Automatentheorie und formale Sprachen Vorlesung – Prof. Dr. K. Indermark – Sommersemester 2001

geTEX't von Matthias Hensler (mh@wspse.de)

1. September 2001

Inhaltsverzeichnis

Kapitel 1

Alphabete, Wörter, Sprachen

1.1 Einführung

Zeichenreihe als Grundobjekt der Informatik

- Präzisierung des Algorithmenbegriffs durch Turing-Maschinen
- Kommunikation mit dem Rechner über Tastatur
- automatische Datenverarbeitung (Bitfolgen)

Grundbegriffe: Σ-Alphabet, nicht-leere endliche Menge.
Beachte: Elemente eines Alphabets können aus Zeichen zusammengesetzt sein (Symbole in Programmiersprachen).

Σ	Alphabet
$a \in \Sigma$	Buchstaben, Zeichen, Charakters, Symbol
Σ^*	Menge der Wörter über Σ, $\Sigma^* := \{(a_1 a_2 \dots a_n) \mid a_i \in \Sigma, n \in \mathbb{N}\}$
$w \in \Sigma^*$	Wort einer Sprache
L	Sprache, Menge von Wörtern
ε	leeres Wort $(n = 0)$

Beispiel: Bitstrings, Dezimalzahl, Programme, Textdatei

1.2 Operationen auf Menge von Worten

- Verkettung/Concatenation

$$\begin{array}{rcl} \_\cdot\_ : \Sigma^* \times \Sigma^* & \to & \Sigma^* \\ (w, v) & \to & w \cdot v := wv \end{array}$$

  Es gilt:

  - Verkettung ist assoziativ
  - ε ist neutrales Element bezüglich „ $\cdot$ “: $v \cdot \varepsilon = \varepsilon \cdot v = v$

Sprachweise: $\langle \Sigma^*, \_ \cdot \_, \varepsilon \rangle$ (Monoid, Halbgruppe mit „ 1 “)

- Länge eines Wortes
  $w = a_1 a_2 \ldots a_n \in \Sigma^* \Rightarrow |w| = n \qquad (n \in \mathbb{N})$
  $|\varepsilon| = 0$
  $|w \cdot v| = |w| + |v|$

- Potenzen eines Wortes
  $w^0 := \varepsilon \qquad \Leftarrow$ muß neutrales Element sein
  $w^{m+1} := ww^m$

- „ Spiegelbild “ eines Wortes
  $\varepsilon^R := \varepsilon$
  $(wa)^R := a(w)^R$

1.3 Grundbegriffe

$\mathfrak{p}(\Sigma^*)$ Potenzmenge von Σ^*, Menge der formalen Sprachen über Σ
$\mathfrak{p}(\Sigma^*) := \{L \,|\, L \subseteq \Sigma^*\}$

$$\left.\begin{array}{l} \emptyset \neq \{\varepsilon\} \\ \{w_1, \ldots, w_n\}, \Sigma^* \\ \text{Menge der Java} - \text{Programme} \\ \text{Menge der URLs} \\ \text{Menge der HTML} - \text{Dateien} \end{array}\right\} \text{Sprachen } \in \mathfrak{p}(\Sigma^*)$$

1.4 Operationen auf $\mathfrak{p}(\Sigma^*)$

boolsche Operationen: $L_1 \cup L_2, L_1 \cap L_2, \overline{L} := \Sigma^* \backslash L$

Komplexprodukt: $L_1 \cdot L_2 := \{w_1 \cdot w_2 | w_i \in L_i\}$
Beachte: „Jeder mit jedem“, $L \cdot L := \{w \cdot v | w, v \in L\}$ (nicht $\{w^2 | w \in L\}$)!

Potenzen einer Sprache: $L^0 := \{\varepsilon\}$ weil $\{\varepsilon\}L = L$
$L^{m+1} = LL^m$

Stern einer Sprache: (Iteration / Repetition)
$L^* := \bigcup_{n \in \mathbb{N}} L^n$

Spiegelbild einer Sprache: $L^R := \{w^R | w \in L\}$

Kapitel 2

Reguläre Ausdrücke und endliche Automaten

2.1 Reguläre Ausdrücke

endliche Beschreibung unendlicher Sprachen mit (Technik) Hilfe regulärer Operationen.

wichtig: Stern-Operator

2.1.1 Definition (Syntax)

Sei Σ ein Alphabet. Die Menge $\text{Reg}E(\Sigma)$ der *regulären Ausdrücken über* Σ ist induktiv definiert durch:

1. $\Lambda \in \text{Reg}E(\Sigma)$
2. $a \in \text{Reg}E(\Sigma)$ für jedes $a \in \Sigma$
3. $(\alpha \vee \beta) \in \text{Reg}E(\Sigma)$ (Alternation)
4. $(\alpha \cdot \beta) \in \text{Reg}E(\Sigma)$ (Concatenation)
5. $(\alpha^*) \in \text{Reg}E(\Sigma)$ (Repetiton)

2.1.2 Vereinfachte Schreibweise

- Präzedenzregeln, um Klammern zu sparen:
  $*$ bindet stärker als $\cdot$
  $\cdot$ bindet stärker als $\vee$
- Der $\cdot$ wird auch weggelassen.

Beispiel: $a \vee b^*c$ statt $(a \vee ((b^*) \cdot c))$.
Beachte: $\text{Reg}E(\Sigma)$ ist selbst eine formale Sprache.
$\text{Reg}E(\Sigma) \subseteq (\Sigma \cup \{\Lambda, (,), \cdot, \vee, ^*\})^*$

2.1.3 Definition (Semantik von $\mathrm{Reg}E(\Sigma)$)

Ein regulärer Ausdruck α beschreibt eine formale Sprache. Statt $L(\alpha)$ schreiben wir hier $[\![\alpha]\!]$
$[\![.]\!] : \mathrm{Reg}E(\Sigma) \to \mathfrak{p}(\Sigma^*)$

- $[\![\Lambda]\!] := \emptyset$
- $[\![a]\!] := \{a\}$
- $[\![(\alpha \cup \beta)]\!] := [\![a]\!] \cup [\![\beta]\!]$
- $[\![(\alpha \cdot \beta)]\!] := [\![a]\!][\![\beta]\!]$
- $[\![(\alpha^*)]\!] := [\![a]\!]^*$

Sprechweise: für $w \in [\![\alpha]\!]$, w ist ein Match für α, α ein Muster (Pattern).
Beachte: Unterschied zwischen Syntax und Semantik

$$[\![a^*]\!] := \{a^n | n \in \mathbb{N}\}$$

Definition: Die Klasse $\mathrm{Reg}L(\Sigma)$ der *regulären Sprachen* über Σ ist induktiv definiert durch

- $\emptyset, \{a\} \in \mathrm{Reg}L(\Sigma)$ für alle $a \in \Sigma$
- $L, L' \in \mathrm{Reg}L(\Sigma) \Rightarrow L \cup L', LL', L^* \in \mathrm{Reg}L(\Sigma)$

Also gilt: $\mathrm{Reg}L(\Sigma) = [\![\mathrm{Reg}E(\Sigma)]\!]$
(siehe `man 7 regex` unter Unix)

2.1.4 Standardprobleme für $\mathrm{Reg}E(\Sigma)$

1. Wortproblem (matching problem)
2. Äquivalenzproblem
3. Leerheitsproblem

2.2 Deterministische endliche Automaten

Definition: Seien Q und Σ nicht-leere, endliche Mengen, $q_0 \in Q$, $F \subseteq Q$ und $\delta : Q \times \Sigma \to Q$.

Dann heißt: $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >$ ein *deterministischer endlicher Automat über* Σ mit

- der *Zustandsmenge* Q,
- dem *Eingabealphabet* Σ,
- der *Transitionsfunktion (Übertragungsfunktion)* δ
- dem *Anfangszustand* q_0 und
- der *Endzustandsmenge* F

Bezeichnung: $\mathfrak{A} \in \text{DFA}(\Sigma), \mathfrak{A} \in \text{DFA}(\text{det. finite aut.})$
Definition: $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \text{DFA}(\Sigma)$ bestimmt die *erweiterte Transitionsfunktion*

$$\overline{\delta} : Q \times \Sigma^* \to Q$$

mit

$$\overline{\delta}(q, \varepsilon) := q$$

$$\overline{\delta}(q, wa) := \delta(\overline{\delta}(q, w), a) \text{ für } w \in \Sigma^*, a \in \Sigma$$

und die von $\mathfrak{A}$ *erkannte Sprache*

$$L(\mathfrak{A}) : \left\{ w \in \Sigma^* | \overline{\delta}(q_0, w) \in F \right\}$$

Bezeichnung: $\mathscr{L}(\Sigma, \text{DFA})$ Klasse der von deterministischen endlichen Automaten erkennbaren Sprachen über Σ.
Ziel: $\mathscr{L}(\Sigma, \text{DFA}) = \text{Reg}\mathscr{L}(\Sigma)$ nachweisen.
wesentliches Hilfsmittel: *nicht deterministische Automaten*.

2.3 Nicht-deterministische endliche Automaten

Definition: Seien Q, Σ, q_0 und F wie bei einem $\mathfrak{A} \in \text{DFA}$, ferner $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$ und $\delta : Q \times \Sigma_\varepsilon \curvearrowright \mathfrak{p}(Q)$. Dann heißt

$$\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >$$

ein *nicht-deterministischer endlicher Automat über* Σ.
Bezeichnung: $\text{NFA}(\Sigma), \text{NFA}$. Es folgt: $\text{DFA}(\Sigma) \subseteq \text{NFA}(\Sigma)$.

Semantik eines $\mathfrak{A} \in \text{NFA}(\Sigma)$. Erweiterung der Semantik von DFA's.
Für $T \subseteq Q$ ist die ε-*Hülle* (bez: $\varepsilon(T)$), induktiv durch

- $T \subseteq \varepsilon(T)$
- $q \in \varepsilon(T) \curvearrowright \delta(q, \varepsilon) \subseteq \varepsilon(T)$

Die *erweiterte Transitionsfunktion*

$$\overline{\delta} : \mathfrak{p}(Q) \times \Sigma^* \to \mathfrak{p}(Q)$$

ist induktiv definiert durch

- $\overline{\delta}(T, \varepsilon) := \varepsilon(T)$
- $\overline{\delta}(T, w) := \varepsilon\left(\bigcup_{q \in \overline{\delta}(T,w)} \delta(q, a)\right)$

$$\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \text{DFA}(\Sigma)$$

$\delta : Q \times \Sigma \to Q$
$\overline{\delta} : Q \times \Sigma^* \to Q$
$\overline{\delta}(q, \varepsilon) := q$

$\overline{\delta}(q, wa) := \delta\left(\overline{\delta}(q, w), a\right)$

$L(\mathfrak{A}) := \{w \in \Sigma^* | \overline{\delta}(q_0, w) \in F\}$

$$\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \mathrm{NFA}(\Sigma)$$

$\delta : Q \times \Sigma_\varepsilon \to \mathfrak{p}(Q)$

$\overline{\delta} : \mathfrak{p}(Q) \times \Sigma^* \to \mathfrak{p}(Q)$
$\overline{\delta}(T, \varepsilon) = \varepsilon(T)$
$\overline{\delta}(T, wa) := \varepsilon\left(\bigcup_{q \in \overline{\delta}(T,w)} \delta(q, a)\right)$
Dann ist die durch $\mathfrak{A}$ erkannte Sprache definiert durch

$$L(\mathfrak{A}) := \{w \in \Sigma^* | \overline{\delta}(\{q_0\}, w) \cap F \neq \emptyset\}$$

Beachte: Für DFAs stimmen beide Definitionen überein.

Grund: $|\delta(q, a)| = 1$ für alle $q \in Q, a \in \Sigma$
$|\delta(q, \varepsilon)| = 0$ für alle $q \in Q, a \in \Sigma$

2.3.1 Der Potenzmengenautomat

Definition: $\mathfrak{A}^P :=< \hat{Q}, \Sigma, \hat{\delta}, \hat{q_0}, \hat{F} >\in \mathrm{DFA}(\varepsilon)$ ist der Potenzmengenautomat von $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \mathrm{NFA}(\varepsilon)$ wenn gilt:

- $\hat{Q} = \{T \subseteq Q \,|\, T = \overline{\delta}(\{q_0\}, w), w \in \Sigma^*\}$
- $\hat{\delta} : \hat{Q} \times \Sigma \to \hat{Q}$
  $\hat{\delta}(T, a) := \overline{\delta}(T, a)$[1]
- $\hat{q_0} := \varepsilon(\{q_0\})$
- $T \in \hat{F} \curvearrowright T \in \hat{Q}$ und $T \cup F \neq \emptyset$

Lemma: Für $\mathfrak{A} \in \mathrm{NFA}(\Sigma)$ gilt: $L(\mathfrak{A}) = L(\mathfrak{A}^P)$
Beweis: (einfach wegen Defintion von $\mathfrak{A}^P$)

$$\begin{aligned} w \in L(\mathfrak{A}) &\curvearrowleft \overline{\delta}(\{q_0\}, w) \cup F \neq \emptyset \\ &\curvearrowleft \overline{\hat{\delta}}(\{q_0\}, w) \in \hat{F} \\ &\curvearrowleft w \in L(\mathfrak{A}^P) \end{aligned}$$

□

[1] $T \subseteq \hat{Q} \curvearrowright T = \overline{\delta}(\{q_0\}, w)$
ferner gilt: $\overline{\delta}(T, \varepsilon) = \varepsilon(T) = T$
$\overline{\delta}(T, a) = \varepsilon\left(\bigcup_{q \in \overline{\delta}(T,\varepsilon)} \delta(q_0, a)\right) = \varepsilon\left(\bigcup_{q \in T} \delta(q, a)\right) = \varepsilon\left(\bigcup_{q \in \overline{\delta}(\{q_0\},w)} \delta(q, a)\right) = \overline{\delta}(\{q_0\}, wa) \in \hat{Q}$

2.4 Synthese und Analyse endlicher Automaten

2.4.1 Synthese

Konstruiere für $\alpha \in \mathrm{Reg}E(\Sigma)$ einen äquivalenten $\mathfrak{A}(\alpha) \in \mathrm{NFA}(\Sigma)$, d.h. $[\![\alpha]\!] = L(\mathfrak{A}(\alpha))$

Der Algorithmus von Thompson

Beh: $\mathfrak{A}(\alpha)$ hat genau 1 Endzustand $q_f \neq q_0$, q_0 ist Quelle, q_f ist Senke.

- $\mathfrak{A}(\Lambda) := \longrightarrow \underset{q_0}{\bullet} \qquad \circledcirc_{q_f}$
- $\mathfrak{A}(a) := \longrightarrow \underset{q_0}{\bullet} \xrightarrow{a} \circledcirc_{q_f}$
- $\mathfrak{A}(\alpha \vee \beta) := \longrightarrow \underset{q_0}{\bullet} \begin{array}{c} \overset{\varepsilon}{\nearrow} \\ \underset{\varepsilon}{\searrow} \end{array} \begin{array}{c} \circ \sim \mathfrak{A}(\alpha) \sim \circledcirc \\ \\ \circ \sim \mathfrak{A}(\beta) \sim \circledcirc \end{array} \begin{array}{c} \overset{\varepsilon}{\searrow} \\ \underset{\varepsilon}{\nearrow} \end{array} \circledcirc_{q_f}$
- $\mathfrak{A}(\alpha\beta) := \longrightarrow \underset{q_0}{\bullet} \sim \mathfrak{A}(\alpha) \sim \circ \sim \mathfrak{A}(\beta) \sim \circledcirc_{q_f}$ (*Beachte:* q_0 Quelle, q_f Senke)
- $\mathfrak{A}(\alpha^*) := \rightarrow \underset{q_0}{\bullet} \xrightarrow{\varepsilon} \circ \sim \mathfrak{A}(\alpha) \sim \circledcirc \xrightarrow{\varepsilon} \circledcirc_{q_f}$, mit ε-Kante von $\circledcirc$ zurück nach $\circ$ und ε-Kante von q_0 nach q_f

Offensichtlich gilt für jedes $\alpha \in \mathrm{Reg}E : [\![\alpha]\!] = L(\mathfrak{A}(\alpha))$.

Korollar: $\mathrm{Reg}L \subseteq \mathscr{L}(\Sigma, \mathrm{DFA})$ (endlicher Automat ist beschreibbar durch eine Turingmaschine, die sich nur nach rechts bewegen kann).

