Buch:
Hopcroft, Motwani, Ullman
Einführung in der Automatentheorie, Formale Sprachen und Komplexitätstheorie
Automaten allgemein
möglicher Klausurtermin: 4.2.09: 18-20 Uhr
Kontrolleinheit
-

Grammatiken
Eine Grammatik G erzeugt Zeichenreihen
Formal:
Def.: Eine Grammatik G ist ein Tupel G = (N,T,P,S)
N endliche Menge von Hilfszeichen (Nichtterminalen)
T endliche Menge von terminalen Zeichen
P endliche Menge von Regeln (Produktionen)
$S \in N$: Axiom(Startsymbol)

Beispiel:
$T = \{V, +, *, 1, (,)\}, N = \{\pi, \alpha, \phi, \tau\}, S \equiv \alpha$
$P : \pi \rightarrow V$
$\pi \rightarrow (\alpha)$
$\phi \rightarrow \pi$
$\phi \rightarrow \phi \uparrow \pi$
$\tau \rightarrow \phi$
$\tau \rightarrow \tau * \phi$
$\alpha \rightarrow \tau$
$\alpha \rightarrow \alpha + \tau$

Eingabe:

V	*	V	↑	V
π		π		π
ϕ		ϕ		
τ			ϕ	
	τ			
	α			

Arbeitsweise: (2 Möglichkeiten)
1. Top-Down (von oben nach unten, reduzieren)
gegeben Zeichenreihe
Man versucht ausgehend von einer Zeichenreihe mit Hilfe der Regeln das Axiom zu erzeugen.
2. Buttom-Up (von unten nach oben, erweitern)
gegeben: Axiom:
-ausgehen von dem Axiom versucht man mit Hilfe der Regeln die Zeichenreihe zu erzeugen.

Variationen:
1. deterministisch vs. nicht-deterministisch
2. Arten der Regeln: Allgemein: Zeichenreihe → Zeichenreihe (beide aus terminale und nicht-terminale Zeichen)
- linke Seite nur ein Nicht-terminalsymbol
- Länge der Zeichenreichen ist beschränkt
- spezielle Formen für linke und rechte Zeichenkette

Sprachen:

Def. 1: Eine Zeichenreihe mit terminalen Zeichen, die mit Hilfe der Regeln aus P auf das Axiom S realsisiert werden kann, heißt von G erzeugtes Wort.
Def. 2: Eine Zeichenreihe mit terminalen Zeichen, die mit Hilfe der Regeln aus P vom Axiom S aus abgeleitet werden kann, heißt von G erzeugtes Wort(„Definition gleichwertig“)
Def. 3: Die Menge der von G erzeugten Wörter heißt Sprache von G L(G).

Zusammenhang zwischen Automaten und Grammatiken
Automat $\underbrace{\rightarrow}_{\text{akzeptiert}}$ Sprache $\underbrace{\leftarrow}_{\text{erzeugt}}$ Grammatik
Ein Automat A ist äquivalent zu einer Grammatik G, genau dann, wenn $L(A) = L(G)$

Vorlesung:

1. Mathematischen Grundlagen
   Diskrete Mathematik, Zeichenreihe, Graphen, Logik, Relationen, Fkt
2. endliche Automaten
   Variationen, deterministisch, nicht-deterministisch, ϵ-Übergänge, Ausgaben
3. Reguläre Grammatiken, Mengen und Ausdrücken
   Definitionen
   Äquivalenz endliche Automaten
   Eigenschaften (Abgeschlossen gegenüber verschiedenen Operationen usw.)
4. Kellerautomaten
   Definition (incl. Varianten)

5. Kontextfreie Grammatiken
   Definition + Eigenschaften
   Äquivalenz: Kellerautomat ↔ kfr. Grammatiken
6. Stapelautomat
7. Turingmaschine
   Definition Variationen , Eigenschaften, Berechenbarkeit
8. Unbeschränkte Grammatiken
   Eigenschaften, Äquivalenz von Turingmaschinen
9. linear beschränkte Automaten
10. kontextsensitive Grammatiken
11. Chomsky
12. LR(k)-Grammatiken
13. Baumautomaten
14. Berechenbarkeitstheorie

§1. Mathematischen Grundlagen

Mengen

Grundmenge G, Elemente aus der Grundmenge
Festlegung: $x \in G$ oder $x \notin M$, $x \in G$
Anzahl der Elemente in M : $|M|$
Schreibweise : $\{a_1, .., a_n\}, \{2, 4, 6, ...\} = \{n \in N | n \mod 2 = 0\}$, charateristischen Prädikat $M = \{x \in G | P(x)\}$
charakteristische Funktion: $X_n : G \to \{0, 1\}$
$M = G \Rightarrow M$ heißt Universalmenge, leere Menge $\emptyset$

Teilmenge

Potenzmenge: die Menge aller Teilmengen von M heißt Potenzmenge von M
$(2^M, P(M)), |M| = n \Rightarrow |P(M)| = 2^n$
Es gilt stets: $\emptyset \in P(M), M \in P(M)$
Komplement: $A^c = \overline{A}$
Schnitt: $A \cap B$
Vereinigung: $A \cup B$
Differenz: $A - B$
Symmetrische Differenz: $A * B = (A - B) \cup (B - A)$
Potenzmenge
Kartesisches Produkt von A und B. $A * B = \{(x, y) | x \in A \wedge y \in B\}$
Eigenschaften: $(A^c)^c = A$
$X_{(A^c)^c}(x) = X_A(x)$

Def.: $A_1, .., A_n$ Mengen, $x_1 \in A_1, ..., x_n \in A_n$
$(x_1, .., x_n)$ gewichtetes Tupel über $A_1, .., A_n$ (Reihenfolge der x_i wichtig)
$(A_1 \times .. \times A_n) = \{(x_1, .., x_n) | x_1 \in A_1, .., x_n \in A_n\}$
Beispiel: A: $\{1, 2, 3\}, B = \{2, 3\}$
$A \times B = \{(1, 2), (1, 3), (2, 2), (2, 3)\}$
$P(A) = \{\emptyset, \{1\}, \{2\}, \{2, 3\}\}$
Relationen:
Eine n-stellige Relation P über den Mengen $A_1, .., A_n$ ist eine Teilmenge von $A_1 \times .. \times A_n (R \subseteq A_1 \times .. \times A_n)$
$R \subseteq M \times M$ = Relation „$\times$“ auf M.
Schreibweise $(x_1, .., x_n)$ auf $(x_1, x_2) \subseteq R \equiv x_1 R x_2$
Eigenschaften(kleiner Relation)
Symmetrisch
antisymmetrisch
asymmetrisch
reflexiv
irreflexiv
transitiv

Sei R eine Relation auf M

1. R heißt Äquivalenzrelation, wenn R symmetrisch, transitiv und reflexiv ist.
2. R heißt Ordnungsrelation, wenn R antisymmetrisch, transitiv und reflexiv ist.

Ist R eine Äquivalenzrelation auf M dann gilt $M = M_1 \cup M2 \cup ...$ mit

1. $M_i \cap M_j = \emptyset$
2. Für alle $a, b \in M_j$ gilt aRb
3. Für alle $a_i \in M_i$ und $b \in M_j$, $i \neq j$, gilt nicht aRb.

Definition: (Transitive Hülle, transitive, reflexive Hülle)
Relation R auf M. Seinen Ferne die Relationen $R^0 := \{(x, x) | x \in M\}$ und $R^i := R^{i-1}R := \{(x, z) \in M \times M |$ Es existiert ein $x \in M$ mit $(x, y) \in R^{i-1}, (y, z) \in R\}$
$R^+ := \{R^1 \cup R^2 \cup ...\}$ transitive Hülle von R
$R^* := \{R^0 \cup R^1 \cup ...\}$ transitiv reflexive Hülle von R

Beispiel: $M = \{1, 2, 3\}$

$R = \{(1,2),(2,2),(2,3)\}$
$R^+ = \{(1,2),(2,2),(2,3),(1,3)\}$
$R^* = \{(1,1),(2,2),(3,3),(1,2),(1,3),(2,3)\}$

Funktionen:

Def: Gegeben: $R \subseteq M \times N$

1. R linkstotal $\Rightarrow$ für alle $x \in M$: (es existiert $y \in N$ mit $(x,y) \in R$)
2. R rechtstotal $\Rightarrow$ für alle $y \in N$: (es existiert $x \in M$ mit $(x,y) \in R$)
3. R rechtseindeutig $\Rightarrow$ für alle $x \in M, y \in N, (x,y) \in R$ und $(x,z) \in R \Rightarrow y = z$
4. R linkseindeutig $\Leftrightarrow$ für alle $x \in N, y \in M, (y,x) \in R$ und $(z,x) \in R \Rightarrow y = z$

Definition:

1. Eine eindeutige Relation $F \subseteq M \times N$ heißt (partielle) Funktion von M nach N.
2. Ist F zusätzlich linkstotal, dann heißt F total definierten Funktion

Definition: Eigenschaften von Funktionen $F \subseteq M \times N$ total definierte Funktion

1. F rechtstotal: F surjektion
2. F linkseindeutig : F injektion
3. F surjektion + injektion : F Bijketion

2-stellige Verknüpfung: $R \subseteq (A \times A) \times B$, z.B: + (3,4) = 7
gegeben: $\circ : A \times A \to B$, 2-stellige Verknüpfung

1. $B = A : \circ$ abgeschlossen
2. $\circ$ heißt assioziativ: $(x \circ y) \circ z = x \circ (y \circ z)$
3. n heißt neutrales Element, wenn für alle $x \in A, n \circ x = x \circ n = x$
4. $x, y \in A$ heißen invers, falls $x \circ y = y \circ x = n$

Monoide, Halbgruppen
Definition: Eine Halbgruppe besteht aus: einer Menge A und einer auf A abgeschlossenen und assoziativen Verknüpfung $\circ$, in Zeichen $H = [A, \circ]$.
Existiert ein neutrales Elemet, so heißt H Monoid.
Realtionen zwischen Halbgruppen ind Monoiden
zug, 2 Halbgruppen, $H_1 = [H, \circ], H_2 = [B, *]$
$f : A \to B$ heißt Homomorphismus von A in B, falls $f(x_1 \circ x_2) = f(x_1) * f(x_2)$.
Bei Monoiden muss zusätzlich gelten: $f(n_1) = n_2$
f: surjektiv $\Rightarrow$ f Epimorphismus
f: injektiv $\Rightarrow$ f Monomorphismus
f: bijektiv $\Rightarrow$ f Isomorphismus
A Alphabet: A ist nichtleere, endliche Menge von Zeichen, die total geordnet sind.
Zeichenreihe: Eine total definierte Funktion $Z : [1...n] \to A$ heißt Zeichenreihe n = Länge.
gegeben: Alphabet A:
A^* Menge aller Zeichenreihen über A einschließlich der leeren Zeichenreihe ϵ
A^+ Menge aller Zeichenreihen über A ohne epsilon
A^n Menge aller Zeichenreihen über A der Länge n
$a^n, a \in A$ Zeichenreihe(Wort) mit $n = a's$
Verknüpfung $\equiv$ Kombination
$a_1...a_n \circ b_1...b_m \equiv a_1...a_n b_1...b_n$ $\quad$ $[A^*, \circ]$ ist eine freies Monoid

Logik:
logische Ausdruck über das Alphabet $A = \{a, .., z, \vee, \wedge, \neg, W, F, \to, \leftarrow, \leftrightarrow, (,)\}$

1. kleine Buchstaben sind logischer Ausdruck
2. W,F sind logischer Ausdruck
3. A, $\neg$ logischer Ausdruck, dann auch (A), nicht A, $(A * B)$

Semantik, gegeben Wahrheitswert: W,F
Definition: Sei $A(x_1, .., x_n)$ eine logischer Ausdruck (n-stellig)

1. $A(x_1, .., x_n)$ heißt erfüllbar(konsistent), falls es es eine Belegung mit Wahrheitswert für die x_i gibt, so dass die Auswertung δ den Wert W ergibt.
2. A heißt unerfüllbar(inkonsistent), wenn es keine einzige derartige Belegung gibt.
3. $A(x_1, .., x_n)$ heißt Tautologie, wenn jede mögliche Belegung den Wert W liefert.

$A \cap (\neg A)$ ist nicht erfüllbar

A	$\neg A$	$A \cap (\neg A)$
W	F	F
W	F	F
F	W	F
F	W	F

logische Ausdrücke heißen „äquivalent“, wenn sich für jede mögliche Belegung der gleiche Wahrheitswert ergibt. $(A \Leftrightarrow B)(A \equiv B)$
Definition:Seien $A_1, ..., A_n, B_1, ..., B_n$ n-stellige logische Ausdrücke. Wenn für jede Belegung der X_i für die der Wahrheitswert der $A_i = W$ ist auch die Wahrheitswerte der $B_i = W$ sind, so heissen die B_i Folgerung aus $A_1, .., A_n$.

Graphen:
Definition: Ein Graph G ist eine zweistellige Relation auf einer Menge V.
V ist (endliche) Menge der Knoten.
Ein Graph kann als Menge von Paaren (x,y) dargestellt werden.
ⓧ→ ⓨ, Kanten können benannt werden(gewichteter Graph).
Matrixdarstellung:
Zeilen und Spalten sind Knoten: eine „1“ bedeutet Kante, eine „0“ bedeutet keine Kante.
bei gewichteten Graphen steht in der Matrix die Benennung der Kante.

	x	y	z
x	0	0	0
y	0	0	1
z	0	0	0

Definition: Ein Graph $G \subseteq V \times V$ „ungerichteter“ Graph, wenn G symmetrisch ist (Alle Verbindungen gehen in beide Richtungen). ⓧ⇆ ⓨ ≈ ⓧ - ⓨ
Weg: Eine Folge: $w : K_1, K_2, ..., K_n = V$ mit K_i sind Knoten und $(K_i, K_{i+1}) \in G$ (zyklusfrei, Weg mit Zyklen)
Ein Graph heißt zusammenhängend, wenn es zwischen zwei Konten eine Weg gibt.
Endliche Automaten:
u.a. Grundlage aller Formalen Spezifikationssprachen (Prozeßmodellierung)
lexikalische Analyse bei Compilern
Definition: (endlicher Automat), (1-Weg-deterministischer endlicher Automat ohne

ϵ-Übergang
Eine EA ist ein Tupel $EA = (Q, \Sigma, q_0, F)$
Q : endliche Menge von Zuständen
Σ : endliche Menge von erlaubten Eingabesymbolen
$\delta : Q \times \Sigma \to Q$, Übergangsfunktion
$q_0 \in Q$,
$F \subseteq Q$, Menge von Endzuständen

Darstellungen:

1. Mengendarstellung
2. Matrixdarstellung
   Zeilen entsprechen den Zuständen, die Spalten den Eingabesymbolen
   in der Matrix stehen die neuen Zustände
3. Graphendarstellung
   Graph: Knoten sind die Zustände
   Existiert eine Regel $\delta(q, a) = q' \Rightarrow$ Es existiert eine Kante zwischen q und q' mit der Benennung „a“.
   Endzustand entspricht Doppelkreis
   q_0 entspricht Kreis mit eingehende unbenannte Kante

Beispiel: Mengendarstellung: $A = (Q, \Sigma, \delta, q_0, F)$
$Q = \{q_0, q_2, q_1\}, \Sigma = \{a, b, c\}, F = \{q_1, q_2\}$
$\delta = \{((q_0, a), q_0), ((q_0, b), q_1), ((q_0, c), q_2)\}$
Matrix:

	a	b	c
q_0	q_0	q_1	q_2
q_1			
q_2			

Ein von einem endlichen Automaten akzeptiertes Wort ist diejenige Zeichenreihe, die auf der Eingebe gelesen wurde, bis der Automat seinen Endzustand erreicht.
Sprache $\approx$ Menge aller akzeptierten Wörter.
Erweiterung der Definition von S auf „mehrere“ Schritte zu $\hat{\delta}$
$\hat{\delta} : Q \times \Sigma^* \to Q$
$\hat{\delta}(q, \epsilon) = q$

$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$
$w \in \Sigma^*, a \in \Sigma, q \in Q$
Definition (akzeptierte Sprache)
Eine Zeichenreihe $w \in \Sigma^*$ wird erst von einen endlichen Automaten akzeptiert, falls $\hat{\delta}(q_0, w) = q' \in F$
Definition: Die von endlichen Automaten akzeptierten Sprachen heißen auch „reguläre Sprachen“.

