, Typ 2, Typ 3), falls eine Typ 0(Typ 1, Typ 2, Typ 3)–Grammatik G existiert mit $L(G) = L$.

<u>Beispiele</u>

1. Beispiel (arithmetische Ausdrücke) ist eine Grammatik vom Typ 2 (kontextfrei).
2. Beispiel (Grammatik für $\{a^n b^n c^n \mid n \geq 1\}$) ist eine Grammatik vom Typ 1 (kontextsensitiv).

<u>$\Box$ – Sonderregel:</u>

Da $|w_1| \leq |w_2|$, kann $\Box$ in keiner Grammatik vom Typ 1, 2, 3 abgeleitet werden. Das wollen wir durch die folgende Sonderregel ändern:

$S \longrightarrow \Box$ ist erlaubt, falls S auf keiner rechten Seite einer Regel vorkommt.

Lemma (Empty String Lemma)
Sei $G = (V, \Sigma, P, S)$ eine Typ 1–Grammatik. Dann existiert eine Typ 1–Grammatik G' mit $L(G) = L(G')$, und S kommt auf keiner rechten Seite vor.

<u>Beweis</u>
Sei $S' \notin V \cup \Sigma$.

$$\begin{array}{lcl} G' & := & (V \cup \{S'\}, \Sigma, P', S') \\ P' & := & P \cup \{S' \longrightarrow \alpha \mid S \longrightarrow \alpha \in P\} \end{array}$$

<u>Beh.:</u> $L(G) = L(G')$

$\subseteq$: Sei $S \Rightarrow_G^* w$ mit erster Regel $S \longrightarrow \alpha$
$\Longrightarrow \quad S \Rightarrow_G \alpha \Rightarrow_G^* w$
Nach Definition gilt: $S' \Rightarrow_{G'} \alpha$
Wegen $P \subseteq P'$ gilt $\alpha \Rightarrow_{G'}^* w$, also $S' \Rightarrow_{G'}^* w$

$\supseteq$: Sei $S' \Rightarrow_{G'}^* w$ mit erster Regel $S' \longrightarrow \alpha$
$\Longrightarrow \quad S \longrightarrow \alpha \in P \quad \Longrightarrow \quad S \Rightarrow_G \alpha$
Wegen $\alpha \Rightarrow_{G'}^* w$ und da α kein S' enthält, folgt:
$\alpha \Rightarrow_G^* w$, also $S \Rightarrow_G^* w$ ■

Korollar:
Eine analoge Aussage gilt auch für Typ 2– und Typ 3–Grammatiken.
Damit ergibt sich der folgende
Satz
Ist L kontextsensitiv (kontextfrei, regulär), dann ist $L \cup \{\square\}$ und $L - \{\square\}$ kontextsensitiv (kontextfrei, regulär).
<u>Beweis</u>
Wir können nach dem Lemma garantieren, dass das Startsymbol auf der rechten Seite nicht vorkommt und die Produktionenmenge um die $\square$ – Sonderregel ergänzen. ■

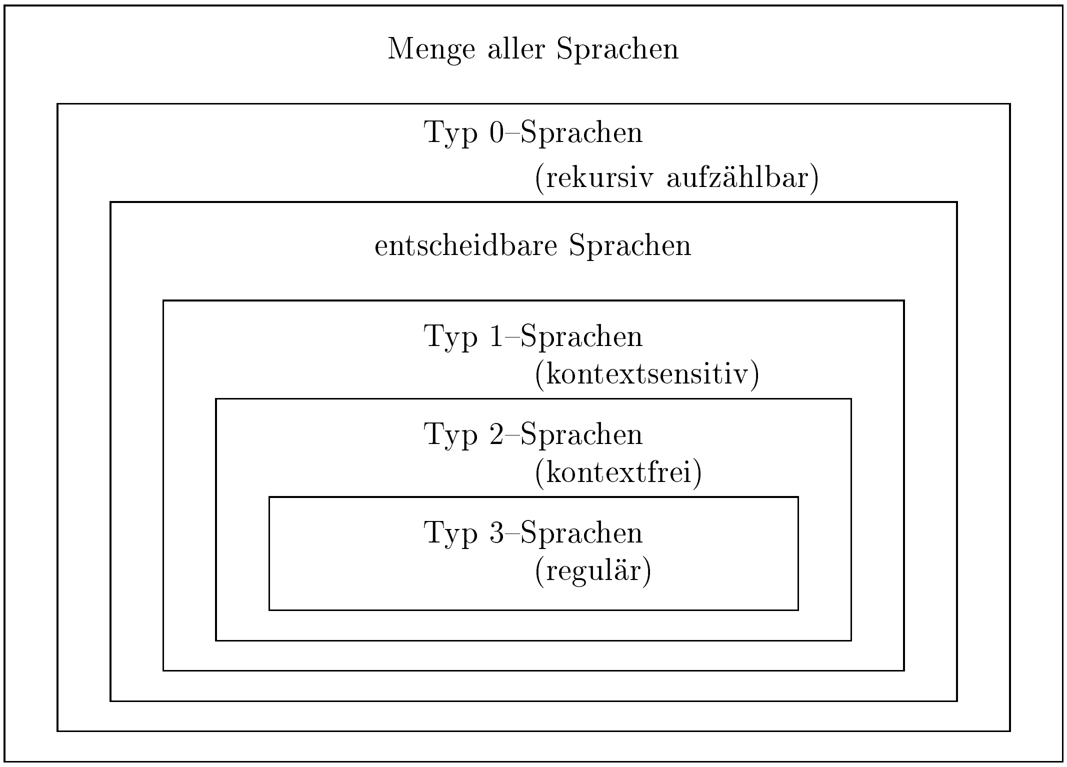


Abbildung 2: Die Chomsky-Hierarchie

Fakt

- Alle Sprachen vom Typ 1, 2, 3 sind entscheidbar, d.h. es existiert ein Algorithmus, der bei der Eingabe von G und w in endlicher Zeit entscheidet, ob $w \in L(G)$ oder nicht.
- Die Menge der Typ 0–Sprachen ist gleich der Menge der rekursiv aufzählbaren Sprachen (semi–entscheidbaren Sprachen), d.h. es existiert ein Algorithmus, der bei Eingaben w aus $L(G)$ stoppt.
- Die Menge aller Typ 0–Grammatiken ist abzählbar, da eine Typ 0–Grammatik ein endliches Objekt ist.
- Die Menge der Typ 0–Sprachen ist abzählbar, da jeder Typ 0–Sprache wenigstens eine Typ 0–Grammatik zugeordnet ist.
- Aber: Die Menge aller Sprachen (über $\{0,1\}^*$) ist nicht abzählbar.
- Also existieren Sprachen (die meisten), die nicht durch eine Grammatik beschreibbar sind!
- Praktische Anwendungen der Theorie der formalen Sprachen in der Informatik:
  - Syntaxanalyse
  - Compilerbau
  - Spezifikation und Analyse von Programmiersprachen

  Besonders gut beherrscht sind reguläre und kontextfreie Sprachen. Man untersucht auch weitere Sprachfamilien zwischen Typ 3 und Typ 2 (lineare kontextfreie Sprachen, deterministisch kontextfreie Sprachen, LL(k) und LR(k)-Sprachen, usw.).
- Es ergibt sich das folgende Problem: viele Fragestellungen in der Praxis führen auf Typ 1– oder Typ 0–Sprachen. Diese sind algorithmisch nur schwierig oder gar nicht handhabbar.
- Ausweg: Man arbeitet mit Typ 2–Grammatiken und erfaßt die „Kontextbedingungen“ und „Sonderfälle“ durch nichtgrammatikalische Zusatzalgorithmen.

  Beispiel: Die Menge der korrekten MODULA-Programme ist nicht kontextfrei (Typverträglichkeit, ausschließliche Verwendung vorher deklarierter Objekte, korrekte Anzahl der Parameter). Trotzdem benutzt man kontextfreie Grammatiken (Syntaxdiagramme), und wir wissen, dass zusätzliche Typüberprüfungen etc. erfolgen müssen.

1.4 Wortprobleme

Im folgenden untersuchen wir die Frage nach der Herleitbarkeit von Wörtern $w \in \Sigma^*$ über einer Grammatik G.

Definition
Sei $G = (V, \Sigma, P, S)$ eine Grammatik. Das Wortproblem für G besteht in der Entscheidung, ob $w \in L(G)$ oder $w \notin L(G)$ für ein beliebiges $w \in \Sigma^*$.

Satz
Das Wortproblem für Typ 1–Sprachen (kontextsensitiv) ist entscheidbar, d.h. es existiert ein Algorithmus, der bei der Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ für ein beliebiges Wort $w \in \Sigma^*$ in endlicher Zeit entscheidet, ob $w \in L(G)$ oder $w \notin L(G)$. („Kontextsensitive Grammatiken sind rekursiv“)

Beweis: Betrachte T_m^n, $n, m \in \mathbb{N}$, wobei:

$$T_m^n = \{y \in (V \cup \Sigma)^* \mid |y| \leq n \text{ und } S \Rightarrow_G^{\leq m} y\}$$

Wir können T_m^n induktiv über m definieren:

$$\begin{array}{lcl} T_0^n & = & \{S\} \\ T_{m+1}^n & = & \mathrm{Abl}_n(T_m^n) \end{array}$$

wobei $\mathrm{Abl}_n(X) = X \cup \{y \in (V \cup \Sigma)^* \mid |y| \leq n \text{ und } y' \Rightarrow_G y \text{ für ein } y' \in X\}$

- Die Darstellung ist nur für Typ 1–Grammatiken korrekt. Bei Typ 0 können aus Wörtern der Länge größer als n unter Umständen Wörter der Länge kleiner oder gleich n abgeleitet werden.
- Es existieren nur endlich viele Wörter der Länge kleiner oder gleich n in $(V \cup \Sigma)^*$, also ist $\bigcup_{m \geq 0} T_m^n$ endlich für alle n. Daher existiert ein m mit $T_m^n = T_{m+1}^n = \ldots$
- Es sei $w \in L(G)$, $|w| = n$. Dann gilt: $w \in \bigcup_{m \geq 0} T_m^n$, d.h. es existiert ein m mit $w \in T_m^n$.

Daher entscheidet folgender Algorithmus, ob $w \in L(G)$ ist:

```
INPUT(G,w);     (|w| = n)
T := {S}
REPEAT
     T1 := T
     T := Abl_n(T1)
UNTIL (w ∈ T) OR (T = T1)
IF w ∈ T THEN WRITE ('' w ∈ L(G)'' )
     ELSE WRITE ('' w ∉ L(G) '')
END
```

Achtung:
Der Algorithmus hat eine exponentielle Laufzeit. Das Wortproblem für kontextsensitive Sprachen ist NP-hart.

Beispiel
Betrachte die Grammatik für $\{a^n b^n c^n \mid n \geq 1\}$. Setze $n = 4$:

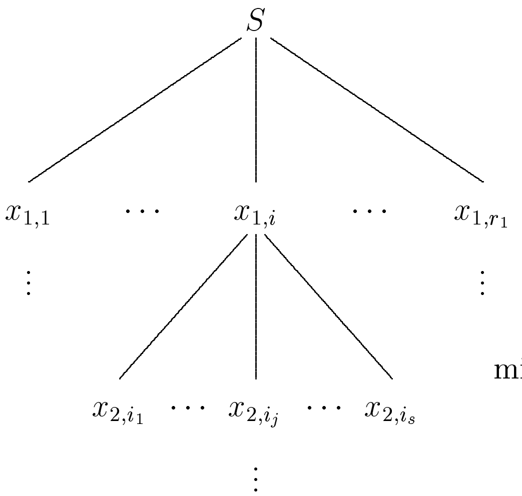
$$\begin{array}{lcl} T_0^4 & = & \{S\} \\ T_1^4 & = & \{S, aSBC, aBC\} \\ T_2^4 & = & \{S, aSBC, aBC, abC\} \\ T_3^4 & = & \{S, aSBC, aBC, abC, abc\} \\ T_4^4 & = & \{S, aSBC, aBC, abC, abc\} = T_3^4 \end{array}$$

d.h. einziges Wort $w \in L(G)$ mit $|w| \leq 4$ ist abc.

1.5 Syntaxbäume

Wir können die Ableitungen in kontextfreien Grammatiken durch Syntaxbäume darstellen. Sei G kontextfreie (Typ 2) Grammatik und $w \in L(G)$. Desweiteren sei $A : S = x_0 \Rightarrow x_1 \Rightarrow \ldots \Rightarrow x_t = w$ eine Ableitung von w.

Syntaxbaum T_w^A für w:



falls $x_1 = x_{1,1} \cdots x_{1,r_1}$

falls $x_{1,i} \in V$ und $x_1 \Rightarrow x_2$
mit der Regel $x_{1,i} \to x_{2,i_1} \cdots x_2, i_s$

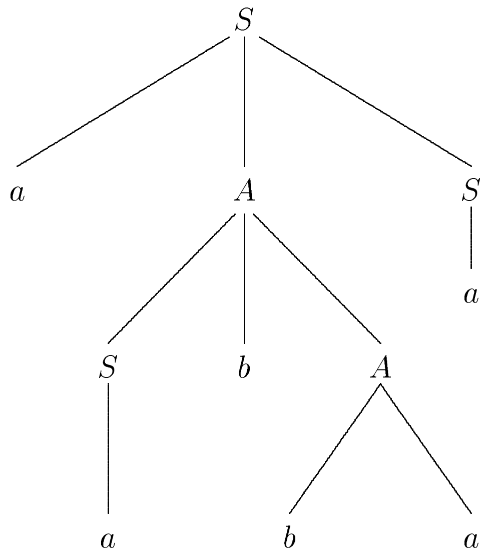
$\Rightarrow$ Die Blätter von T_w^A sind von links nach rechts gelesen mit den Buchstaben von w beschriftet!

Beispiel
Sei $G = (\{S, A\}, \{a, b\}, P, S)$ mit

$$
\begin{array}{rlll}
P = \{ & S & \longrightarrow & aAS \\
 & A & \longrightarrow & SbA \\
 & A & \longrightarrow & SS \qquad\qquad\qquad \text{(kontextfrei!)} \\
 & S & \longrightarrow & a \\
 & A & \longrightarrow & ba \ \}
\end{array}
$$

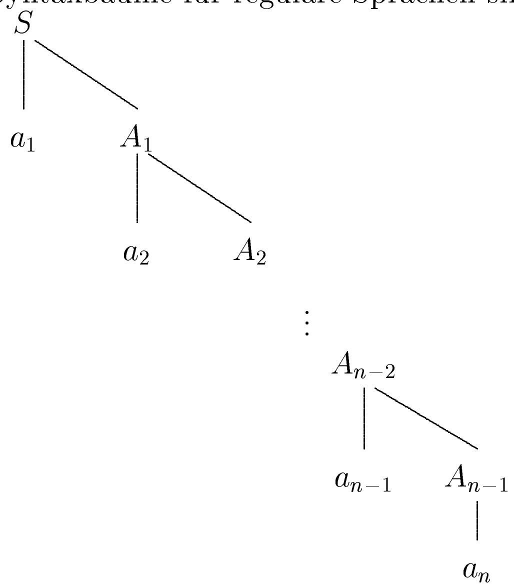
Ableitung für $w = aabbaa$:
$A : S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aabAa \Rightarrow aabbaa$
(angewendete Regeln: $S \longrightarrow aAS, A \longrightarrow SbA, S \longrightarrow a, S \longrightarrow a, A \longrightarrow ba$)

<u>Syntaxbaum T_w^A:</u>



Bemerkung

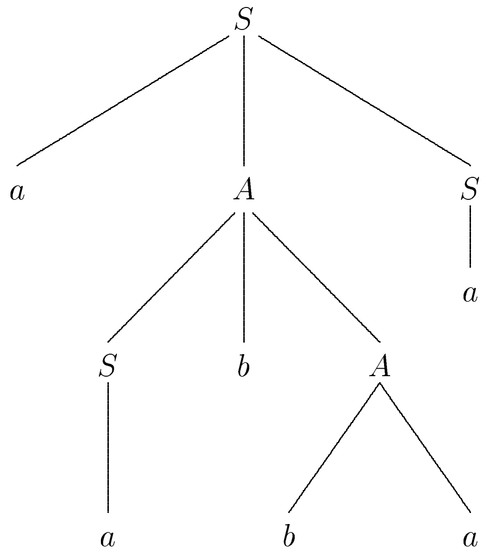
- Syntaxbäume für reguläre Sprachen sind entartet:



- Verschiedene Ableitungen können dem gleichen Syntaxbaum zugeordnet sein.