2.4.2 Analyse

Konstruiere für $\mathfrak{A} \in \mathrm{DFA}(\Sigma)$ einen $\alpha(\mathfrak{A}) \in \mathrm{Reg}E(\Sigma)$ mit $[\![\alpha(\mathfrak{A})]\!] = L(\mathfrak{A})$.

$\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \mathrm{DFA}(\Sigma)$ und $Q = \{q_1, \ldots, q_n\}$.

Für $i, j \in \{1, \ldots, n\}$ und $k \in \{0, 1, \ldots, n\}$ definieren wir ${W_{ij}}^k := \{w \in \Sigma^* | w$ überführt q_i in q_j ohne Benutzung von $q_{k+1}, \ldots, q_n$ als Zwischenzustände.

Dann gilt: $L(\mathfrak{A}) = \bigcup_{q_j \in F} {W_{ij}}^n$

Somit genügt der Nachweis der Regularität der ${W_{ij}}^k$. Dies zeigen wir nun durch Induktion über k:

$k = 0$**:** ${W_{ij}}^0 \subseteq \Sigma \cup \{\varepsilon\}$ (beachte: $\{\varepsilon\} = [\![\Lambda^*]\!]$)
$\curvearrowright {W_{ij}}^0$ ist regulär.

$k-1 \to k$**:** siehe Folie

Korollar: (Satz von Kleene) $\mathscr{L}(\Sigma, \text{DFA}) = \text{Reg}L(\Sigma)$
Korollar: $\text{Reg}L(\Sigma)$ ist abgeschlossen unter $\cap$ und $^{-}$.

Beweis: (Idee):

Komplement: $F' : Q \backslash F$

$L_1 \cap L_2$**:** $\overline{\overline{L_1} \cup \overline{L_2}}$

□

2.5 Das Pumping-Lemma

Hilfsmittel zum Nachweis nicht-regulärer Sprachen.

Satz: (Pumping-Lemma, Iterationslemma)
Sei $L \in \text{Reg}L(\Sigma)$. Dann existiert $k \in N$, so daß für jedes $x \in L$ mit $|x| \geq k$ eine Zerlegung

$$x = uvw$$

mit folgenden Eigenschaften existiert:

1. $v \neq \varepsilon$
2. $|uv| \leq k$
3. $uv^iw \in L$ für alle $i \in \mathbb{N}$, insbesondere für $i = 0$.

Beweis: Sei $\mathfrak{A} \in \text{DFA}(\Sigma)$ mit $L(\mathfrak{A}) = L$ und $k = |Q|$.
Sei $x \in L$ mit $|x| \geq k$. Dann wird beim „Erkennungslauf" von x in $\mathfrak{A}$ mindestens 1 Zustand mehr als einmal besucht.

$$\to \underset{q_0}{\bullet} \xrightarrow{u} \underset{i}{\overset{\overset{v}{\curvearrowleft}}{\bullet}} \xrightarrow{w} \circledcirc$$

Sei i der erste solche „Iterationszustand" und v das Teilwort von dort bis zur ersten Wiederholung. Dann gelten offensichtlich $1-3$.

□

Beachte: Das Pumping-Lemma beschreibt eine notwendige, aber nicht hinreichende Eigenschaft $\rightsquigarrow$ Hilfsmittel zum Nachweis nicht regulärer Sprachen.

Beispiel: $L = \{a^n b^n | n \geq 1\} \notin \text{Reg}L(\{a, b\})$
Beweis: Angenommen, $L \in \text{Reg}L$. Dann existiert $k \in \mathbb{N}$ mit den Eigenschaften des Pumping-Lemma. („Pumping Index").
Für $x = a^k b^k$ muß dann eine Zerlegung existieren:

$$a^k b^k = uvw$$

mit $v \neq \varepsilon$ und $|uv| \leq k$, hieraus folgt sofort $v \in \{a^i | i \geq 1\}$ und $uw = a^{k-|v|} b^k \notin L$. Widerspruch!

□

2.6 Zustandsreduktion endlicher Automaten

Ziel: Konstruktion endlicher Automaten mit minimaler Zustandszahl

2.6.1 Ableitung

Definition: Für $L \subseteq \Sigma^*$ und $w \in \Sigma^*$ heißt

$$d_w(L) := \{v \in \Sigma^* | wv \in L\}$$

die *Ableitung von L nach w*.

Lemma: Sei $L = L(\mathfrak{A})$ für $\mathfrak{A} =< Q, \ldots >\in \mathrm{DFA}(\Sigma)$. Dann gilt für $D(L) := \{d_w(L)|w \in \Sigma^*\} : |D(L)| \leq |Q|$
Beweis: Für $q \in Q$ bezeichne

$$L(q) := L < Q, \Sigma, \delta, \underset{!}{q}, F >$$

Dann gilt: $d_w(L) = d_w(L(q_o)) = L\left(\overline{\delta}(q_0, w)\right)$.

2.6.2 Der Ableitungsautomat

Definition: Der *Ableitungsautomat* $\mathfrak{A}_L =< D(L), \Sigma, \delta, q_0, F >\in \mathrm{DFA}(\Sigma)$ ist für $L \in \mathrm{Reg}L$ definiert durch

- $q_0 := d_\varepsilon(L) = L$
- $F := \{d_w(L)|w \in L\}$ also: $\varepsilon \in d_w(L)$
- $\delta(d_w(L), a) := d_{wa}(L)$

Beachte: $d_w(L) = d_v(L) \curvearrowright d_{wa}(L) = d_{va}(L)$
Also ist die Definition von δ unabhängig vom Repräsentanten w.

Lemma: $L(\mathfrak{A}_L) = L$ ($\mathfrak{A}_L$ erkennt L).
Beweis: $w \in L(\mathfrak{A}_L) \curvearrowright \overline{\delta}(q_0, w) \in F$
$\curvearrowright d_w(L) \in F \curvearrowright w \in L$

□

Korollar:

1. Der Ableitungsautomat ist zustandsminimal.
2. Für $L \subseteq \Sigma^*$ gilt: $L \in \mathrm{Reg}L(\Sigma) \curvearrowright D(L) < \infty$

Zustandsreduktion: Konstruktion eines zustandsminimalen Automaten aus einem gegebenen Automaten durch

- Weglassen nicht erreichbarer Zustände
- Verschmelzen äquivalente Zustände

Dabei heißt:

- $q \in Q$ *erreichbar* $:\curvearrowleftright \exists w \in \Sigma^* : \overline{\delta}(q_0, w) = q$
- $q_1 \sim q_2$ (*äquivalent*): $\curvearrowleftright L(q_1) = L(q_2)$

2.6.3 Der Faktorautomat

Definition: Für $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in \mathrm{DFA}(\Sigma)$ ist der *Faktorautomat*

$$\mathfrak{A}/\sim := < \tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q_0}, \tilde{F} > \in \mathrm{DFA}(\Sigma)$$

wie folgt definiert:
Für $q \in Q$ sei $[q] := \{q' | q' \sim q\}$

- $\tilde{Q} := \{[\overline{\delta}(q_0, w)] | w \in \Sigma^*\}$
- $\tilde{q_0} := [q_0]$
- $\tilde{F} = \{[q] | q \in F\} \cap \tilde{Q}$
- $\tilde{\delta}([q], a) := [\delta(q, a)]$

Beachte: $q \in F.q \sim q' \curvearrowright q' \in F$ und $q \sim q' \curvearrowleftright \delta(q,a) \sim \delta(q', a)$ zeigen die Unabhängigkeit der Definition von den Repräsentanten.
Lemma: Für $\mathfrak{A} \in \mathrm{DFA}(\Sigma)$ gilt: $\mathfrak{A}/\sim = \mathfrak{A}_{L(\mathfrak{A})}$ (bis auf Zustandsnamen).

Insbesondere ist $\mathfrak{A}/\sim$ äquivalent zu $\mathfrak{A}$ und es folgt: Zustandsminimale Automaten sind bis auf Isomorphie eindeutig bestimmt.

Beweis: $\beta\left([\overline{\delta}(q_0, w)]\right) := d_w(L(\mathfrak{A}))$

- $\underline{\beta}$ ist unabhängig vom Repräsentanten $w \in \Sigma^*$:
  $\overline{\delta}(q_0, v) \sim \overline{\delta}(q_1, w) \curvearrowright L(\overline{\delta}(q_0, v)) = L(\overline{\delta}(q_1, w))$
  $\curvearrowright d_w(L(\mathfrak{A})) = d_v(L(\mathfrak{A}))$
- β ist bijektiv, weil die obige Folgerung umkehrbar ist.
- β ist strukturerhaltent:
  - $\beta([q_0]) = \beta\left([\overline{\delta}(q_0, \varepsilon)]\right) = d_\varepsilon(L(\mathfrak{A}))$ „Anfangszustand im Ableitungsautomaten"
  - $\overline{\delta}(q_0, w) = q \in F : \beta\left([\overline{\delta}(q_0, w)]\right) = d_w(L(\mathfrak{A}))$ „Endzustand vom Ableitungsautomaten", weil $w \in L(\mathfrak{A})$.
  - $\tilde{\delta}([q], a) = [\tilde{\delta}(q, a)] \curvearrowright \tilde{\delta}\left([\overline{\delta}(q_o, w)], a\right) = [\overline{\delta}(q_0, wa)] \curvearrowright \delta(d_w(L(\mathfrak{A})), a) = d_{wa}(L(\mathfrak{A}))$

□

Zustandsreduktion

- Weglassen nicht erreichbarer Zustände
- Verschmelzen äquivalenter Zustände

k-Äquivalenz und k-Äquivalenzmatrizen

Definition: Sei $\mathfrak{A} = < Q, \Sigma, \delta, q_0, F > \in$ DFA und jeder Zustand erreichbar;
ferner $k \in \mathbb{N}, q_1, q_2 \in Q$.
Dann sagt man:

- q_1 *k-äquivalent* q_2 $\left(q_1 \overset{k}{\sim} q_2\right)$:
  $\curvearrowleft\!\curvearrowright \forall w \in \Sigma^*, |w| \leq k : w \in L(q_1) \curvearrowleft\!\curvearrowright w \in L(q_2)$

- $\mathfrak{A}$ induziert die Abbildungen $r^k : Q^2 \to \{0,1\}$ mit $r^k(q_1,q_2) = 1 : \curvearrowleft\!\curvearrowright q_1 \overset{k}{\sim} q_2$. Sie können als Matrizen $R^k = \left(r^k(q_i,q_j)\right)_{i,j=1}^{n}$ mit $n = |Q|$ dargestellt werden.
  Beachte: $r^k(q_i,q_j) = r^k(q_j,q_i)$

Berechnung der k-Äquivalenz Matrizen

- $q_1 \overset{0}{\sim} q_2 \curvearrowleft\!\curvearrowright (q_1 \in F \curvearrowleft\!\curvearrowright q_2 \in F)$

- $q_1 \overset{k+1}{\sim} q_2 \curvearrowleft\!\curvearrowright q_1 \overset{0}{\sim} q_2$ und $\delta(q_1,a) \overset{k}{\sim} \delta(q_2,a)$ für alle $a \in \Sigma$

Lemma: Es gibt ein $k \in \mathbb{N}$, so daß für alle $n \in \mathbb{N}$ gilt: $R^k = R^{k+n}$
Beweis: Da $r^k(q_i,q_0) = 0 \curvearrowright r^{k+1}(q_i,q_0) = 0$, muß es ein $k \in \mathbb{N}$ geben, mit $r^k = r^{k+1}$. Dann muß auch $r^k = r^{k+n}$ für alle $n \in \mathbb{N}$ gelten:

Sei $r^k(q_1,q_2) = r^{k+1}(q_1,q_2) = 1$, also $q_1 \overset{k}{\sim} q_2$ und $q_1 \overset{k+1}{\sim} q_2$
Sei ferner $a_1 \ldots a_{k+2} \in L(q_1)$.
$\delta(q_1,a_1) \overset{k}{\sim} \delta(q_2,a_1)$, also auch $\delta(q_1,a_1) \overset{k+1}{\sim} \delta(q_2,a_1)$.
Somit $a_2 \ldots a_{k+2} \in L(\delta(q_2,a_1))$ und $a_1 \ldots a_{k+2} \in L(q_2)$.
Wegen Symmetrie folgt $q_1 \overset{k+2}{\sim} q_2$

□

Markierungsalgorithmus

Eingabe: $\mathfrak{A} = < Q, \Sigma, \delta, q_0, F > \in$ DFA, jeder $q \in Q$ erreichbar.
Ausgabe: äquivalente Zustände.

Verfahren:

- Tabelle aller Zustandspaare $\{q, q'\}$ mit $q \neq q'$

- Markiere alle Paare $\{q, q'\}$ mit $q \in F$ und $q' \notin F$ oder umgekehrt.

- Für jedes unmarkierte Paar $\{q, q'\}$ und $a \in \Sigma$ teste, ob $\{\delta(q,a), \delta(q',a)\}$ markiert. Wenn ja: markiere $\{q, q'\}$.

- Wiederhole den letzten Schritt, bis keine Änderung mehr erfolgt.

- Unmarkierte Paare repräsentieren äquivalente Zustände.

Bemerkung: Implementierung mit $\rho(|Q|^2)$ Zeitkomplexität möglich.

2.7 Entscheidbare Eigenschaften

2.7.1 Deterministische endliche Automaten

- Wortproblem: $w \in L(\mathfrak{A})$? für $w \in \Sigma^*$ und $Q \in \text{DFA}(\Sigma)$
  Durch Eingabe in $|w|+1$ Schritten entscheidbar.

- $\emptyset$-Problem (Leerheitsproblem): $L(\mathfrak{A}) = \emptyset$? für $\mathfrak{A} \in \text{DFA}(\Sigma)$
  Durch Eingabe alle Wörter w mit $|w| < |Q|$ entscheidbar.
  Beachte: $w \in L(\mathfrak{A}), |w| \geq |Q| \curvearrowright \exists v \in L(\mathfrak{A}), |v| < |w|$ (Vergleiche Beweis zum Pumping-Lemma [2.5])
  Verfahren: testen, ob F von q_0 erreichbar. Durchführung in $\rho(|Q|^2)$-Zeit.

- $\sim$-Problem: $L(\mathfrak{A}) = L(\mathfrak{A}')$? für $\mathfrak{A}, \mathfrak{A}' \in \text{DFA}(\Sigma)$
  Reduktion auf $\emptyset$-Problem:
  $L(\mathfrak{A}) = L(\mathfrak{A}') \curvearrowleft\!\!\curvearrowright L(\mathfrak{A}) \subseteq L(\mathfrak{A}')$ und $L(\mathfrak{A}') \subseteq L(\mathfrak{A}) \curvearrowleft\!\!\curvearrowright L(\mathfrak{A}) \cap \overline{L(\mathfrak{A}')} = \emptyset$ und $L(A') \cap \overline{L(\mathfrak{A})} = \emptyset$
  2 Automaten mit $|Q| \cdot |Q'|$ Zuständen auf $\emptyset$ testen. $\sim$-Problem in $\rho\left((|Q| \cdot |Q'|)^2\right)$-Zeit lösbar.

2.7.2 Reguläre Ausdrücke, nicht-deterministische Automaten

Durch Transformation in DFA's sind alle 3 Probleme entscheidbar. Aber der Aufwand steigt:

Wortproblem: $\rho(|\alpha| \cdot |w|)$ Zeit, bzw. $\rho(|Q| \cdot |w|)$.

$\emptyset$-Problem: NFA wie DFA.
RegE $\mapsto$ NFA in $\rho(|\alpha|)$ Zeit mit $|Q| \leq 2|\alpha|$.

$\sim$-Problem: NP-hart.
Grund: Exponentieller Aufwand der P-Mengen-Konstruktion.

2.8 Anwendung

1. Endliche Automaten mit Ausgabe.
   Beispiel: Mealy-Automaten, Moore-Automaten.
   Beschreibung von Schaltwerken.

2. Wortsuche in Texten (siehe Folie 2.15).
   2 Implementierungstechniken:

   (a) NFA-Methode:
   $u = aebwewebba$ bestimmt einen Lauf.
   $\{a\}a\{1\}e\{1,5\}b\{1,6\}w\{1,2\}e\{1,3,5\}w\{1,2\}e\{1,3,5\}b\{1,4,6\}\ldots$

   (b) DFA-Methode

3. Verteilte Systeme / Parallele Prozesse (siehe Folie 2.17)

4. Erweiterte reguläre Ausdrücke: $\cap, ^-$
   Wichtig für Suchmaschinen.

5. Zusammenhang zwischen endlichen Automaten und iterativen Programmen (Flußdiagramme) bzw. regulären Ausdrücken und `WHILE` -Programmen.