Nichtdeterministische endliche Automaten:
Definition (NEA)
Sei NEA ist ein Tupel $(Q, \Sigma, \delta, q_0, F)$
Q endliche Menge von Zuständen
Σ endliche Menge von Eingabesymbole
$q_0 \in Q$ Startzustand
$F \subseteq Q$ Endzustände
$(*)\delta : Q \times \Sigma \to \mathscr{P}(Q) \equiv 2^q$

Erweiterung von δ zu $\hat{\delta}$
$\hat{\delta} : Q \times \Sigma^* \to \mathscr{P}(Q)$

1. $\hat{\delta}(q, \epsilon) = q$
2. $\hat{\delta}(q, w, a) = \{p|$ es gibt einen Endzustand $\sigma \in \delta(q, w)$, für den $p \in \delta(\sigma, w)$ ist.$\}$

Definition der Sprache analog zu einem deterministischen Automaten.

Konstruktionsidee für den Beweis der Äquivalenz von DEA und NEA
$q_1 \xrightarrow{a} q_2$
$q_1 \xrightarrow{a} q_3$
$\Rightarrow q_1 \xrightarrow{a} (q_2, q_3)$

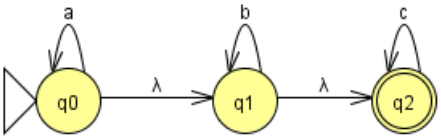
Satz: Gegeben: Sei ein NEA. Dann läßt sich effektiv ein äquivalenter (deterministischer) endlicher Automat Konstruieren mit L(EA) = L(NEA)
Bew.: Gegeben: $NEA = (Q, \Sigma, \delta, q_0, F)$
Der äquivalente EA ist gegeben durch $(Q', \Sigma, \delta', q_0', F')$ mit
1. $Q' = P(Q)$. Elemente von Q' sind bezeichnet durch $[q_1, a_2, ...q_n]$
2. F' ist die Menge aller Zustände aus Q' die einen Zustand auf F enthalten.
3. $q_0' = [q_0]$
4. $\delta'([q_1, q_2, ...], a) = [p_1, p_2, ...p_g] \Leftrightarrow p_1, p_2, ...p_g = \bigcup_{l=1,..,i} \delta(q_l, a)$

EA'S mit ϵ-Übergängen:
Def.: Ein EA mit ϵ-Übergängen (ϵ-NEA) ist erweitert um:
$\delta : Q \times (\Sigma, \epsilon) \to \mathscr{P}(Q)$
Eine Zeichenreihe wird akzeptiert, wenn es einen mit w markierten Pfad gibt, die auch ϵ-Kanten enthalten kann, der von q_i in einem Endzustand führt.
Def.: Die ϵ-Hülle(q) $q \in Q$, ist die Menge aller Knoten p, für die es eine Weg von q nach p gibt, das Kanten müssen mindestens mit ϵ markiert sind (einschließlich q selbst).

Definition von δ (ϵ-NEA) induktiv gegeben durch:
1. $\hat{\delta}(q, \epsilon) = \epsilon$-Hülle(q)
2. $\hat{\delta}(q, wa) = \epsilon$-Hülle(p) mit $w \in \Sigma^*$, $a \in \Sigma$ und P = {p| es existiert ein r aus $\hat{\delta}(q, w)$ und $p \in \delta(r, a)$}
Erweiterung auf Zustandsmenge R
3. $\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$
4. $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$

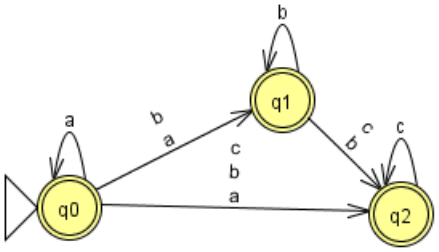
Definition: Sprache eines ϵ-NEA
Die von einem ϵ-NEA akzeptierte Sprache L(ϵ-NEA) ist gegeben durch L(ϵ-NEA) = {w|$\hat{\delta}(q_0, w)$ enthält einen Zustand aus F.}
$\hat{\delta}(q_0, \epsilon) = \epsilon$-Hülle von $q_0 = \{q_0, q_1, q_2\}$
$\hat{\delta}(q_0, a) = \epsilon$-Hülle($\delta(\hat{\delta}(q_0, \epsilon), a)$) = .. = ϵ-Hülle von q_0
$\hat{\delta}(q_0, ab) = \{q_1, q_2\}$
Satz:
Zu jedem ϵ-NEA läßt sich effektiv ein NEA bestimmen mit L(ϵ-NEA) = L(NEA)

NEA mit ϵ-Übergängen
Satz: Gegeben: ϵ-NEA läßt sich effektiv eine NEA-Konstruieren mit L(ϵ-NEA) = L(NEA)
Beweis: ϵ-NEA: $(Q, \Sigma, \delta, q_0, F)$
gegeben: NEA: $(Q, \Sigma, \delta', q_0, F')$
mit
1.$\delta'(q,a) = \hat{\delta}(q,a)$ $^{q\in Q, a\in\Sigma}$
2. $F' = \begin{cases} F \cup q_0 & \text{falls, die } \epsilon\text{-Hülle in } q_0 \text{ einen Zustand auf F enthält.} \\ F & \text{sonst} \end{cases}$
Durch Induktion über $|w|, w \in \Sigma^*$, muss man zeigen
$\delta'(q_0, w) = \hat{\delta}(q_0, w)$



$\delta(q_0, a) = q_0$
$\delta(q_0, \epsilon) = q_1$
$\delta(q_1, b) = q_1$
$\delta(q_1, \epsilon) = q_2$
$\delta(q_2, c) = q_2$
1.Schritt: Bestimmung der ϵ-Hüllen
$\epsilon - \text{Hülle}(q_0) = \{q_0, q_1, q_2\}$
$\epsilon - \text{Hülle}(q_1) = \{q_1, q_2\}$
$\epsilon - \text{Hülle}(q_2) = \{q_2\}$
2. Schritt: Bestimmung von $\hat{\delta}$
$\hat{\delta}(q_0, \epsilon) \overset{\text{Def}}{=} \epsilon - \text{Hülle}(q_0) = \{q_0, q_1, q_2\}$
$\hat{\delta}(q_0, a) = \{q_0, q_1, q_2\}$
$\hat{\delta}(q_0, b) = \hat{\delta}(q_0, \epsilon b) = \epsilon - \text{Hülle}(\delta(\hat{\delta}(q_0, \epsilon), b)) = \epsilon - \text{Hülle}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_0, b)) = \epsilon - \text{Hülle}(\emptyset \cup q_1 \cup \emptyset) = \{q_1, q_2\}$
$\hat{\delta}(q_0, c) = \hat{\delta}(q_0, \epsilon c) = \epsilon - \text{Hülle}(\delta(\hat{\delta}(q_0, \epsilon), c)) = \epsilon - \text{Hülle}(\delta(q_0, q_1, q_2, c)) = \epsilon - \text{Hülle}(\delta(q_0, c) \cup \delta(q_1, c) \cup \delta(q_2, c)) = \epsilon - \text{Hülle}(q_2) = \{q_2\}$
q_2 ist Endzustand in der ϵ-Hülle von q_0 enthalten.
$\Rightarrow \hat{\delta}(q_1, \epsilon) = \epsilon - \text{Hülle}(q_1) = \{q_1, q_2\}$
$\hat{\delta}(q_1, a) = \hat{\delta}(q_1, \epsilon a) = \epsilon - \text{Hülle}(\delta(\hat{\delta}(q_1, \epsilon), a)) = \epsilon - \text{Hülle}(\delta(q_1, q_2, a)) = \epsilon(\delta(q_1, a) \cup \delta(q_2, a)) = \emptyset$
$\hat{\delta}(q_1, b) = \hat{\delta}(q_1, \epsilon b) = \epsilon - \text{Hülle}(\delta(\hat{\delta}(q_1, \epsilon), b)) = \epsilon - \text{Hülle}(\delta(q_1, \epsilon) \cup \delta(q_1, b)) =$

$\epsilon - \text{Hülle}(\{q_1, q_2\})$
$\hat{\delta}(q_1, c) = q_2$



NEA mit ϵ-Übergängen $\equiv$ NEA ohne ϵ-Übergängen $\equiv$ DEA

Reguläre Ausdrücke
Σ endliche Menge von Symbolen
$L, L_1, L_2 \approx$ Menge von Zeichenketten aus Σ^*
Konkatenation von L_1 und $L_2 (L_1 L_2) = \{xy | x \text{ ist aus } L_1, y \text{ ist aus } L_2\}$
Def: $L^0 = \epsilon, L^i = LL^{i-1}, i \geq 1$
Transitive reflexive Hülle(kleene'sche Hülle) von $L(L^*)$ ist die Menge $L^* = \bigcup_{i=0}^{\infty} L^i$
Transitive Hülle(Positive Hülle)
$L^+ = \bigcup_{i=1}^{\infty} L^i$
$\Sigma = \{0, 1\}, L_1 = \{10, 1\}, L_2 = \{011, 11\}$
$L_1 L_2 = \{10011, 1011, 1011, 111\}$
$L = \{10, 11\}$
$L^* = \{\epsilon, 10, 11, 1010, 1011, 1110, 1111, ...\}$
Definition:
Sei Σ ein Alphabet.
Die „regulären Ausdrücke“ über Σ und die von ihren beschriebenen Mengen und rekusiv definiert durch:
1. $\emptyset$ ist ein regulärer Ausdruck und bezeichnet die leere Menge
2. ϵ ist ein regulärer Ausdruck und bezeichnet die Menge ϵ
3. $a, a \in \Sigma$ ist ein regulärer Ausdruck und bezeichnet die Menge$a$

4.Seien r,s reguläre Ausdrücke, die die Menge (Sprache) R, S' bezeichnen
$\Rightarrow (r + s)$ ist regulärer Ausdruck und bezeichnet $R \cup S'$
$\Rightarrow (rs)$ ist regulärer Ausdruck und bezeichnet RS'
$\Rightarrow (r^*)$ ist regulärer Ausdruck und bezeichnet R^*

Klammereinsparung durch Prioritäten:

* hat höhere Priorität als Konkatenation mit „+“
$((0(1^*)) + 0) \approx 01^* + 0$
Kommentar:
Sofern keine Komplikationen auftreten wird zwischen regulären Ausdrücken und der Sprache (Menge), die r bezeichnet nicht unterschieden!!

„Äquivalenz zwischen endlichen Automaten und regulären Ausdrücken“
Satz: Sei r eine beliebiger regulärer Ausdruck
Dann gibt es einen ϵ-NEA für $L(r)$ akzeptiert $(L(r) = L(\epsilon - NEA))$(effektivkonstruierbar)
Beweis:
Induktion über Anzahl der Operatoren in r (aus 4.)
Anfang = Operatoren (Basisarithmetik)
$r = \epsilon$, q_0 ist Endzustand
$r = \emptyset$, q_0 hat keine Verbindung zum Endzustand
$r = a$, ganz normaler Automat

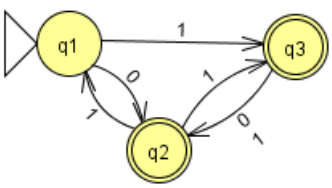
Satz: Sei r ein beliebiger regulärer Ausdruck. Dann gibt es eine ϵ-NEA mit L(ϵ-NEA) = L(r)
Beweis: Induktion über die Anzahl der Operationen in r.
Induktionsanfang: 1. $r = \epsilon \Rightarrow$ EA: $\rightarrow (q_0) \approx (q_0, \emptyset, \emptyset, q_0, q_0)$
2. $r = \emptyset \Rightarrow EA :\rightarrow (q_0) \qquad (q_2)$
3. $r = a, a \in \Sigma \Rightarrow EA :\rightarrow (q_0) \rightarrow (q_2)$
Induktionsschritt: Der Satz gilt für alle regulären Ausdrücke mit regulären Operatoren $i \geq 1$
$\Rightarrow$ ex. α Ausdrücke r_1, r_2 und zugeordnete „Identische“ Automaten EA_1, EA_2
$\Rightarrow$ es ex 3 Möglichkeiten für r:
1. Fall: $r = r_1 + r_2$
es ex. $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$
es ex. $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$
O.B.d.A.: $Q_1 \neq Q_2$
$\Rightarrow M = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$
$\delta : \delta(q_0, \epsilon) = \{q_1, q_2\}, \delta(q, a) = S_1(q, a), q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \{\epsilon\}$
$\delta(q, a) = \delta_2(q, a), q \in Q_2 - \{f_2\}, a \in \Sigma_2 \cup \{\epsilon\}$
$\delta(f_1, \epsilon) = \delta(f_2, \epsilon) = \{f_0\}$
2. Fall: $r = r_1 r_2$
geg.: EA_1, EA_2 minimal
$\Rightarrow M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\})$
$\delta : \delta(q, a) = \delta_1(q, a), q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \{\epsilon\}$
$\delta(f_1, \epsilon) = \{q_2\}$
$\delta(q, a) = \delta_2(q, a), q \in Q_2$ und $a \in \Sigma_2 \cup \{\epsilon\}$
2. Fall: $r = r_1^*$
$\Rightarrow M = (Q_1 \cup q_0, f_0, \Sigma_1, \delta, q_0, \{f_0\})$
$\delta(q_0, \epsilon) = \{q_1, f_0\}, \delta(q, a) = \delta_1(q, a)$
$\delta(f_1, \epsilon) = \{q_1, f_0\}, q \in Q - \{f_1\}, a \in \Sigma_1 \cup \{\epsilon\}$
Beispiel:
Gegeben: regulärer Ausdruck: $01^* + 1 \approx (0(1^*)) + 1 = r$
$r_1 = r_3 r_4$ mit $r_3 = 0, r_4 = 1^*$
$r_3 :\rightarrow (q_3) \underbrace{\rightarrow}_{0} (q_4)$
$r_4 = r_5^*$ mit $r_5 = 1$
$r_5 :\rightarrow (q_5) \underbrace{\rightarrow}_{1} (q_5)$
$\Rightarrow r_4 = r_5^* : [Bild]$
$r_1 = r_3 r_4 : [Bild]$
$r = r_1 + r_2 : [Bild]$

Satz: Wenn L von DEA akzeptiert wird (L = L(DEA)), dann wir L durch einen regulären Ausdruck beschrieben.
Bew.: DEA $\equiv$ M $= (\{q_1, .., q_n\}, \Sigma, \delta, q_1, F)$
Zustände sollen durchnummeriert sein.
R_{ij}^K: Menge aller Zeichenreihen, die DEA von Zustand q_i in den Zustand q_j bringen, ohne durch einen Zustand zu gehen(„Betretenünd „verlassen“'), der mit etwas größerem als K nummeriert ist.
$[Bild]$
$\Rightarrow R_{ij}^n$: Menge Zeichenreihen, die q_i in q_j überführen
Zulässige mögliche Wege(Situationen) für R_{ij}^K
1. Auf dem Weg von q_i nach q_j liegt kein K.
$\Rightarrow$ die Eingaben, die M veranlassen, von q_i nach q_j zu gehen, liegen in R_{ij}^{K-1}.
2. Die Eingaben, die M veranlassen, von q_i nach q_j zu gehen,(ohne einen höher als K nummerierten Zustand zu passieren), setzt sich aus folgenden Teilwegen zusammen:
1. eine Zwischenreihe aus R_{ik}^{n-1} (bringt M ausgehend von q_i zum ersten Mal nach q^k)
2. beliebig viele Zeichenreihen aus: R_{kk}^{k-1}(bringt M ausgehend von q_k zurück nach q^k, ohne eine höheren Zustand oder q_k selbst zu passieren)
3.eine Zustandsreihe aus R_{kj}^{k-1} (bringt M von q_k nach q_j)

$\Rightarrow$ Rekursive Definition für R_{ij}^K:
$R_{ij}^K = R_{ij}^{k-1} \cup R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$

$$R_{ij}^0 = \begin{cases} \{a|\delta(q,a) = q_j\} & i \neq j \\ \{a|\delta(q,a) = q_j\} \cup \{\epsilon\} & \text{falls } i \neq j \end{cases}$$

Zusagen durch Induktion:
für alle i,j und k gibt es einen regulären Ausdruck r_{ij}^k der die Sprache R_{ij}^K repräsentiert.
Alle Operationen von der Def in R_{ij}^k entsprechen den „Operationen“ der regulären Ausdrücke
Beispiel für Anwendung:

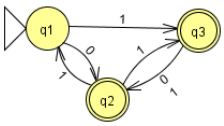


zur Vereinfachung wird genutzt $(r + s)t = t + st, (\epsilon + r)^* = r^*$

Tabelle der r_{ij}^k:

	k =0	k=1	k=2
r_{11}^k	ϵ	ϵ	$(00)^*$
r_{12}^k	0	0	$0(00)^*$
r_{13}^k	1	1	0^*1

Gegeben: EA

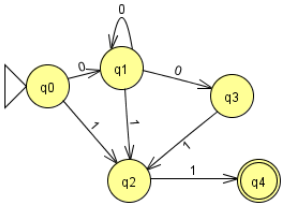


$$R_{ij}^0 = \begin{cases} a|\delta(q_j,a) = q_j i \neq j \\ a|\delta(q_i,a) = q_j \cup \epsilon, i = j \end{cases} \qquad R_{ij}^k = R_{ij}^{k-1}(R_{kk}^{k-1})R_{kj}^k - 1 \cup R_{ij}^{k-1}$$

	K = 0	K = 1	K = 2	K=3
r_{11}^K	ϵ	ϵ	$(00)^*$	
r_{12}^K	0	$\epsilon^* + 0 + 0 = 0$	$0(00)^*$	$0 * 1(\epsilon + (0+1)0^*1)^*(0+1)^* + 0(00)^*$ $= 0 * 1((0+1)0^*1)^*(0+1)^* + 0(00)^*$
r_{13}^K	1	1	0^*1	$0 * 1(\epsilon + (0+1)0^*1)^*(\epsilon + (0+1)0^*1) + 0^*1$ $= 0^*1((0+1)0^*1)^*$
r_{21}^K	0	0	$0(00)^*$	
r_{22}^K	ϵ	$\epsilon + 00$	$(00)^*$	
r_{23}^K	1	$1 + 01$	0^*1	
r_{31}^K	$\emptyset$	$\emptyset$	$(0+1)(00)^*0$	
r_{32}^K	0+1	$0 + 14$	$(0+1)(00)^*$	
r_{33}^K	ϵ	ϵ	$\epsilon + (0+1)0^*1$	

Berechnung der Sprache L(EA),
1.Berechne zu jedem Zustand n = 3
2.Bestimmung aller Anfangszustände, und den Endzuständen
3. Berechne alle $4 r_{ij}^n \approx r_{12}^3, r_{13}^3$
4. Verknüpfung mit „+“
Die von EA’s akzeptierten Sprachen heissen auch „regulären“ Sprachen

Minimierung von EA’s



$L(EA) = \{11\} \cup \{00^*11\} \cup \{00^*011\}$
Zustand q_3 kann gelöscht werden.