  <u>Beispiel</u>
  Betrachte Grammatik von oben mit $w = aabbaa$
  $A' : S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$
  (angewendete Regeln: $S \longrightarrow aAS, S \longrightarrow a, A \longrightarrow SbA, A \longrightarrow ba, S \longrightarrow a$)

Dazu gehört der Syntaxbaum $T_w^{A'}$:



Die Ableitung A hatte die Eigenschaft, dass stets die am weitesten links stehende Variable ersetzt wurde. („Linksableitung“)

Satz
Sei G eine kontextfreie Grammatik. Dann sind folgende Aussagen äquivalent:

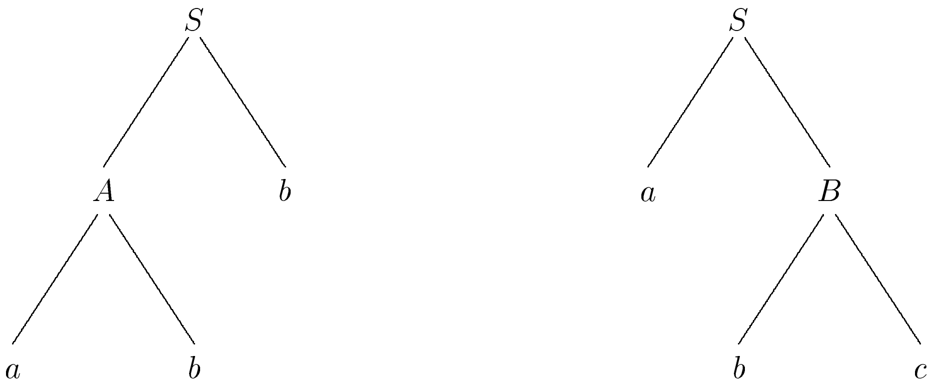
1. $w \in L(G)$
2. Es gibt eine Ableitung von w
3. Es gibt einen Syntaxbaum für w
4. Es gibt eine Linksableitung von w

Beweis:
(1) $\Rightarrow$ (2): trivial
(2) $\Rightarrow$ (3): nach Konstruktion
(3) $\Rightarrow$ (4): Offensichtlich läßt sich einem Syntaxbaum eine Linksableitung zuordnen.
(4) $\Rightarrow$ (1): trivial

Bemerkung

- Analog zu den Linksableitungen lassen sich auch Rechtsableitungen betrachten.

- Zu einem Wort $w \in L(G)$ kann es unterschiedlich strukturierte Syntaxbäume geben.
  Beispiel
  Sei G gegeben mit $S \longrightarrow aB, S \longrightarrow Ac, A \longrightarrow ab, B \longrightarrow bc$ und $w = abc$:



d.h. w hat verschiedene Linksableitungen in G!

Definition
Eine kontextfreie Grammatik G heißt mehrdeutig ("ambiguous")
$=_{def}$ es existiert ein Wort $w \in L(G)$ mit zwei verschieden strukturierten Syntaxbäumen (d.h. zwei verschiedenen Linksableitungen).
Eine kontextfreie Sprache L heißt inhärent mehrdeutig ("inherently ambiguous")
$=_{def}$ jede Grammatik G mit $L = L(G)$ ist mehrdeutig.

1.6 Backus-Naur-Form

Backus und Naur entwickelten einen Formalismus zum kompakten Niederschreiben von kontextfreien Grammatiken:

Schreibe anstelle von

$$\begin{array}{ccc} A & \longrightarrow & \beta_1 \\ A & \longrightarrow & \beta_2 \\ & \vdots & \\ A & \longrightarrow & \beta_n \end{array}$$

kurz: $A \longrightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$
(Oft wird auch $\longrightarrow$ durch $::=$ ersetzt.) Diese Darstellung heißt abgekürzt BNF.

Erweiterte Backus-Naur-Form: (EBNF)

1. $A \longrightarrow \alpha[\beta]\gamma$
   steht für
   $$\begin{array}{ccc} A & \longrightarrow & \alpha\gamma \\ A & \longrightarrow & \alpha\beta\gamma \end{array}$$
   (β kann, muß aber nicht zwischen α und γ stehen)

2. $$A \longrightarrow \alpha\{\beta\}\gamma$$

steht für

$$\begin{array}{rcl} A & \longrightarrow & \alpha\gamma \\ A & \longrightarrow & \alpha B\gamma \\ B & \longrightarrow & \beta \\ B & \longrightarrow & \beta B \end{array}$$

(β kann zwischen α und γ beliebig oft (auch 0 mal) vorkommen)

2 Reguläre Sprachen

Wir geben drei verschiedene äquivalente Charakterisierungen regulärer Sprachen an:

- mittels endlicher Automaten (Rabin, Scott)
- mittels regulärer Ausdrücke (Kleene)
- mittels Äquivalenzrelationen (Myhill, Nerode)

2.1 Deterministische und Nichtdeterministische endliche Automaten

Das Berechnungsmodell des endlichen Automaten modelliert ein endliches Gedächtnis. Die Automaten werden angesetzt auf Eingabeworte und „erkennen“ bzw. „akzeptieren“ diese oder nicht.

Definition

Ein *(deterministischer) endlicher Automat* (DFA) M ist ein 5-Tupel $M = (Q, \Sigma, \delta, q_0, F)$ mit:

Q	endliche Menge von *Zuständen*
Σ	endliches Eingabealphabet, $Q \cap \Sigma = \emptyset$
$\delta : Q \times \Sigma \longrightarrow Q$	*Überführungsfunktion*
$q_0 \in Q$	*Startzustand*
$F \subseteq Q$	Menge der *Endzustände*

<u>Beispiel</u>

$M = (Q, \Sigma, \delta, q_0, F)$ mit:

$$Q = \{q_0, q_1, q_2, q_3\}$$
$$\Sigma = \{a, b\}$$
$$F = \{q_3\}$$

δ	a	b
q_0	q_1	q_3
q_1	q_2	q_0
q_2	q_3	q_1
q_3	q_0	q_2

Zur Veranschaulichung endlicher Automaten benutzt man den sogenannten *Zustandsgraphen*.

Definition

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein endlicher Automat. Der Zustandsgraph $G_M = (V, E)$ von M ist ein gerichteter Graph mit:

- $V = Q$

- $(q, q') \in E \iff$ es existiert $a \in \Sigma$ mit: $\delta(q, a) = q'$
- (q, q') wird mit a bewertet, falls $\delta(q, a) = q'$
- q_0 wird charakterisiert durch $\longrightarrow q_0$
- $q \in F$ wird charakterisiert durch q

Beispiel
Zustandsgraph zum endlichen Automaten M aus dem Beispiel:

$$
\begin{tikzcd}
{} \arrow[r] & q_0 \arrow[r, shift left, "a"] \arrow[d, shift left, "a"] & q_1 \arrow[l, shift left, "b"] \arrow[d, shift left, "b"] \\
& q_3 \arrow[u, shift left, "b"] \arrow[r, shift left, "b"] & q_2 \arrow[l, shift left, "a"] \arrow[u, shift left, "a"]
\end{tikzcd}
$$

Ein endlicher Automat M „erkennt“ („akzeptiert“) ein Wort $w = w_1 \ldots w_n \in \Sigma^*$, $w_i \in \Sigma$, falls M gestartet in q_0 bei Eingabe von $w_1 \ldots w_n$ die Zustände $q_1, \ldots, q_n$ durchläuft, also $\delta(q_{i-1}, w_i) = q_i$ für $1 \leq i \leq n$ gilt und $q_n \in F$.

Definition
Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA.
Wir erweitern $\delta : Q \times \Sigma \longrightarrow Q$ zur Funktion $\tilde{\delta} : Q \times \Sigma^* \longrightarrow Q$ gemäß

$$\tilde{\delta}(q, \Box) := q$$

$$\tilde{\delta}(q, aw) := \tilde{\delta}(\delta(q, a), w)$$

für alle $a \in \Sigma$, $w \in \Sigma^*$, $q \in Q$

Die von M akzeptierte Sprache $L(M) \subseteq \Sigma^*$ ist definiert als

$$L(M) := \{w \in \Sigma^* \mid \tilde{\delta}(q_0, w) \in F\}.$$

Beispiel
Sei M der Automat aus obigem Beispiel.
$L(M) = \{w \in \{a, b\}^* \mid [(\text{Anzahl der } a\text{'s in } w) - (\text{Anzahl der } b\text{'s in } w)] \bmod 4 = 3\}$

Satz
Jede durch einen endlichen Automaten erkannte Sprache ist regulär (vom Typ 3).

Beweis:
Sei $L \subseteq \Sigma^*$, $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA mit $L = L(M)$.
Wir geben eine Typ 3–Grammatik $G = (V, \Sigma', P, S)$ an mit $L(G) = L$:

$V := Q,\ \Sigma' := \Sigma,\ S := q_0$

P: Gilt $\Box \in L$, dann $q_0 \longrightarrow \Box \in P$,
$q \longrightarrow aq' \in P$, falls $\delta(q,a) = q'$
$q \longrightarrow a \in P$, falls $\delta(q,a) = q'$ und $q' \in F$

Wir zeigen $L(G) = L(M)$:

$w = w_1 \dots w_n \in L(M)$

$\Longleftrightarrow$ es existieren Zustände $q_0, \dots, q_n \in Q$ mit $q_0 =$ Startzustand,
$\delta(q_{i-1}, w_i) = q_i$, $(i = 1, \dots, n)$ und
$q_n \in F$

$\Longleftrightarrow$ es existieren Variablen $q_0, \dots, q_n \in V$ mit $q_0 =$ Startsymbol,
$q_{i-1} \longrightarrow w_i q_i$, $(i = 1, \dots n-1)$ und
$q_{n-1} \longrightarrow w_n$

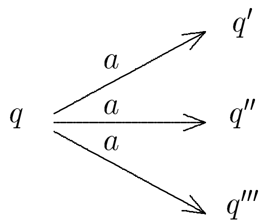
$\Longleftrightarrow$ $w \in L(G)$ ■

Definition
Ein *nichtdeterministischer endlicher Automat* (NFA) M
ist ein 5-Tupel$M = (Q, \Sigma, \delta, Q_0, F)$ mit

Q	endliche Menge von *Zuständen*
Σ	endliches Eingabealphabet, $Q \cap \Sigma = \emptyset$
$\delta \subseteq (Q \times \Sigma \times Q)$	*Überführungsrelation* (d.h. $\delta : Q \times \Sigma \longrightarrow 2^Q$)
$Q_0 \subseteq Q$	Menge von *Startzuständen*
$F \subseteq Q$	Menge der *Endzustände*

Veranschaulichung :

Wir lassen folgende Situation zu :



d.h. ein NFA „kann" im Zustand q bei Eingabe von a in verschiedene Zustände übergehen („nichtdeterministische Auswahl"); alle Möglichkeiten sind zulässig.
Wir können δ wieder verallgemeinern zu $\tilde{\delta}$:

$\tilde{\delta}$: $2^Q \times \Sigma^* \longrightarrow 2^Q$

$\tilde{\delta}(Q', \Box) := Q'$ für alle $Q' \subseteq Q$

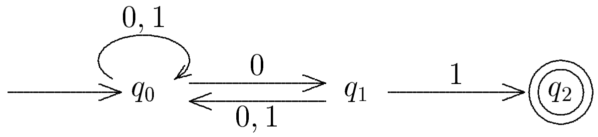
$\tilde{\delta}(Q', ax) := \bigcup_{q \in Q'} \tilde{\delta}(\delta(q,a), x)$ für alle $a \in \Sigma$, $x \in \Sigma^*$

Definition
Ein NFA M akzeptiert $w \in \Sigma^*$ $=_{def}$ $\tilde{\delta}(Q_0, w) \cap F \neq \emptyset$
d.h. es existiert eine Berechnung, die in einen Endzustand führt.

Bezeichnung: $L(M) = \{w \in \Sigma^* \mid \tilde{\delta}(Q_0, w) \cap F \neq \emptyset\}$

Beispiel

M :



$L(M) = \{w \in \{0,1\}^* \mid w = w'01\}$

Satz (Rabin, Scott)
Für jeden NFA M existiert ein DFA M', so daß $L(M) = L(M')$.