Kapitel 3

Kontextfreie Grammatiken und Kellerautomaten

Anwendung: Compilerbau (Syntaxanalyse), `XML`.

3.1 Kontextfreie Grammatiken

Definition: Seien N und Σ nicht-leere endliche Mengen mit $N \cap \Sigma = \emptyset$. Sei $P \subseteq N \times (N \cup \Sigma)^*$ mit $|P| < \infty$ und $S \in N$.
Dann heißt $G =< N, \Sigma, P, S >$ eine *kontextfreie Grammatik*.
Bezeichnung: $G \in \mathrm{CFG}(\Sigma)$ oder $G \in \mathrm{CFG}$.

3.1.1 Bezeichnungen und Konventionen

$A, B, C, \ldots \in N$	Nichtterminalsymbole
$a, b, c, \ldots \in \Sigma$	Terminalsymbole
$S \in N$	Startsymbole
$X, Y, Z, \ldots \in X := N \cup \Sigma$	Symbole
$\alpha, \beta, \gamma, \ldots X^*$	Satzformen
$u, v, w, \ldots \in \Sigma^*$	Terminalwörter
$A \to \alpha := (A, \alpha) \in P$	Regel, Produktion

Definition: Sei $G =< N, \Sigma, P, S > \in \mathrm{CFG}$ und $\pi = A \to \alpha \in P$.
π bestimmt eine *Ableitungsrelation*

$$\Rightarrow_\pi \subseteq X^* \times X^*$$

mit $\beta_1 \Rightarrow_\pi \beta_2$: es existiert ein Kontext $\gamma_1, \gamma_2 \in X^*$, so daß $\beta_1 = \gamma_1 A \gamma_2$ und $\beta_2 = \gamma_1 \alpha \gamma_2$.

Dann heißt das Tripel $(\gamma_1, \pi, \gamma_2)$ auch *Ableitungsschritt*. G bestimmt die *Ableitungsrelation* durch

$$\Rightarrow_G := \bigcup_{\pi \in P} \Rightarrow_\pi$$

und damit die *von G erzeugte Sprache* $L(G) := \{w \in \Sigma^* | S \overset{*}{\Rightarrow}_G w\}$ mit $\overset{*}{\Rightarrow}_G := \bigcup_{n=0}^{\infty} \overset{n}{\Rightarrow}_G$ (reflexive und transitive Hülle).

Es folgt: $w \in L(G) \curvearrowleft\!\!\curvearrowright \exists \alpha_0, \alpha_1, \ldots, \alpha_n \in X^*$ und $\pi_1, \ldots, \pi_n \in P : S = \alpha_0 \Rightarrow_{\pi_1} \alpha \cdots \Rightarrow_{\pi_n} \alpha_n = w \; (n \geq 1)$.
Bezeichnung: $\mathrm{CFL}(\Sigma) := \{L \subseteq \Sigma^* \,|\, \exists G \in \mathrm{CFG}(\Sigma) : L(G) = L\}$

3.1.2 Ableitungsbäume

Sei $G =< N, \Sigma, P, S >\in$ CFG.
Darstellung von Ableitungen durch Bäume.

1. Eine Regel $\pi = A \to X_1 \ldots X_n$ bestimmt den *Regelbaum* $\overset{A}{\underset{X_1 \ldots X_n}{/\backslash}}$, Spezialfall: $\overset{A}{\underset{\varepsilon}{|}}$ für $n = 0$.

2. Eine Ableitung $A \Rightarrow \alpha_1 \ldots \Rightarrow \alpha_n$ bestimmt den *Ableitungsbaum* durch entsprechendes Verkleben der Regelbäume.

Folgerung: Ein Ableitungsbaum repräsentiert eine Klasse von Ableitungen, die sich nur in der Reihenfolge von Ableitungsschritten unterscheiden. Ein Ableitungsbaum repräsentiert die relevante syntaktische Struktur. Ein Compiler bestimmt zu einem Programm (geben als Zeichenreihe) einen *Ableitungsbaum* als Basis der Codegenerierung.
(weitere Anwendungen: `HTML`, `XML`).

3.1.3 Rechts- und Linksableitung

Definition: Eine Ableitung heißt *Rechtsableitung* (*Linksableitung*), wenn jeder Ableitungsschritt (α, π, β) ein *Rechtsableitungsschritt*, d.h. $\beta \in \Sigma^*$ (, bzw. ein *Linksableitungsschritt*, d.h. $\alpha \in \Sigma^*$).

Schreibweise: $\underset{r}{\Rightarrow}$ bzw. $\underset{l}{\Rightarrow}$
Folgerung: Ein Baum hat genau eine Rechts- bzw. genau eine Linksableitung.
(siehe Folie 3.4)

3.1.4 Ein- und Mehrdeutigkeit

Definition:

- $G \in \mathrm{CFG}(\Sigma)$ heißt *eindeutig*: $\curvearrowleft\!\!\curvearrowright$ Zu jedem $w \in L(G)$ gibt es genau eine Rechtsableitung.

$$S \underset{r}{\Rightarrow} \alpha_1 \underset{r}{\Rightarrow} \ldots \underset{r}{\Rightarrow} w$$

- G heißt *mehrdeutig*: $\curvearrowleft\!\!\curvearrowright G$ nicht eindeutig.

Bemerkung: Syntaktische Mehrdeutigkeit (fehlende Klammern) wird durch Präzedenzregeln beseitigt.

3.2 Einseitig lineare Grammatiken

Definition: Sei $G =< N, \Sigma, P, S >\in$ CFG.

- Dann heißt diese Grammatik G *linkslinear*, wenn für jedes $\pi = A \to \alpha$ gilt: $\alpha = Bw$ oder $\alpha = w$.

- Gilt stattdessen $\alpha = wB$ oder $\alpha = w$ für jedes $\pi \in P$, so heißt G *rechtslinear*.
- G *einseitig linear*: $\curvearrowright G$ rechts- oder G linkslinear.

Ziel: $\{L(G)|G :, \text{linkslinear}\} = \{L(G)|G :, \text{rechtslinear}\} = \{L(G)|G \text{einseitig} - \text{linear}\} = \text{Reg}L$
Satz: $\mathscr{L}(\Sigma, \text{NFA}) = \{L \subseteq \Sigma^* | L = L(G), G \text{rechtslinear}\}$.
Beweis:

„$\supseteq$" Sei $G =< N, \Sigma, P, S >$ rechtslinear. Auffassung von G als $\mathfrak{A}(G) =< Q, \Sigma, \delta, q_0, F > \in$ NFA:
$Q := N \cup \{q_f\}$ $\quad$ (q_f neu)
$q_0 := S$ $F := \{q_f\}$
$\delta(A, w) \ni B$ falls $A \to wB \in P$
$\delta(A, w) \ni q_f$ falls $A \to w \in P$
Offensichtlich gilt: $L(G) = L(\mathfrak{A}(G))$.
Beachte: Wortübergänge durch Zwischenzustände beseitigen (siehe Folie 3.5).

„$\subseteq$" $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F > \in$ NFA kann ebenso als rechtslineare Grammatik aufgefasst werden.

$$\underset{q}{\circ} \xrightarrow{a} \underset{q'}{\circledcirc} \qquad \mapsto \qquad q \to aq' | q' \to \varepsilon$$

$q \in F \mapsto$ neue Regel $q \to \varepsilon$

□

Korollar: $\text{Reg}L(\Sigma) \subseteq \text{CFL}(\Sigma) := \mathscr{L}(\Sigma, \text{CFG})$

Für $|\Sigma| \geq 2$ ist diese Inklusion echt, weil $\{a^n b^n | n \in N\} = L(G)$ mit $G = (S \to aSb|\varepsilon)$.
Lemma: $L \in \text{Reg}L(\Sigma) \curvearrowright L^R \in \text{Reg}L(\Sigma)$
Beweis: Zu $\mathfrak{A} \in \text{DFA}(\Sigma)$ konstruieren wir $\mathfrak{A}^R \in \text{NFA}(\Sigma)$ mit $L(\mathfrak{A}^R) = L(\mathfrak{A})^R$.
Idee: ersetze $\underset{q}{\bullet} \xrightarrow{a} \underset{q'}{\bullet}$ durch $\underset{q}{\bullet} \xleftarrow{q} \underset{q'}{\bullet}$,
sowie $\to \underset{q_0}{\bullet}$ durch $\circledcirc_{q_0}$
und ergänze diesen NFA durch

$$\begin{array}{ccl} & & \overset{\varepsilon}{\nearrow} \circledcirc_{q_1} \\ \to \underset{q_0}{\bullet} & & \vdots \qquad \forall q_i \in F \\ & & \underset{\varepsilon}{\searrow} \circledcirc_{q_n} \end{array}$$

□

Korollar: $\{L \subseteq \Sigma^* | L = L(G), G \text{rechtslinear}\}$
$\{L \subseteq \Sigma^* | L = L(G), G \text{linkslinear}\}$

Beweis: L rechtslinear $\curvearrowright L^R$ rechtslinear
$\curvearrowright L = L^{RR}$ linkslinear (Regeln spiegeln).
(siehe Folie 3.6)

□

3.3 Normalformen von Kontextfreien Grammatiken

Hilfsmittel: Vorgänger von Satzformen.

$$\alpha, \beta \in X^*, X = \Sigma \cup N$$

$\alpha \Rightarrow \beta$ α direkte Vorgänger von β
$\alpha \stackrel{*}{\Rightarrow} \beta$ α Vorgänger von β

Definition: Sei $G =< N, \Sigma, P, S >\in$ CFG und $L \subseteq X^*$. Dann ist

die direkte Vorgängermenge von L definiert durch

$$\underline{\text{Pre}}_G(L) := \{\alpha \in X^* \mid \exists \beta \in L, \alpha \Rightarrow_G \beta\}$$

der Vorgängerabschluß von L definiert durch

$$\underline{\text{Pre}}^*_G(L) := \{\alpha \in X^* \mid \exists \beta \in L, \alpha \stackrel{*}{\Rightarrow}_G \beta\}$$

Lemma: Sei $G =< N, \Sigma, P, S >\in$ CFG und $L \subseteq X^*$.
Dann gilt: $L \in \text{Reg}L(X) \curvearrowright \underline{\text{Pre}}^*_G(L) \in \text{Reg}L(X)$.

Beweis: Sei $\mathfrak{A} =< Q, X, \delta, q_0, F >\in$ NFA mit $L(\mathfrak{A}) = L$.
Konstruiere $\mathfrak{A}' =< Q, X, \delta', q_0, F >\in$ NFA durch

1. $\delta(q,a) \subseteq \delta'(q,a)\ \forall (q,a) \in Q \times X_\varepsilon$
2. Wenn $A \to \alpha \in P$ und $\overline{\delta'}(\{q\}, \alpha) \ni q'$, so füge $\delta'(q,A) \ni q'$ hinzu.
   Ergänze δ' um solche Transitionen, solange dies möglich ist.
   Der Prozeß terminiert, weil Q und X endlich sind.

Die Korrektheit ist offensichtlich. (Erläuterungen siehe Folie 3.7).

□

Bemerkung: Die Zeitkomplexität der Automatenkonstruktion liegt in $\rho(|Q|^6)$.

Definition: Sei $G \in$ CFG(Σ) und $A \in N$.

1. A heißt *produktiv*: $\curvearrowright \exists w \in \Sigma^* : A \stackrel{*}{\Rightarrow} w$
2. A heißt *erreichbar*: $\curvearrowright \exists \alpha, \beta \in X^* : S \stackrel{*}{\Rightarrow} \alpha A \beta$

Folgerung: *A produktiv* $\curvearrowright A \in \underline{\text{Pre}}^*_G(\Sigma^*)$
A erreichbar $\curvearrowright S \in \underline{\text{Pre}}^*(X^* A X^*)$

Definition: $G \in$ CFG(Σ) heißt *reduziert*: $\curvearrowright$
entweder $P = \emptyset$
oder jedes $A \in N$ ist *produktiv* und *erreichbar*.

Lemma: Jedes $G \in \mathrm{CFG}(\Sigma)$ läßt sich in eine äquivalente reduzierte Grammatik $G' \in \mathrm{CFG}(\Sigma)$ transformieren.

Beweis: Stelle für jedes $A \in N$ fest, ob A *produktiv* und *erreichbar*. Unterscheide zwei Fälle:

1. S ist *nicht produktiv*. Wähle: $P' = \emptyset$.
2. S ist *produktiv*. Weglassen aller $A \in N$, die *nicht produktiv* oder *nicht erreichbar* sind, sowie aller Regeln, in denen solche A's vorkommen.

□

Korollar: Das $\emptyset$-Problem von CFG's ist entscheidbar.

3.3.1 Elimination von ε-Regeln

Definition: $G \in \mathrm{CFG}(\Sigma)$ heißt ε-*frei*: $\curvearrowleft\!\curvearrowright$

$A \to \varepsilon \in P \curvearrowright A = S$ und
S auf keiner rechten Regelseite.

Lemma: $X \overset{*}{\Rightarrow} \varepsilon \curvearrowleft\!\curvearrowright X \in \underline{\mathrm{Pre}}^*(\{\varepsilon\})$.

Satz: Jedes $G \in \mathrm{CFG}(\Sigma)$ läßt sich in eine äquivalente ε-*freie Grammatik* $G' \in \mathrm{CFG}(\Sigma)$ transformieren.

Beweis: Konstruiere mit $\underline{\mathrm{Pre}}^*(\{\varepsilon\})$ die Menge

$$N_\varepsilon := \{A \in N \,|\, A \overset{*}{\Rightarrow} \varepsilon\}$$

und damit $G' = < N', \Sigma, P', S' > \in \mathrm{CFG}$:

- $N' := N \mathbin{\dot\cup} \{S'\}$ (neues Startsymbol)
- $P' := \{S' \to S\} \cup \{S' \to \varepsilon \,|\, S \in N_\varepsilon\}$
  $\cup \{A \to X_1 \ldots X_k \,|\, k \geq 1, X_i \in (N \cup \Sigma),$
  $\exists \alpha_0, \ldots, \alpha_k \in N_\varepsilon^* : A \to \alpha_0 X_1 \alpha_1 \ldots X_k \alpha_k \in P\}$

Beachte: Wegen $k \geq 1$ entfallen ε-Regeln und G' ist ε-*frei*.
Bleibt zu zeigen: $L(G') = L(G)$

1. $w = \varepsilon :\ \varepsilon \in L(G') \curvearrowleft\!\curvearrowright S' \to \varepsilon \in P' \curvearrowleft\!\curvearrowright S \in N_\varepsilon$
   $\curvearrowleft\!\curvearrowright S \overset{*}{\Rightarrow}_G \varepsilon \curvearrowleft\!\curvearrowright \varepsilon \in L(G)$
2. $w \neq \varepsilon$
   (a) $w \in L(G') \curvearrowright S \overset{*}{\Rightarrow}_{G'} w \curvearrowright S \overset{*}{\Rightarrow}_G w \curvearrowright w \in L(G)$
   weil Ableitung in G' mit $L_i \overset{*}{\Rightarrow} \varepsilon$ aufgefüllt werden kann.