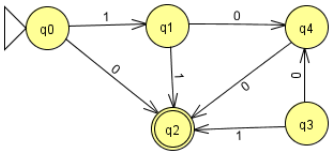
Def. Zwei Zustände p,q heißen äquivalent, wenn für alle Zeichenreihen $w \in \Sigma^*$: $\hat{\delta}(p,q) \in F \Leftrightarrow \hat{\delta}(q,w) \in F$ (es muss nicht der gleiche Endzustand sein)
Zwei Zustände p,q heißen unterscheidbar, falls mindestens eine Zeichenreihe $w \in \Sigma^*$ mit der Eigenschaft, dass es in p strikt und in einem Endzustand endet und der Weg mit w markiert ist und für q kein dazugehöriger Weg existiert und umgekehrt.
Gegeben: Ein EA ... Anfangszustände äquivalent sind
Ges. Algorithmus der zwei Zustände auf Äquivalenz überprüft. „Tabelle-filling-Algorithmus“
Konstruiere eine Tabelle $T : Q \times Q \to a, \text{nä}$

$$T(q,q') = \begin{cases} \text{nä} & \text{falls } (q \in F \wedge q' \notin F) \text{ oder } (q \notin F \wedge q' \in F) \\ \text{ä} & \text{sonst} \end{cases}$$

Schleife: repeat:

T(q,q’):= nä , falls ein $a \in \Sigma$ existiert mit $T(\delta(q,a), \delta(q',a)) = \text{nä}$

until „keine Veränderung mehr“



T	q_0	q_1	q_2	q_3	q_4
q_0		nä	nä	nä	nä
q_1			nä	ä	nä
q_2				nä	nä
q_3					nä
q_4					

2. Durchlauf bringt keine Veränderung

Der Algorithmus erstelle „Blöcke“(Mengen) die mit einem Zustand q selbst und aus allen zu q äquivalenten Zuständen bestehen(disjunkte Blöcke).
Berechnung des Minimalen Automaten: $EA = (Q, \Sigma, \delta, q_0, F)$ Ausgangsautomat
1. Eliminiere alle Zustände, die aus q_0 nicht erreichbar sind.
2. Ermittle mit dem „Table-filling Algorithmus“ alle Paare äquivalenter Zustände Zustände mit „ä“ markiert
3. Partitioniere Q in Blöcke
Sind (q,p) und (p,r) äquivalent, dann ist auch (q,r) äquivalent
4. Die Blöcke sind die Zustandsmengen des Minimalautomaten
5. δ' des Minimalautomaten ist gegeben durch: Sei SÄ ein Blcok: Dann gilt für ein beliebiges $a \in \Sigma$: 1 .Entweder ex ein Block T, so dass für alle Zustände q aus S’ gilt,

dass $\delta(q, a)$ in T liegt oder 2. $\delta(q, a)$ ist für keine Zustand q definiert.
6. Der Anfangszustand ist derjenige, in dem q_0 liegt
7. Alle Blöcke die einen „alten“ Endzustand enthalten, sind „neue“ Endzustände

Endliche Automaten mit Ausgabe
2. Möglichkeiten
1.Ausgabe an die Zuständen und 2. an die Kanten

$\Rightarrow$ 2 Automatentypen: 1. Moore-Maschine, 2. Mealy-Maschine

EA's mit Ausgabe
1. Ausgabe im Zustand
Moore-Maschine
2. Ausgabe im Zustandsübergang
Mealy-Maschine

Definition:
Eine Moore-Maschine ist eine Tupel
$MO - EA = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
mit Δ endliches Alphabet (Ausgabesymbole)
$\lambda : Q \to \Delta$
Bemerkung:
1. Die Ausgabe eines MO-EA die Antwort mit der Eingabe $a_1, .., a_n, n \geq 0$ ist $\lambda(q_0), .., \lambda(q_n)$ wobei die $q_0, ., q_n$ die Zustandsfolge ist für die $\delta(q_i, a) = q_j$
2. Es gibt keine Endzustände.
ein klassischer EA ist „äquivalent“ zu einem Moore-Automat, für die $\Delta = \{0, 1\}$ und ein Zustand als Endzustand interpretiert wird wenn seine Ausgabe 1 ist.
3. Ausgabe erfolgt bereits im Startzustand, d.h. für die Eingabe ϵ wird $\lambda(q_0)$ ausgegeben.
Beispiel:
Gesucht ist eine Moore-Maschine, für die eine bestimmte Zeichenreihe(interpretiert als ganze Zahl) den Rest mod 3 ausgibt.
Def. Ein Mealy-Maschine ist ein Tupel $ME - EA = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$
mit $\lambda : Q \times \Sigma \to \Delta$
Die Ausgabe ein ME-EA zzgl der Eingabe $a_1, .., a_n$ ist
$\lambda(q_0, a_1), \lambda(q_1, a_2), ..\lambda(q_{n-1}, a_n)$ wobei die $q_0, .., q_n$ diejenige Folge von Zuständen ist, für die gilt: $\delta(q_{i-1}, a_i) = q_i$
Vergleich von Moore- und Mealy-Maschine
1. Für die Eingabe ϵ liefert eine Mealy-Maschine stets ϵ, die Moore-Maschine $a \in \Delta$
2. Gegeben Weg mit n Kanten $\Rightarrow$ Länge der Ausgabe bei Moore: $n + 1$
$\Rightarrow$ Länge der Ausgabe bei Mealy: n
Äquivalenz??
„De Facto“ liegt der Unterschied in der 1. Ausgabe
Definition: Sei M eine Mealy- oder Moore-Maschine. Die Ausgabe: $T_M(w)$, $w \in \Sigma^*$ ist eine Ausgabe, die von M bei „Abarbeiten“ von w ausgegeben wird.
T_{ME} und T_{MO} erzeugen für ein w nie die gleiche Ausgabe.
Definition: ME-EA und MO-EA sind äquivalent, wenn $bT_{ME}(w) = T_{MO}(w)$
für alle $w \in \Sigma^*$ und für b ist die Ausgabe von MO in q_0.
Satz: $MO = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ ließt sich effektiv in eine „äquivalente“ ME-EA überführen.

$ME - EA = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$ mit $\lambda'(q, a) = \lambda(\delta(q, a)), q \in Q, a \in \Sigma$.
Satz. $ME = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$.
Dann heißt sich effektiv eine „äquivalente“ Moore-Maschine konstruieren.
Beweis:
$MO - EA = (Q', \Sigma, \Delta, \delta', q_0')$
$q_0' = [q_0, b_0]$, b_0 beliebig aus Δ
$Q' = \{[a, b], q \in Q, b \in \Delta\}$
$\lambda'([q, b]) = b$
Eigenschaften von Regulären Mengen (und damit für EA's)
Pumping Lemma:
Gegeben: reguläre Sprache R:
$\Rightarrow$ Es existiert ein DEA $M(Q, \Sigma, \Delta, q_0, F)$ mit R = L(M)
$|Q| = n$
Betrachte eine Eingabe, die n oder mehr als n Symbole umfasst.
$a_1, ..a_m, m \geq n$
Sei $\delta(q_0, a_1, .., a_n) = q_i$
$\Rightarrow$ M nimmt m+1 Zustände an
$\Rightarrow$ von diesen m+1 Zuständen müssen mindestens 2 doppelt sein.
..
$\Rightarrow a_{j+1}..a_k$ hat mindestens die Länge 1 und ist nicht länger als n.
..
Annahme: $q_n \in F$
$\Rightarrow a_1a_2...a_j...a_m \in L(M)$
$\Rightarrow a_1a_2..a_k(a_{j+1}..a_n)a_{n+1}..a_m \in L(M)$

$\Rightarrow$ Pumping-Lemma für reguläre Mengen:
Sei L eine reguläre Menge.
Dann gibt es eine Konstante n, so dass sich ein Wort $z \in L$ mit der Eigenschaft $|z| \geq n$
schrieben lässt (aufspalten) in z = uvw
1. $v \neq \epsilon$
2. $|uv| \leq n$
und dass ferner für alle $i \geq 0$ gilt: $uv^iw \in L$.
Anwendung des Punping-Lemmas
Gegeben: Sprache(Menge) $L = (a^nb^n)$
zu zeigen: L ist keine reguläre Sprache:
Vorgehen: L ist regulär und führen Widerspruchsbeweis
Wähle für n den Wert aus dem Pumping-Lemma
2. Betrachte alle mögliche Aufspalten von z aus L

$z = a^n b^n = uvw$
mit den Randbedingungen $|uv| \leq n, v \neq \epsilon$
$|uv| \leq n \Rightarrow uv$ kann nur a's enthalten
$\Rightarrow$ Pumpinglemma: $uv^2w \in L$ $(i = 2)$(v enthält mindestens ein a)
uv^2w enthält mindestens ein a mehr als uvw. aber die gleich Anzahl von b's.
$\Rightarrow$ in uv^2w ist die Anzahl der a's und b's verschieden.
$\Rightarrow uv^2w \notin L$ Widerspruch!!

Pumping-Lemma
Prinzipielle Vorgehensweise
1. Gegeben se eine Sprache L, von der gezeigt werden soll, dass sie nicht regulär ist.
2. Wähle eine konkrete und feste Zeichenreihe z aus L(genügend lang, Auswahl darf von der Konstanten n des PumpingLemma abhängen)
3. Zeige, daß für jede mögliche Aufspaltung von z ein $z = uvw$ mit den Randbedingungen $|uv| \leq n, v \neq \epsilon$, es eine Zeichenreihe $uv^i w$ gibt, die nicht in L liegt, $i = 0, 1..$
$\Rightarrow L$ ist nicht regulär
Abschlußeigenschaften von regulären Mengen
Eine reguläre Menge heißt abgeschlossen(bzgl. einer Operation), wenn die Anwendung dieser Operatoren wieder eine reguläre Menge liefert.
Satz: Die regulären Menge sind abgeschlossen unter Vereinigung , Konkatenation, und Kleensche Hüllenbildung
Beweis: unmittelbar aus der Definition der regulären Ausdrücke bzw. regulären Mengen.
Satz: Die regulären Mengen sind abgeschlossen unter Komplementbildung, d.h. Sie $L \subseteq \Sigma^*$ eine reguläre Menge $\Rightarrow \Sigma^* - L$ ist eine reguläre Menge.
Beweis: Sei $L \subseteq \Sigma^*$ die akzeptierte Sprache für den DEA M.
$M = (Q, \Sigma_1, \delta, q_0, F)$
Setze $\Sigma = \Sigma_1$(o.B.d.A)
Annahme: Es existieren Symbole die nicht in Σ sind.$\Rightarrow$ diese Symbole sind „Sackgassen“ und können gelöscht werden.

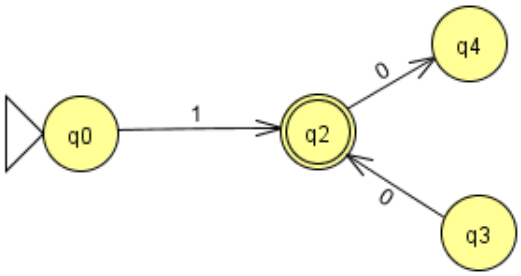


Abbildung 1: Bild 1

q_4 und q_3 dürfen gelöscht werden
Existieren Symbole in Σ, die nicht in Σ_1 sind, so treten sind in keinem Wort aus $L = L(M)$ auf.
Konstruktion aus EA M’ mit $L(M') = \Sigma^* - L$
$M' = (Q, \Sigma, \delta, q_0, Q - F)$, Bildung des Komplementes der Endzustände
$\Rightarrow w \in L(M') \Leftrightarrow \delta(q_0, w) \in Q - F$, d.h. w ist in $\Sigma^* - L$

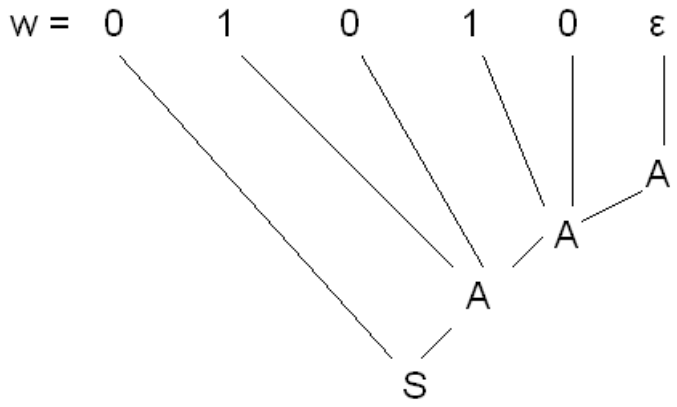
Satz: Die regulären Mengen sind unter Schnittbildung abgeschlossen.
Beweis: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
Substitution und Homomorphismus
Substitution:
Beispiel: Gegeben: 2 Alphabete Σ und Δ mit $\Sigma = \{0, 1\}, \Delta = \{a, b\}$
Die Substitution f ist gegeben durch: $f(0) = a, f(1) = b^*$
f läßt sich erweitern auf Zeichenketten, z.B. $f(010) = ab^*a$ (reguläre Menge)
Analog: Erweiterung auf Sprachen (Mengen von Zeichenketten)
Sei $L = 0^*(0 + 1^*)1^* \Rightarrow f(L) = a^*(a + b^*)b^* = a^*b^*$
Generell:
Seien Σ, Δ zwei Alphabete
Eine Substitution f eine Abbildung von Σ auf Teilmenge Δ^*
f läßt sich auf Zeichenketten erweitern durch
1. $f(\epsilon) = \epsilon$
2. $f(xa) = f(x)f(a)$
f läßt sich auf Sprachen durch
$f(L) = \bigcup_{x \in L} f(x)$
Satz: Die Klasse der regulären Menge ist unter Substitution abgeschlossen.
Sei $R \subseteq \Sigma^*$ eine reguläre Menge.
Sei ferne für jedes a aus Σ. $R_0 \subseteq \Delta^*$ eine reguläre Menge
Sei $f : \Sigma \rightarrow \Delta^*$ eine Substitution gegen durch $f(a) = Ra$
Betrachte die regulären Ausrücke die R und die R_a repräsentieren.
$\Rightarrow$ Ersetze nun bei der Substitution jedes auftreten a durch den regulären Ausdruck für Ra.
$\Rightarrow$ Es gilt: Die Substitution einer Vereinigung eines Produktes oder einer Hülle ist gleich der Vereinigung dem Produkt bzw. Hülle der Substitution, d.h. $f(L_1 \cup L_2) = f(L_1) \cup f(L_2)$
Beweis durch Induktion über die Anzahl der Operatoren.
Allgemein: gegeben: ist ein regulärer Ausdruck
$\Rightarrow$ durch die Ersetzung von einzelnen Symbolen durch reguläre Ausdrücke entstehen Zeichenreihen , die nur die für reguläre Ausdrücke erlaubten Operationen enthalten.
Homomorphismus:
Eine Homomorphismus h ist eine Substitution bei der h(a) eine einzige Zeichenreihe für jedes a enthält.
Beispiel: $h(0) = aa, h(1) = aba$
$h(010) = aaabaaa$
Erweiterungen: analog zur Substitution
z.B. $L_1 = (01)^* \Rightarrow h(L_1) = (aaaba)^*$