Beweis:
Sei $M = (Q, \Sigma, \delta, Q_0, F)$ ein NFA.
Wir konstruieren einen DFA M' mit $L(M') = L(M)$:
$M' := (\hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F})$ mit

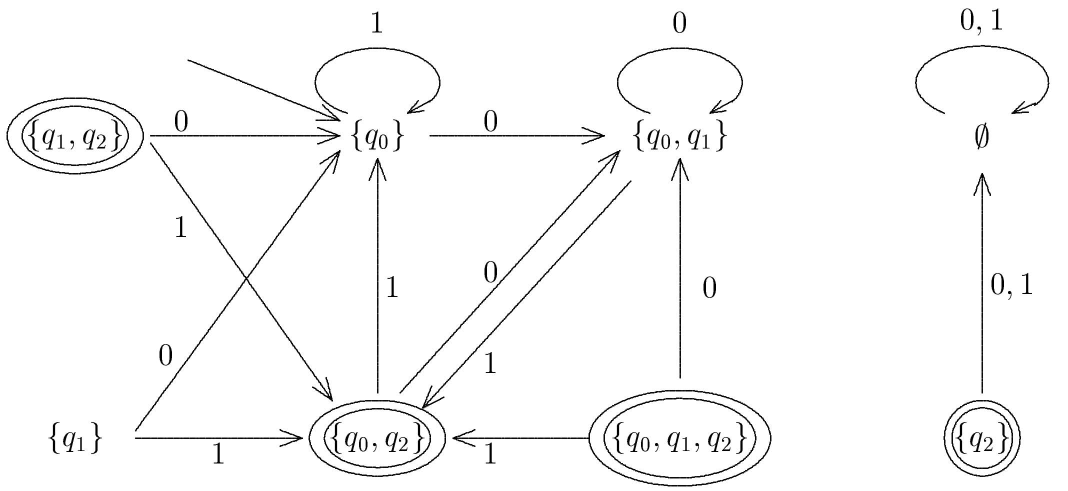
$$\begin{array}{lcl} \hat{Q} & := & 2^Q \\ \hat{q_0} & := & Q_0 \\ \hat{F} & := & \{Q' \subseteq Q \mid Q' \cap F \neq \emptyset\} \\ \hat{\delta}(Q', a) & := & \bigcup\limits_{q \in Q'} \delta(q, a) = \tilde{\delta}(Q', a) \text{ für alle } Q' \in \hat{Q}, a \in \Sigma \end{array}$$

Für $w = a_1 \dots a_n \in \Sigma^*$ gilt:

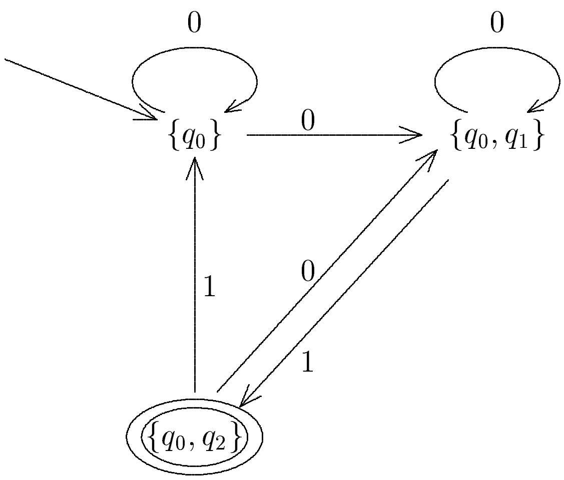
$$\begin{array}{ll} & w \in L(M) \\ \iff & \tilde{\delta}(Q_0, w) \cap F \neq \emptyset \\ \iff & \text{es existiert eine Folge } Q_1, \dots, Q_n \subseteq Q \text{ mit} \\ & \tilde{\delta}(Q_0, a_1) = Q_1 \\ & \tilde{\delta}(Q_1, a_2) = Q_2 \\ & \qquad \vdots \\ & \tilde{\delta}(Q_{n-1}, a_n) = Q_n \\ & \text{und } Q_n \cap F \neq \emptyset \\ \iff & \tilde{\hat{\delta}}(Q_0, w) \in \hat{F} \\ \iff & w \in L(M') \end{array}$$

ACHTUNG:
Wir können nur für sehr wenige Berechnungsmodelle nachweisen, dass Nichtdeterminismus keinen Gewinn an Rechenkraft bringt.

Beispiel Ein DFA, der den NFA aus dem vorherigen Beispiel simuliert:



Wir können alle Zustände streichen, die nicht vom Startzustand erreicht werden können. Dadurch verändert sich die Berechnungskraft des Automaten nicht:



ACHTUNG:
Hat ein NFA n Zustände, so kann der simulierende DFA 2^n Zustände haben.
$\Rightarrow$ Umformung NFA $\longrightarrow$ DFA kann exponentielle Laufzeit besitzen.

Satz Für jede reguläre Grammatik G existiert ein NFA M mit $L(G) = L(M)$.

Beweis:
Sei $G = (V, \Sigma, P, S)$.
Betrachte $M = (Q, \Sigma, \delta, S', F)$ mit

$$\begin{array}{lcl}
Q & = & V \cup \{q_e\} \\
S' & = & \{S\} \\
F & := & \{S, q_e\}, \text{ falls } S \longrightarrow \Box \in P \\
 & & \{q_e\}, \text{ sonst} \\
B & \in & \delta(A, a), \text{ falls } A \longrightarrow aB \in P \\
q_e & \in & \delta(A, a), \text{ falls } A \longrightarrow a \in P
\end{array}$$

Es gilt:

$$\begin{array}{ll}
 & a_1 \ldots a_n \in L(G) \\
\Longleftrightarrow & \text{es existiert eine Folge von Variablen } A_1, \ldots A_{n-1} \text{ mit} \\
 & S \Rightarrow a_1 A_1 \Rightarrow a_1 a_2 A_2 \Rightarrow \ldots \Rightarrow a_1 \ldots a_{n-1} A_{n-1} \Rightarrow a_1 \ldots a_n \\
\Longleftrightarrow & \text{es existiert eine Folge von Zuständen } A_1, \ldots A_{n-1} \text{ mit} \\
 & A_1 \in \delta(S, a_1), A_2 \in \delta(A_1, a_2), \ldots, q_e \in \delta(A_{n-1}, a_n) \\
\Longleftrightarrow & a_1 \ldots a_n \in L(M)
\end{array}$$

Damit haben wir insgesamt:
SATZ:
Jede von einem DFA erkannnte Sprache ist regulär und jede reguläre Sprache kann von einem DFA erkannt werden.

Beweis:
DFA $\longrightarrow$ reguläre Sprache $\longrightarrow$ NFA $\longrightarrow$ DFA

2.2 Reguläre Ausdrücke

Reguläre Ausdrücke sind spezielle Formeln zur Definition von Sprachen.

Definition (Reguläre Ausdrücke)
Sei Σ ein endliches Alphabet.
Ein regulärer Ausdruck über Σ ist ein Wort über dem Alphabet $\Sigma \cup \{\Box\} \cup \{(,), \emptyset, |, *\}$ mit:

1. $\emptyset$ ist ein regulärer Ausdruck
2. $\Box$ ist ein regulärer Ausdruck
3. a ist ein regulärer Ausdruck (für alle $a \in \Sigma$)
4. Sind α, β reguläre Ausdrücke, dann auch
   (a) $\alpha\beta$
   (b) $(\alpha|\beta)$
   (c) $(\alpha)^*$
5. Sonst gibt es keinen regulären Ausdruck über Σ.

Wir können induktiv über den Aufbau jedem regulären Ausdruck γ eine Sprache $L(\gamma)$ zuordnen:

Definition
Sei γ ein regulärer Ausdruck über Σ. Es gilt:

1. $L(\gamma) = \emptyset$, falls $\gamma = \emptyset$
2. $L(\gamma) = \{\Box\}$, falls $\gamma = \Box$
3. $L(\gamma) = \{a\}$, falls $\gamma = a$ für $a \in \Sigma$
4. (induktiv:)
   (a) $L(\gamma) = L(\alpha)L(\beta)$, falls $\gamma = \alpha\beta$
   (b) $L(\gamma) = L(\alpha) \cup L(\beta)$, falls $\gamma = (\alpha|\beta)$
   (c) $L(\gamma) = L(\alpha)^*$, falls $\gamma = (\alpha)^*$

Beispiel
Sei $\gamma = ((a|b))^*a$. Dann gilt

$$\begin{aligned} L(\gamma) &= L(((a|b))^*a) \\ &= L(((a|b))^*)\{a\} \\ &= L((a|b))^*\{a\} \\ &= (L(a) \cup L(b))^*\{a\} \\ &= (\{a\} \cup \{b\})^*\{a\} \\ &= (\{a,b\})^*\{a\} \\ &= \{w \mid w \in \{a,b\}^* \text{ und } w = w'a\}. \end{aligned}$$

Bemerkungen

- Alle endlichen Sprachen sind durch reguläre Ausdrücke beschreibbar:
  Sei $L = \{w_1, \dots, w_k\}$.
  Dann gilt $L = L(\gamma)$ für $\gamma = (\dots((w_1|w_2)|w_3)\dots|w_k)$
- Sei L eine Sprache und $L = L(\gamma)$ für einen regulären Ausdruck.
  Dann existieren unendlich viele γ' mit $L = L(\gamma')$. Man wähle zum Beispiel $\gamma' = (\alpha|\emptyset), \dots$

Satz (Kleene)
Die Menge der durch reguläre Ausdrücke beschreibbaren Sprachen ist genau die Klasse der regulären Sprachen.

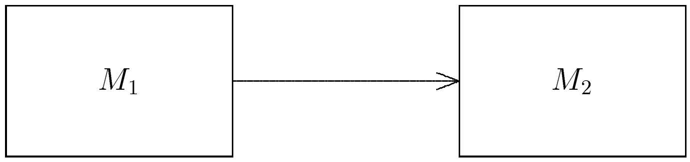
Beweis:

$\subseteq$: Sei $L = L(\gamma)$, γ ein regulärer Ausdruck über Σ. Wir konstruieren induktiv über den Aufbau von γ einen NFA M mit $L = L(M)$:

Induktionsanfang: $\gamma = \emptyset, \Box, a \in \Sigma$: trivial

Induktionsschritt:

1. Sei $\gamma = \alpha\beta$.
   Nach Induktionsvoraussetzung existieren NFA M_1, M_2 mit $L(M_1) = L(\alpha)$ und $L(M_2) = L(\beta)$;
   M:

   „Serienschaltung" von M_1 und M_2:
   Seien $M_i = (Q_i, \Sigma, \delta_i, Q_{i0}, F_i)$, $i = 1, 2$, $Q_1 \cap Q_2 = \emptyset$.
   Definiere $M = (Q, \Sigma, \delta, Q_0, F)$ mit

   $$\begin{array}{lcl} Q & := & Q_1 \cup Q_2 \\ Q_0 & := & Q_{10}, \text{ falls } \square \notin L(\alpha) \\ & & Q_{10} \cup Q_{20}, \text{ sonst} \\ F & := & F_2 \\ \delta(q, x) & := & \delta_2(q, x), \text{ falls } q \in Q_2 \\ & & \delta_1(q, x), \text{ falls } q \in Q_1 \text{ und } \delta_1(q, x) \cap F_1 = \emptyset \\ & & \delta_1(q, x) \cup Q_{20}, \text{ falls } q \in Q_1 \text{ und } \delta_1(q, x) \cap F_1 \neq \emptyset \end{array}$$

   Offenbar gilt: $L(M) = L(M_1)L(M_2) = L(\alpha\beta)$.

2. Sei $\gamma = (\alpha|\beta)$.
   M sei der folgende „Vereinigungsautomat":

   $$\begin{array}{lcl} Q & := & Q_1 \cup Q_2 \\ \delta & := & \delta_1 \cup \delta_2 \\ Q_0 & := & Q_{10} \cup Q_{20} \\ F & := & F_1 \cup F_2 \end{array}$$

   Offenbar gilt: $L(M) = L(M_1) \cup L(M_2) = L(\alpha) \cup L(\beta) = L(\alpha|\beta)$.

3. Sei $\gamma = (\alpha)^*$.
   Sei M_1 ein NFA mit $L(M_1) = L(\alpha)$. O.B.d.A. sei $\square \in L(M_1)$. (Ansonsten betrachte M_1' wie M_1, aber mit zusätzlichem Anfangs- und gleichzeitig Endzustand. Dann ist offenbar $L(M_1') = L(M_1) \cup \{\square\}$.)

   $$\begin{array}{lcl} Q & := & Q_1 \\ Q_0 & := & Q_{10} \\ F & := & F_1 \\ \delta(q, x) & := & \delta_1(q, x), \text{ falls } \delta_1(q, x) \cap F_1 = \emptyset \\ & & \delta_1(q, x) \cup Q_0; \text{ sonst („Rückkopplung")} \end{array}$$

   Offenbar gilt: $L(M) = L(M_1)^* = L(\alpha)^* = L(\gamma)$.

$\supseteq$: Sei $L = L(M)$ für einen DFA $M = (Q, \Sigma, \delta, q_1, F)$.
Wir konstruieren einen regulären Ausdruck γ mit $L(\gamma) = L(M)$:
Sei $Q = \{q_1, \ldots, q_n\}$ und $F = \{q_{i_1}, \ldots, q_{i_m}\}$.
Betrachte $R_{i,j}^k$ für alle $i, j \in \{1, \ldots, n\}$, $k \in \{0, \ldots, n\}$.

$R^k_{i,j} := \{w \in \Sigma^* \mid \tilde{\delta}(q_i, w) = q_j$ und kein Zwischenzustand hat Index $> k\}$
Offenbar gilt: $L(M) = \bigcup_{q_j \in F} R^n_{1,j}$.

Behauptung:
Die Mengen $R^k_{i,j}$ sind durch reguläre Ausdrücke beschreibbar.

Vollständige Induktion über k:
$k = 0$:

$$\begin{aligned} R^0_{i,j} &= \{a \in \Sigma \mid \delta(q_i, a) = q_j\}, \quad i \neq j \\ R^0_{i,i} &= \{\Box\} \cup \{a \in \Sigma \mid \delta(q_i, a) = q_i\} \end{aligned}$$

$R^0_{i,j}$, $R^0_{i,i}$ sind endlich $\Rightarrow$ durch reguläre Ausdrücke beschreibbar

$k \to k+1$:
Es gilt $R^{k+1}_{i,j} = R^k_{i,j} \cup R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j}$.
Ist $L(\alpha^k_{i,j}) = R^k_{i,j}$, dann gilt $L(\alpha^{k+1}_{i,j}) = R^{k+1}_{i,j}$ für $\alpha^{k+1}_{i,j} = (\alpha^k_{i,j} \mid \alpha^k_{i,k+1}(\alpha^k_{k+1,k+1})^* \alpha^k_{k+1,j})$
$\Rightarrow$ Für $\gamma := (\alpha^n_{1,i_1} | \dots | \alpha^n_{1,i_m})$ gilt: $L(\gamma) = L(M)$ ■.

2.3 Äquivalenzrelationen und Minimalautomaten

Man kann einer Sprache $L \subseteq \Sigma^*$ eine Äquivalenzrelation R_L auf Σ^* zuordnen.
Definition
Sei $L \subseteq \Sigma^*$.
$xR_Ly \;:\Longleftrightarrow\;$ für alle $z \in \Sigma^*$ gilt $xz \in L \Longleftrightarrow yz \in L$.

Bemerkung

1. R_L ist Äquivalenzrelation:
   (a) reflexiv
   (b) symmetrisch
   (c) transitiv
2. Aus xR_Ly folgt $x \in L \Longleftrightarrow y \in L$ (setze $z = \Box$).

Äquivalenzrelationen erzeugen Einteilung in Äquivalenzklassen.
Index(R_L) := Anzahl der Äquivalenzklassen von R_L (= Anzahl Σ^*/R_L)

Beispiel
Sei $L = \{a^n b^n \mid n \geq 1\}$

$$\begin{aligned} R_L\colon\ [\Box] &= \{\Box\} \\ [a] &= \{a\} \\ [aa] &= \{a^2\} \\ &\vdots \\ [a^k] &= \{a^k\} \\ &\vdots \end{aligned}$$

$\Rightarrow$ Index$(R_L) = \infty$

Satz (Myhill, Nerode)
Eine Sprache L ist regulär genau dann, wenn der Index von R_L endlich ist.

Beweis:

$\Rightarrow$) Sei L regulär, M ein DFA $M = (Q, \Sigma, \delta, q_0, F)$ mit $L(M) = L$.
Wir ordnen M die Äquivalenzrelation R_M zu:
$wR_Mv \;:\Longleftrightarrow \tilde{\delta}(q_0, w) = \tilde{\delta}(q_0, v)$, d.h. w und v überführen M in den gleichen Zustand.

Behauptung: $R_M \subseteq R_L$ („Verfeinerung“ von R_L).
Sei $wR_Mv \Rightarrow \tilde{\delta}(q_0, w) = \tilde{\delta}(q_0, v)$
Sei $z \in \Sigma^*$ beliebig.

$$\begin{array}{lcl} wz \in L & \Longleftrightarrow & \tilde{\delta}(q_0, wz) \in F \\ & \Longleftrightarrow & \tilde{\delta}(\tilde{\delta}(q_0, w), z) \in F \\ & \Longleftrightarrow & \tilde{\delta}(\tilde{\delta}(q_0, v), z) \in F \\ & \Longleftrightarrow & \tilde{\delta}(q_0, vz) \in F \\ & \Longleftrightarrow & vz \in L \end{array}$$

also wR_Lv.