(b) $w \in L(G)$: Wir zeigen durch Induktion über die Ableitungslänge r, daß gilt:

$$A \overset{r}{\Rightarrow}_G w \in {}^1\Sigma^+ \curvearrowright A \overset{*}{\Rightarrow}_{G'} w$$

$r = 1:$ $\quad A \rightarrow w, w \neq \varepsilon \curvearrowright A \rightarrow w \in P$

$\quad\quad \curvearrowright A \rightarrow w \in P' \curvearrowright A \overset{*}{\Rightarrow}_{G'} w$

$r \rightarrow r+1:$ $\quad A \Rightarrow_G X_1 \ldots X_k \overset{r}{\Rightarrow}_G w \neq \varepsilon$

Dann existieren Ableitungen $X_i \overset{r_i}{\Rightarrow}_G w_i$ mit $w_1 \ldots w_k$, $r_i \leq r$
Falls $w_i = \varepsilon$, so $X_i \in N_\varepsilon$. Durch entsprechendes Löschen entsteht

$$A \Rightarrow_{G'} X'_1 \ldots X'_j \overset{*}{\Rightarrow} w'_1 \ldots w'_j = w.$$

$\square$

Beispiel: Transformation in ε-freie Grammatik $G' \in$ CFG

$$N_\varepsilon := \{A \in N | A \overset{*}{\Rightarrow} \varepsilon\}$$

- $N' := N \cup \{S'\}$
- $P' := \{S' \rightarrow S\} \cup \{S' \rightarrow \varepsilon | S \in N_\varepsilon\}$
  $\cup \{A \rightarrow X_1 \ldots X_k | k \geq 1, X_i \in (N \cup \Sigma),$
  $\exists \alpha_0, \ldots, \alpha_k \in N_\varepsilon^* : A \rightarrow \alpha_0 X_1 \alpha_1 \ldots X_k \alpha_k \in P\}$

Beispiel:

$$A \rightarrow aBa \qquad B \rightarrow b|\varepsilon$$

wird transformiert nach

$$A \rightarrow aBa|aa \qquad B \rightarrow b$$

Definition: Eine Regel der Form $A \rightarrow B$ heißt *Kettenregel*.
Satz: Jedes $G = < N, \Sigma, P, S > \in$ CFG läßt sich in eine äquivalente ε-*freie* Grammatik $G' = < N', \Sigma, P', S' > \in$ CFG ohne Kettenregeln transformieren.

Beweis: o.B.d.A sei G ε-*frei*.
Dann definieren wir $G' = < N, \Sigma, P', S >$ durch
$A \rightarrow \alpha \in P' : \curvearrowleftright \alpha \neq N$ und es gibt $B \rightarrow \alpha \in P$ mit $A \in \underline{\text{Pre}}^*(B)$.

Die Korrektheit folgt, da wegen ε-Freiheit von G gilt:

$$A \in \underline{\text{Pre}}^*(B) \curvearrowleftright A \Rightarrow A_1 \Rightarrow A_2 \ldots \Rightarrow A_n = B$$

$\square$

[1] nichtleeres Wort

3.3.2 Die Chomsky-Normalform

Definition: $G =< N,\Sigma,P,S >\in$ CFG ist in *Chomsky-Normalform* ($G \in$ CNF): $\curvearrowleftright$ Jede Regel von P hat die Form

$\bullet A \to BC$ mit $B,C \in N$
oder $\bullet A \to a$ mit $a \in \Sigma$
$\bullet S \to \varepsilon$

Im letzten Fall darf S auf keiner rechten Regelseite auftreten.

Satz: Jedes $G =< N,\Sigma,P,S >\in$ CFG läßt sich in eine äquivalente Grammatik $G' \in$ CNF transformieren.

Beweis: o.B.d.A. sei G ε-*frei* und ohne Kettenregeln. Außerdem können wir annehmen, daß für $k \geq 2$ gilt

$$A \to X_1 \ldots X_k \in P \curvearrowright X_i \in N$$

Denn, $X_i = a$ kann ersetzt werden durch neues $C_i \in N$ unter hinzufügen von $C_{\to}a$.

Bleibt die Elimination von Regeln der Form

$$A \to B_1 \ldots B_k \text{ mit } k \geq 3$$

Ersetzen durch

$$\begin{array}{rcl} A & \to & B_1C_1 \\ C_1 & \to & B_2C_2 \\ C_2 & \to & B_3C_3 \\ \vdots & & \vdots \\ C_{k-2} & \to & B_{k-1}B_k \end{array}$$

mit neuen $\mathbb{N}$-Symbolen $C_1,\ldots,C_{k-2}$

$\square$

3.3.3 Die Greibach Normalform, Linksrekursion

Definition: $G =< N,\Sigma,P,S >\in$ CFG ist in *Greibach-Normalform* ($G \in$ GNF): $\curvearrowleftright$ Jede Regel von der Form

- $A \to aB_1 \ldots B_n$ oder
- $A \to a$
- $S \to \varepsilon$ (S auf keiner rechten Regelseite)

Satz: Jede $G \in$ CFG läßt sich in eine äquivalente Grammatik $G' \in$ CFG transformieren.

Beweis: Elimination *linksrekursiver* Nichtterminalsymbole.

Definition: $A \in N$ *linksrekursiv*: $\Leftrightarrow A \overset{+}{\Rightarrow} A\alpha$ für ein $\alpha \in X^*$

Spezialfall: Direkte *Linksrekursion*

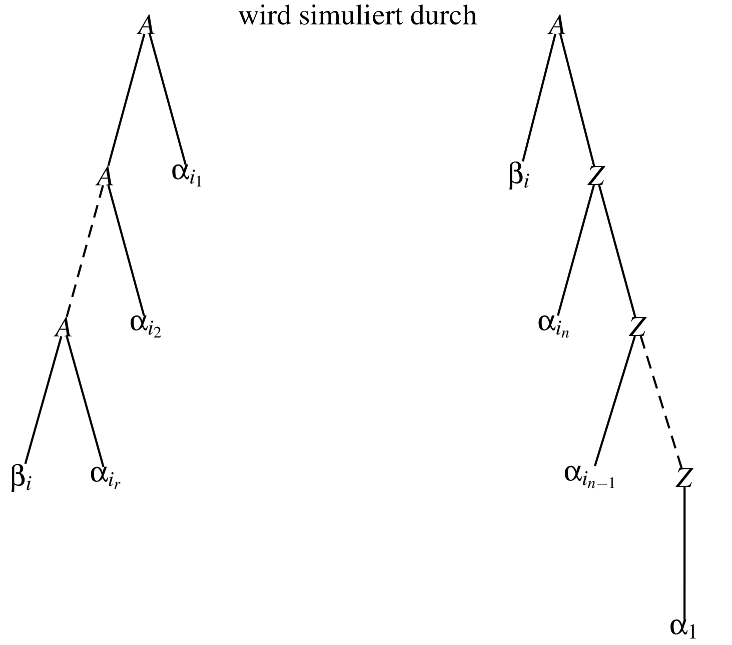
$$\left.\begin{array}{ll} \text{Seien} & A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_r \\ \text{und} & A \to \beta_1 | \ldots | \beta_s \end{array}\right\} \text{alle Regeln der Form } A \to A\alpha (\alpha_i \neq \varepsilon)$$

Dann läßt sich dieser Regelsatz äquivalent ersetzen durch

$A \to \beta_1 Z | \ldots b_s Z | \beta_1 | \ldots | \beta_s$

$Z \to \alpha_1 Z | \ldots \alpha_r Z | \alpha_1 | \ldots | \alpha_r$

Grund:



Bemerkung: Linksrekursion stört die Syntaxanalyse

3.4 Abschlußeigenschaften von CFL, Pumping-Lemma

Hilfsmittel: Substitutionssatz, Pumping-Lemma

Definition: Sei Σ ein Alphabet und für jedes $a \in \Sigma, \Sigma_a$ ein weiteres Alphabet.
Dann heißt $\phi : \mathfrak{p}(\Sigma^*) \to \mathfrak{p}((\bigcup_{a \in Sigma} \Sigma_a)^*)$ eine *Substitutionsabbildung*, falls

- $\phi(\{\alpha\}) \subseteq \Sigma_a^*$
- $\phi(\{\varepsilon\}) = \{\varepsilon\}$
- $\phi(\{a_1 \ldots a_k\}) = \phi(\{\alpha_1\}) \cdot \phi(\{\alpha_2\}) \cdot \ldots \cdot \phi(\{\alpha_k\})$
- $\phi(L) = \bigcup_{w \in L} \phi(\{w\})$

ϕ ist also durch die „*Substitutionssprachen*" $\phi(\{a\})$ bestimmt. Statt $\phi(\{w\})$ schreiben wir auch $\phi(w)$.

3.4.1 Substitutionssatz

Sei $\phi : \mathfrak{p}(\Sigma^*) \to \mathfrak{p}(\bigcup_{a\in\Sigma}\Sigma_a)^*$ eine *Substitutionsabbildung*. Dann gilt:
$L \in \mathrm{CFL}(\Sigma), \phi(a) \in \mathrm{CFL}(\Sigma_a)\, \forall a \in \Sigma$
$\curvearrowright \phi(L) \in \mathrm{CFL}(\bigcup_{a\in\Sigma}\Sigma_a)$

Beweis: Kombination kontextfreier Grammatiken mit disjunkten Nichtterminal-Symbolmengen. Ersetze $a \in \Sigma$ durch S_a.

Korollar: $\mathrm{CFL}(\Sigma)$ ist unter regulären Operationen abgeschlossen.

Beweis: Seien $L_1, L_2 \in \mathrm{CFL}(\Sigma)$ und a, b Buchstaben mit $\phi(a) := L_1$ und $\phi(b) := L_2 (\Sigma = \{a,b\})$.
Dann gilt:

$$\begin{aligned} \phi(\{a,b\}) &= L_1 \cup L_2 \\ \phi(ab) &= L_1 L_2 \\ \phi(\{a\}^*) &= L_1^* \end{aligned}$$

Da $\{a,b\}, \{ab\}, \{a\}^* \in \mathrm{CFL}(\{a,b\})$ folgt die Behauptung.

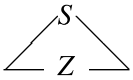
$\square$

3.4.2 Pumping-Lemma (für CFL)

uvwxy-**Theorem**

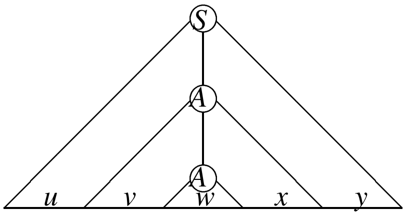
Sei $L \in \mathrm{CFL}(\Sigma)$. Dann existiert $k \geq 1$, so daß für alle $z \in L$ mit $|z| \geq k$ gilt: es existiert eine Zerlegung $z = uvwxy$ mit $|vwx| \leq k$, $vx \neq \varepsilon$ und $uv^iwx^iy \in L$ für alle $i \in \mathbb{N}$.

Beweis: o.B.d.A. sei $G = < N, \Sigma, P, S > \in$ CNF mit $L(G) = L$.
Sei $n := N, k := 2^n$ und $z \in L$ mit $|z| \geq k$.

Ein Ableitungsbaum 

hat mind. 2^n Blätter.

Da $G \in$ CNF, muß ein Pfad p maximaler Länge mindestens $n+1$ Kanten, also $n+2$ Knoten haben. p besitzt daher mindestens $n+1$ Knoten, markiert durch Nichtterminalsymbole, also 2 Knoten mit gleichem Nichtterminal-Symbol.



Wähle auf p den letzten Knoten, dessen Marke $A \in N$ sich wiederholt. Dann hat sein Teilbaum höchstens $2^n = k$ Blätter.

- $|vwx| \leq 2^n = k$
- $vx \neq \varepsilon$
- $uv^i wx^i y \in L$ für alle $i \in \mathbb{N}_0$
  wegen der Ersetzbarkeit der A-Teilbäume.

□

Lemma: $L = \{a^n b^n c^n | n \geq 1\} \notin \mathrm{CFL}(\{a,b,c\})$.

Beweis: (durch Widerspruch) Wäre L kontextfrei, so existiert ein *Pumping-Index* $k \in \mathbb{N}$ und für $a^k b^k c^k$ eine Zerlegung $uvwxy$ mit den *Pumping-Eigenschaften*, insbesondere $uwy \in L$. Da $vx \neq \varepsilon$, muß $|uwy| < 3k$ sein. Da $|vwx| \leq k$, kann vx nicht gleichzeitig ein a und ein c enthalten. Also muß $uwy = a^k \ldots$ oder $uwx = \ldots c^k$. Dies ist ein Widerspruch

□

Satz: CFL ist weder unter Durchschnitt, noch unter Komplement abgeschlossen.

Beweis: $\left\{ \begin{array}{c} S \to Sc|Ac \\ A \to aAb|ab \end{array} \right\}$ und $\left\{ \begin{array}{c} S \to aS|aA \\ A \to bAc|bc \end{array} \right\}$ erzeugen die Sprachen $\{a^n b^n c^p | n,p \geq 1\}$, bzw. $\{a^p b^n c^n | n,p \geq 1\}$, deren Schnitt nicht kontextfrei ist.
Da CFL unter $\cup$ abgeschlossen ist, kann CFL nicht unter komplement abgeschlossen sein, denn:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

□

3.5 Entscheidbare Eigenschaften von CFG

Satz: Das Wortproblem und das $\emptyset$-Problem sind für CFG entscheidbar.

Beweis:

$$\begin{array}{lcl} w \in L(G) & \curvearrowright & S \in \underline{\mathrm{Pre}}^*(\{w\}) \\ L(G) = \emptyset & \curvearrowright & S \notin \underline{\mathrm{Pre}}^*(\Sigma^*) \end{array}$$

□

Bemerkung:

1. Das Wortproblem ist in $\rho(n^3)$ entscheidbar (siehe: CYK-Algorithmus [3.7])
2. Das Äquivalenzproblem ist nicht entscheidbar.

3.6 Kellerautomaten

Kellerspeicher (Stack, Stapel, *push down store*): wichtige Datenstruktur für die sequentielle Behanlung strukturierter Objekte

Charakterisierung von CFL durch nicht-deterministische Kellerautomaten, keineÄquivalenz zu deterministischen Kellerautomaten

Bedeutung deterministischer Kellerautomaten für den Compilerbau:

- Syntaxanalyse
- Datenkeller
- Prozedurkeller (Speichertechnik für rekursive Prozeduren)

Wir betrachten nun zusätzlich zum Eingabeband und dem Zustand noch das Symbol, welches wir auf dem *Kellerspeicher* lesen. Der Kellerspeicher ist ein Stapelspeicher, auf dem wir oben Elemente hinzufügen und lesen könenn (LIFO-Prinzip: Last In - First Out).
Definition: Seien Q, Σ, Γ nicht-leere endlichen Mengen von *Zuständen*, *Eingabe*- und *Kellersymbolen*; ferner: $q_0 \in Q$ *Anfangszustand*, $Z_0 \in \Gamma$ *Kellerstartsymbol* und $\delta : Q \times \Sigma_\varepsilon \times \Gamma \to \mathfrak{p}_f(Q \times \Gamma^*)$ *Transitionsfunktion* und $F \subseteq Q$ *Entzustandsmenge*.

Dann heißt $\mathfrak{A} =< Q, \Sigma, \Gamma, \delta, q_0, Z_0, F >$ ein *Kellerautomat über* Σ, Bezeichnung: $\mathfrak{A} \in \mathrm{PDA}(\Sigma)$.