Homomorphismus: ist Spezialfall der Substitution
$\Rightarrow$ der Satz gilt auch für Homomorphismen
Inverser Homomorphismus
$h^{-1}(L) = \{x|h(x) \in L\}$
Urbild des Homomorphismus einer Sprache L
$\Rightarrow h^{-1}(w) = \{x|h(x) = w\}$
Beispiel: $L_2 = (ab + ba)^*a$
$\Rightarrow h^{-1}(L_2) := \{1\}$
Satz:
Satz gilt nicht für inversen Homomorphismus

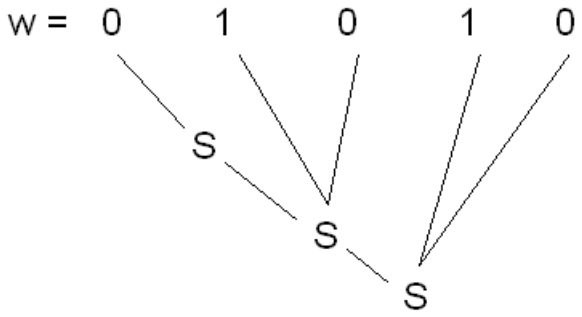
Satz: Die Menge von Sätzen, die von einem EA mit n Zuständen akzeptiert wird, ist genau dann:
1. nicht leer, wenn EA eine Satz der Länge $< n$ akzeptiert.
2. unendlich, wenn EA Sätze der Länge l akzeptiert mit $n \leq l < 2n$
Beweis:
1. a) Rückrichtung trivial
b) Es existiert Menge von Sätzen. Seien ferner w das kürzeste aller anderen akzeptierten Wörter.
$\Rightarrow |w| < n$
Denn sei $|w| \geq n \overset{Pumpinglemma}{\Rightarrow} w$ ist zerlegbar in $w = xyz$
$\Rightarrow w' = xz$ („abpumpen“) ist ein Wort der Sprache.
Widerspruch zur Annahme w ist das kürzeste Wort.
2. a) Annahme: $w \in L(EA)$ mit $n \leq |w| < 2n$.
$\Rightarrow$ wegen Pumpinglemma ist xy^iz ($w = xyz$) für alle i ein Wort der Sprache.
b) Annahme L(EA) ist unendlich und es existiert kein Wort mit einer Länge zwischen n und $2n - 1$.
$\Rightarrow w$ hat mindestens die Länge 2n.
$\overset{Pumpinglemma}{\Rightarrow} w = xyz$ und $w' = xz \in L(EA) \Rightarrow$ Widerspruch

Reguläre Grammatiken
Definition: (Grammatik allgemein)
Eine (allgemeine) Grammatik G ist ein Tupel, $G = (T, N, P, S)$
T: ist eine endliche Menge von Terminalen.
N: ist eine Menge von Nichtterminalen(Variablen, Hilfssymbole)
P: ist eine Menge von Regeln(Produktionen)
$S \in N$: Startsymbol (Axiom)
Die durch G erzeugte Sprache ist gegeben durch:
$L(G) = \{w \in T^* | S \overset{P}{\Rightarrow} w\} \overset{P}{\Rightarrow} \approx$ reduzieren „oder“ ableiten
Definition: Eine rechtslineare Grammatik G ist eine allgemeine Grammatik mit den Einschränkungen für die Regeln in P:
Alle Regeln besitzen eine der beiden Formen: $A \to wB$ oder $A \to w$, $(a, b \in N, w \in T^*)$
Definition: Eine links-lineare Grammatik ist eine allgemeine Grammatik mit den Einschränkungen für die Regeln in P:
Alle Regeln besitzen eine der beiden Formen: $A \to Bw$ oder $A \to w$, $(a, b \in N, w \in T^*)$
Eine rechts- oder links-lineare Grammatik wird auch reguläre Grammatik genannt.

Beispiel: Gegeben: Sprache: $L = 0(10)^*$
Beh.: L wird von der rechts-linearen Grammatik
L_r mit $P = S \to 0A, A \to 10A, A \to \epsilon$
erzeugt.



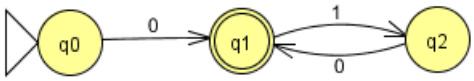
Beh: L wird von der links-linearen Grammatik L_l mit $P = S \to S10, S \to 0$ erzeugt.



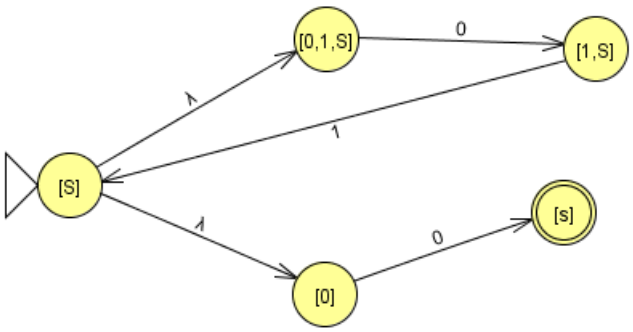
Satz: Wenn L eine reguläre Grammatik besitzt, so ist L eine reguläre Menge.
Bemerkung (Teil 1)
Annahme: $L = L(G)$ für ein rechts-lineare Grammatik $G = (T, N, P, S)$
$\Rightarrow$ Konstruktion eines ϵ-NEA $M = (Q, T, S, [S], [\epsilon])$
der die Ableitungen von G simuliert.
Q: Besteht aus Symbolen $[a]$ mit:
i. $a = S$ oder
ii. a ist ein (nicht notwendigerweise echtes)Suffix einer rechten Seite einer Produktion von P.
δ : i. $a \in N \Rightarrow \delta([A], \epsilon) = [a] | A \to \alpha \in P$
ii. $a \in T, \alpha \in T^* \cup T^*N \Rightarrow \delta([a\alpha], a) = [\alpha]$
Durch Induktion über eine Ableitungsfolge läßt sich zeigen:

$\delta([s], w)$ enthält $[\alpha] \Leftrightarrow S \Rightarrow [\alpha]xA \Rightarrow xya$ mit
1.$[] \rightarrow ya \in P$
2. $a = S'$ und $w = \epsilon$
$[\epsilon]$ ist einziger Endzustand.
$\Rightarrow$ M akzeptiert $w \Leftrightarrow S' \overset{*}{\Rightarrow} xA \Rightarrow w$
$\Rightarrow$ M akzeptiert $w \Leftrightarrow w \in L(h)$

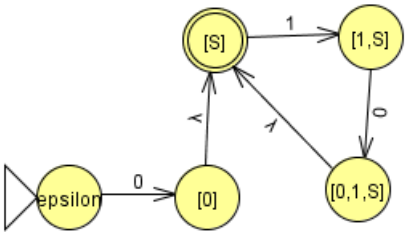
Teil 2:
Bemerkung: Spiegelt man eine links-lineare Grammatik, w enthält man eine (nicht identische!) rechts-lineare Grammatik und umgekehrt.
$G = (N, T, P, S), G' = (N, T, P', S)$
$P' = A \rightarrow \alpha | A \rightarrow \alpha^R \in P$
Beispiel: links-lineare Grammatik:
$L = 0(10)^*$
$S \rightarrow S10, S \rightarrow 0 \overset{Spiegeln}{\Rightarrow} S \rightarrow 01S, S \rightarrow 0 \Rightarrow L' = (01)^*0$
$\Rightarrow$ Angegebene Konstruktion:
$L = 0(10)^*$



Automat zu L’

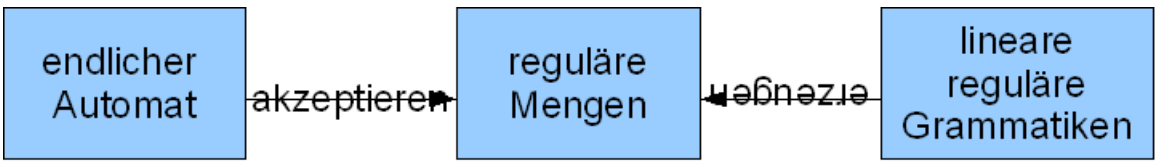


Umkehrung der Kanten + Tausch von Anfangs- und Endzustand.
$\Rightarrow$

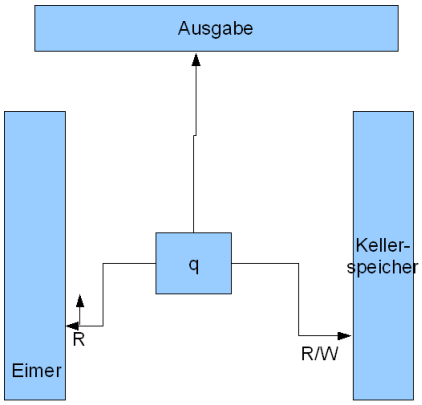


$0(10)^*$

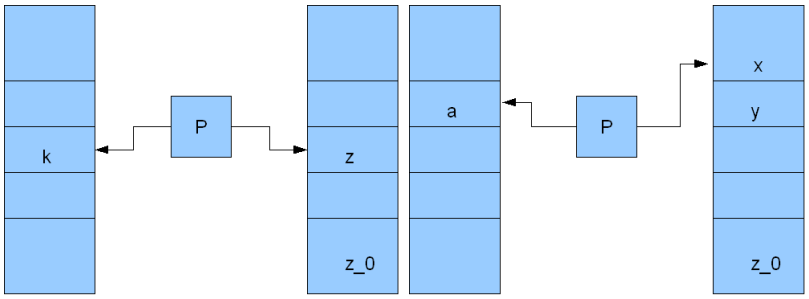
Satz:
Wenn L eine reguläre Menge ist, so wird L von einer link-regulären Grammatik und von einer rechts-linearen Grammatik erzeugt.
Beweis: Sei $L = L(M)$ für einen DEA: $M = (Q, \Sigma, \delta, q_0, F)$
1. $q_0 \notin F$
$\Rightarrow L = L(G)$ für die nicht-lineare Grammatik $G = (Q, \Sigma, P, q_0)$
$G = (N, T, P, S)$
$P : p \rightarrow aq$, falls $\delta(p, a) = q$
$p \rightarrow a$ falls $\delta(p, a) \in F$
$\Rightarrow \delta(p, w) = q \Leftrightarrow p \overset{*}{\Rightarrow} w1$
$\Rightarrow$ Falls wa von M akzeptiert wird$(wa \in L(M))$
dann existiert ein p mit $\delta(q_0, w) = p$ und $\delta(p, a)$ ist Endzustand.
$\Rightarrow q_0 \overset{*}{\Rightarrow} wp$ und es existiert eine Produktion $p \rightarrow a$ in $P \Rightarrow q_0 \overset{*}{\Rightarrow} wa$
Umkehrung: Es gelte $q_0 \overset{*}{\Rightarrow} x$
$\Rightarrow q_0 \overset{*}{\Rightarrow} wp$
$\Rightarrow \delta(q_0, w) = p$ und $\delta(p, a)$ ist Endzustand
$\Rightarrow a \in L(M) \Rightarrow L(M) = L(G) = L$
Fall 2:
$q_0 \in F \Rightarrow \epsilon \in L \Rightarrow$ von Fall 1 konstruierte Grammatik erzeugt $L = \epsilon$
$\Rightarrow$ Führe eine Axiom S und die Regeln $S \rightarrow q_0$, $S \rightarrow \epsilon$ ein.



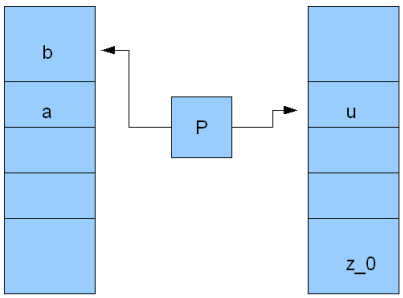
Kellerautomaten:



Definition: 1-Weg-nichtdeterministischer Kellerautomat[1](1-N-KA) ist ein 7-Tupel
$M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
mit:
Q: endliche Menge von Zuständen
Σ: (endliche) Eingabealphabet
Γ: (endliches) Kelleralphabet
$q_0 \in Q$: Startzustand
$z_0 \in \Gamma$ Kelleranfangssymbol
$F \subseteq Q$
$\delta : Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow P(Q \times \Gamma^*)$



[1]in Büchern häufig als Pushdownautomat(engl.) bezeichnet
http://de.wikipedia.org/wiki/Kellerautomat

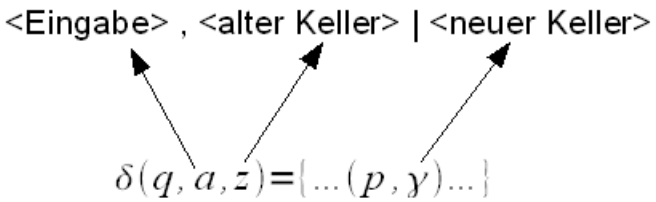


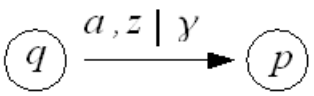
$(p, xy) \in \delta(q, a, z)$
$(r, \epsilon) \in \delta(q, b, z)$

Definition: Konfigurationen eine KA-M
1. Eine Konfiguration K_m eines KA-M ist ein Tupel$(q, w, \gamma) \in (Q \times \Sigma^* \times \Gamma^*)$
2. Für einen KA-M heißt $\overset{\vdash}{M} \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Sigma^* \times \Gamma^*)$ Konfigurationsübergang
$\Leftrightarrow (q, aw, z\tilde{\gamma}) \vdash (p, w, \tilde{\gamma})$
$(p, \gamma') \in \delta(q, a, z)$
Bemerkung: $\overset{+}{\vdash}, \overset{*}{\vdash}$ sind die transitive bzw. transitive Fortsetzung von $\vdash$
Darstellungen:

δ	(q_1, z_1)	(a_i, z_i)	..	(ϵ, z_x)
$\rightarrow q_0$	$(p, \gamma_1), (p, \gamma_2)$			
q_1				
..				
$*q_m$				

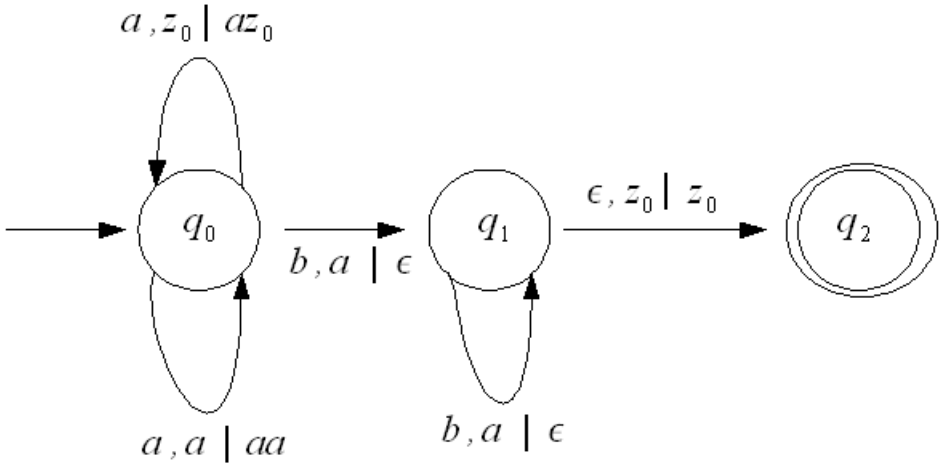
graphisch:
- Knoten sind die Zustände
- Kanten verbinden Quell- und Zielzustand
- Markierungen beinhaltet:





Beispiel1:

δ	(a, z_1)	(a,a)	(b,a)	(ϵ, z_0)
q_0	(q_0, az_0)	(q_0, aa)	(q_1, ϵ)	
q_1			(q_1, ϵ)	(q_2, z_0)
$*q_2$				



$(q_0, aabb, z_0) \vdash (q_0, abb, az_0) \vdash (q_0, bb, aaz_0) \vdash (q_0, b, az_0) \vdash (q_1, \epsilon, z_0) \vdash (q_2, \epsilon, z_0)$, Automat akzeptiert!!