Es gilt

$$\begin{array}{lcl} \text{Index}(R_L) & \leq & \text{Index}(R_M) \\ & = & \text{Anzahl der Zustände von } M\text{, die von } q_0 \text{ erreicht werden können} \\ & \leq & \#Q \\ & < & \infty. \end{array}$$

$\Leftarrow$) Sei $\text{Index}(R_L) = k < \infty$.
$\Rightarrow$ es existieren endlich viele Wörter $w_1, \ldots, w_k$ mit $\Sigma^* = [w_1] \cup [w_2] \cup \ldots \cup [w_k]$.
Wir konstruieren DFA $M_L = (Q, \Sigma, \delta, q_0, F)$ mit

$$\begin{array}{lcll} Q & = & \{[w_1], \ldots, [w_k]\} & \\ \delta([w], a) & = & [wa] & \text{für alle } a \in \Sigma \\ q_0 & = & [\square] & \\ F & = & \{[w_i] \mid w_i \in L, 1 \leq i \leq k\} & \end{array}$$

Behauptung: $L(M_L) = L$:
Es gilt $\tilde{\delta}([\square], w) = [w]$ (nach Konstruktion) und

$$\begin{array}{lcl} v \in L(M_L) & \Longleftrightarrow & \tilde{\delta}(q_0, v) \in F \\ & \Longleftrightarrow & \tilde{\delta}([\square], v) \in F \\ & \Longleftrightarrow & [v] \in F \\ & \Longleftrightarrow & v \in L \qquad \blacksquare \end{array}$$

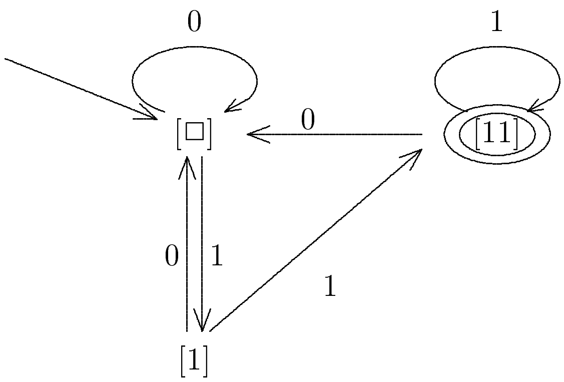
Beispiel
Sei $L = \{a^n b^n \mid n \geq 1\}$
L ist nicht regulär, da $\text{Index}(R_L) = \infty$
Beispiel

$$\begin{aligned}
\text{Sei } L \ &= \{w \in \{0,1\}^* \mid w \text{ endet mit } 11\}. \\
[\square] \ &= \{v \mid v \text{ endet nicht mit } 1\} \\
[1] \ &= \{v \mid v \text{ endet mit } 1, \text{ aber nicht mit } 11\} \\
[11] \ &= \{v \mid v \text{ endet mit } 11\}
\end{aligned}$$

$\Rightarrow \{0,1\}^* = [\square] \cup [1] \cup [11]$
$\Rightarrow \text{Index}(R_L) = 3 < \infty$
$\Rightarrow L$ ist regulär
Der Äquivalenzautomat M_L hat die Zustände $[\square]$, $[1]$, $[11]$:

$$\begin{array}{ccccc}
\delta([\square], 0) & = & [\ 0\ ] & = & [\square] \\
\delta([\square], 1) & = & [\ 1\ ] & & \\
\delta([1], 0) & = & [\ 10\ ] & = & [\square] \\
\delta([1], 1) & = & [\ 11\ ] & & \\
\delta([11], 0) & = & [\ 110\ ] & = & [\square] \\
\delta([11], 1) & = & [\ 111\ ] & = & [\ 11\ ]
\end{array}$$

$q_0 = [\square]$
$F = \{[11]\}$
Zugehörige graphische Darstellung :



Verschiedene DFA können die gleiche Sprache erkennen:
Definition
Sei $L \subseteq \Sigma^*$.
$M = (Q, \Sigma, \delta, q_0, F)$ heißt Minimalautomat für L, falls

1. $L(M) = L$
2. Für alle DFA $M' = (Q', \Sigma, \delta', q_0', F')$ mit $L(M') = L$ gilt
   $\#Q' \geq \#Q$

Satz
Sei $L \subseteq \Sigma^*$ regulär.
Dann existiert ein Minimalautomat M_L für L, der bis auf Isomorphie (also Umbenennung der Zustände) eindeutig bestimmt ist.

Beweis:

1. Existenz:
   Behauptung: Der Äquivalenzautomat M_L für L ist Minimalautomat für L.
   Beweis: Wir wissen aus dem Satz von Myhill-Nerode:
   $R_M \subseteq R_L = R_{M_L}$ für alle M mit $L(M) = L(M_L)$
   $\Rightarrow \#Q_M \geq \#Q_{M_L}$ für alle M mit $L(M) = L$.

2. Eindeutigkeit:
   Wir können sukzessive Zustände von zwei Minimalautomaten mit minimaler Zustandszahl identifizieren (und damit eine Isomorphie angeben) ■

Wir wollen testen, ob ein gegebener DFA M ein Minimalautomat ist:
O.B.d.A. sei M zusammenhängend, d.h. $\tilde{\delta}(q_0, \Sigma^*) = Q$.
$M = (Q, \Sigma, \delta, q_0, F)$ ist nicht minimal
$\iff$ es ex. $q \neq q' \in Q$, so daß für alle $w \in \Sigma^*$: $\tilde{\delta}(q, w) \in F \iff \tilde{\delta}(q', w) \in F$
q und q' können identifiziert werden, ohne die von M erkannte Sprache zu verändern.

<u>ALGORITHMUS:</u>
<u>Eingabe:</u>
DFA $M = (Q, \Sigma, \delta, q_0, F)$ (mit $\tilde{\delta}(q_0, \Sigma^*) = Q$)
<u>Ausgabe:</u>
Angabe aller identifizierbaren Zustandspaare:

1. Stelle Tabelle aller Zustandspaare (q, q') mit $q, q' \in Q$ und $q \neq q'$ auf

2. Markiere alle Paare (q, q') mit $q \in F$ und $q' \notin F$ (oder umgekehrt)

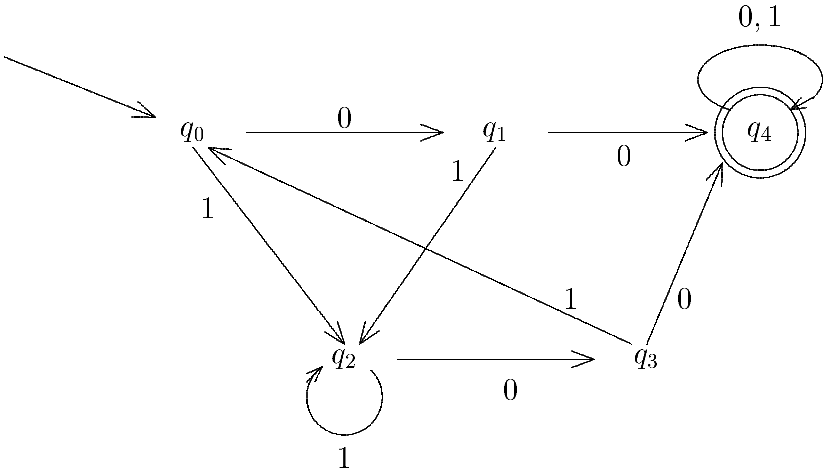
3. Teste für jedes noch unmarkierte Paar (q, q') und jedes $a \in \Sigma$, ob $(\delta(q, a), \delta(q', a))$ bereits markiert ist.
   Wenn ja: markiere (q, q')

4. Wiederhole den letzten Schritt, bis sich keine Änderung mehr ergibt.

5. Sind (q, q') nicht markiert, so können q und q' zu einem Zustand verschmolzen werden.

<u>Bemerkung</u>
Man kann den Algorithmus in Zeitkomplexität $O(n^4 m)$, $n = \#Q$, $m = \#\Sigma$ ausführen.

Beispiel:



Das ergibt die folgende Tabelle aller Zustandpaare (Schritt 1):

	q_0	q_1	q_2	q_3
q_4	.	.	.	.
q_3	.	.	.	
q_2	.	.		
q_1	.			

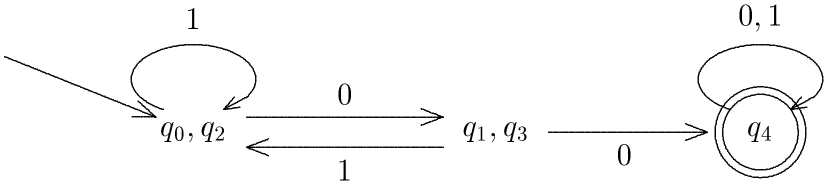
Schritt 2 ergibt die folgende Tabelle:

	q_0	q_1	q_2	q_3
q_4	+	+	+	+
q_3	.	.	.	
q_2	.	.		
q_1	.			

Nach Schritt 3 erhält man die folgende Tabelle:

	q_0	q_1	q_2	q_3
q_4	+	+	+	+
q_3	+	.	+	
q_2	.	+		
q_1	+			

Die Tabelle ändert sich nicht mehr
$\Rightarrow$ Wir können (q_0, q_2) und (q_1, q_3) identifizieren und erhalten den Minimalautomat:



2.4 Periodizitätseigenschaften

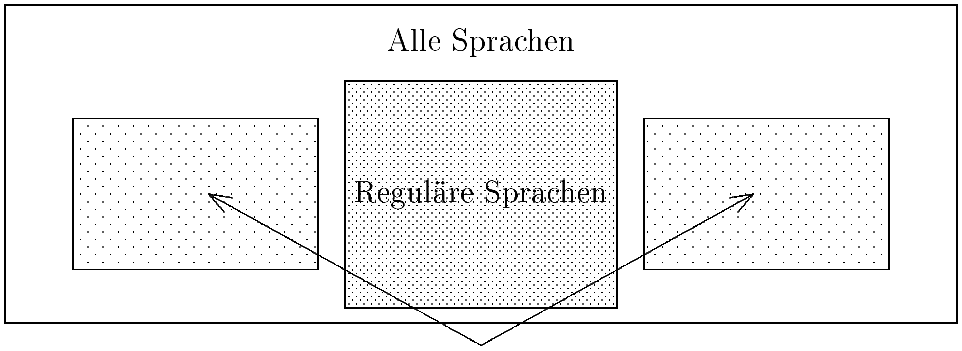
Wichtigstes Hilfsmittel zum Nachweis der Nichtregularität.

Satz (Pumping Lemma, uvw-Theorem, Schleifenlemma, Iterationslemma)
Sei L eine reguläre Sprache.
Dann existiert ein $n \in \mathbb{N}$, so dass sich alle $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$ mit:

1. $|v| \geq 1$
2. $|uv| \leq n$
3. Für alle $i \in \mathbb{N}$ gilt: $uv^iw \in L$

Beweis:
Da L regulär ist, existiert ein DFA M für L.
Sei n die Anzahl der Zustände von M und $x \in L$ mit $|x| \geq n$
$\Rightarrow$ M akzeptiert x.
Beim Abarbeiten von x durchläuft M genau $|x|+1$ Zustände (einschließlich Startzustand); wegen $|x| \geq n$ durchläuft M einen Zustand zweimal (Schubfachprinzip).
Wir setzen $x = uvw$ so, dass die Zustände nach Lesen von u und nach Lesen von uv übereinstimmen und Bedingungen (i) und (ii) erfüllt sind.
$\Rightarrow$ M erreicht den gleichen Endzustand beim Lesen von uw, uvw, uv^2w, ...
$\Rightarrow$ Bedingung (3)

ACHTUNG:
Das Pumping Lemma liefert keine äquivalente Charakterisierung von Typ 3–Sprachen. Seine Anwendung dient nur zum Nachweis der Nichtregularität.



Sprachen, die nach Pumping-Lemma nicht regulär sind

Beispiel
Sei $L = \{a^n \mid n \text{ ist Primzahl}\}$
Behauptung: L ist nicht regulär.
Annahme: L ist regulär.
$\Rightarrow$ Hinreichend lange $x = a^i \in L$ können zerlegt werden als $x = uvw$ mit $u = a^p$, $v = a^q$,

$w = a^r$ und $i = p + q + r$. Nach dem Pumping Lemma ist mit a^i auch jedes $a^p a^{nq} a^r$ in L für alle $n \in I\!N$
Das ist ein Widerspruch, da für $n = p + 2q + r + 2$ gilt $p + nq + r = (q + 1)(p + 2q + r)$ ■

Beispiel
Sei $L = \{a^n b^n \mid n \geq 1\}$
Behauptung: L ist nicht regulär
Annahme: L ist regulär.
$\Rightarrow$ es existiert ein n, so dass jedes $x \in L$ mit $|x| \geq n$ gemäß Pumping Lemma zerlegbar ist.
Betrachte $x = a^n b^n = uvw$.
Da $|uv| \leq n$, gilt $v = a^i$ mit $i \geq 1$. Aus $u = a^j$ und $w = a^{n-i-j} b^n$ folgt dann mit (3) des Pumping Lemmas $uv^2w = a^{j+2i+n-i-j} b^n = a^{i+n} b^n \in L$, Widerspruch!

■

2.5 Abschlußeigenschaften

Wir untersuchen nun den Abschluß der Menge der regulären Sprachen gegenüber gewissen Operationen. Wichtigste Beweistechnik ist die Simulation.

Satz
Die Menge der regulären Sprachen ist abgeschlossen bzgl.:

1. Vereinigung $(L \cup L')$
2. Durchschnitt $(L \cap L')$
3. Komplement $(\overline{L} = \Sigma^* - L)$
4. Produkt (LL')
5. Sternoperation $(L^* = \bigcup_{n \geq 0} L^n)$

Beweis:
(1), (4), (5) folgen aus dem Satz von Kleene:
Sind L, L' durch reguläre Ausdrücke α, β beschreibbar, so sind $L \cup L'$, LL', L^* beschreibbar durch die regulären Ausdrücke $(\alpha|\beta)$, $\alpha\beta$ und $(\alpha)^*$.
(3): Sei L akzeptiert von DFA $M = (Q, \Sigma, \delta, q_0, F)$.
Betrachte $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$.
Offenbar gilt: $L(\overline{M}) = \Sigma^* - L = \overline{L}$, d.h. $\overline{L}$ ist regulär.
(2): Wegen $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$ folgt (2) aus (1) und (3).
Direkte Konstruktion:
Seien $M = (Q, \Sigma, \delta, q_0, F)$, $M' = (Q', \Sigma, \delta', q_0', F')$ DFAs für L und L'.
Betrachte den „Kreuzproduktautomat“
$\hat{M} = (Q \times Q', \Sigma, \hat{\delta}, (q_0, q_0'), F \times F')$ mit
$\hat{\delta}((z, z'), a) = (\delta(z, a), \delta(z', a))$.
Offenbar gilt $L(\hat{M}) = L \cap L'$.

2.6 Algorithmische Eigenschaften

Wir können algorithmische Verfahren zur Lösung vieler Probleme für reguläre Sprachen bzw. endliche Automaten angeben.