3.6.1 Semantik

Konfigurationsmenge $Q \times \Sigma^* \times \Gamma^*$ (Kellerspitze links)

Einzelschrittrelation $(q, w, \alpha) \vdash (q', w', \alpha')$

- Σ-Schritt: $(q, aw, Z\alpha) \vdash (q', w, \beta\alpha)$, falls $\delta(q, a, Z) \ni (q', \beta)$
- ε-Schritt: $(q, w, Z\alpha) \vdash (q', w, \beta\alpha)$, falls $\delta(q, \varepsilon, Z) \ni (q', \beta)$

Startkonfiguration für $w \in \Sigma^* : (q_0, w, Z_0)$

3.6.2 Die von $\mathfrak{A}$ erkannten Sprachen

$$\begin{aligned}
L(\mathfrak{A}, F) &:= \{w \in \Sigma^* | (q_0, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \alpha), q \in F\} \\
L(\mathfrak{A}, \varepsilon) &:= \{w \in \Sigma^* | (q_0, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \varepsilon)\} \\
L(\mathfrak{A}, F, \varepsilon) &:= \{w \in \Sigma^* | (q_0, w, Z_0) \overset{*}{\vdash} (q, \varepsilon, \varepsilon), q \in F\}
\end{aligned}$$

Bemerkung: Die Erkennungsarten sind äquivalent. (Stichwort: Simulation)

$$\mathscr{L}(\Sigma, \mathrm{PDA}) := \{L \subseteq \Sigma^* | \exists \mathfrak{A} \in \mathrm{PDA}(\Sigma), L = L(\mathfrak{A}, F)\}$$

(Beispiel: siehe Folie 3.8)

Ziel: $\mathrm{CFL}(\Sigma) = L(\Sigma, \mathrm{PDA})$
Satz: $\mathrm{CFL}(\Sigma) \subseteq \mathscr{L}(\Sigma, \mathrm{PDA})$

Beweis: Sei $G =< N, \Sigma, P, S >\in$ CFG. Dann definiere $\mathfrak{A}_G =< Q, \Sigma, \Gamma, \delta, q_0, Z_0, F >\in \mathrm{PDA}(\Sigma)$ durch
$Q := \{q\}, \Gamma := N \cup \Sigma, Z_0 := S, q_0 = q, F = \emptyset$
$\delta(q, a, a) := \{(q, \varepsilon)$ für alle $a \in \Sigma$ (Vergleichsschritt)
$\delta(q, \varepsilon, A) := \{(q, \beta) | A \to \beta \in P\}$ für alle $A \in N$ (Ableitungsschritt)

Behauptung: $L(G) = L(\mathfrak{A}_G, \varepsilon)$ (*) $A \overset{*}{\Rightarrow} w \curvearrowleft (q, wv, A\alpha) \overset{*}{\vdash} (q, v, a)$
„$\curvearrowright$“ Induktion über die Ableitungslänge n
$n = 1, A \to w \curvearrowright (q, wv, A\alpha) \vdash (q, wv, w\alpha) \overset{*}{\vdash} (q, v, \alpha)$
$n \rightsquigarrow n+1, A \Rightarrow v_0 A_1 v_1 \ldots A_r v_r \overset{n}{\Rightarrow} w$
Dann muß $A_i \overset{n_i}{\Rightarrow} w_i, w = v_0 w_1 v_1 w_2 \ldots v_r\, n_i \leq n$, so daß nach Induktionsvoraussetzung:

$$(q, v_0 w_1 v_1 \ldots v_r v, A\alpha) \vdash (q_0, v_0 w_1 \ldots v_r v, v_0 A_1 \ldots v_r \alpha) \overset{*}{\vdash} (q, v, \alpha)$$

„$\curvearrowleft$“ Induktion über die Länge der Berechnung (Übung)

Es folgt: $L(G) = L(\mathfrak{A}_G, \varepsilon)$

□

(Beispiel siehe Folie 3.9)

Satz: $\mathscr{L}(\Sigma, \mathrm{PDA}) \subseteq \mathrm{CFL}(\Sigma)$
Beweis: „ Schöning “

Korollar: CFL ist unter Schnitt mit regulären Sprachen abgeschlossen.
Beweis: Für $L \in \mathrm{CFL}(\Sigma)$ und $R \in \mathrm{Reg}L(\Sigma)$ existiert $\mathfrak{A}_L \in \mathrm{PDA}(\Sigma)$ und $\mathfrak{A}_R \in \mathrm{DFA}(\Sigma)$ mit $L = L(\mathfrak{A}_L)$ und $R = L(\mathfrak{A}_R)$. Konstruiere $\mathfrak{A}_L \| \mathfrak{A}_R \in \mathrm{PDA}(\Sigma)$

Parallelkonstruktion: $Q := Q_L \times Q_R$

$$\delta((q_1, q_2), a, Z) \ni ((q_1', q_2'), \alpha)$$

falls $\delta_L(q_1, a, Z) \ni (q_1', \alpha)$ und $\delta_R(q_2, a) = q_2'$
$F := F_L \times F_R$

□

Ziel: Der Deklarationszwang von Variablen ist keine kontextfreie Eigenschaft.

Lemma: $L = \{ww | w \in \{a, b\}^+\} \notin \mathrm{CFL}$
Beweis: Zunächst zeigen wir $L' = \{a^m b^n a^m b^n | m, n \geq 1\} \notin \mathrm{CFL}$
Wäre $L' \in$ CFL, so ergäbe sich mit *Pumping-Index k*:
$a^k b^k a^k b^k = uvwxy$ mit $|vwx| \leq k, vx \neq \varepsilon$

also auch $uvy \in L'$. Dies is ein Widerspruch, weil die Zahl der vorderen und hinteren a's bzw. b's nicht mehr gleich ist.
$L' = L \cap \{a\}^+\{b\}^+\{a\}^+\{b\}^+$ ist also ebenfalls nicht kontextfrei.

Korollar: Die Menge P der Pascal-Programme ist **nicht** kontextfrei.

Beweis: $L :=$ { program T (output); var w: integer; begin $w := 1$ end. $|w \in \{a,b\}^+\}$
Also: $L \subseteq P$
R sei bestimmt durch den regulären Ausdruck
program T (output); var $(a \vee b)^+$; begin $(a \vee b)^+ := 1$ end.
Dann gilt: $L = P \cap R$. h sei eine *Substitutionsabbildung* mit

$$h(a) = a,\ h(b) = b,\ h(x) = \varepsilon\,\text{sonst}$$

Dann $h(L) = \{ww | w \in \{a,b\}^+\} \notin$ CFL, was einen Widerspruch zu der Annahme $P \in$ CFL darstellt.

□

3.6.3 Deterministische Kellerautomaten

Definition: $\mathfrak{A} =< Q, \Sigma, \Gamma, \delta, q_0, Z_0, F >\in \mathrm{PDA}(\Sigma)$ heißt *deterministisch*, in Zeichen: $\mathfrak{A} \in \mathrm{DPDA}(\Sigma)$, wenn gilt:

1. $|\delta(q,a,Z)| \leq 1$ für alle $(q,a,Z) \in Q \times \Sigma_\varepsilon \times \Gamma$
2. $|\delta(q,\varepsilon,Z)| = 1 \curvearrowright |\delta(q,a,Z)| = 0$

Folgerung: Zu jeder Konfiguration (q,w,α) gibt es höchstens eine Folgekonfiguration. ε-Schritte möglich.

$$L(\mathfrak{A}) := L(\mathfrak{A}, F)$$
$$\mathscr{L}(\Sigma, \mathrm{DPDA}) := \mathscr{L}(\Sigma, \mathrm{DPDA}, F) \supsetneq \mathscr{L}(\Sigma, \mathrm{DPDA}, \varepsilon)$$

Satz: $\mathscr{L}(\Sigma, \mathrm{DPDA})$ ist unter Komplement abgeschlossen. (Schade: da unerwünscht)

Beweis: (Idee): Zu jedem $\mathfrak{A} \in \mathrm{DPDA}(\Sigma)$ ist ein äquivalenter $\tilde{\mathfrak{A}} \in \mathrm{DPDA}(\Sigma)$ konstruierbar, so daß gilt: Für jedes $w \in \Sigma^*$ gibt es $q \in \tilde{Q}$ und $\alpha \in \tilde{\Gamma}^*$ mit

$$(\tilde{q}_0, w, \tilde{Z}_0) \overset{*}{\vdash} (q, \varepsilon, \alpha)\ \text{(Endkonfiguration)}$$

Dazu müssen Schleifenkonfigurationen eliminiert werden.

$(q,w,\alpha) \in Q \times \Sigma^* \times \Gamma^*$ heißt *Schleifenkonfiguration*, falls für jedes $i \in \mathbb{N}$ ein $q_i \in Q$ und ein $\alpha_i \in \Gamma^*$ existiert, so daß $|\alpha_i| \geq |\alpha|$ und $(q,w,\alpha) \overset{i}{\vdash} (q_i, w, \alpha_i)$.
Diese Eigenschaft ist entscheidbar (Kern des Beweises) $\curvearrowright$ Elimination.

□

Folgerung: $\mathscr{L}(\Sigma, \mathrm{DPDA}) \subsetneq \mathscr{L}(\Sigma, \mathrm{PDA}) = \mathrm{CFL}$

3.7 Der Algorithmus von Cocke, Yonnger und Kasami

effiziente Lösung des Wortproblems für beliebige CFL, dynamische Programmierung, $\rho(n^3)$-Zeit, $\rho(n^2)$-Platz.
O.B.d.A. $G \in$ CNF (auf beliebige CFG übertragbar).

Für $G =< N, \Sigma, P, S >\in$ CNF $(A \to BC, A \to a)$ und $w = a_1 a_2 \ldots a_n$ mit $n > 0$ definieren wir:

$$\begin{aligned} w_{ij} &:= a_i a_{i+1} \ldots a_j \text{ für } 1 \le i \le j \le n \\ N_{ij} &:= \{A \in N | A \overset{*}{\Rightarrow} w_{ij}\}, \end{aligned}$$

so daß gilt:

$$w \in L(G) \curvearrowleftright S \in N_{1n}$$

Berechnung von N_{1n}:

Bestimme	N_{11}	N_{22}	$\ldots$		N_{nn}
dann	N_{12}	N_{23}		N_{n-1}	N_n
dann	N_{13}	N_{24}	$\ldots$	N_{n-2}	N_n
bis	N_{1n}				

mit Hilfe von

1. $A \in N_{ii} \curvearrowleftright A \to a_i \in P$
2. $A \in N_{ij}$ mit $i < j \curvearrowleftright \exists k : i \le k < j, \exists A \to BC \in P : B \in N_{ik}, C \in N_{k+1\,j}$

(siehe auch Folie 3.10, Beispiel 3.11)

3.7.1 Komplexität des CYK-Algorithmus

Zeit äußere "for i"-Schleife $c_1 \cdot n$ Schritte
"for k" $c_2 \cdot d$ Schritte
innere "for i" $c_2 \cdot (n-d)d$ Schritte
"for d" $c_2 \cdot \sum_{d=1}^{n-1}(n-d)d$ Schritte

Insgesamt: $c\sum_{d=1}^{n}(n-d)d = c\left(n\sum_{d=1}^{n} d - \sum_{d=1}^{n} d^2\right) = c\left(n^2\frac{n+1}{2} - n\frac{(n+1)(2n+1)}{6}\right) = c\frac{n^3-n}{6}$. Also $\rho(n^3)$.

Platz $\rho(n^2)$

3.8 Erweiterte kontextfreie Grammatiken (EBNF)

höherer Beschreibungskomfort durch die Verwendug von regulären Ausdrücke als rechte Regelseiten; äquivalente Erweiterung.

Definition: $G =< N, \Sigma, P, S >$ heißt *erweiterte CFG*, in Zeichen: $G \in$ ECFG,
wenn $< N, \Sigma, \emptyset, S >\in$ CFG und $P : N \to \mathrm{Reg}E(N \cup \Sigma)$.

Schreibweise: $A \to \alpha$, falls $\alpha = P(A)$

Semantik: P repräsentiert die Regelmenge $\tilde{P}$ mit $A \to \tilde{\alpha} \in \tilde{P} : \curvearrowright \tilde{\alpha} \in [\![P(A)]\!]$.

Beachte: $\tilde{P}$ ist im allgemeinen unendlich. $L(G) := L\big(< N, \Sigma, \tilde{P}, S >\big)$

Satz: $\mathrm{CFL}(\Sigma) = \mathscr{L}(\Sigma, \mathrm{ECFG})$

Beweis: (Idee) "$\subseteq$": nach Definition.
"$\supseteq$": Transformation von ECFG nach CFG durch Elimination regulärer Ausdrücke:

Disjunktion: Regelalternation

Stern: neues Nichtterminal mit $A \to \alpha A | \varepsilon$

□

(siehe auch Folie 3.12)

3.9 Rekursive endliche Automaten (Syntaxdiagramme)

Syntaxdiagramme entsprechen endlichen Automaten die sich rekursiv aufrufen können (Beispiel siehe Folien 3.13 und 3.14). Dabei sind neben den gewohnten Transitionen $q_1 \xrightarrow{a} q_2$ auch Transitionen der Form $q_1 \to \boxed{q_3} \to q_2$ möglich.

Definition: $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >$ heißt *rekursiver endlicher Automat* über Σ, falls

$$\delta : Q \times (\Sigma_\varepsilon \cup Q) \to \mathfrak{p}(\mathfrak{A})$$

und sonst wie ein endlicher Automat.

Bezeichnung: $\mathfrak{A} \in \mathrm{RFA}(\Sigma)$.

3.9.1 Semantik

Konfiguration: $Q \times \Sigma^*$
Transitionsrelation: $\vdash \subseteq (Q \times \Sigma^*)^2$ definiert als kleinste Relation für die gilt:

1. $p \in \delta(q, a) \curvearrowright (q, aw) \vdash (p, w)$
2. $p \in \delta(q, \varepsilon) \curvearrowright (q, w) \vdash (p, w)$
3. $p \in \delta(q, r), (r, w) \overset{*}{\vdash} (s, v), s \in F \curvearrowright (q, w) \vdash (p, v)$

$L(\mathfrak{A}) := \{w \in \Sigma^* | (q_0, w) \vdash (p, \varepsilon), p \in F\}$
Beispiel für den Erkennungslauf siehe Folie 3.15.

Satz: $\mathscr{L}(\Sigma, \mathrm{RFA}) = \mathscr{L}(\Sigma, \mathrm{PDA})$
Beweis:

„$\subseteq$" $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in$ RFA(Σ). Simulation von $\mathfrak{A}$ durch $\mathfrak{A}' < Q, \Sigma, Q, \delta', q_0, q_0, F >\in$ PDA(Σ)
Idee: Keller für „Rücksprungadressen" verwenden:

$$
\begin{aligned}
\delta' &: Q \times \Sigma_\varepsilon \times Q \to \mathfrak{p}_f(Q \times Q^*), p \in \delta(q,a) \\
&\delta'(q,a,s) := \{(p,s) | p \in \delta(q,a)\} \\
&\delta'(q,\varepsilon,s) := \{(p,s) | p \in \delta(q,\varepsilon)\} \\
&\qquad\qquad \cup \{(r,ps) | p \in \delta(q,r)\} \\
&\qquad\qquad \cup \{s,\varepsilon) | q \in F\}
\end{aligned}
$$

„$\supseteq$" Für $L \in \mathscr{L}(\Sigma, \text{PDA})\ \exists G =< N, Z, P, S >\in$ CFG mit $L = L(G)$.
Konstruiere $\mathfrak{A} =< Q, \Sigma, \delta, q_0, F >\in$ RFA durch: $Q := N \cup \{q_f\}, q_0 := S, F = \{q_f\}$ und

$C \in \delta(A,B)$	falls	$A \to BC$
$q_f \in \delta(A,a)$	falls	$A \to q$
$q_f \in \delta(S,\varepsilon)$	falls	$S \to \varepsilon$

$\square$

Kapitel 4

Turingmaschinen und aufzählbare Sprachen

4.1 Chomsky-Grammatiken

Verallgemeinerung kontextfreier Grammatiken.

Definition: Seien N, Σ, X und S wie bei kontextfreien Grammatiken (CFG). Sei ferner $P \subseteq X^* N X^* x X^*, P$ endlich. Dann heißt
$G = < N, \Sigma, P, S >$ eine *Chomsky-Grammatik*.

4.1.1 Semantik

$\pi = \alpha_1 \to \alpha_2 \in P$ bestimmt die *Ableitungsrelation* $\Rightarrow_\pi \subseteq X^* x X^*$ durch
$\beta_1 \Rightarrow_\pi \beta_2 : \curvearrowleft\!\!\curvearrowright$ es existiert $\gamma, \delta \in X^*$ mit $\beta_1 = \gamma \alpha_1 \delta$ und $\beta_2 = \gamma \alpha_2 \delta$.

Ableitungsrelation von $G : \Rightarrow_g := \bigcup_{\pi \in P} \Rightarrow_\pi$
G erzeugt $L(G) := \{w \in \Sigma^* | S \overset{*}{\Rightarrow}_G w\}$

4.1.2 Klassifikation von Chomsky-Grammatiken und Sprachfamilien

Sei $G = < N, \Sigma, P, S >$ eine Chomsky-Grammatik und $i \in \{0, 1, 2, 3\}$. Dann heißt G vom Typ i oder auch eine Typ i-Grammatik, wenn P die Eigenschaft (i) besitzt mit

(0) Keine Einschränkung

(1) Jedes $\pi \in P$ hat die Form $\gamma A \delta \to \gamma \beta \delta$ mit $\beta \neq \varepsilon$ oder $S \to \varepsilon$. Ferner $S \to \varepsilon, \alpha \to \beta \in P \curvearrowright S$ nicht in β.

(2) Jedes $\pi \in P$ hat die Form $A \to \alpha$.

(3) Entweder: Jedes $\pi \in P$ hat die Form $A \to Bw$ oder $A \to w$ oder: Jedes $\pi \in P$ hat die Form $A \to wB$ oder $A \to w$

Es folgt:

Grammatiken vom Typ 0 sind genau die Chomsky-Grammatiken
Grammatiken vom Typ 1 heißen auch *kontextsensitiv*
Grammatiken vom Typ 2 sind genau die kontextfreien Grammatiken
Grammatiken vom Typ 3 sind genau die einseitig linearen Grammatiken

$L \subseteq \Sigma^*$ heißt vom Typ $i :\curvearrowright \exists G$ vom Typ $i : L(G) = L$.
Bezeichnung: $\mathscr{L}(\Sigma) := \{L \subseteq \Sigma^* | L \text{ vom Typ } i\}$

4.1.3 Chomsky-Hierarchie

$$\mathscr{L}_3(\Sigma) \subsetneqq \mathscr{L}_2(\Sigma)^{1} \subsetneqq \mathscr{L}_1(\Sigma) \subsetneqq \mathscr{L}_0(\Sigma)$$

falls $|\Sigma| > 1; |\Sigma| = 1 \curvearrowright \mathscr{L}_3(\Sigma) = \mathscr{L}_2(\Sigma)$.