Definition: Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ ein 1-N-KA

(1)Dann heißt:

$L(M) := \{w \in \Sigma^* : \exists p \in F, \gamma \in \Gamma^* \text{ mit } (q_0, w, z_0) \vdash_M^* (p, \epsilon, \gamma)\}$ der von M unter erreichen eines Endzustandes akzeptierte Sprache.

(2)Dann heißt $L(M) := w \in \Sigma^* : (q_0, w, z_0) \overset{*}{\vdash} (p, \epsilon, \epsilon)$ die von M unter leeren des Kellers akzeptierte Sprache.

Satz: Für jede Sprache $L = L(M_2), M_2$: 1-N-KA

(1) Existiert ein 1-N-KA M_1 mit $L = N(M_1)$

(2) Für jede Sprache L mit $L = N(M_1)$, M_1 1-N-KA existiert eine 1-N-KA M_2 mit $L = L(M_2)$.

Beweis: (1): $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

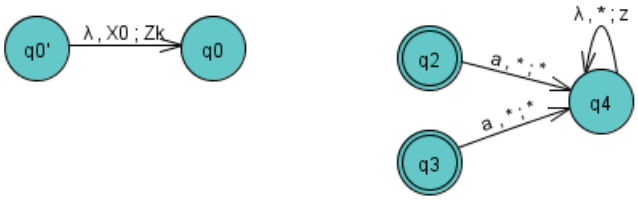
Sei $x_0 \notin \Gamma$ neues Kellersymbol, q_0', $q_E \notin Q$ neue Zustände, $M_1 := (Q \cup q_0', q_E, \Sigma, \Gamma \cup x_0, q_0' x_0, \emptyset)$

δ' wie δ, bis auf

$\delta(q_0', \epsilon, x_0) = \{(q_0, z_0, x_0)\}$, und für alle $p \in F, \gamma \in \Gamma^*$

$\delta'(p, \epsilon, \gamma) = \{(q_E, \epsilon)\}$

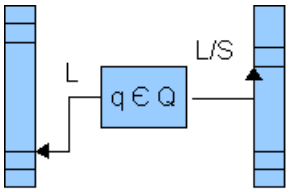
$\delta(q_E, \epsilon, \gamma) = \{(q, \epsilon)\}$



Sei $w \in L(M_2) zzs : q \in N(M_1)$
$(q_0, w, z_0) \vdash M_2^*(p, \gamma), p \in F$
$(q_0', w, x_0) \vdash M_1(q_0, w, z_0 x_0) \vdash M_1, M_2^*(p, \epsilon, \gamma x_0) \vdash (q_E, \epsilon, \gamma' x_0) \vdash M_1^*(q_E, \epsilon, \epsilon)$
$w \in N(M_1) \Rightarrow w \in L(M_2) analog$
Beweis (2): $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$
$M_2 := (Q \cup \{q_0', Q_E\}, \Sigma, \Gamma \cup \{x_0\}, \delta', q_0', x_0, \{q_E\})$
δ' wie δ, bis auf:
1) $\delta'(q_0', \epsilon, x_0) = \{(q_0, z_0, x_0)\}$
$\delta'(p, \epsilon, x_0) = \{(q_E, \epsilon)\}, \forall q \in Q$

Definition: Ein 1-D-KA ist bis auf das δ ein 1-N-KA.
Hier gelten folgende Änderungen:
$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times \Gamma^*$
und für alle $q \in Q$, $z \in \Gamma$ mit $\delta(q, \epsilon, z) \neq \emptyset$ muss gelten: $\forall a \in \Sigma : \delta(q, a, z)$ nicht definiert.
Bemerkung: $Sprachen_{EA} \subsetneq Sprachen_{1-N-KA}$

Kellerautomat

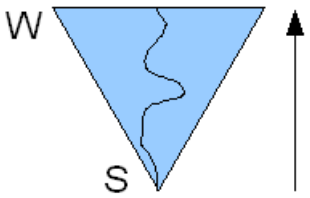


Varianten:
Deterministisch, nicht-deterministisch, 1-Weg, 2-Weg
sind alle unterschiedlich
⇒ es gibt nicht den Kellerautomaten
Zunächst:
1-ND-KA

Kontextfreie Grammatiken
Kontextfreie Grammatiken sind in der Praxis die wichtigsten Grammatiken. (Programmiersprachen)
Definition: kontextfreie Grammatik G eine allgemeine Grammatik G = (N,T,P,S) mit der Einschränkung:
Für alle Regeln aus P gilt: $A \rightarrow \alpha, A \in N, \alpha \in (N \cup T)^*$.
Erzeugte Sprache $L(G) = \{w | w \in T^* \text{ und } S \Rightarrow_p^+ w\}$
Normalformen:
Satz:(Chomsky-Normalform)
Zu jeder kontextfreien Grammatik G mit $\epsilon \notin L(G)$ läßt sich eine äquivalente kontextfreie Grammatik G’ erzeugen, bei der alle Produktionen P die Form:
$A \rightarrow BC; A, B, C \in N$
$A \rightarrow a; a \in T$
besitzen Satz: (Greibach-Normalformal)
Zu jeder kontextfreien Grammatik G mit $\epsilon \notin L(G)$ läßt sich eine äquivalente kontextfreie Grammatik G’ erzeugen, bei der alle Produktionen P die Form:
$A \rightarrow a\alpha; A \in N, a \in T, \alpha \in N^*$
besitzen.

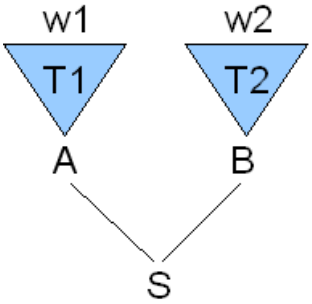
Pumpinglemma für Kontextfreie Sprachen(uvwxy-Lemma)
Sei L eine kontextfreie Sprache. Dann existiert eine von L abhängige Konstante n, so daß für jedes Wort $z \in L, |z| \geq n$, als z = uvwxy schreiben läßt mit folgenden Randbedingungen:
1) $|vx| \geq 1$

2) $|vwx| \leq n$
3) $\forall i \geq 0 : uv^i wx^i y \in L$
Beweis: (Ableitungsbaum)



Hilfssatz: Sei L = L(G) mit G in Chomsky-Normalform
$\Rightarrow$ Existiert in einem Ableitungsbaum für ein Wort für ein Wort $w \in L(G)$. Kein Pfad P mit $|P| > i \Rightarrow |w| \leq 2^{i-1}$.
Beweis: CNF $\Rightarrow$ Ableitungsbaum ist ein binärer Baum, d.h. alle Knoten haben, 2,1 oder 0 Nachfolger. $\Rightarrow$ Wegen Beschränkung der Tiefes des Baumes lässt sich die Anzahl der Blätter abschätzen.
1.: $i = 1$:

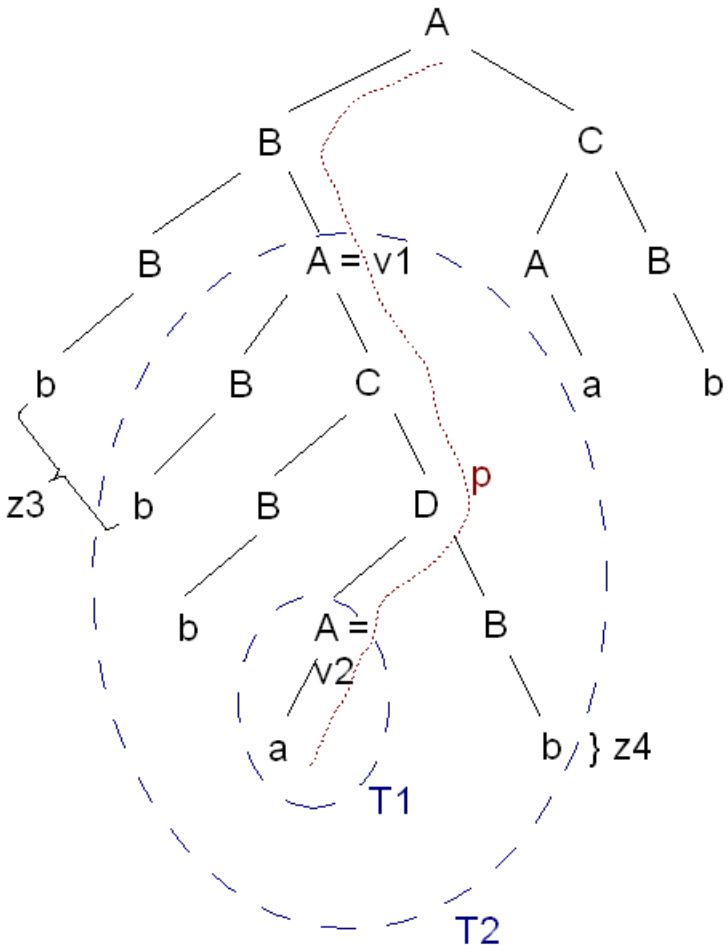
$|w| = 1 = 2^0 = 2^{1-1}$
2.: $i > 1$:



Annahme: Aussage gelte für T_1 und T_2. Erweiterung um die Regel $S \rightarrow AB \Rightarrow w = w_1 w_2, |P| = |P_{1,2}| + 1 \Rightarrow w \leq 2 * 2^{i-2} = 2^{i-1}$

Beweis Pumpinglemma:
2. Spezialfälle:
1. $L(G) = \emptyset$, Satz gilt, da L(G) keine Zeichenreihe enthält.
2. $L(G) = \epsilon$, Satz gilt, da keine Zeichenreihe existiert mit $|z| \geq n$.

Generell:
Sei $|N| = K$ und $n = 2^K$
Sei $z \in L(G)$ mit $|z| \geq n = 2^k$
$\Rightarrow$ Der Ableitungsbaum für z enthält eine Pfad P mit $|P| \geq k + 1$
$\Rightarrow$ In P existiert mindestens 2 Knoten mit dem gleichen Nichtterminal-Zeichen.
Beispiel:
$G(\{A, B, C, D\}, \{a, b\}, \{A \to BC, A \to a, B \to BA, B \to b, C \to AB, C \to BD, D \to AB\}, A)$
Beispiel:

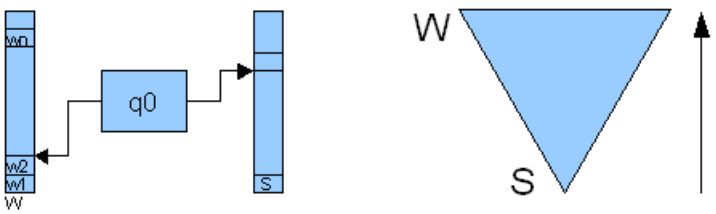


Allgemein: Sei P der längste Pfad.
Es existieren in P zwei Knoten V_1, V_2 mit
1. V_1, V_2 repräsentieren gleiches Nichtterminalzeichen, z.B: A
2. V_1 sei näher an der Wurzel als V_2.

3. V_1, V_2 seien vom Blatt ausgesehen, die nächsten Knoten mit A.
4. Die Länge $|\overline{v_1 a}| \leq n+1$ (keine anderen „doppelten“ Nichtterminale)
T_1 Unterbaum mit Wurzel V_1
1. Sei z_1 die Front von T_1, d.h. $z_1 = bbab$
2. Sei z_2 die Front von T_2, d.h. $z_2 = a$
$\Rightarrow z_1 = z_3 z_2 z_4$, mit $z_3 = bb, z_4 = b$
$\Rightarrow z = b z_3 z_2 z_4 ab \Rightarrow A \Rightarrow^* z_3 A z_4$ und $A \Rightarrow^* z_2$ mit $|z_3 z_2 z_4| \leq 2^k = n$.
$\Rightarrow A \Rightarrow^* z_3^i z_2 z_4^i \forall i \geq 0$
$z_3 = v, z_2 = w, z_4 = x$

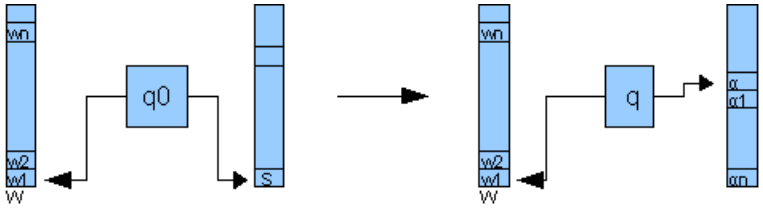
Anwendung:
$L = \{a^j b^j c^j | j \geq 1\}$ ist nicht kontextfrei
Annahme: L ist kontextfrei:
Finde ein Wort aus L für das das Pumping-Lemma nicht gilt:
Wir betrachten das Wort $z \in L$ mit $z = a^n b^n c^n$, n Konstante aus dem Lumping-Lemma
$\Rightarrow |z| = 3n \geq n$
(Voraussetzung erfüllt)
Betrachte alle mögliche Zerlegungen von z in z = uvwxy mit den Randbedingungen des Pumpinglemmas.
Randbedingungen: $|vwx| \leq n$ und $|vx| \geq 1$
vx kann nicht gleichzeitig a's und c's enthalten.
1. Fall:
v und x bestehen nur aus a's und b's:
$\Rightarrow [PL] uv^0 wx^0 y = uxy \in L$
Der uwy enthält weniger a's als b's und c's $\Rightarrow uxy \notin L$ Widerspruch.
2. Fall: v und x bestehen aus a's und b's, analog
3. Fall: analog

Zusammenhänge:
Sei L eine kontextfreie Sprache, dann läßt sich effektiv ein 1-N-KA(!) M Konstruieren mit L = L(M).

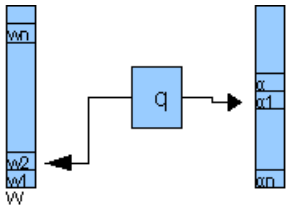


Simulation der Ableitung durch M(Normalform von Greibach).

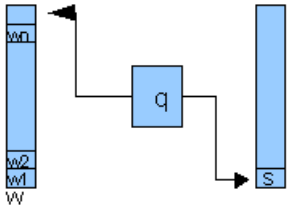
$S \rightarrow \overbrace{a}^{\in T}\overbrace{\alpha_1\alpha_2..\alpha_n}^{\in N}$



Vergleiche w_1 mit a
1.Fall $w_1 = a \Rightarrow$ Lösche a und gehe zu w_2



2. Fall: $w \neq a$ M bleibt stehen ohne zu akzeptieren
$\Rightarrow ... \Rightarrow$



kontextfreie Grammatik $\Leftrightarrow$ 1-N-KA
[$Bild1$]
.. durch leeren den Keller
Ersetzen Σ durch das Aktion S der Grammatik

a. Σ ist entweder ein Nichtterminalsymbol wähle Produktion aus P, so das linke Seite aus Kellersymbol
   Ersetze aber das Kellersymbol durch die rechte Seite der Regel

b. ein Kellersymbol $\in \Gamma$
   Vergleiche Kellersymbol mit dem gelesenen Symbol aus der Eingabe

   1. keine Übereinstimmung: Stop
   2. Übereinstimmung: lösche oberes Kellersymbol und gehe auf Eingabe einen Schritt weiter

Sei G = (N,T,P,S) kontextfreie Grammatik
Der „äquivalente“ 1-N-KA ist gegeben duruch
1-N-KA = $(\{q, T, N \cup T, S, q, Z_0, \emptyset\}$
δ:
1. $\delta(q, \epsilon, A) = \{(q, \beta) | A \to \beta \in P\}$
2. $\delta(q, a, a) = \{(q, \epsilon)\}$
3. $\delta(q, \epsilon, z_0) = \{(q, S)\}$

Satz (ohne Beweis)
Ist L(M) = L für eine 1-N-KA M,dann ist L kontextfrei.
Eigenschaften von kontextfreien Grammatiken.
1. Abschlusseigenschaften:
Satz: Kontextfreie Sprachen sind unter Vereinigung, Konkatenation und Hüllenbildung abgeschlossen.
Beweis:
Annahme: Seien L_1, L_2 kontextfreie Sprachen, die durch dir Grammatiken
$G_1 = (N_1, T_1, P_1, S_1)$ und $G_2 = (N_2, T_2, P_2, S_2)$ erzeugt werden.
Mit $N_1 \cap N_2 = \emptyset; S_3, S_4, S_5 \notin N_1 \cup N_2$.