Satz
Die folgenden Probleme für reguläre Sprachen (endliche Automaten) sind entscheidbar, also algorithmisch lösbar:

1. Wortproblem
   Gegeben $w \in \Sigma^*$, G reguläre Grammatik (M endlicher Automat)
   Frage: $w \in L(G)$? ($w \in L(M)$?)

2. Leerheitsproblem
   Gegeben G reguläre Grammatik (M endlicher Automat)
   Frage: $L(G) = \emptyset$? ($L(M) = \emptyset$?)

3. Endlichkeitsproblem
   Gegeben G reguläre Grammatik (M endlicher Automat)
   Frage: $\#L(G) < \infty$? ($\#L(M) < \infty$?)

4. Schnittproblem
   Gegeben G, G' reguläre Grammatiken (M, M' endliche Automaten)
   Frage: $L(G) \cap L(G') = \emptyset$? ($L(M) \cap L(M') = \emptyset$?)

5. Äquivalenzproblem
   Gegeben G, G' reguläre Grammatiken (M, M' endliche Automaten)
   Frage: $L(G) = L(G')$? ($L(M) = L(M')$?)

Beweis:

1. bereits gezeigt für Typ 1, Typ 2, Typ 3 Grammatik.
   Ist L durch DFA M gegeben, so hat folgendes Verfahren lineare Laufzeit:
   x wird Buchstabe für Buchstabe im Zustandsgraphen von M verfolgt. Ist der erreichte Zustand ein Endzustand, so gilt $x \in L$.

2. $L(M) = \emptyset \iff$ Im Zustandsgraph von M existiert kein Pfad vom Startzustand zu einem terminalen Zustand. D.h. ist L durch M gegeben, überprüfe, ob ein Weg der Länge kleiner oder gleich der Anzahl der Zustände vom Startzustand zu einem terminalen Zustand existiert.

   Für Grammatiken können wir das Leerheitsproblem mit dem Pumping Lemma lösen:
   Sei n aus Pumping Lemma. Dann gilt
   $L \neq \emptyset \iff$ es existiert ein $w \in L$ mit $|w| < n$.
   Bew. $\Leftarrow$) trivial!
   $\Rightarrow$): Sei $L(G) \neq \emptyset$, $w \in L$ mit minimaler Länge.
   Annahme: $|w| \geq n$
   $\Rightarrow w = uvx$, $|v| \geq 1$ und $ux \in L$,
   Widerspruch zur Minimalität von w, denn $|ux| < |w|$,
   d.h. ein Entscheidungsalgorithmus für das Leerheitsproblem muß nur für alle Wörter w mit $|w| < n$ überprüfen, ob $w \in L$.

3. Sei n aus dem Pumping Lemma. Dann gilt
   $\#L = \infty \iff$ es existiert ein $w \in L$ mit $n \leq |w| < 2n$.
   Bew. $\Leftarrow$): Sei $x \in L$ mit $n \leq |x| < 2n$
   $\Rightarrow x = uvw$ und $\{uv^iw \mid i \in I\!N\} \subseteq L$
   $\Rightarrow \#L = \infty$.
   $\Rightarrow$): Sei $\#L = \infty$
   Falls das kürzeste Wort $x \in L$ mit $|x| \geq n$ Länge $> 2n$ hat,
   $\Rightarrow x = uvw$ und $uw \in L$, wegen $|v| \leq |uv| \leq n$ gilt $|uw| \geq n$ im Widerspruch zur Konstruktion von x,
   d.h. der Entscheidungsalgorithmus für das Endlichkeitsproblem überprüft alle Wörter x mit $n \leq |x| < 2n$, ob $x \in L$.

4. Wir können aus G, G' (M, M') effektiv $\tilde{G}$ ($\tilde{M}$) konstruieren mit $L(\tilde{G}) = L(G) \cap L(G')$ ($L(\tilde{M}) = L(M) \cap L(M')$)
   $\Rightarrow$ Schnittproblem führt auf Leerheitsproblem

5. Wir konstruieren zu M, M' Minimalautomaten und testen diese auf Isomorphie.
   Es gilt
   $L = L' \iff (L \cap \overline{L'}) \cup (\overline{L} \cap L') = \emptyset$
   $\Rightarrow$ Wir können das Problem nach Ausführung von Schnitt-, Vereinigungs- und Komplementbildung auf das Leerheitsproblem reduzieren.

Bemerkung zur Effizienz

- Alle DFA-basierten Algorithmen haben Komplexität $O(n^2)$, ($n = \#Q$)
- Sind Sprachen durch NFA, Grammatiken oder reguläre Ausdrücke beschrieben, so ist z.B. das Äquivalenzproblem NP-hart!

3 Kontextfreie Sprachen

- Reguläre Sprachen sind in der Anwendung recht eingeschränkt.
  Z.B. ist $L = \{a^n b^n \mid n \geq 1\}$ nicht regulär.
- Die nächsthöhere Klasse in der Chomsky Hierarchie bilden kontextfreie Sprachen (Typ 2).
- Kontextfreie Sprachen sind sehr gut geeignet zur Beschreibung von Klammerungen.
  <u>Beispiel</u>
  1. Beliebig geklammerte arithmetische Ausdrücke:
     $$\begin{array}{lll} E \longrightarrow T & \mid & E+T \\ T \longrightarrow F & \mid & T*F \\ F \longrightarrow a & \mid & (E) \end{array}$$
     <u>Klammerungen:</u> (und)
  2. Anweisungen in MODULA:

<Anweisung>	$\longrightarrow$	<While–Anweisung> | <If–Anweisung>
<While–Anweisung>	$\longrightarrow$	WHILE <Bedingung> DO <Anweisung> END
<IF–Anweisung>	$\longrightarrow$	IF <Bedingung> THEN <Anweisung> END
<Bedingung>	$\longrightarrow$	...

  <u>Klammerungen:</u>
  DO und END
  THEN und END

- Die Klasse der kontextfreien Sprachen ist echt umfangreicher als die Klasse der regulären Sprachen.
  <u>Beispiel</u>
  $L = \{a^n b^n \mid n \geq 1\}$ ist kontextfrei: $S \longrightarrow ab|aSb$

3.1 Normalformen

Wir wollen nun kontextfreie Sprachen auf eine möglichst einfache Normalform bringen.
<u>Vorbemerkungen:</u>

1. Wir wissen bereits, dass jede kontextfreie Sprache □-frei gemacht werden kann.
2. Wir können alle Regeln $A \longrightarrow B$, A, B Variablen eliminieren:
   - Ziehen „Ketten“ zusammen:

Sei
$$\begin{array}{rcl} B_1 & \longrightarrow & B_2 \\ B_2 & \longrightarrow & B_3 \\ & \vdots & \\ B_{k-1} & \longrightarrow & B_k \\ B_k & \longrightarrow & B_1 \end{array}$$

wobei $B_1, \ldots, B_k$ Variablen sind.
Wir ersetzen $B_1, \ldots, B_k$ durch eine bisher nicht vorkommende Variable B.
Bemerkung
Ketten sind algorithmisch einfach aufzuspüren, da die Anzahl der Variablen endlich ist.

- Numerieren Variablen $V = \{A_1, \ldots, A_n\}$ so, dass für $A_i \longrightarrow A_j \in P$ stets gilt: $i < j$;
starten mit $k = n-1, \ldots 1$ und eliminieren Regeln $A_k \longrightarrow A_{k'}$ für $k' > k$ wie folgt:
Gilt $A_{k'} \longrightarrow x_1|x_2|\ldots|x_s$, dann fügen wir Regeln $A_k \longrightarrow x_1|x_2|\ldots|x_s$ hinzu und streichen $A_k \longrightarrow A_{k'}$.
So erhalten wir eine kontextfreie Grammatik ohne Regeln der Art $A \longrightarrow B$, A, B Variablen, die die gleiche Sprache erzeugt.

Definition (Chomsky Normalform (CNF))

Eine kontextfreie Grammatik G mit $\Box \notin L(G)$ heißt *Chomsky Normalform*
$=_{def}$ alle Regeln haben die Form $A \longrightarrow BC$ oder $A \longrightarrow a$
für Variablen A, B, C und terminales Symbol a.

Bemerkung

1. Die Ableitungsbäume für CNF-Grammatiken sind binäre Bäume.
2. Es gilt:
Jedes $w \in L(G)$ wird in genau $2|w| - 1$ Ableitungsschritten erzeugt.

Satz
Zu jeder kontextfreien Grammatik G mit $\Box \notin L(G)$ existiert eine CNF-Grammatik G' mit $L(G) = L(G')$.

Beweis:
Sei $G = (V, \Sigma, P, S)$.
O.B.d.A. haben alle Regeln von P die Form:

1. $A \longrightarrow a \qquad (A \in V, a \in \Sigma)$
2. $A \longrightarrow x \qquad (A \in V, x \in (V \cup \Sigma)^*, \quad |x| \geq 2)$

Für jedes $a \in \Sigma$ fügen wir eine neue Variable A_a zu V und die Regel $A_a \longrightarrow a$ zu P hinzu. Wir ersetzen dann jedes Vorkommen eines Terminals a auf der rechten Seite einer Regel vom Typ (2) durch die Variable A_a.

$\Rightarrow$ Wir haben nur noch Regeln $A \longrightarrow a$
$A \longrightarrow B_1 \ldots B_k, \qquad k \geq 2$

Wir müssen lediglich noch Regeln der Form $A \longrightarrow B_1 \ldots B_k$, $k \geq 3$ umwandeln.
Wir fügen neue Variablen $C_2, \ldots, C_{k-1}$ hinzu und ersetzen:

$$A \longrightarrow B_1 \dots B_k$$

durch:

$$\begin{aligned} A &\longrightarrow B_1 C_2 \\ C_2 &\longrightarrow B_2 C_3 \\ &\vdots \\ C_{k-1} &\longrightarrow B_{k-1} B_k \end{aligned}$$

<u>Beispiel</u>
Betrachte $L = \{a^n b^n c^m \mid n, m \geq 1\}$. L ist kontextfrei:

$$\begin{array}{lcl} S & \longrightarrow & AB \\ A & \longrightarrow & ab|aAb \\ B & \longrightarrow & c|cB \end{array}$$

Umformen in CNF ergibt:

$$\begin{array}{lcl} S & \longrightarrow & AB \\ A_a & \longrightarrow & a \\ A_b & \longrightarrow & b \\ A_c & \longrightarrow & c \\ B & \longrightarrow & c \end{array} \qquad \begin{array}{lcl} B & \longrightarrow & A_c B \\ A & \longrightarrow & A_a A_b \\ A & \longrightarrow & A_a D \\ D & \longrightarrow & A A_b \end{array}$$

Eine weitere wichtige Normalform für kontextfreie Grammatiken ist die Greibach-Normalform:

Definition (Greibach Normalform (GNF))

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit $\Box \notin L(G)$ ist in Greibach Normalform
$=_{def}$ alle Regeln haben die Form $A \longrightarrow aB_1 \dots B_k$
für $a \in \Sigma$, $B_i \in V$, $1 \leq i \leq k$

<u>Bemerkung</u>
Die GNF erweitert reguläre Grammatiken: für reguläre Grammatiken gilt $k = 0$ oder $k = 1$.

Satz
Zu jeder kontextfreien Grammatik G mit $\Box \notin L(G)$ existiert eine GNF-Grammatik G' mit $L(G) = L(G')$.

3.2 Pumping Lemma

Satz (Pumping Lemma, $uvwxy$-Theorem)
Sei L eine kontextfreie Sprache. Dann existiert ein $n \in I\!N$, so dass sich alle Wörter $z \in L$, $|z| \geq n$, zerlegen lassen in $z = uvwxy$ mit

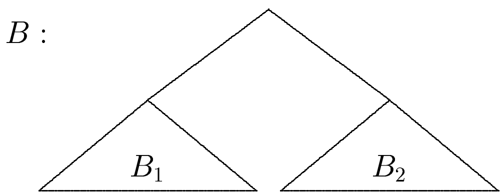
1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. für alle $i \geq 0$ gilt: $uv^iwx^iy \in L$.

Zum Beweis verwenden wir das folgende Lemma:

Lemma
Sei B ein binärer Baum mit mindestens 2^k Blättern.
Dann existiert in B mindestens ein Pfad der Länge $\geq k$.

<u>Beweis:</u>
(Induktion über k)

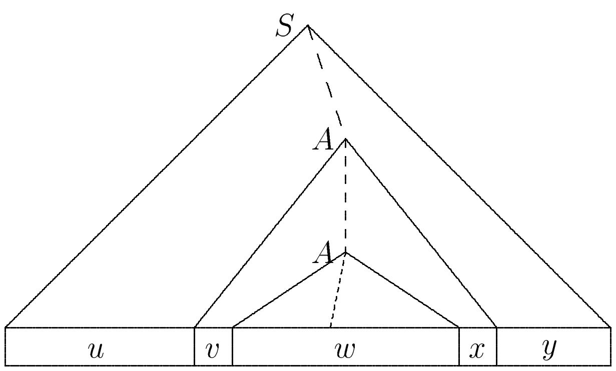
$k = 0$: trivial (jeder Baum hat einen Pfad der Länge ≥ 0)
$k-1 \to k$: Sei B Baum mit $\geq 2^k$ Blättern:



$\Rightarrow$ B_1 oder B_2 hat $\geq 2^{k-1}$ Blätter.
$\Rightarrow$ In B_1 oder B_2 existiert ein Pfad der Länge $\geq k-1$
$\Rightarrow$ In B existiert ein Pfad der Länge $\geq k$. ■

<u>Beweis:</u> (des Pumping Lemmas)
Sei $G = (V, \Sigma, P, S)$ Grammatik für $L - \{\square\}$ in CNF.
Sei $k := \#V + 1$ und $n := 2^k$.
Betrachte Ableitungsbaum T für beliebiges $z \in L(G)$ mit $|z| \geq n$

$\Rightarrow$ T ist ein binärer Baum, da G in CNF
alle Knoten bis auf die Knoten der letzten Schicht haben 2 Söhne
T hat $\geq n = 2^k$ Blätter
$\Rightarrow$ wegen des Lemmas existiert ein Pfad der Länge $\geq k$
Sei p ein Pfad maximaler Länge in T, d.h. $|p| \geq k$
$\Rightarrow$ wenigstens eine Variable A kommt zweimal vor (Schubfachprinzip)
bestimme erstes doppeltes Vorkommen von A von unten nach oben
$\Rightarrow$ oberes Vorkommen von A hat von Blättern den Abstand $\leq k$
Betrachte Teilwörter, die aus den beiden A's abgeleitet werden können.