Definition: Sei $G =< N, \Sigma, P, S >$ vom Typ 0.
G heißt *normiert*, wenn gilt:

- Für jedes $\pi = \alpha_1 \to \alpha_2 \in P$ gibt es $\gamma, \delta \in N^*, A \in N, \beta \in X^*$ mit $\alpha_1 = \gamma A \delta$ und $\alpha_2 = \gamma\beta\delta$.
  Beachte: $\beta = \varepsilon$ erlaubt, im Gegensatz zu Typ 1.
- Σ-Symbole nur in Regeln der Form $A \to a$.

4.1.4 Abschlußeigenschaften von $\mathscr{L}_0(\Sigma)$

G normiert $(\pi = \gamma\underline{A}\delta \to \gamma\underline{\beta}\delta)$

Satz: Zu jedem $G =< N, \Sigma, P, S >$ vom Typ 0 läßt sich eine äquivalente Grammatik $G' =< N', \Sigma, P', S' >$, G' normiert, konstruieren.

Beweis:

1. Terminalsymbol-Bedingung:
   $a \in \Sigma \mapsto A_a$ neues Nichtterminalsymbol. (a durch A_a ersetzen und $A_a \to a$ hinzufügen).
2. Simulation von $A_1 \dots A_n \to B_1 \dots B_m$ $(n \neq 1, m \in \mathbb{N})$:
   durch Symbolersetzungsregeln mit Hilfe neuer Nichtterminal-Symbole A_i'.

$$\begin{array}{rcl}
A_1 \dots A_n & \to & A_1' A_2 \dots A_n \\
A_1' A_2 \dots A_n & \to & A_1' A_2' \dots A_n \\
\vdots & & \vdots \\
A_1' \dots A_{n-1}' A_n & \to & A_1' A_2' \dots A_n' \\
A_1' \dots A_n' & \to & B_1 A_2' \dots A_n' \\
B_1 A_2' \dots A_n' & \to & B_1 B_2 A_3' \dots A_n' \\
\vdots & &
\end{array}$$

[1] folgt aus CNF

Zwei Fälle zu unterschieden:

Fall 1 ($n \leq m$)**:** $B_1 \ldots B_{n-1}A'_n \to B_1 \ldots B_m$

Fall 2 ($n > m$)**:**

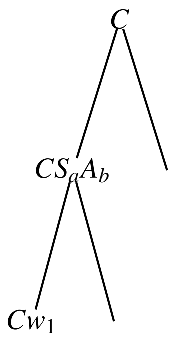
$$\begin{array}{rcl} B_1 \ldots B_m A'_{m+1} \ldots A'_n & \to & B_1 \ldots B_m A'_{m+2} \ldots A'_n \\ \vdots & & \vdots \\ B_1 \ldots B_m A'_n & \to & B_1 \ldots B_m \end{array}$$

□

Satz: $\mathscr{L}(\Sigma)$ ist unter Substitution abgeschlossen, d.h. ist $\phi : \mathfrak{p}(\Sigma^*) \to \mathfrak{p}((\bigcup_{a\in\Sigma}\Sigma_a)^*)$ eine *Substitutionsabbildung*, so gilt: $L \in \mathscr{L}_0(\Sigma), \phi(a) \in \mathscr{L}_0(\Sigma_a)\, \forall a \in \Sigma \curvearrowright \phi(L) \in \mathscr{L}_0(\bigcup_{a\in\Sigma}\Sigma_a)$.

Beweis: Konstruktion von CFG nicht anwendbar: neue Ableitungen wegen unzulässiger Satzformen.
Idee: Sequentialisierung der Ableitungen mit Kontrollsymbol.
$L = L(G)$ mit $G =< N, \Sigma, P, S >$ vom Typ 0. $\phi(a) =: L_a = L(G_a)$ mit $G_a =< N_a, \Sigma, P_a, S_a >$ vom Typ 0. O.B.d.A.: G, G_a normiert, N, N_a disjunkt.

Konstruktion von $\overline{G}$ (Typ 0) mit $L(\overline{G}) = \phi(L)$:
$\overline{N} := N \cup (\uplus_{a\in\Sigma} N_a) \uplus \{A_a | a \in \Sigma\} \uplus \{\overline{S}, C\}$
$\overline{\Sigma} := \bigcup_{a\in\Sigma}\Sigma_a$
$\overline{P} := P_0 \cup P_1 \cup P_2 \cup (\bigcup_{a\in\Sigma} P_a) \cup \{\overline{S} \to CS, C \to \varepsilon\}$
P_0 entstehe aus P durch Ersetzen von a durch $A_a (a \in \Sigma)$
$P_1 := \{CA_a \to CS_a | a \in \Sigma\}$



$P_2 := \{C_a \to aC | a \in \Sigma\}$

1. $\phi(L) \subseteq L(\overline{G}), \overline{S} \Rightarrow CS \overset{*}{\Rightarrow} CA_{a_1} \ldots A_{a_r} \Rightarrow CS_{a_1}A_{a_2} \ldots A_{a_r} \overset{*}{\Rightarrow} Cw_1 A_{a_2} \ldots A_{a_r} \overset{*}{\Rightarrow} w_1 CA_{a_2} \ldots \overset{*}{\Rightarrow} w_1 \ldots w_r C \Rightarrow w_1 \ldots w_r$

2. $L(\overline{G}) \subseteq \phi(L)$: keine Ableitungen neuer Wörter, weil G normiert und daher A_a auf keiner linken Regelseit, weil N-Alphabet disjunk und weil C die Teilableitungen der G_a sequentialisiert.

□

Korollar: $\mathscr{L}_0(\Sigma)$ ist unter regulären Operationen abgeschlossen: $L, L' \in \mathscr{L}_0(\Sigma) \curvearrowright L \cup L', LL', L^* \in \mathscr{L}_0(\Sigma)$.
Beweis: wie für CFL.

□

Satz: $\mathscr{L}_0(\Sigma)$ ist unter Durchschnitt abgeschlossen.
Beweis: $L_i = L(G_i)$ und $G_i = < N_i, \Sigma, P_i, S_i >$ vom Typ 0 $(i = 1,2)$. O.B.d.A. sei $N_1 \cap N_2 = \emptyset$.
Konstruktion von $G = < N, \Sigma, P, S >$:
$N := N_1 \cup N_2 \cup \{A_a | a \in \Sigma\} \uplus \{S, C_1, C_2\}$
$P := P_1 \cup P_2 \cup Q$
$Q := \{S \to C_1 S_1 C_2 S_2 C_1, C_2 a \to A_a C_2, b A_a \to A_a b, C_1 A_a a \to a C_1, C_1 C_2 C_1 \to \varepsilon | a, b \in \Sigma\}$

□

Bemerkung: Da $\mathscr{L}_0(\Sigma) = \mathscr{L}(\Sigma, \mathrm{TM})$ (siehe 4.2.2), ist $\mathscr{L}_0(\Sigma)$ nicht unter Komplement abgeschlossen.

4.1.5 Kontextsensitive Grammatiken und Sprachen

Definition: $G = < N, \Sigma, P, S >$ heißt von *wachsender Länge*: $\curvearrowleftright$ für jede Regel $\alpha \to \beta \in P$ gilt: $|\alpha| \leq |\beta|$, es sei denn, daß $S \to \varepsilon \in P$ und S auf keiner rechten Regelseite steht.

Kontextsensitive Regel: $\alpha A \beta \to \alpha\gamma\beta$ und $\gamma \neq \varepsilon$.

Korollar: Eine Typ 1-Grammatik ist von *wachsender Länge*.
Satz: Zu jeder Grammatik von *wachsender Länge* läßt sich eine äquivalente Typ 1-Grammatik konstruieren.
Beweis: Normierung von Typ 0-Grammatiken.

□

Zur Erinnerung:	$A_1 \ldots A_n \to B_1 \ldots B_m$
neue N-Symbole:	$A'_1, \ldots, A'_n$
	$A_1 \ldots A_n \to A'_1 A_n \ldots A_n$
	$\vdots$
	$A'_1 \ldots A'_{n-1} A_n \to A'_1 \ldots A'_n$
$m \geq n$:	$B_1 \ldots B_{n-1} A'_n \to B_1 \ldots B_m$
$m < n$:	$B_1 \ldots B_m A'_{m+1} \ldots A'_n \to B_1 \ldots B_m A'_{m+2} \ldots A'_n$

4.1.6 Platzbedarf

Definition: Sei $G = < N, \Sigma, P, S >$ eine Typ 0-Grammatik, $\delta = (S \Rightarrow \alpha_0 \Rightarrow \alpha_1 \ldots \Rightarrow \alpha_n)$ eine Ableitung von G und $w \in \Sigma^*$. Dann heißt:

- $\underline{\mathrm{pl}}_G(\delta) := \underline{\max}\{|\alpha_i| \,|0 \leq i \leq n\}$ der *Platzbedarf* von δ, und
- $\underline{\mathrm{pl}}_G(w) := \underline{\min}\{\underline{\mathrm{pl}}_G(\delta) | \delta = (S \Rightarrow \ldots \Rightarrow w)\}$ der *Platzbedarf* von w bezüglich G.

Folgerung:

1. $\underline{\mathrm{pl}}_G(w) \geq |w|$
2. G Typ 1, $w \neq \varepsilon \curvearrowright \underline{\mathrm{pl}}_G(w) = |w|$

4.1.7 Platzbedarfssatz

Sei G vom Typ 0 und $p \in \mathbb{N}, p > 0$, so daß für alle Worte $w \in L(G) \setminus \{\varepsilon\}$:

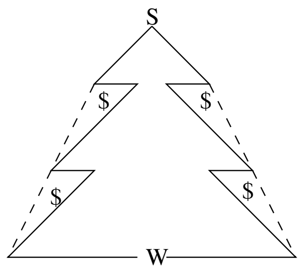
$$\underline{\mathrm{pl}}_G(w) \leq p|w|$$

Dann ist $L(G) \in \mathscr{L}_1$.

Beweis: Sei $p = 1$, also für $w \in L(G) \setminus \varepsilon$

$$\underline{\mathrm{pl}}_G(w) = |w|$$

Idee: Elimination verkürzender Regeln durch Auffüllen mit Hilfssymbolen.
Konstruktion einer äquivalenten Grammatik von *wachsender Länge*: $G' =< N \uplus \{\$\}, \Sigma, P', S >$



1. $\alpha \to \beta \in P$ mit $|\alpha| = |\beta| + i \, (i > 0)$
   $\curvearrowright \alpha \to \$^i\beta \in P'$
2. $\alpha \to \beta \in P$ mit $|\alpha| + i = |\beta| \, (i \in \mathbb{N})$
   $\curvearrowright \$^\delta\alpha \to \beta \in P'$ für alle $\delta = 0, 1, \ldots, i$
3. $\$X \to X\$, X\$ \to \$X \in P'$ für alle $X \in X$.

Wegen $\underline{\mathrm{pl}}_G(w) = |w|$ lassen sich die eingeschobenen \$-Symbole löschen.
$p > 1$: Induktion über p. $N^P \subseteq N'$.

□

4.1.8 Abschlußeigenschaften von $\mathscr{L}_1(\Sigma)$ mit Hilfe des Platzbedarfssatzes

Satz: $\mathscr{L}_1(\Sigma)$ ist unter ε-freier Substitution ($\varepsilon \in \phi(a)$) abgeschlossen.
Beweis: Platzbedarf der Ausgangsgrammatiken mit $p = 1 \curvearrowright$ Platzbedarf der Substitutionsgrammatik $|w| + 1 \leq 2|w|$ wegen zusätzlichen Kontrollsymbol C und fehlender ε-Regeln.

□

Korollar: $\mathscr{L}_1(\Sigma)$ ist unter regulären Operationen abgeschlossen.
Beweis: Reduktion auf ε-freie Typ 1-Sprachen.

$$L \in \mathscr{L}_1(\Sigma) \curvearrowright L\setminus\{\varepsilon\} \in \mathscr{L}_1(\Sigma)$$

Für ε-freie $L, L' \in \mathscr{L}_1(\Sigma)$ folgt

$$L \cup L', LL', L^* \in \mathscr{L}_1(\Sigma)$$

wie für Typ 0-Sprachen durch ε-freie Substitution. Falls $\varepsilon \in L$ und/oder $\varepsilon \in L'$, so schließt man über die ε-freie Teilsprachen, zum Beispiel:

$$(L \uplus \{\varepsilon\})(L' \uplus \{\varepsilon\}) = LL' \cup L \cup L' \cup \{\varepsilon\}$$

□

Korollar: $\mathscr{L}_1(\Sigma)$ ist unter $\cap$ abgeschlossen.
Beweis: $L_1, L_2 \in \mathscr{L}_1(\Sigma)$ sind mit linearen Platzbedarf ($p = 1$) erzeugbar. Die $\cap$-Konstruktion für Typ 0-Grammatiken hat dann einen Platzbedarf von $2|w| + 3 \leq 5|w|$

□

Ein lange ungelöstes Problem ist der Abschluß von $\mathscr{L}_1$ unter Komplement. 1987: $\mathscr{L}_1(\Sigma)$ ist unter Komplement abgeschlossen.

Korollar: $\mathscr{L}_1(\Sigma) \subsetneqq \mathscr{L}_0(\Sigma)$

4.2 Turingmaschinen, linear beschränkte Automaten

Ziel: $\mathscr{L}_0(\Sigma) = \mathscr{L}(\Sigma, \mathrm{TM})$
$\mathscr{L}_1(\Sigma) = \mathscr{L}(\Sigma, \mathrm{LBA})$

Modell einer Turingmaschine (TM): besteht aus einem unendlichen Band, welches größtenteils mit Blanks beschriftet ist, sowie wenige Symbole. Weiterhin gibt es eine *endliche Kontrolle* welche aus endlichen *Quintupel* besteht.

4.2.1 Nicht deterministisch erkennende TM

Definition: Seien die folgenden nicht-leeren, endlichen Menge gegeben:

Q *Zustandsmenge*
Σ *Eingabealphabet*
Γ *Arbeitsalphabet* mit $\Sigma \subseteq \Gamma$
Ferner:
$q_0 \in Q$ *Anfangszustand*
$\Box \in \Gamma\setminus\Sigma$ *Blank*
$F \subseteq Q$ *Endzustandsmenge* und die *Transitionsfunktion*
$\delta : Q \times \Gamma \to \mathfrak{p}(Q \times \Gamma \times \{L, N, R\})$

Dann heißt $\mathfrak{A} =< Q, \Sigma, \Gamma, q_0, \Box, F, \delta >$ eine nicht-deterministische erkennende Turingmaschine.
Bezeichnung: $\mathfrak{A} \in \mathrm{TM}(\Sigma)$.

4.2.2 Semantik

Konfigurationsmenge: $\underline{\mathrm{Conf}}(\mathfrak{A}) := Q \times \Gamma^* \times \Gamma \times \Gamma^*$

Einzelschrittrelation: $\vdash_{\mathfrak{A}} \subseteq \underline{\mathrm{Conf}}(\mathfrak{A})^2$ sei definiert wie folgt:

- $\delta(q,X) \ni (q',X',N) \curvearrowright (q,\alpha,X,\beta) \vdash (q',\alpha,X',\beta)$
- $\delta(q,X) \ni (q',X',L) \curvearrowright (q,\alpha Y,X,\beta) \vdash (q',\alpha,Y,X'\beta)$
  $\delta(q,X) \ni (q',X',L) \curvearrowright (q,\varepsilon,X,\beta) \vdash (q',\varepsilon,\Box,X'\beta)$
- $\delta(q,X) \ni (q',X',R) \curvearrowright (q,\alpha,X,Y\beta) \vdash (q',\alpha X',Y,\beta)$
  $\delta(q,X) \ni (q',X',R) \curvearrowright (q,\alpha,X,\varepsilon) \vdash (q',\alpha X',\Box,\varepsilon)$

Bemerkung: beiderseits unendliches Band, fast überall stehen Blanks (□'s).