1. Vereinigung:
   $G_3 = (N_1 \cup N_2 \cup \{S_3\}; T_1 \cup T_2, P_3, S_3)$
   $P_3 = P_1 \cup P_2 \cup \{S_3 \to S_1 | S_2\}$

2. Konkatenation:
   $G_4 = (N_1 \cup N_2 \cup \{S_4\}; T_1 \cup T_2, P_4, S_4)$
   $P_4 = P_1 \cup P_2 \cup \{S_1 \rightarrow S_1 S_2\}$

3. Hüllenbildung:
   $G_5 = (N_1 \cup \{S_5\}, T_1, P_5, S_5)$
   $P_5 = P_1 \cup \{S_5 \rightarrow S_1 S_5 | \epsilon\}$

sonstige Eigenschaften:
Nicht-Det.

	abgeschlossen	nicht-abgeschlossen
Substitution	√	
Homomorphismus	√	
Inv. Homomorphismus	√	
Schnitt		√
Komplement		√
Schnitt mit Reg. Menge	√	
Vereinigung,Konkatenation	√	

Det.

	abgeschlossen	nicht-abgeschlossen
Substitution	√	
Homomorphismus		√
Inv.Homomorphismus	√	
Schnitt		√
Komplement	√	
Schnitt mit Reg. Menge	√	
Vereinigung,Konkatenation		√

$a^n b^n$ Kontextfrei, det, 1-D-KA
$a^n b^{2n}$ Kontextfrei, det, 1-D-KA
$a^n b^n \cup a^n b^{2n}$ kontextfrei, n-det, 1-N-KA

Vereinigung von Kontextfreien Grammatiken
1. Eliminierung von nutzlosen (nicht-erreichbare) Symbole
Definition: Ein Symbol $X \in N \cup T$ eine Grammatik $G = (N, T, P, S)$ heißt nützlich, wenn es eine Ableitung der Form $S \rightarrow \alpha X \beta \overset{*}{\Rightarrow} w, w \in T^*$
$\Rightarrow$ nutzlose Symbole besitzen keinen Einfluß auf die Sprache (nutzlos $\Leftrightarrow$ nicht nützlich)

$\Rightarrow$ nutzlose Symbole können entfernt werden
Definition: Ein Symbol X heißt erzeugend, wenn es eine Zeichenreihe $w \in T^*$ gibt, mit $X \overset{*}{\Rightarrow} w$
Konstruktion der Menge der erzeugenden Symbole
Induktion:
1. Jedes Symbol aus T ist erzeugend.
2. Gegeben Sei eine Produktion $A \rightarrow \alpha_1 ... \alpha_n$ und jedes α_i ist erzeugend, dann ist auch A erzeugend.
Falls $A \rightarrow \epsilon \in P$, dann ist A erzeugend
Beispiel: $G = (N, T, P, S)$ mit P: $S \rightarrow AB|a, A \rightarrow b$
gem. 1 $\Rightarrow$ a,b erzeugend
gem. 2 $\Rightarrow$ A, S erzeugend
$\Rightarrow$ Erzeugenden Symbole sind a, b, A, S
Definition:
Ein Symbol $X \in N \cup T$ heißt erreichbar, wenn es für gewisse $\alpha, \beta \in (N \cup T)^*$ eine Ableitung $S \overset{*}{\Rightarrow} \alpha X \beta$ gibt.
Konstruktion der Menge der erreichbaren Symbole (Induktion)
1. S ist erreichbar
2. Gegeben eine Produktion $A \rightarrow \alpha_1 ... \alpha_n$, und A ist erreichbar $\Rightarrow$ alle α_i sind erreichbar.
Erreichbare Symbole
1. S ist erreichbar
Gem. 2 $\Rightarrow$ A,B, a sind erreichbar
Gem. 2 $\Rightarrow$ b ist erreichbar
$\Rightarrow$ Erreichbaren Symbole $\{S, A, B, a, b\}$
Satz: Sei G = (N,T,P,S) eine kontextfreie Grammatik mit $L(G) \neq \emptyset$. Dann läßt sich effektiv eine kontextfreie Grammatik $G_1 = (N_1, T_1, P_1, S)$ mit $L(G) = L(G_1)$ erzeugen, die keine nutzlosen Symbole enthält.
Beweis:
Konstruktion:
1.Schritt:
a) Bestimme alle Symbole, die nichts erzeugen und eliminiere sie
b) Eliminiere alle Produktionen, die mindestens ein nicht-erzeugendes Symbol enthalten.$\Rightarrow$ $G' = (N', T', P', S)$ (Bemerkung: S muss erzeugend sein, denn $L(G) \neq \emptyset$)
2. Schritt:
Eliminiere alle Symbole, die in G' nicht erreichbar sind.(einschließlich der zugehörigen Produktionen), Ergebnis ist G_1.
Beispiel:

Schritt 1: Erzeugende Symbole sind {a,b,A,S}

a) $\Rightarrow$ B wird eliminiert

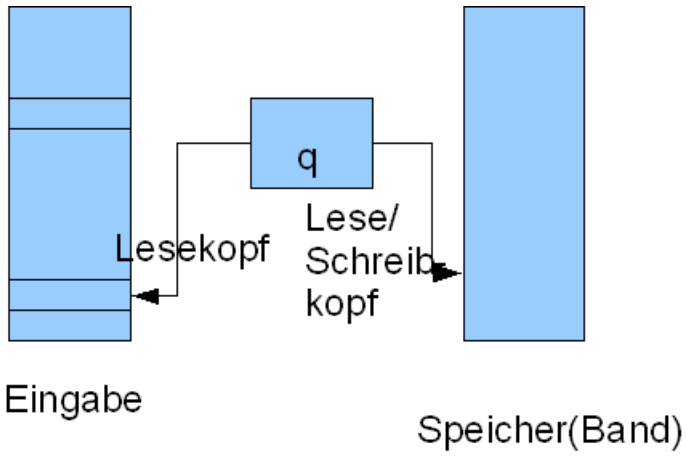
b) Elimination von $S \rightarrow AB, \Rightarrow P : S \rightarrow a, A \rightarrow b$

Schritt 2: In der neuen Grammatik sind nur noch S und a erreichbar.
$\Rightarrow$ A wird eliminiert
$\Rightarrow A \rightarrow b$ wird eliminiert
$\Rightarrow P_1 : S \rightarrow a$

Reihenfolge der Schritte ist wesentlich!!!
Beispiel in umgekehrter Reihenfolge:
Schritt 2 : Alle Symbole sind erreichbar $\Rightarrow$ G bleibt unverändert
Schritt 1 : $\Rightarrow S \rightarrow a, A \rightarrow b$

Kellerautomaten
1-ND-KA, $\approx$ kontextfreie Sprachen $\approx$ kontextfreie Grammatiken, exponentielle Laufzeit zur Verifikation, $a^n b^n + a^n b^{2n}$
1-D-KA, lineare Laufzeit, $a^n b^n$
2-ND-KA
2-D-KA, $a^n b^n c^n$

Turingmaschinen
Alan Turing 1936



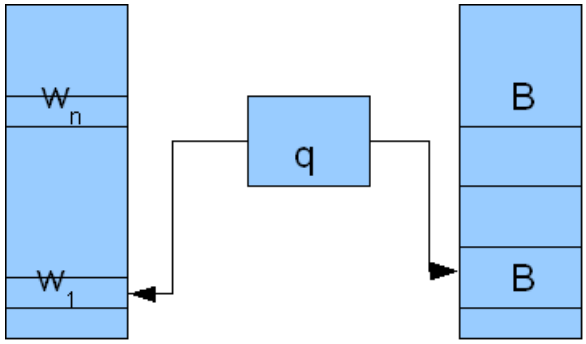
Arbeitsweise:
In Abhängigkeit vom gelesenen Symbol auf der Eingabe, dem Zustand der Kontrolleinheit und dem gelesenen Symbol auf dem Speicherband kann die Turingmaschine(TM)

1. Das Symbol im Speicher verändern.
2. Den Lese/Schreibkopf für den Speicher um eine Position nach oben/unten bewegen.
3. Den Lesekopf der Eingabe um eine Position nach oben verschieben.
4. Den Zustand der Kontrolleinheit ändern.

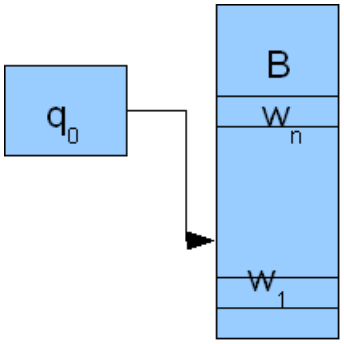
Definition: Eine Turingmaschine TM ist eine Tupel $TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$
Q: endliche Menge an Zuständen
Σ: endliche Menge Eingabesymbolen
Γ: Bandalphabet
$q_0 \in Q$: Startzustand

$F \subseteq Q$: Endzustände
$\delta : Q \times \Gamma \times \Sigma -> Q \times \Gamma \times \{L, R\}$
$B \in \Gamma (B \notin \Sigma)$: „Blank“

Startkonfiguration:

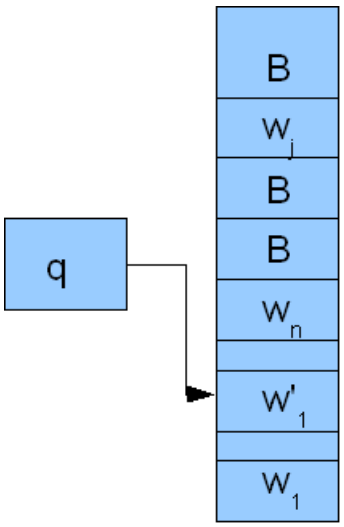


Variante 1:
Eine Turingmaschine ohne separate Eingabe ist ein Tupel $TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit 1. $\Sigma \subseteq \Gamma \ \{B\}$
2. $Q \times \Gamma -> Q \times \Gamma \times \{L, R\}$
Anfangskonfiguration:

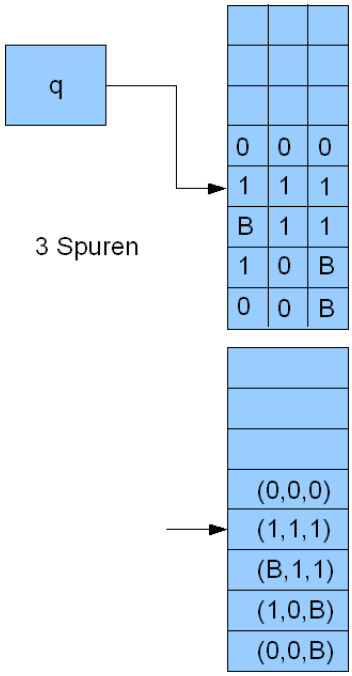


$\underbrace{w_1 ... w_i' ... w_n BB}_{\alpha_1} q \underbrace{w_j}_{\alpha_2}$
$\alpha_1 q \alpha_2$
Trennung gibt an, wo der Kopf steht.

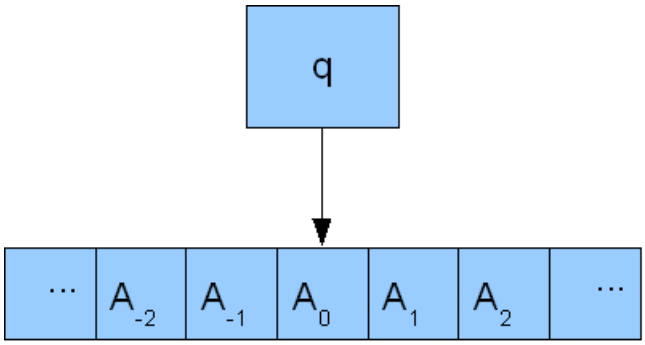
Satz: Jede Turing-Maschine mit separatem Eingabeband läßt sich simulieren durch eine TM ohne Eingabeband.
Zustandsbeschreibung:



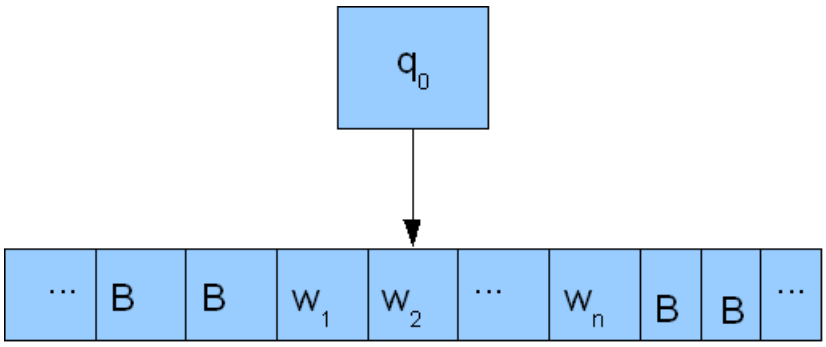
$\alpha_1 q \alpha_2, \alpha_1 \alpha_2 \approx$ relevanter Inhalt des Bandes, das „linkeste“ Symbol von α_2 wird gelesen.
Zustandsübergänge:
Gegeben: $X_1 X_2 ... X_{i-1} q X_i .. X_n$ und $\delta(q, X_i) = (p, Y, L)$
$\vdash_M X_1 X_2 ... X_{i-2} P X_{i-1} Y X_{i+1} ... X_n$
$\delta(q, X_i) = (p, Y, R)$
$\vdash_u X_1 X_2 .. X_{i-1} Y X_{i+1} .. X_n$
$\vdash_M^*$ transitiv reflexive Hülle von $\vdash_M$
Die von einer Turingmaschine TM abzeptierte Sprache L(TM) ist gegeben durch:
$L(TM) = \{w | w \in \Sigma^* \text{ und } q_0 w \vdash_{TM}^* \alpha_1 p \alpha_2 \text{ und } p \in F, \alpha_1, \alpha_2 \in \Gamma^*\}$
Variante 2:
Auf das Sondersymbol „B“ kann verzichtet werden.
Variante 3:
Mehrspurige Turing-Maschinen



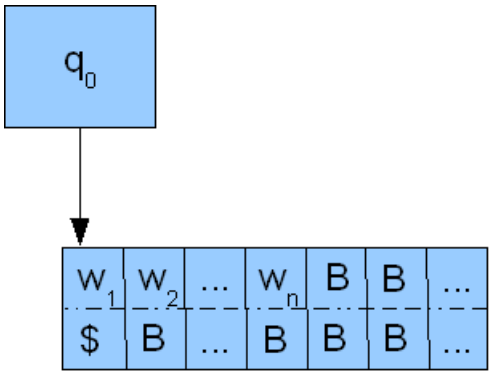
Satz: Jede mehrspurige TM M kann durch eine einspurige TM M simuliert werden.
$\Gamma' = (\alpha_1, ..., \alpha_n)|n$ Anzahl der Spuren, $\alpha_i \in \Gamma$
Variante 4:
Turing-Maschine mit beidseitig unendlichem Band



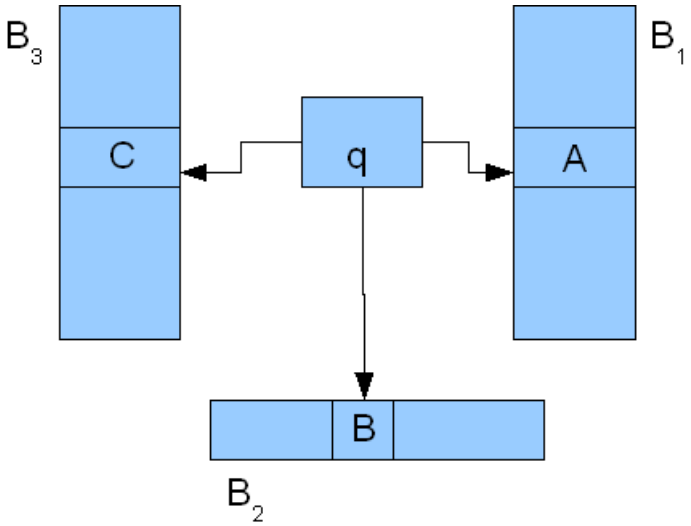
Satz: Jede Turing-Maschine M mit einem beidseitig unendlichen Band kann durch eine Turing-Maschine M’ mit einem einseitig unendlichen Band simuliert werden.



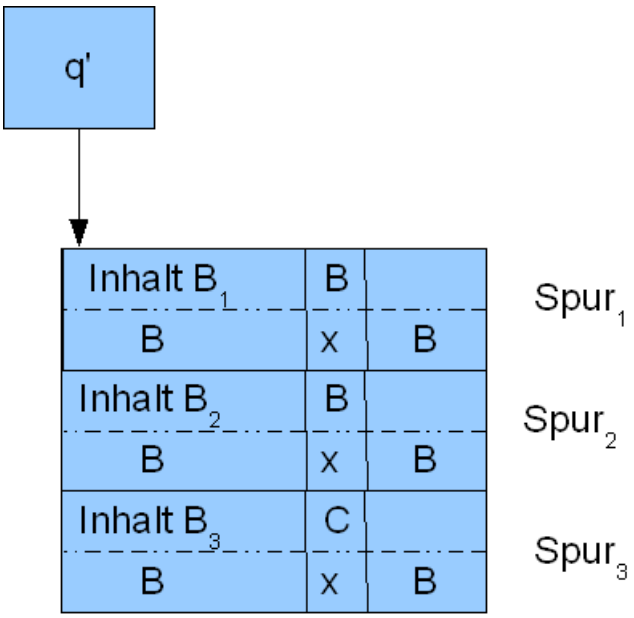
M’ besitzt ein einseitig unendliches Band mit 2 Spuren.