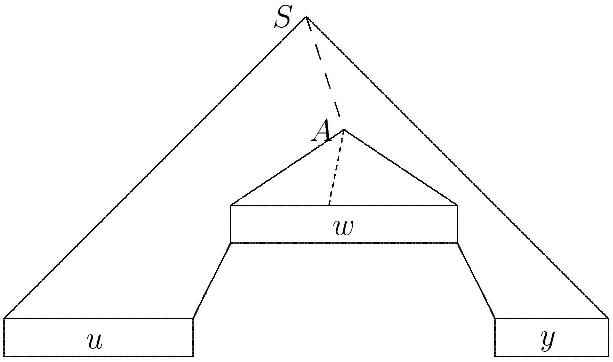


Behauptung:
Die Zerlegung $z = uvwxy$ hat die gewünschten Eigenschaften (1) – (3):

zu (1): Da G in CNF
$\Rightarrow$ aus dem oberen A wird gemäß $A \longrightarrow BC$ abgeleitet
$\Rightarrow |v| > 0$ oder $|x| > 0$
$\Rightarrow$ (1)

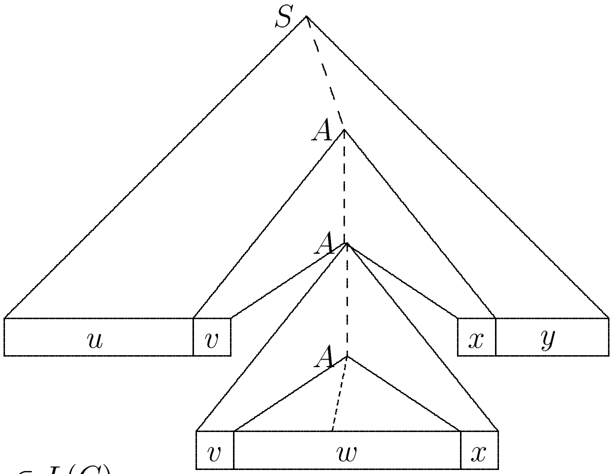
zu (2): Oberes A hat von Blattebene Abstand $\leq k$
$\Rightarrow |vwx| \leq 2^k = n$

zu (3): Wir können den Ableitungsbaum aufgrund des Doppelvorkommens von A auf verschiedene Arten modifizieren:
zum Beispiel:



$\Rightarrow uwy = uv^0wx^0y \in L(G)$

oder:



$\Rightarrow uv^2wx^2y \in L(G)$

allgemein gilt: $uv^iwx^iy \in L(G)$ für alle $i \geq 0$ ■

Anwendung des Pumping Lemma:

1. $L = \{a^m b^m c^m \mid m \geq 1\}$
   Behauptung: L ist nicht kontextfrei.
   Beweis:
   Annahme: L ist kontextfrei
   $\Rightarrow$ es existiert ein n, so dass jedes $z = a^m b^m c^m$ mit $|z| = 3m \geq n$ gemäß Pumping Lemma zerlegbar ist.
   Wähle $z = a^n b^n c^n = uvwxy$
   wegen (2) kann vwx nicht gleichzeitig a's, b's und c's enthalten
   wegen (1) ist $vx \neq \Box$
   wegen (3) gilt $uv^0wx^0y = uwy \in L$, aber $uwy \neq a^j b^j c^j$ für alle j
   $\Rightarrow$ Widerspruch. ■

   Bemerkung
   L ist kontextsensitiv
   $\Rightarrow$ {kontextfreie Sprachen} $\subset$ {kontextsensitive Sprachen}
   (echt enthalten!)

2. $L = \{a^n \mid n \text{ ist Quadratzahl}\}$
   Behauptung : L ist nicht kontextfrei.
   Beweis:

   Annahme: L ist kontextfrei
   $\Rightarrow$ es existiert ein n mit $a^{k^2} = z = uvwxy$, $k^2 \geq n$
   und $uv^iwx^iy \in L$ für alle $i \geq 0$
   $\Rightarrow k^2 + i * |vx|$ Quadratzahl für alle $i \geq -1$, Widerspruch. ■

Beobachtung:
Über einem Alphabet mit 1 Element kann das Pumping Lemma für kontextfreie Sprache in das Pumping Lemma für reguläre Sprachen überführt werden.

Satz
Jede kontextfreie Sprache über einem Alphabet Σ mit $\#\Sigma = 1$ ist regulär.

3.3 Abschlußeigenschaften

Satz

a) Die Menge der kontextfreien Sprachen ist abgeschlossen bezüglich:
   (1) Vereinigung ($L \cup L'$)
   (2) Produkt (LL')
   (3) Sternoperation ($L^* = \bigcup_{n \geq 0} L^n$)
b) Sie ist nicht abgeschlossen bezüglich:
   (1) Durchschnitt ($L \cap L'$)
   (2) Komplement ($\overline{L}$)

Beweis:
Seien $L_i \quad (i=1,2)$ kontextfreie Sprachen,
$G_i = (V_i, \Sigma, P_i, S_i)$, $(V_1 \cap V_2 = \emptyset)$ kontextfreie Grammatiken für L_i

zu a(1): $G := (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \longrightarrow S_1 | S_2\}, S)$ für ein $S \notin V_1 \cup V_2$ ist kontextfreie Grammatik für $L_1 \cup L_2$

zu a(2): $G := (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \longrightarrow S_1 S_2\}, S)$ ist kontextfreie Grammatik für $L_1 L_2$

zu a(3): $G := (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \longrightarrow \Box | S_1 S\}, S)$ ist kontextfreie Grammatik für L_1^*

zu b(1): $L_1 = \{a^i b^j c^j \mid i,j \geq 1\}$ ist kontextfrei;
$L_2 = \{a^i b^i c^j \mid i,j \geq 1\}$ ist kontextfrei;
aber $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 1\}$ ist nicht kontextfrei.

zu b(2): Annahme: Kontextfreie Sprachen sind abgeschlossen bezüglich Komplement
$\Rightarrow L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ wäre ebenfalls kontextfrei,
im Widerspruch zu b(1). $\Box$

3.4 Algorithmische Eigenschaften

Satz
Die folgenden Probleme für kontextfreie Sprachen sind algorithmisch lösbar:

1. Wortproblem:
   Gegeben: kontextfreie Grammatik G, $w \in \Sigma^*$
   Frage: $w \in L(G)$?

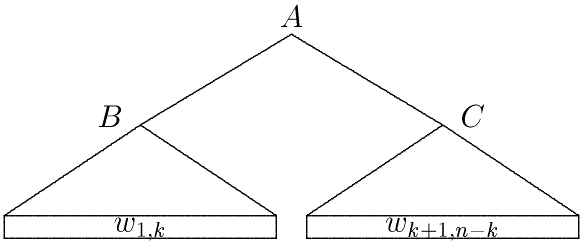
2. Leerheitsproblem:
   Gegeben : kontextfreie Grammatik G
   Frage: $L(G) = \emptyset$?

Beweis: (1) haben wir bereits für Typ 1–, Typ 2– und Typ 3–Sprachen gezeigt, für kontextfreie Sprachen können wir auch einen effizienten Algorithmus angeben, den „CYK – Algorithmus" (Cooke, Younger, Kasami):
Sei G eine kontextfreie Grammatik in CNF, $w = a \in \Sigma$.

$\Rightarrow$ w aus $A \in V$ ableitbar
$\Longleftrightarrow$ es existiert eine Regel $A \longrightarrow a$

Sei nun $w = a_1 \dots a_n, \ (n \geq 2)$
$\Rightarrow$ w aus $A \in V$ ableitbar
$\Longleftrightarrow$ es existiert ein k, $1 \leq k < n$, und eine Regel $A \longrightarrow BC$,
so dass $a_1 \dots a_k$ aus B ableitbar und $a_{k+1} \dots a_n$ aus C.
$\Rightarrow$ Ableitungsproblem für Wörter der Länge n kann überführt werden in zwei Ableitungsprobleme der Längen k und $n-k$
Wegen $w \in L(G) \Longrightarrow w$ aus S ableitbar, liefert die Lösung des Ableitungsproblems auch eine Lösung des Wortproblems.
$\longrightarrow$ „Dynamisches Programmieren"

Wir untersuchen systematisch alle Teilwörter von w auf Ableitbarkeit aus Variablen von G, speichern erzielte Information in einer Tabelle und können bei der Untersuchung von Teilwort der Länge $m < n$ auf bereitgestellte Information über Teilwörter der Länge $< m$ zurückgreifen.
Bezeichnung: Sei $w = a_1 \dots \underbrace{a_i \dots a_{i+j-1}}_{w_{i,j} = \text{ Teilwort der Länge } j} \dots a_n$



Wir arbeiten mit Tabelle $T[1 \dots n, 1 \dots n]$
$T[i,j] = \{A \in V \mid A \longrightarrow w_{i,j}\}$; es genügt, die obere Dreiecksmatrix zu betrachten.

CYK – Algorithmus

Eingabe: $w = a_1 \dots a_n$, G kontextfreie Grammatik
Ausgabe: $w \in L(G)$, falls $S \in T[1,n]$
$w \notin L(G)$, sonst

FOR $i := 1$ TO n DO (* Fall $j = 1$ *)
    $T[i,1] := \{A \mid A \longrightarrow a_i \in P\}$
END;
FOR $j := 2$ TO n DO (* Fall $j > 1$ *)
    FOR $i := 1$ TO $n+1-j$ DO
        $T[i,j] := \emptyset$;
        FOR $k := 1$ TO $j-1$ DO
            $T[i,j] := T[i,j] \cup \{A \mid A \longrightarrow BC \in P,\ B \in T[i,k],\ C \in T[i+k, j-k]\}$;
        END;
    END;
END;
IF $S \in T[1,n]$ THEN WriteString('$w \in L(G)$')
    ELSE WriteString('$w \notin L(G)$')
END;

Analyse: Laufzeit $O(n^3)$
Bemerkung:
Der CYK – Algorithmus leitet einen Ableitungsbaum von den Blättern beginnend ab („bottom up")

Beispiel

$$\begin{array}{rcl} \text{Sei } L & = & \{a^n b^n c^m \mid n, m \geq 1\} \\ P & : & S \longrightarrow AB \\ & & A \longrightarrow ab|aAb \\ & & B \longrightarrow c|cB \end{array}$$

1. Umformung in CNF :

$$\begin{array}{lcl} S & \longrightarrow & AB \\ A & \longrightarrow & A_a A_b | A_a F \\ B & \longrightarrow & c|A_c B \\ A_a & \longrightarrow & a \\ A_b & \longrightarrow & b \\ A_c & \longrightarrow & c \\ F & \longrightarrow & A A_b \end{array}$$

2. Anwendung des CYK – Algorithmus:
   Sei $w = aaabbbcc$. Das ergibt die folgende Tabelle:

w	a	a	a	b	b	b	c	c
1	A_a	A_a	A_a	A_b	A_b	A_b	A_c, B	A_c, B
2			A				B	
3			F					
4		A						
5		F						
6	A							
7	S							
8	S							

   Wegen $S \in T[1,8]$ gilt $aaabbbcc \in L(G)$.

Satz
Die folgenden Probleme für kontextfreie Grammatiken sind nicht entscheidbar.

1. Schnittproblem
   Gegeben: G_1, G_2 kontextfrei
   Frage: $L(G_1) \cap L(G_2) = \emptyset$?

2. Äquivalenzproblem
   Gegeben: G_1, G_2 kontextfrei
   Frage: $L(G_1) = L(G_2)$?

3. Mehrdeutigkeitsproblem
   Gegeben: G kontextfrei
   Frage: Ist G mehrdeutig, d.h. existiert $w \in L(G)$ mit zwei verschiedenen Ableitungsbäumen?

(ohne Beweis)

3.5 Pushdown Automaten

Wir erweitern den endlichen Automaten, um ein Berechnungsmodell für kontextfreie Sprachen zu erhalten.
Wir müssen den NFA mit einem Speicher ausstatten. Der Speicher darf nicht zu „mächtig“ sein, sonst erhält man eine Turing Maschine (TM).
Wir verwenden einen Pushdown Speicher (Keller, Stack).
Zur Erinnerung:

Kellerspeicher:	last in – first out Speicher
2 Operationen:	PUSH (Einkellern)
	POP (Auskellern)

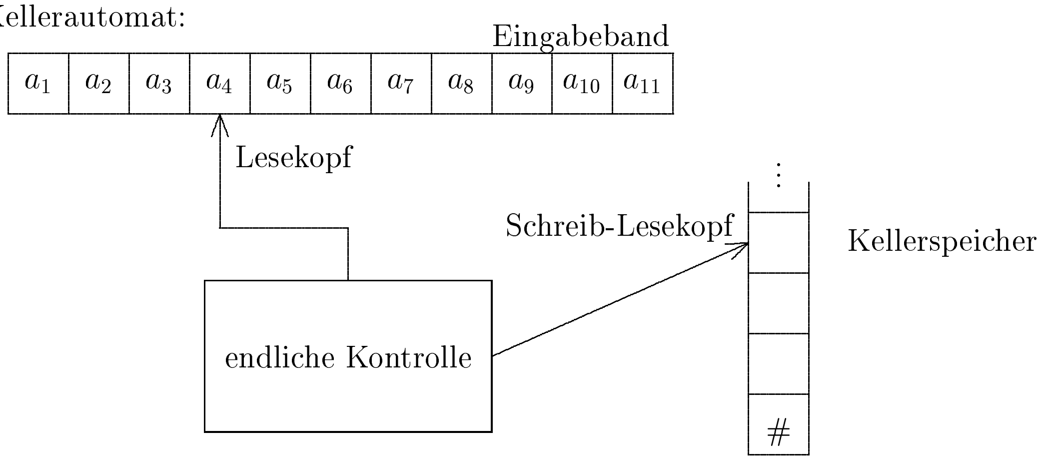
Definition
Ein Pushdownautomat (Kellerautomat, Stackautomat, ...) (PDA) ist ein 6-Tupel

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \#)$$

mit

Q	$=$	endliche Menge von *Zuständen*
Σ	$=$	endliches *Eingabealphabet*
Γ	$=$	endliches *Kelleralphabet*
δ	:	$Q \times (\Sigma \cup \{\Box\}) \times \Gamma \longrightarrow P_e(Q \times \Gamma^*)$ *Überführungsfunktion*,
		$(P_e(Q \times \Gamma^*) =$ Menge der endl. Teilmengen von $(Q \times \Gamma^*))$
$q_0 \in Q$	$=$	Startzustand
$\# \in \Gamma$	$=$	unterstes Kellerzeichen (zu Beginn jeder Rechnung im Keller)

Kellerautomat:



Arbeitsweise:

- $(q', B_1 \dots B_k) \in \delta(q, a, A)$ bedeutet:
  liest M im Zustand q das Eingabesymbol a und hat Zugriff auf oberstes Kellerzeichen A, dann kann M in Zustand q' übergehen und im Keller das oberste Zeichen A durch die Zeichenkette $B_k, \dots, B_1$, $(k \geq 0)$ ersetzen; M hat nun Zugriff auf das zuletzt geschriebene Kellerzeichen B_1.
  Ausführung der Operation POP: Wähle $k = 0$.
  Ausführung der Operation PUSH: $B_1 \dots B_k = BA$.

- $(q', B_1 \dots B_k) \in \delta(q, \Box, A)$ bedeutet:
  M kann ohne vom Eingabeband zu lesen in den Zustand q' übergehen und das Kellerzeichen A durch die Zeichen $B_k, \dots, B_1$ ersetzen. (B_1 zuletzt)

- M akzeptiert $w \in \Sigma^*$,
  falls der Keller nach einer möglichen Abarbeitung von w leer ist.

Formaler:
$K \in Q \times \Sigma^* \times \Gamma^*$ heißt Konfiguration von PDA M.
Dabei beinhaltet die Σ^*-Komponente die Beschreibung des noch zu lesenden Teils der Eingabe, die Γ^*-Komponente den aktuellen Kellerinhalt (oberstes Zeichen steht ganz links).
$K \mapsto K'$: „K' geht in einem Schritt aus K hervor“:

$$(q_0, a_1 \dots a_n, A_1 \dots A_m) \mapsto \begin{cases} (q', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m), \\ \text{falls } (q', B_1 \dots B_k) \in \delta(q, a_1, A_1) \\ (q', a_1 \dots a_n, B_1 \dots B_k A_2 \dots A_m), \\ \text{falls } (q', B_1 \dots B_k) \in \delta(q, \Box, A_1) \end{cases}$$

$\mapsto^*$ bezeichnet die reflexive und transitive Hülle von $\mapsto$.