Anfangskonfiguration: $w \in \Sigma^*$

$$\mathfrak{K}(w) := \begin{cases} (q_0,\varepsilon,a,v) & \text{, falls } w = av \\ (q_0,\varepsilon,\Box,\varepsilon) & \text{, falls } w = \varepsilon \end{cases}$$

Endkonfiguration: $(q,\alpha,X,\beta)\ , q \in F$

Die von $\mathfrak{A}$ erkannte Sprache: $L(\mathfrak{A}) := \{w \in \Sigma^* | \mathfrak{K}(w) \vdash_{\mathfrak{A}}^* (q,\alpha,X,\beta), q \in F\}$

Beachte: Erreichen von $q \in F$ genügt. Ein Anhalten der Turingmaschine ist nicht gefordert.

Die Klasse der durch TM **erkennbaren Sprachen:** $\mathscr{L}(\Sigma,\mathrm{TM})$

Satz: $\mathscr{L}(\Sigma,\mathrm{TM}) = \mathscr{L}_0(\Sigma)$

Beweis:

4.2.3 I. $\mathscr{L}(\Sigma,\mathrm{TM}) \subseteq \mathscr{L}_0(\Sigma)$

$\mathfrak{A} =< Q,\Sigma,\Gamma,q_0,\Box,F,\delta > \in \mathrm{TM}(\Sigma)$
Simulation von $\mathfrak{A}$ durch eine Typ 0-Grammatik. *Idee:* Erzeugen einer beliebigen Anfangskonfiguration mit hinreichend vielen Blanks an den Rändern (bei Ableitungen sind Ränder nicht erkennbar). Wir benutzen zwei Spuren: Die erste Spur hält die Eingabe für die Erzeugung und die zweite Spur simuliert die Turingmaschine.

Für $a_1 \dots a_r \in \Sigma^*$, $m,n \in \mathbb{N}$ erzeugt man

$$(\varepsilon,\Box)^m q_0 (a_1,a_1)(a_2,a_2)\dots(a_r,a_r)(\varepsilon,\Box)^n \qquad (*)$$

Dazu $G =< N,\Sigma,P,S >$ mit
$N := Q \cup (\Sigma_\varepsilon \times \Gamma) \cup \{S,C_1,C_2\}$

P :
Erzeugen von (*)
$\quad S \to (\varepsilon,\Box)S | q_0 C_1$

$C_1 \to (a,a)C_1|C_2 \qquad (a \in \Sigma)$
$C_2 \to (\varepsilon,\Box)C_2|\varepsilon$

Simulation der Turingmaschine:

$q(a,X) \to q'(a,X')$ falls $\delta(q,X) \ni (q',X',N)$
$q(a,X) \to (a,X')q'$ falls $\delta(q,X) \ni (q',X',R)$
$(b,Y)q(a,X) \to q'(b,Y)(a,X)$ falls $\delta(q,X) \ni (q',X',L)$

löschen von q durch 2. Spur, ursprünglicher Wert bleibt übrig:

$q \to \varepsilon$ falls $q \in F$
$(a,X) \to a$ $(a \in \Sigma_\varepsilon$

Es folgt: $L(G) = L(\mathfrak{A})$.

4.2.4 II. $\mathscr{L}_0(\Sigma) \subseteq \mathscr{L}(\Sigma, \mathrm{TM})$

$L \in \mathscr{L}_0(\Sigma)$ werde erzeugt von der Typ 0-Grammatik $G =< N,\Sigma,P,S >$. Simulation von G durch $\mathfrak{A} \in \mathrm{TM}(\Sigma)$:
neben Eingabewort Ableitungen von G simulieren und anschließend vergleichen.

Form der Berechnung:

q_0w Anfangskonfiguration
$\downarrow *$
$[w][s_{q_G}]$ Beginn der Simulation von G
$\downarrow$
$[w][s_{q_i}]$ Wahl der i-ten Regel
$\downarrow *$
$[w][\alpha_{q_G}]$ Simulation der i-ten Regel beendet
$\downarrow *$
$[w][v_{q_v}]$ Vergleich $w = v$?
$\downarrow *$
$q_{erk}[\Box \dots \Box]$

Konstruktion von $\mathfrak{A} \in \mathrm{TM}(\Sigma)$ für die Typ 0-Grammatik $G =< N,\Sigma,P,S >$ mit s Regeln der Form $X_1 \dots X_n \to Y_1 \dots Y_m$ $(n \geq 1, m \geq 0)$
$\mathfrak{A} =< Q,\Sigma,\Gamma,q_0,F,\Box,\delta >$ sei definiert durch

$Q = \{q_0, q_0^1, \dots, q_0^n\}$ (Anfangsphase)
$\cup\{q_G, q_1, \dots, q_s\}$ (Simulations von G, Regelauswahl)
$\cup\{q_i^{Z_1\dots Z_k}, q_{i_{Z_1\dots Z_l}} | 1 \leq k \leq n, 0 \leq l \leq m, Z_\mu \in X \cup \{]\}\}$
für alle $1 \leq i \leq s$ (Simulation von $X_1 \dots X_n \to Y_1 \dots Y_m$)
$\cup\{q_v, q^a, q_a, \vec{q}, q_{erk} | a \in \Sigma \cup \{[\}\}$ (Vergleichsphase)
$\Gamma := \Sigma \dot\cup N \dot\cup \{\Box, [,]\}$
$F := \{q_{erk}\}$
$\delta : Q \times \Gamma \to \mathfrak{p}_f(Q \times \Gamma \times \{-1,0,1\})$

sei mit Quintupeln dargestellt: statt $\delta(q,X) \ni (q',X',p)$ schreibt man $qXX'pq'$

$$\left.\begin{array}{ll}
q_0 aaLq_0 & (a \in \Sigma) \\
q_0\Box[Rq_0^1 & \\
q_0^1 aaRq_0^1 & \\
q_0^1\Box]Rq_0^2 & \\
q_0^2\Box[Rq_0^3 & \\
q_0^3\Box SRq_0^4 & \\
q_0^4\Box]Nq_G &
\end{array}\right\}\text{Anfangsphase} \quad \text{erzeugt}\,[w][s_{q_G}]$$

$$\left.\begin{array}{lll}
q_G]]Nq_i & & \text{(nichtdeterministisch)} \\
q_i]\Box Lq_i^] & & \\
q_i^{Z_1\ldots Z_k}Z\Box Lq_i^{ZZ_1\ldots Z_k} & (1 \le k < n, Z, Z_\mu \in X \cup \{]\}) & \\
q_i^{Z_1\ldots Z_n}ZZ_nLq_i^{ZZ_1\ldots Z_{n-1}} & & \text{(ND)} \\
q_i^{X_1\ldots X_n}ZZRq_{i_{Y_1\ldots Y_m}} & & \text{(ND)} \\
q_{i_{Z_1\ldots Z_m}}ZZ_1Rq_{i_{Z_2\ldots Z_m Z}} & & \\
q_{i_{Z_1\ldots Z_l}}\Box Z_1 q_{i_{Z_2\ldots Z_l}} & & (1 \le l \le m) \\
q_{i_\varepsilon}\Box\Box Lq_G & & \\
q_{i_\varepsilon}ZZRq_{i_\varepsilon} & & (\text{falls} m = 0)
\end{array}\right\}\text{Simulation}$$

$$\left.\begin{array}{ll}
q_G]]Nq_V & \text{(ND)} \\
q_V]\Box Lq_V & \\
q_V a]Lq^a & (a \in \Sigma \cup \{[\}) \\
q^a bbLq^a & (b \in \Sigma \cup \{[,\Box\}) \\
q^a]\Box Lq_a & \\
q_a a]R\vec{q} & (a \in \Sigma) \\
\vec{q}\,bbR\vec{q} & (b \in \Sigma \cup \{[,\Box\}) \\
\vec{q}]]Nq_V & \\
q_[[[Nq_{erk} & (q_{erk}[\Box\ldots\Box])
\end{array}\right\}\text{Erkennungsphase}$$

$\Box$

4.2.5 Platzbedarf für Turingmaschinen

Definition: Sei $\mathfrak{A} \in \mathrm{TM}(\Sigma), \gamma = (\kappa_0 \vdash \kappa_1 \vdash \ldots \vdash \kappa_n)$ eine Berechnung und $w \in L(\mathfrak{A})$. Dann ist

- $|\kappa_i| := |\alpha X \beta|$ falls $\kappa_i = (q,\alpha,X,\beta)$

- $\underline{\mathrm{bv}}_{\mathfrak{A}}(\gamma) := \underline{\max}\{|\kappa_i| | 0 \le i \le n\}$

- $\underline{\mathrm{bv}}_{\mathfrak{A}}(w) := \underline{\min}\{\underline{\mathrm{bv}}_{\mathfrak{A}}(\gamma) | \gamma \text{ erkennt } w\}$

der *Bandverbrauch* von $\mathfrak{A}$ für γ, bzw. w.

Definition: $\mathfrak{A} \in$ TM heißt *linear beschränkt* ($\mathfrak{A} \in$ lbTM), wenn $p \geq 1$ existiert, so daß für alle $w \in L(\mathfrak{A})\backslash\varepsilon$

$$\underline{\mathrm{bv}}_{\mathfrak{A}}(w) \leq p \cdot |w|$$

Korollar: $\mathscr{L}_1(\Sigma) = \mathscr{L}(\Sigma, \mathrm{lbTM})$.
Grund: Simulationen erhalten linearen Platzbedarf/Bandverbrauch.

Definition: $\mathfrak{A} \in \mathrm{LBA}(\Sigma)$ („*linear beschränkter Automat*"): $\curvearrowleft \mathfrak{A} \in$ lbTM mit $p = 1$, d.h. nur Eingabefelder werden benutzt.

Satz: $\mathscr{L}_1(\Sigma) = \mathscr{L}(\Sigma, \mathrm{LBA})$
Beweis: (Idee) Kompression des benutzten Bandbereichs durch Vergrößerung des Arbeitsalphabets $\Gamma' := \Gamma^p$.

□

4.2.6 Deterministische TM, k-Band-TM

Reduktion nicht-deterministische TM auf deterministische TM in 2 Schritten:

1. Simulation einer nicht-deterministischen TM durch eine 3-Band-deterministische-TM.
2. Reduktion von k-Band-deterministischen-TM auf 1-Band-deterministische-TM.

deterministische k-Band-TM

Definition: $\mathfrak{A} = < Q, \Sigma, \Gamma, q_0, \Box, F, \delta > \in \mathrm{k-dTM}$, wenn: $\delta : Q \times \Gamma^k \dashrightarrow Q \times (\Gamma \times \{L, N, R\})^k$ und sonst wie TM.

Konfiguration: $Q \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$

Einzelschrittfunktion: simultanes Arbeiten auf k Bändern gemäß δ.

Anfangskonfiguration: für $w \in \Sigma^*$:

$$\kappa_w := \begin{cases} (q_0, (\varepsilon, a, v)(\varepsilon, \Box, \varepsilon)^{k-1}) & \text{falls}\, w = av \\ (q_0, (\varepsilon, \Box, \varepsilon)^k) & \text{falls}\, w = \varepsilon \end{cases}$$

$L(\mathfrak{A}) := \{w \in \Sigma^* | \kappa_w \overset{*}{\vdash} (q, \ldots) \text{ mit } q \in F\}$

Satz: Zu jedem $\mathfrak{A} \in$ TM läßt sich ein äquivalentes $\mathfrak{A}' \in 3-\mathrm{dTM}$ konstruieren.

Beweis: Für $\delta_{\mathfrak{A}} : Q \times \Gamma \to \mathfrak{p}_f(Q \times \Gamma \times \{L, N, R\})$, gelte $r := \underline{\max}\{|\delta(q, X)| | (qX) \in Q \times \Gamma\}$

Bezeichnung der Alternativen von $\mathfrak{A}$ durch Elemente von $[r] := \{1, \ldots, r\}$.

Berechnung von $\mathfrak{A}$ bestimmt $\zeta \in [r]^*$.

Umgekehrt: Auswahl einer Berechnung durch ein solches ζ als Steuerwort.

Band 1: Eingabe, wiederholtes Lesen.

Band 2: Erzeugung der Zahlenfolgen $\zeta \in [r]^*$.

Band 3: Simulation von $\mathfrak{A}$ gemäß Band 2.

Sukzessive Berechnung aller $\zeta \in [r]^*$. Für jedes ζ Kopie der Eingabe auf Band 3. Dann Simulation gemäß Band 2.

□

Satz: Zu jedem $\mathfrak{A} \in \mathrm{k-dTM}$ läßt sich eine äquivalente $\mathfrak{A}' \in 1-\mathrm{dTM}$ konstruieren.

Beweis: Simulation von k Bändern durch 1 Band mit $2k$ Spuren für k Bänder und Kopfpositionen. Arbeitsalphabet $\Gamma' := (\Gamma \times \{\square, k\})^k$.
Simulation eines Arbeitsschrittes von $\mathfrak{A}$:

Lesephase: Suchen der k gelesenen Symbole, speichern in endlicher Kontrolle. Problem: i-ter Kopf rechts oder links vom Hauptkopf (Simulationskopf)? Lösung: Positionsvektor $\{l,r\}^k$ in endlicher Kontrolle.
$Q' := \underbrace{Q \times \Gamma^k}_{\delta-\text{Argument}} \times \{l,r\}^k \times \{1,\ldots,k\}$

Schreibphase: analog zur Lesephase.

□

Korollar: $\mathscr{L}(\Sigma,\mathrm{TM}) = \mathscr{L}(\Sigma,\mathrm{dTM})$.
Bemerkung: Für LBA's ist diese Frage offen: das *LBA-Problem*.

4.3 Aufzählbare und entscheidbare Sprachen

Intuitiv: $L \subseteq \Sigma^*$ ist *aufzählbar*, wenn es ein effektives Verfahren (Programm) gibt, welches genau die Elemente von L erzeugt (aufzählbar = rekursiv aufzählbar).
Präzisierung durch TM mit Ausgabe:

$$\mathfrak{A} =< Q,\Sigma,\Gamma,q_0,F,\square,\delta >\in \mathrm{dTM}(\Sigma)$$

$\mathfrak{A}$ berechnet $f_{\mathfrak{A}} : \Sigma^* \dashrightarrow \Sigma^*$ mit $f_{\mathfrak{A}}(w) = v : \curvearrowleft q_0 w \square \overset{*}{\vdash} \alpha q v \square \beta i \not\vdash$
Bezeichnung: Konfiguration statt (q,α,X,β) kurz: $\alpha q X \beta$. Endkonfiguration $\kappa \not\vdash \curvearrowleft \kappa$ hat keine Folgekonfiguration.

Beachte: Endzustände ohne Bedeutung.

4.3.1 Aufzählbare Sprachen

Definition: $L \subseteq \Sigma^*$ heißt *aufzählbar*, wenn $\mathfrak{A} \in \mathrm{dTM}(\Sigma)$ existiert mit $\underline{\mathrm{Def}}(f_{\mathfrak{A}}) = \Sigma^*$ und $\underline{\mathrm{Bild}}(f_{\mathfrak{A}}) = L$ oder wenn $L = \emptyset$.

Bemerkung: Aufzählung von Σ^* (etwa der Länge nach), es gibt eine Aufzählung von L.

Satz: Für $L \subseteq \Sigma^*$ gilt: L aufzählbar $\curvearrowleft\!\!\curvearrowright L \in \mathscr{L}(\Sigma, \mathrm{TM})$.

Beweis: (Idee)

1. $L = \emptyset$: L aufzählbar
   $L \in \mathscr{L}(\Sigma, \mathrm{TM})$, zB. $\mathfrak{A}$ mit $F = \emptyset$

2. $L \neq \emptyset$
   „$\curvearrowright$“ $\mathfrak{A} \in \mathrm{dTM}(\Sigma)$ mit $f_{\mathfrak{A}}$ total und $\underline{\mathrm{Bild}}(f_{\mathfrak{A}}) = L$.
   Konstruktion von $\mathfrak{A}' \in \mathrm{dTM}(\Sigma)$ mit $L(\mathfrak{A}') = L$.
   Idee: Eingabe von $w \in \Sigma^*$ speichern, L aufzählbar und jedes aufgezähltes Wort mit w vergleichen; bei Übereinstimmung Endzustand, andernfalls weiter aufzählen.