M’ arbeitet auf der oberen Spur, wenn M rechts von „A_0“ arbeitet. Die Relevanz einer Spur wird im Zustand vermerkt.
M: $q \in Q$, M’ : (q,o), (q,u)
Variante 5:
Mehrbändige Turing-Maschine:

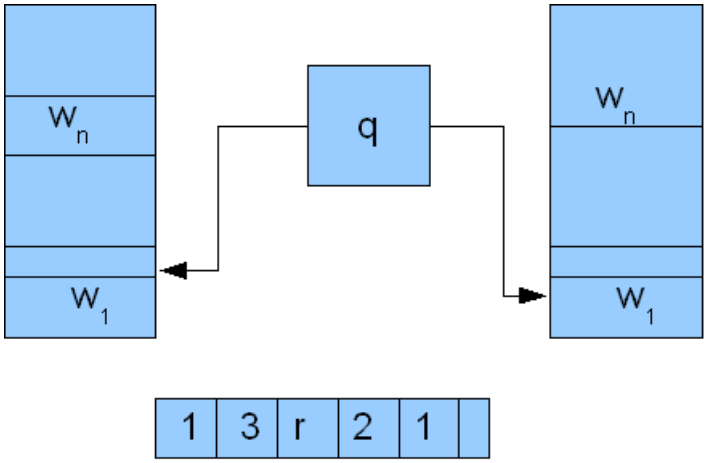


K-Bänder und K Lese/Schreibköpfe
In Abhängigkeit vom Zustand q und den K gelesenen Symbolen kann M
1. Den Zustand ändern
2. Die gelesenen Bandsymbole ändern
3. Jeden Kopf(unabhängig voneinander) um eine Position bewegen.
Satz: Jede mehrbändige Turingmaschine M kann durch eine einbändige Turing-Maschine M’ simuliert werden.
M’:



Für jedes Band besitzt M’ 2 Spuren
i) Spur 1 enthält den Inhalt des Bandes
ii) Spur 2 besitzt eine einzige Markierung die die Position des Kopfes anzeigt.
M’ zunächst die Markierung für B_1(2. Spur) und führt dann die Operation von M auf Spur 1 durch und versetzt die Markierung.
M’ sucht danach die Markierung für B_2(4. Spur) und führt dann die Operation von M auf Spur 3 durch und versetzt die Markierung.
M’ such abschließend die Markierung für B_3(6. Spur) und führt dann die Operation von M auf Spur 5 durch und versetzt die Markierung.
M’ nimmt den neuen Zustand von M an.

Satz:(Variantenteiler)
Jede nichtdeterministische TM m kann durch eine deterministische Turingmaschine simuliert werden.
Konstruktion:
Ein jeder Zustand und für jedes Bandsymbol gibt es es feste Anzahl an Alternativen.
Sei r die Minimalanzahl der Ableitungen(feste Zahl)
Jede Berechnungsfolge kann durch eine Folge von Ziffern $1, .., r$ dargestellt werden, aber nicht jede derartige Ziffernfolge ist eine zulässige Berechnungsfolge.
Konstruktion von M':
M hat 3 Bänder:
1. Band: enthält Eingabe.
2. Band: M' generiert systematisch Ziffernfolgen aus $1, ..., r$ (kürzeste Folge zuerst)
3. Band: Arbeitsband
M' kopiert Eingabe aus Band 1 auf Band3
M' arbeitet gemäß der Ziffernfolgen von Band 2.



Existiert keine Regel gemäß Ziffer, dann löscht M' den Inhalt von Band 3, generiert die nächste Möglichkeit der Ziffernfolge und beginnt von vorne.

Churche These
Alle Variationen von TM bringen keine neuen Möglichkeiten
ab 1930: viele Ansätze zum Begriff der „Berechenbarkeit“
Turing-Maschine
λ-Kalkül
Registermaschine
Theorie der rekursiven Funktionen

Kombinatorische Logik
Post-System
Alle besitzen die gleiche Mächtigkeit
These von Church
Alle „intuitiv“ berechenbaren Funktionen sind Turing-berechenbar
Nicht beweisbar, aber anerkannt.

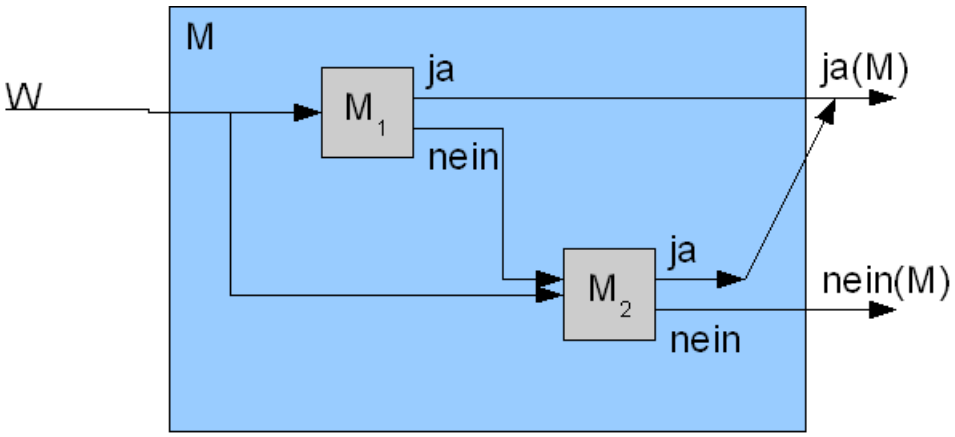
Gegeben: TM
$L(TM) = \{w | w \in \Sigma^* \text{ und } q_0 w \rightarrow^*_{TM} \alpha_1 p \alpha_2 \text{ mit } p \in F\}$
Definition:
Die Sprache, die von einer (beliebigen)TM erkannt wird heißt rekursiv aufzählbar.
Anmerkung: Eine Turingmaschine kann für eine Eingabe nicht anhalten.
Unterklasse:
Definition: Eine Sprache heißt rekursiv, wenn es mindestens eine Turingmaschine gibt, die die Sprache abzeptiert und TM ist eine Turingmaschine, die für jede Eingabe hält.
Eigenschaften:
Satz:

a) Die Vereinigung zweier rekursiver Sprachen ist rekursiv

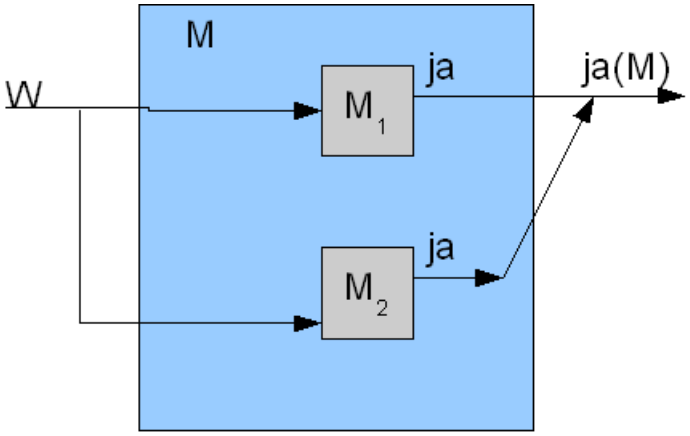
b) Die Vereinigung zweier rekusiv aufzählbarer Sprachen ist rekursiv aufzählbar.

Beweis:
a) Gegeben: L_1, L_2 rekursive Sprachen.
M_1, M_2 mit $L(M_1) = L_1, L(M_2) = L_2, M_1, M_2$ halten für jede Eingabe.
Konstruktion von M mit den Eigenschaften

1. M simuliert zunächst M_1
2. akzeptiert M_1, dann akzeptiert auch M
3. Verwirft M_1, so simuliert M die Maschine M_2
   (a) akzeptiert M_2,so akzeptiert M
   (b) verwirft M_2, dann verwirft auch M

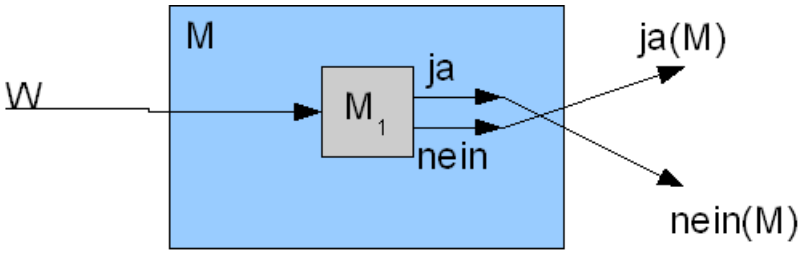


Konstruktion geht nicht für b)



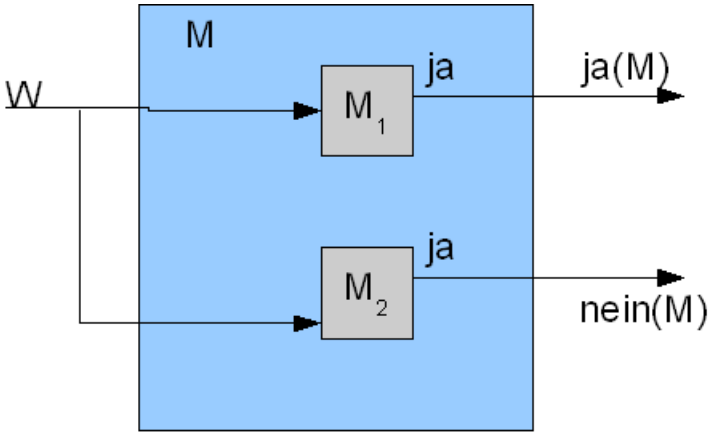
Satz:
Das Komplement einer rekursiven Sprache ist rekursiv
Beweis: Gegeben TM m , die auf allen Eingaben hält ,$L = L(M)$ (L ist rekursiv)
Konstruktion von M’ aus M mit den folgenden Eigenschaften

1. M’ stoppt ohne zu akzeptieren, wenn M bzgl der Eingabe in einen Endzustand übergeht.
2. M’ geht in einen Endzustand, wenn M stoppt ohne zu akzeptieren.



Satz: Wenn eine Sprache L und ihr Komplement $\overline{L}$ beide rekursiv aufzählbar sind, denn ist L(und damit $\overline{L}$) rekursiv.
Beweis: M_1, M_2 mit $L(M_1)$, $L(M_2) = \overline{L}$.
Konstruktion von M mit der Eigenschaft:

1. M akzeptiert w, wenn w von M_1 akzeptiert wird.
2. M verwirft w, wenn w von M_2 akzeptiert wird.



Es gilt: $w \in L(M_1)$ oder $w \in L(M_2)$
$\Rightarrow$ eine der beiden TM akzeptiert immer.
Konsequenzen aus den Sätzen
Seien L und $\overline{L}$ eine Paar von Komplementären Sprachen. Dann gilt eine der folgenden Aussagen:

1. Sowohl L als auch $\overline{L}$ sind rekursiv
2. Weder L noch $\overline{L}$ sind rekursiv aufzählbar
3. Entweder L oder $\overline{L}$ sind rekursiv aufzählbar und die jeweils andere Sprache ist nicht rekursiv aufzählbar.

Ziel Konstruktion eine Funktion(Sprache), die nicht TM-Berechenbar ist.
1. Schritt: Wieviele Turing-Maschinen gibt es?
Antwort liefert eine Codierung(Gödelisierung)
O.B.d.A. soll gelten Bandalphabet {0,1,B}, Eingabealphabet {0,1}, $Q = \{q_1, .., q_n\}$
q_1 Anfangszustand, q_2 Endzustand
$\Rightarrow TM(Q, 0, 1, 0, 1, B, \delta, q_1, B, q_2)$
Abkürzungen: $0 \approx X_1, 1 \approx X_2, B \approx X_3, L \approx R_1, R = R_2$

Die Regeln aus δ besitzen eine allgemeine Form
$\delta(q_i, X_j) = (q_k, X_l, R_m)$, Codierung dieser Regel
als binärer Zeichenreihen sei $=0^i10^j0^K10^l10^m$, („1“ ist Trennzeichen)
$\Rightarrow$ innerhalb einer Regel treten nie „11“ auf
$\Rightarrow$ Benutze „11“ um die Regeln zu trennen.
$\Rightarrow$ Alle Regeln der TM können durch die binäre Zeichenreihe 111 L(Regel 1) 11 L(Regel 2) 11 ... 11 L(Regel r)111

Gödelisierung(Codierung) von Turingmaschinen
1. Schritt: Codierung einer einzelnen Regel: $\delta(q_i, X_j) = (q_k, X_l, R_m)$
$X_1 = 0, X_2 = 1, X_3 = B$
$R_1 = L, R_2 = R$
Decodierung ist eindeutig $0^i10^j10^n10^l10^m$, (die 1sen sind Trennsymbole)
2. Schritt: Codierung aller Regeln.
a) Trenne die Regeln durch „11“
b) Maskiere Anfang und Ende der Maschinencodierung durch „111“
$\Rightarrow$ Kodierung von δ: 111 „Regel 1“ 11 „Regel 2“ 11 ... 11 „Regel n“111
Reihenfolge der Regel ist irrelevant
$\Rightarrow$ Zu einer TM existieren in der Regel mehrere Codierungen, die such nur in der Reihenfolge der Regeln unterscheiden.
$\Rightarrow$ Decodierung führt stets zur gleichen TM.
Jede binäre Zeichenreihe beschreibt eine ganze Zahl.
$\Rightarrow$ Jeder TM können eine oder mehrere ganze Zahlen zugeordnet werden, aber es gibt keine ganze Zahl, die unterschiedliche TM repräsentiert.
Anzahl der ganzen Zahlen: abzählbar unendlich
Anzahl der Funktionen: überabzählbar unendlich
$\Rightarrow$ Es muß Funktionen geben, die nicht Turing-berechenbar sind.
$\Rightarrow$ Es muß Funktionen geben, die nicht intuitiv-berechenbar sind.
Effektive Konstruktion einer nicht rekursiv aufzählbaren Sprache (nicht Turing-berechbar)
Methode: Diagonalisierungsverfahren:
Gegeben sei die (unendliche) Liste aller Zeichenreihen(Worte) über $(0 + 1)^*$ in Kanonischer Ordnung.
0,1,00,01,10,11,...
Sei ferner w_i das i-te Wort in dieser Liste.
Sei ferner M_j die Turing-Maschine, deren Codierung die ganze Zahl j in Binärdarstellung ist.
Bilde eine Tabelle.

$j \rightarrow$ ...

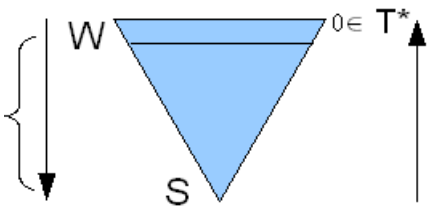
	1	2	3	4	...
1	**0**	0	0	0	...
2	0	**0**	1	0	...
3	0	1	**1**	1	...
4	1	0	1	**0**	...

Tabelle ist mit 0,1 erfüllt:

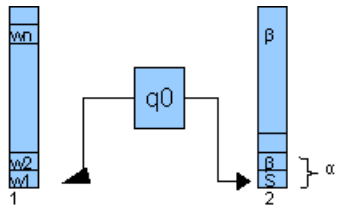
$0 : w_i \notin L(M_j)$
$1 : w_i \in L(M_j)$

Betrachte die Diagonale der obigen Tabelle
Die Sprache L_D ist folgendermaßen definiert.
L_D ist die Menge der Zeichenreihen w_i, so daß $w_i \notin L(M_i)$
$w_i \in L_D \Leftrightarrow (i,i) - te$ Eintrag Eintrag $= 0$.
Annahme:
L_D wird von der Turing-Maschine M_j akzeptiert.
$\Rightarrow$ i) $w_j \in L_D = L(M_j) \Rightarrow$ (j,j)-Eintrag $= 0 \Rightarrow w_j \notin L(M_j) \Rightarrow$ Widerspruch
ii) $w_j \notin L_D = L(M_j) \Rightarrow (j,j)$-Eintrag $= 1 \Rightarrow w_j \in L(M_j) \Rightarrow$ Widerspruch
w_j ist weder in L_D noch nicht in L_D
$\Rightarrow$ Annahme: $L_D = L_j$ war falsch
(gilt: für jedes beliebige „j“)
$\Rightarrow$ Es gibt n der Tabelle keine Turingmaschine, die L_D akzeptiert.