Definition
$L(M) := \{w \in \Sigma^* \mid (q_0, w, \#) \mapsto^* (q, \Box, \Box) \text{ für ein } q \in Q\}$ ist die vom PDA M akzeptierte Sprache.

Beispiel

1. $L = \{w\$w^R \mid w \in \{a, b\}^* \text{ mit } w^R = a_n \dots a_1, \text{ falls } w = a_1 \dots a_n\}$.
   Wir wollen einen PDA M beschreiben mit $L(M) = L$:
   $M = (\{q_0, q_1\}, \{a, b, \$\}, \{\#, A, B\}, \delta, q_0, \#)$
   Wir schreiben anstelle von $(q', x) \in \delta(q, a, A)$ kurz: $qaA \longrightarrow q'x$:

$$\delta: \begin{array}{ll} q_0 a\# \longrightarrow q_0 A\# & q_0 aA \longrightarrow q_0 AA \\ q_0 aB \longrightarrow q_0 AB & q_0 b\# \longrightarrow q_0 B\# \\ q_0 bA \longrightarrow q_0 BA & q_0 bB \longrightarrow q_0 BB \\ q_0 \$\# \longrightarrow q_1 \# & q_0 \$A \longrightarrow q_1 A \\ q_0 \$B \longrightarrow q_1 B & q_1 aA \longrightarrow q_1 \Box \\ q_1 bB \longrightarrow q_1 \Box & q_1 \Box\# \longrightarrow q_1 \Box \end{array}$$

   Betrachte $ba\$ab$:

$$\begin{array}{rcl}(q_0, ba\$ab, \#) & \mapsto & (q_0, a\$ab, B\#) \\ & \mapsto & (q_0, \$ab, AB\#) \\ & \mapsto & (q_1, ab, AB\#) \\ & \mapsto & (q_1, b, B\#) \\ & \mapsto & (q_1, \Box, \#) \\ & \mapsto & (q_1 \Box, \Box)\end{array}$$

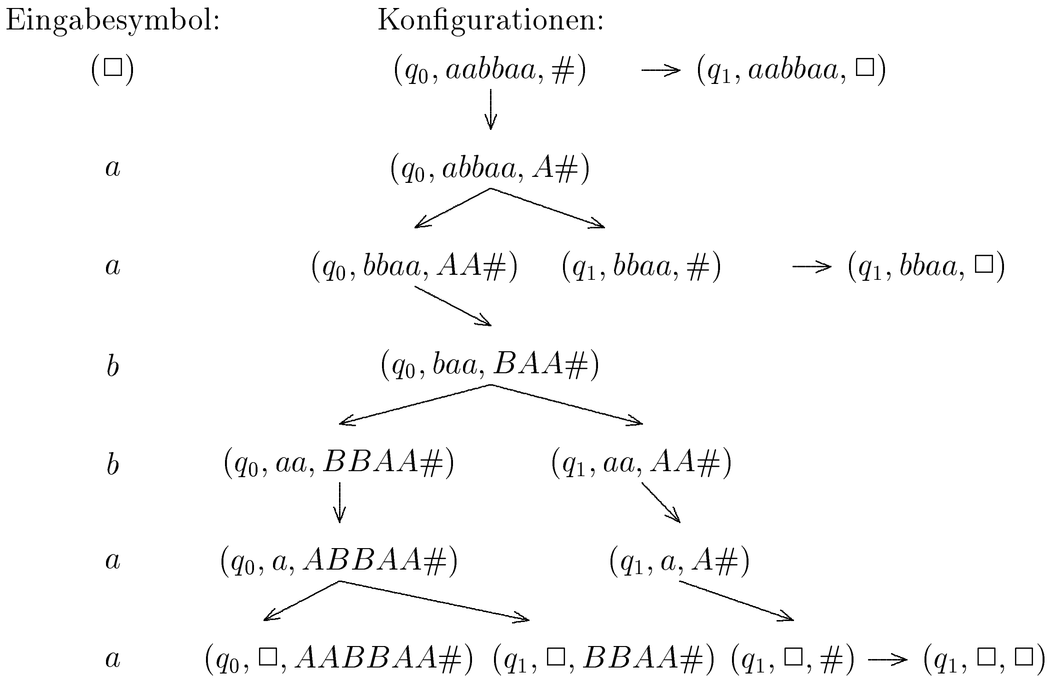
$\Rightarrow ba\$ab \in L(M)$.

Man überlegt sich leicht allgemein, dass $L(M) = L$ gilt.

2. $L' = \{ww^R \mid w \in \{a, b\}^*\}$
   Betrachte: $M' = (\{q_0, q_1\}, \{a, b\}, \{\#, A, B\}, \delta', q_0, \#)$
   δ' ist definiert wie δ, nur anstelle der Übergänge von $q_0\$\#$, $q_0\$A$ und $q_0\$B$ nehmen wir dazu:

   $$\begin{array}{ll}\delta' : & q_0 aA \longrightarrow q_1\Box \\ & q_0 bB \longrightarrow q_1\Box \\ & q_0 \Box\# \longrightarrow q_1\Box\end{array} \qquad M' \text{ ist nichtdeterministisch!}$$

   Betrachte *aabbaa*:

   Wir brauchen den Nichtdeterminismus, um die Wortmitte zu „erraten“.

   Es gilt: $L(M') = L'$.

   Bemerkung
   Es existiert kein deterministischer PDA M mit $L(M) = L'$.

Satz
L ist kontextfrei genau dann, wenn ein PDA M mit $L(M) = L$ existiert.

Beweis:
$\Rightarrow$)
Sei $G = (V, \Sigma, P, S)$ kontextfreie Grammatik für L.
Wir konstruieren einen PDA M, der Ableitungen von G mit Kellerinhalt simuliert.
Sei $M = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$
Definiere δ mit Hilfe von P:

1. Für $A \longrightarrow \alpha, \quad \alpha \in (V \cup \Sigma)^*$ setzen wir $(q, \alpha) \in \delta(q, \Box, A)$
   D.h.: ist oberstes Kellersymbol eine Variable, dann wird ohne zu lesen eine P-Regel angewendet.

2. Für $a \in \Sigma$ setzen wir $(q, \Box) \in \delta(q, a, a)$
   D.h.: ist oberstes Kellerzeichen ein Terminalsymbol und stimmt dieses mit dem Eingabesymbol überein, so wird es aus dem Keller gestrichen.

Es gilt für alle $w \in \Sigma^*$:
$w \in L(G)$
$\Longleftrightarrow$ es existiert eine Ableitung in G der Form:
$S \Rightarrow \ldots \Rightarrow w$
$\Longleftrightarrow$ es existiert eine Folge von Konfigurationen von M der Form:
$(q, w, S) \mapsto \ldots \mapsto (q, \Box, \Box)$
$\Longleftrightarrow w \in L(M)$.

$\Leftarrow$)
Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, \#)$ ein PDA und $L = L(M)$.
O.B.d.A. gelte $qaA \rightarrow q'B_1 \ldots B_k$ mit $k \leq 2$,

$$\begin{array}{ll} \text{ansonsten} & qaA \rightarrow q'B_1 \ldots B_k \text{ mit } k > 2 \\ \text{ersetzen durch} & qaA \rightarrow q_1 B_{k-1} B_k \\ & q_1 \Box B_{k-1} \rightarrow q_2 B_{k-2} B_{k-1} \\ & \quad\vdots \qquad\qquad \vdots \\ & q_{k-2} \Box B_2 \rightarrow q' B_1 B_2 \end{array}$$

Wir konstruieren eine Grammatik G, die Rechenschritte von M durch eine Linksableitung simuliert:

$$\begin{array}{lcl} G & = & ((\{S\} \cup Q) \times \Gamma \times Q, \Sigma, P, S) \\ \text{mit} & & \\ P & = & \{S \longrightarrow (q_0, \#, q) \mid q \in Q\} \\ & \cup & \{(q, A, q') \longrightarrow a \mid (q', \Box) \in \delta(q, a, A)\} \\ & \cup & \{(q, A, q') \longrightarrow a(q_1, B, q') \mid (q_1, B) \in \delta(q, a, A), q' \in Q\} \\ & \cup & \{(q, A, q') \longrightarrow a(q_1, B, q_2)(q_2, C, q') \mid (q_1, BC) \in \delta(q, a, A), q', q_2 \in Q\} \end{array}$$

Achtung
G kann $\Box$-Regeln enthalten, da $a = \Box$ möglich. In diesem Fall eliminieren wir die $\Box$-Regeln wie gehabt.

Behauptung
Für alle $w \in \Sigma^*$ gilt:
$(q, A, q') \Rightarrow^* w$ genau dann, wenn $(q, w, A) \mapsto^* (q', \Box, \Box)$

Beweis:
Sei $a \in \Sigma \cup \{\Box\}$.

$$\begin{aligned}(q, A, q') \Rightarrow a \quad &\Longleftrightarrow \quad (q, A, q') \longrightarrow a \in P \\ &\Longleftrightarrow \quad (q', \Box) \in \delta(q, a, A) \\ &\Longleftrightarrow \quad (q, a, A) \mapsto (q', \Box, \Box)\end{aligned}$$

Für $|w| > 1$ führen wir den Beweis per Induktion:
$\Leftarrow$)
Induktion über Zahl n der Rechenschritte von M

$n = 1$: klar (s.o.)

$n \longrightarrow n+1$: $w = ay$, $a \in \Sigma \cup \{\Box\}$ und
$(q, ay, A) \rightarrow (q_1, y, \alpha) \rightarrow^+ (q', \Box, \Box)$
(dabei ist α der Kellerinhalt)

1. Fall: $\alpha = \Box$.
kann nicht eintreten, da $(q_1, y, \Box)$ ohne Folgekonfiguration

2. Fall: $\alpha = B$.
$\Rightarrow (q_1, B, q') \Rightarrow^* y$ (nach Induktionsvoraussetzung)
und in P muß die folgende Regel existieren:
$(q, A, q') \longrightarrow a(q_1, B, q')$
$\Rightarrow (q, A, q') \Rightarrow a(q_1, B, q') \Rightarrow^* ay = w$

3. Fall: $\alpha = BC$.
Wir können $(q_1, y, BC) \mapsto^* (q', \Box, \Box)$ in zwei Teile zerlegen:
$(q_1, y, BC) \mapsto^* (q_2, y_2, C)$ und
$(q_2, y_2, C) \mapsto^* (q', \Box, \Box)$ (1) wobei $y = y_1 y_2$;
für y_1 gilt: $(q_1, y_1, B) \mapsto^* (q_2, \Box, \Box)$ (2)
Nach Ind. Vor. folgt $(q_1, B, q_2) \Rightarrow^* y_1$ (aus (2)),
und $(q_2, C, q') \Rightarrow^* y_2$ (aus (1)).
In P muß die folgende Regel existieren:
$(q, A, q') \longrightarrow a(q_1, B, q_2)(q_2, C, q')$ (1. Rechenschritt)
$\Rightarrow (q, A, q') \Rightarrow a(q_1, B, q_2)(q_2, C, q')$
$\Rightarrow^* ay_1(q_2, C, q') \Rightarrow^* ay_1y_2 = ay = w$

$\Rightarrow$)
Induktion über die Länge k der Linksableitung:
$k = 1$: klar (s.o.)
$k - 1 \longrightarrow k$:

1. Fall: $(q, A, q') \Rightarrow a \Rightarrow^* w$
$\Rightarrow w = a$, im Widerspruch zu $k > 1$

2. Fall: $(q, A, q') \Rightarrow a(q_1, B, q') \Rightarrow^* ay = w$
$\Rightarrow (q_1, B) \in \delta(q, a, A)$
und nach Ind. Vor. $(q_1, y, B) \mapsto^* (q', \Box, \Box)$
$\Rightarrow (q, ay, A) \mapsto (q_1, y, B) \mapsto^* (q', \Box, \Box)$

3. Fall: $(q, A, q') \Rightarrow a(q_1, B, q_2)(q_2, C, q') \Rightarrow^* ay = w$
$\Rightarrow (q_1, BC) \in \delta(q, a, A)$ und
nach Ind. Vor. $(q_1, y_1, B) \mapsto^* (q_2, \Box, \Box)$,
$(q_2, y_2, C) \mapsto^* (q', \Box, \Box)$ mit $y = y_1 y_2$
$\Rightarrow (q, ay_1y_2, A) \mapsto (q_1, y_1y_2, BC) \mapsto^* (q_2, y_2, C) \mapsto^* (q', \Box, \Box)$

Insgesamt erhalten wir:

$$\begin{aligned} w \in L(M) &\iff (q_0, w, \#) \mapsto^* (q, \Box, \Box) \quad \text{für ein } q \in Q \\ &\iff S \Rightarrow (q_0, \#, q) \Rightarrow^* w \quad \text{für ein } q \in Q \\ &\iff w \in L(G) \end{aligned}$$ ■

3.6 Deterministisch kontextfreie Sprachen

Wir haben gezeigt:
L ist kontextfrei $\iff$ es existiert ein nichtdeterministischer PDA M mit $L = L(M)$.
Welche Auswirkung hat die Beschränkung auf deterministische PDAs?

Definition
Ein PDA M heißt deterministisch
$=_{def}$ für alle $q \in Q, a \in \Sigma, A \in \Gamma$ gilt:
$\#\delta(q, a, A) + \#\delta(q, \Box, A) \leq 1$, und es existiert eine Menge $F \subseteq Q$ von Zuständen, so dass M akzeptiert, falls M einen Zustand $q \in F$ erreicht.
Bei deterministischem PDA sind Konfigurationsbäume linear, und $\mapsto$ ist partielle Funktion mit $(q_0, w, \#) \mapsto k_1 \mapsto k_2 \mapsto \ldots \mapsto k_n \mapsto \ldots$

Beispiel
$L = \{a_1 \ldots a_n \$ a_n \ldots a_1 \mid a_i \in \Sigma\}$ wird von einem deterministischen PDA akzeptiert.

Definition
$L \subseteq \Sigma^*$ heißt deterministisch kontextfrei
$=_{def}$ L wird von einem deterministischen PDA akzeptiert.

Satz
Die Menge der deterministisch kontextfreien Sprachen ist abgeschlossen bezüglich Komplement.

Beweis:
Sei L deterministisch kontextfrei und M ein deterministischer PDA mit $L = L(M)$. Wir konstruieren einen deterministischen PDA M' mit $L(M') = \overline{L}$:

Problem:	deterministische PDAs können nach Lesen der Eingabe durch $\Box$-Bewegungen akzeptierende oder nichtakzeptierende Zustände erreichen.
Idee:	$M' = (Q \times \{1,2,3\}, \Sigma, \delta', q_0, \#, Q \times \{3\})$
$(q,1)$	wird erreicht, wenn M Zustand q erreicht und seit dem letzten Lesen eines Buchstabens einen akzeptierenden Zustand erreicht hat.
$(q,2)$	wird erreicht, falls M Zustand q erreicht, ohne seit dem letzten Lesen eines Buchstabens einen akzeptierenden Zustand erreicht zu haben.