   „$\curvearrowleft$“ $\mathfrak{A} \in \mathrm{dTM}(\Sigma)$ mit $L(\mathfrak{A}) = L$.
   Konstruktion von $\mathfrak{A}'' \in \mathrm{dTM}(\Sigma)$ mit $f_{\mathfrak{A}''}$ total mit $\underline{\mathrm{Bild}}(f_{\mathfrak{A}''}) = L$.
   Idee:

   (a) $\mathfrak{A}$ so modifizieren, daß Eingabe bei Erkennung ausgegeben wird. Dann gilt: $\underline{\mathrm{Def}}(f_{\mathfrak{A}'}) = \underline{\mathrm{Bild}}(f_{\mathfrak{A}'}) = L$.

   (b) Damit $\mathfrak{A}'$ bei jeder Eingabe ein Wort aus L ausgibt, folgender Trick: Teil der Eingabe als Zähler zur Beschränkung der Berechnungslänge.

   $$\Sigma = \{a_1, a_2, \ldots, a_r\} \,\mathrm{mit}\, r \geq 2$$

   Eingabe hat immer die Form:
   $(*)$ $\quad a_1^n a_i w$ bzw. a_1^n mit $i \neq 1$ und $n \in \mathbb{N}$
   $\mathfrak{A}''$ simuliert dann $\mathfrak{A}'$ mit Eingabe w, bzw. ε und höchstens n Schritten. Eine Ausgabe von $\mathfrak{A}'$ ist auch Ausgabe von $\mathfrak{A}''$.
   Wurde nach n Schritten keine Ausgabe berechnet, so wird irgendein $v \in L$ als Ausgabe berechnet: $\mathfrak{A}'$ mit Längenbeschränkung und Eingabe von $\varepsilon, a_1, a_2, \ldots$ bis zur ersten Ausgabe laufen lassen

   Es folgt: $f_{\mathfrak{A}''}$ total mit $\underline{\mathrm{Bild}}(f_{\mathfrak{A}}) = L$.

$\square$

Satz: $\mathscr{L}_0(\Sigma) \subsetneqq \mathfrak{p}(\Sigma^*)$.
Beweis: Die Menge der Typ 0-Grammatiken über Σ läßt sich nach ihrer Größe abzählen. Also ist $\mathscr{L}_0(\Sigma)$ abzählbar (d.h. gleichmächtig mit $\mathbb{N}$).
$\mathfrak{p}(\Sigma^*)$ ist jedoch überabzählbar (*Diagonalverfahren* nach *Cantor*).

Einfachster Fall: $|\Sigma| = 1$, $\Sigma^* = \mathbb{N}3 = aaa$.
$L \subseteq \Sigma^* \mapsto \underline{\text{char}}_L : \mathbb{N} \to \{0,1\}, \underline{\text{char}}_L(n) = 1 \curvearrowleftright n \in L$.
Angenommen, $\mathfrak{p}(\mathbb{N})$ wäre abzählbar. Dann gäbe es eine Funktion $f : \mathbb{N}^2 \to \{0,1\}$ mit $f(i,j) = 1 \curvearrowleftright \underline{\text{char}}_{L_i}(j) = 1$

	0	1	2	3	4	5	6	...
0	0	1	0	0	1	0		
1	1	0	1	0	0	0		
2	0	1	1	1	0	0		
3	1	1	1	0	1	0		
4					$\ddots$			
$\vdots$								

Definiere $L_{\text{dia}} \subseteq \mathbb{N}$ durch
$$\underline{\text{char}}_{L_{\text{dia}}}(j) := 1 - f(j,j)$$
Dann kann L_{dia} nicht in der Abzählung vorkommen.

□

4.3.2 Entscheidbare Sprachen

Eine aufzählbare Sprache $L \subseteq \Sigma^*$ kann zwar durch ein $\mathfrak{A} \in \text{dTM}(\Sigma)$ als $L(\mathfrak{A})$ erkannt werden. Da aber die Berechnung von $\mathfrak{A}$ bei Eingabe von $w \in \Sigma^*$ nicht terminieren muß, läßt sich im allgemeinen auch nicht entscheiden, ob $w \in L$ oder $w \notin L$.

Intuitiv: $L \subseteq \Sigma^*$ ist entscheidbar, wenn es ein effektives Verfahren gibt, welches für jedes $w \in \Sigma^*$ feststellt, ob $w \in L$ oder $w \notin L$ gilt.

Präzisierung durch eine dTM **mit Ausgabe**

Definition: $L \subseteq \Sigma^*$ heißt *entscheidbar*: $\curvearrowleftright$ es gibt $\mathfrak{A} \in \text{dTM}(\Sigma)$ mit $\underline{\text{Def}}(f_{\mathfrak{A}}) = \Sigma^*$ und $L = f_{\mathfrak{A}}^{-1}(\varepsilon)$

Satz: Für $L \subseteq \Sigma^*$ gilt:
L entscheidbar $\curvearrowleftright L$ und $\Sigma^* \backslash L$ aufzählbar.

Beweis:

„$\curvearrowright$" L entscheidbar, $\mathfrak{A} \in \text{dTM}$ mit $f_{\mathfrak{A}}$ total und $L = f_{\mathfrak{A}}^{-1}(\varepsilon)$.
Konstruiere $\mathfrak{A}'$ mit $L(\mathfrak{A}') = L$ und $\mathfrak{A}''$ mit $L(\mathfrak{A}'') = \Sigma^* \backslash L$ durch passende Wahl der erkennenden Endzustände.

„$\curvearrowleft$" $\mathfrak{A}_i \in \text{dTM}(\Sigma), f_{\mathfrak{A}_i}$ total, $\underline{\text{Bild}}(f_{\mathfrak{A}_1}) = L$, bzw. $\underline{\text{Bild}}(f_{\mathfrak{A}_2}) = \Sigma^* \backslash L$.
Konstruiere „*Endscheidungs-Turingmaschine*" nach *Reißverschlußprinzip*:
$w \in \Sigma^*$, berechne $f_{\mathfrak{A}_1}(\varepsilon), f_{\mathfrak{A}_2}(\varepsilon), f_{\mathfrak{A}_1}(a_1), f_{\mathfrak{A}_2}(a_1), \ldots$
bis $f_{\mathfrak{A}_i}(v) = w$.

□

Satz: $\mathrm{DEC}(\Sigma) := \{L \subseteq \Sigma^* | L \text{ entscheidbar } \} \subsetneqq \mathscr{L}_0(\Sigma)$

Beweis: (Idee) Die „*Selbstanwendungsmenge*“ ist nicht entscheidbar (*Diagonalisierungsprinzip*).

□

Satz: Jede Typ 1-Sprache über Σ ist entscheidbar.
Beweis: Sei G eine Typ 1-Grammatik über Σ und $w \in \Sigma^*$.

Fall 1: $w = \varepsilon \in L \curvearrowleft\!\!\curvearrowright S \to \varepsilon$ in P

Fall 2: $w \neq \varepsilon, w \in L \curvearrowleft\!\!\curvearrowright \exists$ Ableitung $S \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$ mit $|\alpha_i| \leq |w|$. Nur endlich viele solcher Ableitungen (können einfach durchgetestet werden).

□

Kapitel 5

Unentscheidbare Probleme

Ziel: Die folgenden Probleme sind unentscheidbar:

- Das Wortproblem für Typ 0-Beschreibungen (Haltproblem für Turingmaschinen, siehe Vorlesung „Berechenbarkeit und Komplexität“).
- Das $\emptyset$-Problem für Typ 1-Beschreibungen.
- Das Äquivalenzproblem für Typ 2-Beschreibungen.

Bemerkung: Das Äquivalenzproblem für DPDA's wurde kürzlich als entscheidbar nachgewiesen.
Hilfsmittel: Das Postsche Korrespondenzproblem (vgl. „BuK“).
Seien $w = (w_1, w_2, \ldots, w_n)$ und $v = (v_1, v_2, \ldots, v_n)$ mit $w_i, v_i \in \Sigma^*$ und $1 \leq i \leq n$.
Dann heißt $(i_1, \ldots, i_k) \in \{1, \ldots, n\}^k$ eine Lösung von PCP(w, v), fall $w_{i_1} w_{i_2} \ldots w_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$.
Es gilt: Das PCP ist unentscheidbar.
Beweis: (Idee) Reduktion des Haltproblems für TM auf das PCP.

5.1 Das $\emptyset$-Problem von lbTM

Satz: Das $\emptyset$-Problem von lbTM ist unentscheidbar.

Beweis: Reduktion des PCP auf das $\emptyset$-Problem von lbTM.
Sei $w = (w_1, \ldots, w_n), v = (v_1, \ldots, v_n) \in (\Sigma^*)^n$. Konstruktion von $\mathfrak{A}(w, v) \in$ lbTM mit

$$\text{PCP}(w, v)\,\text{lösbar} \curvearrowright L(\mathfrak{A}(w, v)) \neq \emptyset$$

$\mathfrak{A}(w, v)$ erzeugt bei Eingabe von $u \in \Sigma^+$ alle Indexfolgen $(i_1, \ldots, i_k) \in \{1, \ldots, n\}^k$ mit $k \leq |u|$. (Idee: Eingabe als Zähler verwenden) und prüft jeweils, ob $w_{i_1} w_{i_2} \ldots w_{i_k} = v_{i_1} v_{i_2} \ldots v_{i_k}$.
$\mathfrak{A}$ erkennt u, falls eine solche Indexfolge auftritt. Platzbedarf linear in $|u|$.
Wäre $\emptyset$-Problem für lbTM entscheidbar, so auch PCP: Widerspruch.

$\square$

5.2 Schnittproblem für kontextfreie Grammatiken

Satz: Es ist nicht entscheidbar, ob für $G_1, G_2 \in \mathrm{CFG}$ gilt: $L(G_1) \cap L(G_2) = \emptyset$.
Beweis: Reduktion des PCP auf dieses Schnittproblem.
Sei $w = (w_1, \dots, w_n)$ und $v = (v_1, \dots, v_n) \in (\Sigma^*)^n$. Konstruiere $G_w, G_v \in \mathrm{CFG}$, so daß

$$\mathrm{PCP}(w,v)\,\text{lösbar} \curvearrowright L(G_w) \cap L(G_v) \neq \emptyset$$

$\tilde{\Sigma} := \Sigma \uplus \{b_1, \dots, b_n\}$ „Indexsymbole“
$G_w := S_w \to w_i S_w b_i | w_i b_i \qquad (1 \leq i \leq n)$
G_v analog

Es folgt: $L_w := L(G_w) = \{w_{i_1} \dots w_{i_k} b_{i_k} \dots b_{i_1} | k \geq 1, 1 \leq i_\mu \leq n\}$, L_v analog und

$$L_w \cap L_v \neq \emptyset \curvearrowright \exists\, i_1, \dots, i_k \in \{1, \dots, n\}$$
$$w_{i_1} \dots w_{i_k} b_{i_k} \dots b_{i_1} = v_{i_1} \dots v_{i_k} b_{1_k} \dots b_{i_1}$$

also $w_{i_1} \dots w_{i_k} = v_{i_1} \dots v_{i_k} \curvearrowright \mathrm{PCP}(w,v)$ lösbar.

□

Korollar: Das Schnittproblem für DPDA’s ist nicht entscheidbar.
Beweis: G_w und G_v sind durch deterministische Kellerautomaten simulierbar.
Idee: w_i’s in den Keller lesen und dann mit den b_i’s löschen.

□

5.3 Äquivalenzproblem für CFG

Satz: Es ist nicht entscheidbar, ob für $G_1, G_2 \in \mathrm{CFG}$ gilt:

$$L(G_1) = L(G_2)$$

Beweis: Reduktion des Schnittproblems für DPDA’s auf das Äquivalenzproblem von CFG.
Beachte:

1. Zu $\mathfrak{A} \in \mathrm{DPDA}(\Sigma)$ läßt sich $\overline{\mathfrak{A}} \in \mathrm{DPDA}(\Sigma)$ konstruieren mit $L(\overline{\mathfrak{A}}) = \overline{L(\mathfrak{A})}$.
2. Zu $\mathfrak{A} \in \mathrm{PDA}(\Sigma)$ läßt sich $G_{\mathfrak{A}} \in \mathrm{CFG}(\Sigma)$ konstruieren mit $L(G_{\mathfrak{A}}) = L(\mathfrak{A})$.
3. Zu $G_1, G_2 \in \mathrm{CFG}(\Sigma)$ läßt sich $G_\cup \in \mathrm{CFG}(\Sigma)$ konstruieren mit $L(G_\cup) = L(G_1) \cup L(G_2)$.

Hieraus folgt für $\mathfrak{A}_1, \mathfrak{A}_2 \in \mathrm{DPDA}(\Sigma)$:

$$\begin{aligned}
L(\mathfrak{A}_1) \cup L(\mathfrak{A}_2) = \emptyset \quad &\curvearrowright \quad L(\mathfrak{A}_1) \subseteq \overline{L(\mathfrak{A}_2)} \\
&\curvearrowright \quad L(\mathfrak{A}_1) \subseteq L(\overline{\mathfrak{A}}_2) \\
&\curvearrowright \quad L(\mathfrak{A}_1) \cup L(\overline{\mathfrak{A}}_2) = L(\overline{\mathfrak{A}}_2) \\
&\curvearrowright \quad L(G_{\mathfrak{A}_1}) \cup L(G_{\overline{\mathfrak{A}}_2}) = L(G_{\overline{\mathfrak{A}}_2}) \\
&\curvearrowright \quad L(G_\cup) = L(G_{\overline{\mathfrak{A}}_2})
\end{aligned}$$

Die Entscheidbarkeit der Äquivalenzproblems für CFG würde somit die Entscheidbarkeit des $\cap$-Problems von DPDA’s implizieren.

□

5.4 Mehrdeutigkeitsproblem

Satz: Es ist nicht entscheidbar, ob ein $G \in$ CFG mehrdeutig ist, oder nicht.
Beweis: Reduktion des PCP auf das Mehrdeutigkeitsproblem.
Seien $w = (w_1, \ldots, w_n)$ und $v = (v_1, \ldots, v_n) \in (\Sigma^*)^n, \tilde{\Sigma} := \Sigma \cup \{b_1, \ldots, b_n\}$. Konstruktion von

$$\begin{array}{rcll} G_{(w,v)} & = & S \rightarrow A_w | A_v & \\ & & A_w \rightarrow w_i A_w b_i | w_i b_i & (1 \leq i \leq n) \\ & & A_v \rightarrow v_i A_v b_i | v_i b_i & (1 \leq i \leq n) \end{array}$$

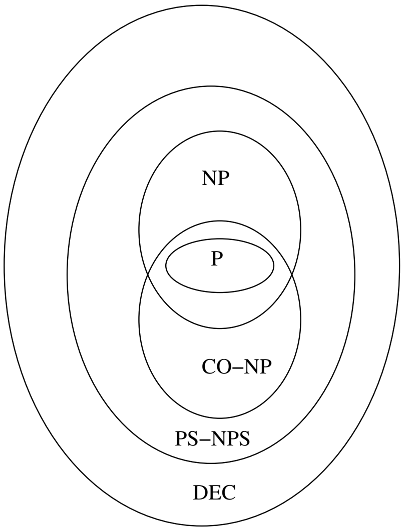
Es folgt:

$$\begin{array}{rcl} (i_1, \ldots, i_k) \,\text{löst}\, \mathrm{PCP}(w,v) & \curvearrowright & w_{i_1} \ldots w_{i_k} = v_{i_1} \ldots v_{i_k} \\ & \curvearrowright & w_{i_1} \ldots w_{i_k} b_{i_k} \ldots b_{i_1} = v_{i_1} \ldots v_{i_k} b_{i_k} \ldots b_{i_1} = Z \\ & \curvearrowright & S \Rightarrow A_w \stackrel{*}{\Rightarrow} Z \,\text{und}\, S \Rightarrow A_v \stackrel{*}{\Rightarrow} Z \\ & \curvearrowright & G_{(w,v)} \,\text{mehrdeutig} \end{array}$$

□

Zusammenfassung siehe Folie 4.4

5.5 Komplexität der entscheidbaren Probleme



5.5.1 Äquivalenzproblem bei Typ 3-Beschreibungen

DFA**:** $\rho(n^2)$

NFA/Reg$E$**:** exponentiell, NP-hart, PS-vollständig

5.5.2 Wortproblem

DFA: linear

CFG: $\rho(n^3)$

CSG: exponentiell, NP-hart, PS-vollständig

Index