Anfangszustand: $(q_0, 1)$, falls $q_0 \in F$
$(q_0, 2)$, sonst

Liest M in Zustand q bei Kellersymbol A den Buchstaben $a \in \Sigma$, so soll dies M' im Zustand $(q,1)$ ebenfalls tun; im Zustand $(q,2)$ soll M' jedoch mit einer $\Box$-Bewegung in Zustand $(q,3)$ wechseln, ohne den Keller zu verändern, und erst dann das Lesen des nächsten Buchstabens simulieren; kann M im Zustand $(q,2)$ nicht weiterarbeiten, geht M' ebenfalls in Zustand $(q,3)$.
M' erreicht akzeptierenden Zustand $\Longleftrightarrow$ M am Ende (leeres Leseband) keinen akzeptierenden Zustand erreicht.
$\Rightarrow L(M') = \overline{L(M)} = \overline{L}$

Korollar
Es existieren kontextfreie Sprachen, die nicht deterministisch kontextfrei sind.

Beweis:
Menge der kontextfreien Sprachen ist nicht abgeschlossen bezüglich Komplement.

Beispiel
$L = \{a_1 \dots a_n a_n \dots a_1 \mid a_i \in \Sigma\}$ ist nicht deterministisch kontextfrei.

Satz
Die Menge der deterministisch kontextfreien Sprachen ist nicht abgeschlossen bezüglich:

1. Durchschnitt
2. Vereinigung
3. Produkt
4. Sternoperation

Beweis:

zu (1) : $L_1 = \{a^n b^n c^m | n, m \geq 1\}$ und
$L_2 = \{a^n b^m c^m | n, m \geq 1\}$ sind deterministisch kontextfrei,
aber $L_1 \cap L_2$ ist nicht kontextfrei, also auch nicht deterministisch kontextfrei.

zu (2) : Beweis durch Widerspruch:
Annahme: det. kontextfreie Sprachen abg. gegen Vereinigung
$\Rightarrow$ (nach deMorgan-Regeln) deterministisch kontextfreie Sprachen abgeschlossen bzgl. Durchschnitt
$\Rightarrow$ Widerspruch zu (1).

zu (3) : durch geeignetes Beispiel

zu (4) : durch geeignetes Beispiel

Bemerkung: Für deterministisch kontextfreie Sprachen ist kein Pumping Lemma bekannt. Um zu zeigen, dass L nicht deterministisch kontextfrei ist, genügt es zu zeigen, dass $\overline{L}$ nicht kontextfrei ist.

Satz

Die folgenden Probleme sind für durch deterministische PDAs gegebene deterministisch kontextfreie Sprachen nicht entscheidbar:

1. Schnittproblem $(L \cap L' = \emptyset?)$
2. Inklusion $(L \subseteq L'?)$
3. $L \cap L'$ deterministisch kontextfrei?
4. $L \cap L'$ kontextfrei?
5. $L \cup L'$ deterministisch kontextfrei?

Satz

Für kontextfreie Sprachen ist nicht entscheidbar, ob sie deterministisch kontextfrei sind.

Beweis:

Können das Problem auf unentscheidbares Endlichkeitsproblem für TM reduzieren.

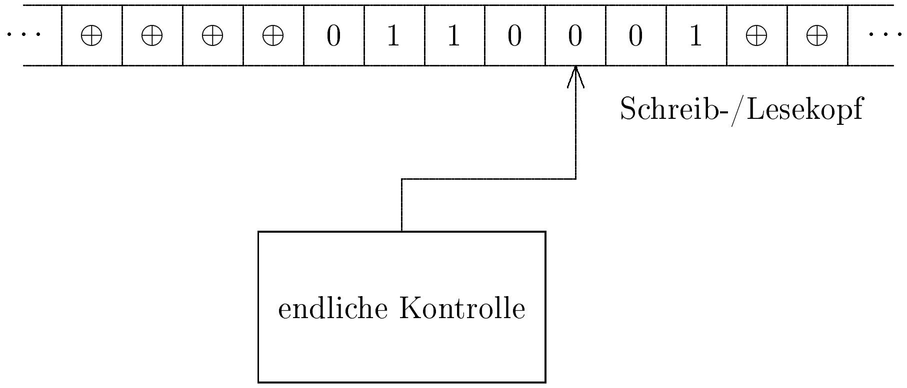
Bemerkung: Es gilt :

1. Deterministisch kontextfrei Sprachen stimmen mit LR(k)-Sprachen überein.
2. Für deterministische kontextfreie Sprachen ist das Wortproblem in der Zeit $O(n)$ lösbar.

4 Kontextsensitive und Typ 0–Sprachen

Wir wollen nun Typ 1– und Typ 0–Sprachen durch geeignete Berechnungsmodelle charakterisieren.

Turing-Maschine:



Definition
Eine nichtdeterministische Turing Maschine ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, \oplus, F)$ mit:

Q	$=$	endliche *Zustandsmenge*
Σ	$=$	endliches *Eingabealphabet*
$\Gamma \supset \Sigma$	$=$	*Arbeitsalphabet*
δ	$:$	$Q \times \Gamma \longrightarrow P(Q \times \Gamma \times \{L, N, R\})$ *Überführungsfunktion*
q_0	$:$	*Startzustand*
$\oplus$	$\in$	$\Gamma - \Sigma$ *Blank* (leeres Bandsymbol)
$F \subseteq Q$	$=$	Menge der *Endzustände*

M heißt <u>deterministische</u> TM
$=_{def}$ die Überführungsfunktion δ ist eine Funktion
$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, N, R\}$

<u>Vorstellung:</u> $(q', b, r) \in \delta(q, a)$
bedeutet: im Zustand q geht M bei gelesenem Symbol a in Zustand q' über, überschreibt a durch b und führt eine Bewegung $r \in \{L, N, R\}$ durch, wobei:

L	$=$	ein Schritt nach links
N	$=$	keine Bewegung
R	$=$	ein Schritt nach rechts

Definition
Eine <u>Konfiguration</u> der TM M ist ein Wort $k \in \Gamma^* Q \Gamma^*$
<u>Vorstellung:</u> Konfigurationen sind „Momentaufnahmen“
<u>$k = \alpha q \beta$</u> bedeutet:
$\alpha\beta$ ist nichtleerer bzw. bereits besuchter Teil des Bandes, q ist Zustand und der Lese-Schreibkopf steht auf dem ersten Symbol von β.

Startkonfiguration bei Eingabe $w \in \Sigma^*$: $q_0 w$

$\ldots \oplus \ldots \oplus w_1 \ldots w_n \oplus \ldots \oplus \ldots$

Definition
Die Relation $\mapsto$ wird auf der Menge der Konfigurationen von M so definiert:

$$a_1 \ldots a_m q b_1 \ldots b_n \mapsto \begin{cases} a_1 \ldots a_m q' c b_2 \ldots b_n, \\ \qquad \text{falls } (q', c, N) \in \delta(q, b_1) \text{ und } m \geq 0, n \geq 1 \\ a_1 \ldots a_m c q' b_2 \ldots b_n, \\ \qquad \text{falls } (q', c, R) \in \delta(q, b_1) \text{ und } m \geq 0, n \geq 2 \\ a_1 \ldots a_{m-1} q' a_m c b_2 \ldots b_n, \\ \qquad \text{falls } (q', c, L) \in \delta(q, b_1) \text{ und } m, n \geq 1 \\ a_1 \ldots a_m c q' \oplus, \\ \qquad \text{falls } (q', c, R) \in \delta(q, b_1) \text{ und } n = 1 \\ q' \oplus c b_2 \ldots b_n, \\ \qquad \text{falls } (q', c, L) \in \delta(q, b_1) \text{ und } m = 0 \end{cases}$$

Beispiel
Eine TM, die die Eingabe $w \in \{0,1\}^*$ als Binärzahl interpretiert und 1 hinzuaddiert.
$M = (\{q_0, q_1, q_2, q_e\}, \{0,1\}, \{0,1,\oplus\}, \delta, q_0, \oplus, \{q_e\})$ mit:

$$\begin{array}{lcl} \delta(q_0, 0) & = & (q_0, 0, R) \\ \delta(q_0, 1) & = & (q_0, 1, R) \\ \delta(q_0, \oplus) & = & (q_1, \oplus, L) \\ \delta(q_1, 0) & = & (q_2, 1, L) \\ \delta(q_1, 1) & = & (q_1, 0, L) \\ \delta(q_1, \oplus) & = & (q_e, 1, N) \\ \delta(q_2, 0) & = & (q_2, 0, L) \\ \delta(q_2, 1) & = & (q_2, 1, L) \\ \delta(q_2, \oplus) & = & (q_e, \oplus, R) \end{array}$$

Betrachte etwa $w = 101$:

$$\begin{array}{ll} q_0 101 & \mapsto 1 q_0 01 \mapsto 10 q_0 1 \mapsto 101 q_0 \oplus \\ & \mapsto 10 q_1 1 \oplus \mapsto 1 q_1 00 \oplus \mapsto q_2 110 \oplus \\ & \mapsto q_2 \oplus 110 \oplus \mapsto \oplus q_e 110 \oplus \end{array}$$

Definition
M sei eine TM, $M = (Q, \Sigma, \Gamma, \delta, q_0, \oplus, F)$.
$L(M) := \{w \in \Sigma^* \mid q_0 w \mapsto^* \alpha q \beta,\ \alpha, \beta \in \Gamma^*, q \in F\}$ ist die von M akzeptierte Sprache.

Für die Charakterisierung der Typ 1–Sprachen betrachten wir *linear beschränkte* Turing-Maschinen.

Definition
Eine nichtdeterministische TM heißt linear beschränkt (LBA)
$=_{def}$ für alle $a_1 \dots a_n \in \Sigma^*$ und alle Konfigurationen $\alpha q \beta$
mit $q_0 a_1 \dots a_{n-1} \hat{a_n} \mapsto^* \alpha z \beta$
gilt: $|\alpha\beta| = n$

Statt Σ wird das Alphabet Σ' benutzt, $\Sigma' = \Sigma \cup \{\hat{a},\ a \in \Sigma\}$. Wir repräsentieren die Eingabe $a_1 \dots a_n$ durch $a_1 \dots \hat{a_n}$, um das Ende der Eingabe zu markieren. Den Anfang kann man zu Beginn der Arbeit markieren.

LBA (= Linear Bounded Automaton): wir verlassen den Teil des Bandes, auf dem die Eingabe steht, nicht.

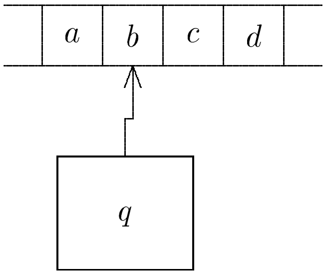
Satz
Die von linear beschränkten nichtdeterministischen TM akzeptierten Sprachen sind genau die kontextsensitiven Sprachen (Typ 1).

<u>Beweis:</u>
$\Leftarrow$)
Sei A Typ 1–Sprache, $A = L(G)$, $G = (V, \Sigma, P, S)$; wir beschreiben eine TM M mit $L(M) = A$:
Bei der Eingabe von $w = a_1 \dots a_n$ wählt M nichtdeterministisch eine Produktion $u \longrightarrow v \in P$, dann sucht M ein beliebiges Vorkommen von v; falls M ein solches Vorkommen findet, ersetzt M v durch u (u ist nicht länger als v, da G kontextsensitiv; ist u kürzer, so werden alle Bandsymbole rechts von u entsprechend nach links verschoben).
Enthält der nichtleere Teil des Bandes nur noch das Startsymbol S, dann stoppt M in einem Endzustand, ansonsten werden die nichtdeterministischen Ersetzungsschritte wiederholt.

Es gilt: $w \in L(G)$	$\Longleftrightarrow$	es existiert Ableitung $S \Rightarrow \dots \Rightarrow w$
	$\Longleftrightarrow$	es existiert eine Berechnung von M, die diese Ableitung in umgekehrter Richtung simuliert
	$\Longleftrightarrow$	$w \in L(M)$

Da für $u \longrightarrow v \in P$ gilt: $|u| \leq |v|$, ist M linear beschränkt.

$\Rightarrow$)
Sei $A = L(M)$, M ein LBA.
Wir beschreiben eine Grammatik, die auf Wörtern, die Konfigurationen von M darstellen, operiert:
Betrachte Alphabet $\Delta = \Gamma \cup (Q \times \Gamma)$
(damit wir Konfigurationen durch Wörter der Länge n darstellen können).
Stelle Konfiguration



dar als: $a(q,b)cd$.
Es gilt: $|a(q,b)cd|_\Delta = 4 = |abcd|$.
Wir können nun δ-Übergänge von M

$$(q', b, L) \in \delta(q, a)$$

durch kontextsensitive Produktion beschreiben:

$$c(q,a) \longrightarrow (q', c)b \qquad \forall c \in \Gamma$$

und erhalten die Produktionenmenge P'.
Gilt $k \mapsto^* k'$ für M, dann ist $\hat{k} \Rightarrow^* \hat{k'}$ in P', wobei $\hat{k}$ die obige Darstellung von k bezeichnet.
Wir erhalten $G = (V, \Sigma, P, S)$ mit $L(G) = A$ und $V = (\{S, K\} \cup (\Delta \times \Sigma))$

$$\begin{array}{lll}
(1)\ P & = & \{S \longrightarrow K(\hat{a}, a) \mid a \in \Sigma\} \\
(2) & \cup & \{K \longrightarrow K(a,a) \mid a \in \Sigma\} \\
(3) & \cup & \{K \longrightarrow ((q_0, a)a) \mid a \in \Sigma\} \\
(4) & \cup & \{(\alpha_1, a)(\alpha_2, b) \longrightarrow (\beta_1, a)(\beta_2, b) \mid \ \alpha_1\alpha_2 \longrightarrow \beta_1\beta_2 \in P'; a, b \in \Sigma\} \\
(5) & \cup & \{((q,a),b) \longrightarrow b \mid q \in F, a \in \Gamma, b \in \Sigma\} \\
(6) & \cup & \{(a,b) \longrightarrow b \mid a \in \Gamma, b \in \Sigma\}
\end{array}$$

(1), (2) und (3) ermöglichen Ableitungen der Form:
$S \Rightarrow^* ((q_0, a_1), a_1)(a_2, a_2) \ldots (a_{n-1}, a_{n-1})(\hat{a_n}, a_n)$,
(4) simuliert die Rechnung von M mittels P' auf der ersten Komponente, bis ein Endzustand erreicht wird:
$\ldots \Rightarrow^* (\gamma_1, a_1) \ldots (\gamma_{k-1}, a_{k-1})((q, \gamma_k), a_k)\ (\gamma_{k+1}, a_{k+1}) \ldots (\gamma_n, a_n)$
mit $q \in F$ Endzustand, $\gamma_i \in \Gamma$, $a_i \in \Sigma$.
Mit (5) und (6) können die jeweiligen ersten Komponenten gelöscht werden:
$\ldots \Rightarrow^* a_1 \ldots a_n$.
Offenbar sind alle Regeln Produktionen vom Typ 1.

Bemerkung: Nichtdeterminismus ist bei LBA Maschinen wichtig.
Es ist unklar, ob deterministische LBAs gleiche Leistungskraft besitzen wie nichtdeterministische LBAs.
(„LBA-Problem:“ LBA = DLBA?)

Satz
Die durch allgemeine TM akzeptierten Sprachen sind genau die Typ 0–Sprachen.

Beweis: Analog, wie für Typ 1.