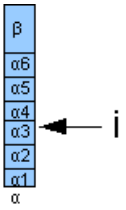
Nicht eingeschränkte Grammatiken
Eine Grammatik G = (N,T,P,S), bei der die Regeln aus P die Form $\alpha \rightarrow \beta, \alpha, \beta \in (N \cup T)^*, \alpha \neq \epsilon$
besitzen, heißen nicht eingeschränkte Grammatiken. (Typ-0-Grammatiken, allgemeine Grammatiken, Semi-Thue-Systeme)
Satz: $L = L(G)$ für eine Typ-0-Grammatik, $G = (N,T,P,S)$ ist, dann ist L eine rekursiv-aufzählbare Sprache.
Beweis: Konstruktion einer nicht deterministischen zwei-bändigen TM, die L akzeptiert.
1. Band: Eingabeband(wird nicht verändert)
2. Band: (speichert Satzformen: $\alpha \in G$)
Satzformen:



Die Turingmaschine simuliert Ableitungen:
$\Rightarrow$ Initialisierung: TM speichert „S“ auf Band2
Rekursives Vorgehen:
1. Wähle beliebige Position i in α (α ist Zeichenreihe ohne B)



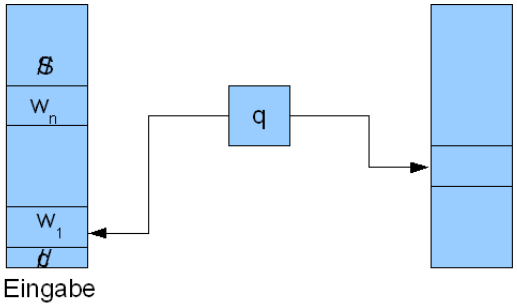
2. Wähle nichtdeterministisch eine Produktion: $\beta - \gamma \in \Gamma$ aus.

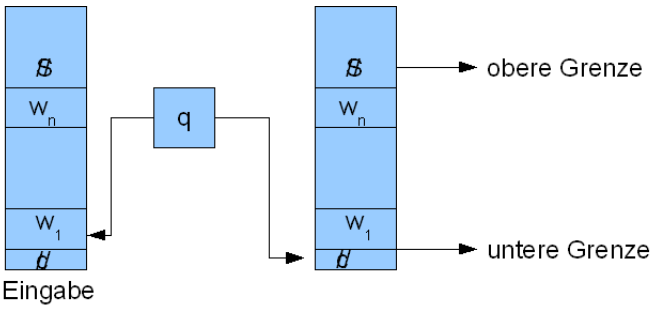


3. Überprüfe, ob ab der Position i, die Zeichenreihe β in α vorkommt. Falls ja, ersetze β durch γ. Gegebenenfalls durch „Platz machen“ bzw. „Zusammenschieben“.
4. Vergleich den Inhalt von Band 2 mit dem von Band 1.
i) Inhalte identisch $\Rightarrow$ akzeptiere
ii) Wähle neues i oder neue Produktion oder starte neu
Satz (ohne Konstruktion)
Sei L eine rekursiv-Aufzählbare Sprache, so gilt L = L(G) für eine Typ-0-Grammatik G.

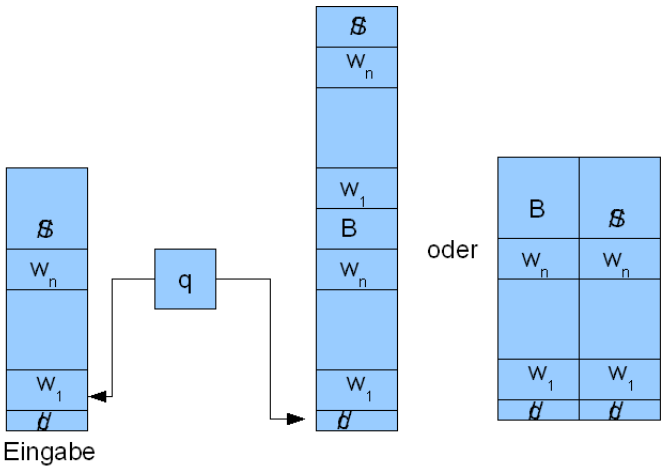
$L(EA's) \subset L(1 - ND - KA) \subset L(TM)$
$2Keller \approx TM$

$L(EA) \subset L(1-ND-EA) \subset .. \subset L(TM)$
Linear-beschränkter Automaten
Ein linear beschränkter Automat(LBA) ist eine nicht-deterministische TM mit den beiden Bedingungen:

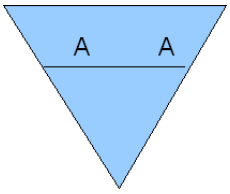




1: Das Eingabealphabet enthält zwei Sonderzeichen $\not C$, $\not S$ linke und rechte Randmarkierung
2. Der LBA kann sich nicht über die Randmarkierungen hinüber bewegen bzw. sie überschreiben.
Ein LBA ist eine Tupel: $LBA = (Q, \Sigma, \Gamma, \delta, q_0, F, \not C, \not S),\ \not C,\ \not S \in \Sigma$
$L(LBA) = \{w | w \in (\Sigma - \not c, \not S)^* \text{ und } q_0\ \not C w\ \not S \vdash^*_{LBA} \alpha q \beta, q \in F\}$
Annahme: Jeder LBA gemäß obiger Definition kann eine LBA simulieren, bei dem der Arbeitsbereich auf K+n, n = Länge der Eingabe, K = Konstante(unabhängig von n), beschränkt ist.



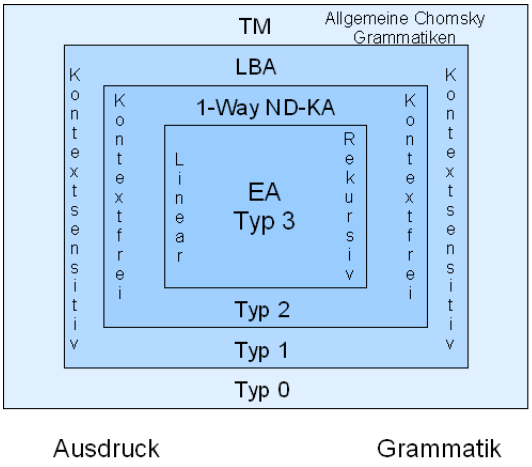
Kontextfreie Sprachen/Grammatiken
Def. Eine Kontextfreie Grammatik $K \subseteq$ ist eine allgemeine Grammatik ksG = (N,T,P,S) insbesondere die Regeln aus P $\alpha \to \beta$
gilt: $|\alpha| \leq |\beta|, \alpha, \beta \in (N \cup T)^*$
Die erzeugte Sprachen heißt kontextsensitive Sprache.
Satz(Normalformen)
Jeder ksG läßt sich durch eine ksG beschreiben, bei der alle Regeln die Form $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2, \beta \neq \epsilon, \beta \in (N \cup T)^+, A \in N$ besitzen.
$A \to \beta$

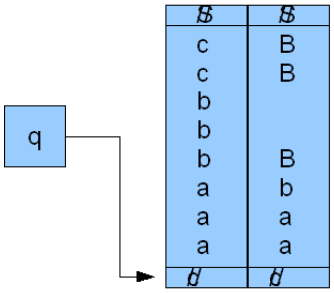


Satz: Ist L eine ksG, dann wir L durch eine LBA erkannt.
Beweis: Analog zu dem Beweis der Aquivalenz von rekursiv aufzählbaren Sprachen und TM's.
2. Beispiel:
1. Spur Eingabe: Spur 1 wird mit $\not C w_1 ... q_n \not S$ initialisiert.
2. Spur Arbeitsband: Spur 2 wird mit $\not C SB..B \not S$ initialisiert.
Länger der Spuren ist gleich lang.
Anschließend wird auf Spur 2 suksessive:

1. Eine beliebige Position auf gesucht.
2. Auf dieser Position eine mögliche Regel angewandt(Ableitung).

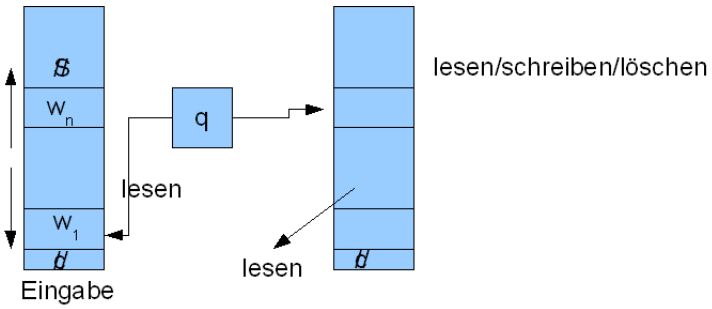
3. Musste hierzu $\not S$ überschrieben werden, so bleibt der LBA ohne zu akzeptieren stehen.
   $\Rightarrow$ LBA akzeptiert $w = w_1..w_n$, wenn in einer Ableitung $S \Rightarrow^* w$ gibt, so daß keine Zwischensatzform länger als w ist.
   Andererseits: Es kann keine Ableitung $S \Rightarrow^* \alpha \Rightarrow^* w$ geben mit $|\alpha| > |w|$





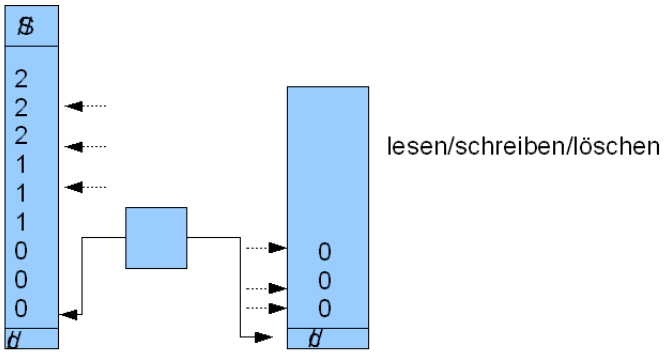
Weitere Automatenmodelle
Stapelautomaten:
Notation: Kellerautomat $\approx$ Pushdown-Automat
Stapelautomat $\approx$ Stackautomat
Definition:
Ein Stapelautomat (SA) ist ein Kellerautomat KA mit den zusätzlichen Eigenschaften.

1. Die Eingabe ist eine 2-Weg-Eingabe mit Anfangs- und Endmarkierungen.
2. Der Lese-/Schreibkopf kann zusätzlich zu dem Schreib- und Löschoperationen am oberen Ende sich im Speicher auf und ab bewegen und in dieser Phase nur lesen.

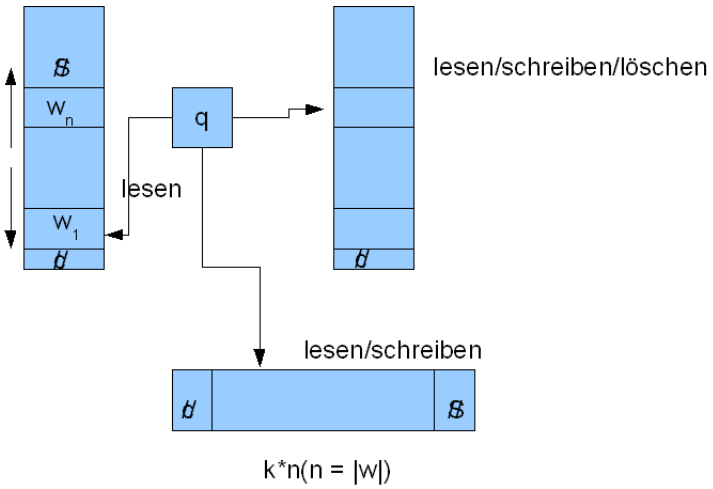


Das Übergangsverhalten ist bestimmt durch den aktuellen Zustand q, gelesenes Symbol auf der Eingabe, gelesenes Symbol im Stack.
In einem Schritt der der SA

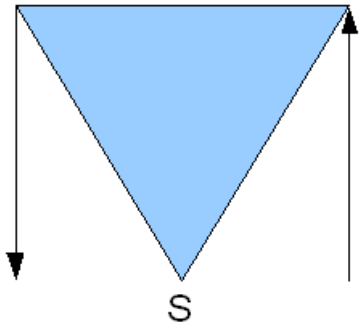
1. den Zustand ändern
2. den Lesekopf der Eingabe um eine Symbol (beide Richtungen= bewegen)
3. Auf dem Stack eine der folgenden Operationen vornehmen:
   (a) Hinzufügen eines Symbols am dem oberen Rand des Stacks(„push“)
   (b) Entfernen eines oberen Symbols von Stack („pop“)
   (c) Bewegen um eine Position (beliebige Richtung) ohne ein Symbol zu verändern, löschen oder hinzuzufügen



Hilfskellerautomaten:

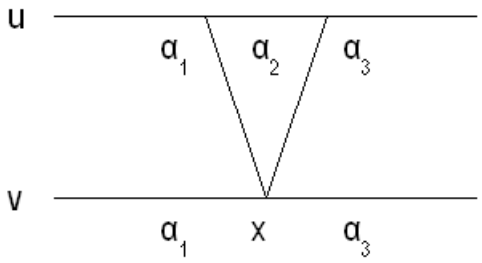

lesen/schreiben/löschen
q
lesen
lesen/schreiben
k*n(n = |w|)

I. Zusammenhang zwischen kontextfreien Grammatiken und Programmiersprachen
Anwendung: Definition der Syntax durch kontextfreie Grammatiken
Systemanalyse (generell): $O = O(n^3)$: n Länge des Programms
(Praxis): Deterministische Grammatiken: $O = O(n)$
$w \in T^*$ (Programm)



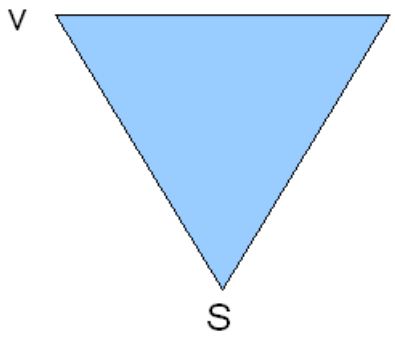
gegeben: kontextfreie Grammatik: G = (N,T,P,S)
$A = N \cup T$

Definition:
Sei $u = \alpha_1, \alpha_2, \alpha_3 \in A^*$, $X \in N, v = \alpha_1 X \alpha_3$
Sei ferner $(X, \alpha_2) \in P, (X \to \alpha_2)$
Dann sagt man; „u ist unmittelbar reduzierbar auf v“ bzw. „u ist unmittelbar aus v ableitbar“

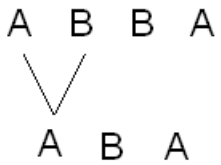


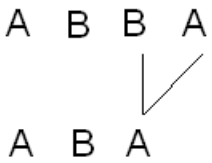
Notation: $u \Rightarrow v(u \to v), v \Rightarrow u(v \to u)$
Sprache: (Die Menge aller Programme): $L(G) = \{w | w \in T^* \text{ und } S \Rightarrow_P^+ w(w \Rightarrow_P^+ S)\}$
Syntaxanalyse: Überprüfung, ob eine Zeichenreihe(Programm) auf das Axiom S reduzierbar ist. Syntaxanalyse: Überprüfung, ob eine Zeichenreihe(Programm) aus dem Axiom S ableitbar ist.
Definition: Die Zeichenreihen $v \in A^*$ mit
$S \Rightarrow_p^* v(v \Rightarrow_P^* S)$ heißen Satzformen.

Programm und Axiom sind auch Satzformen.
Jedes Programm ist eine Satzform, aber nicht jede Satzform ist ein Programm.



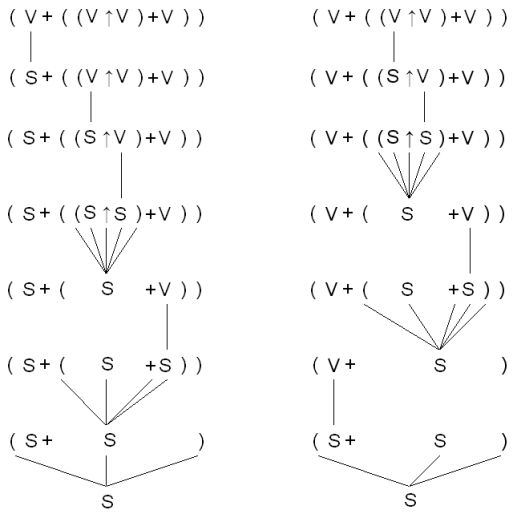
Eindeutigkeit:
Beispiel: $A \to AB, A \to BA \in P$
$u = ABBA, v = ABA$



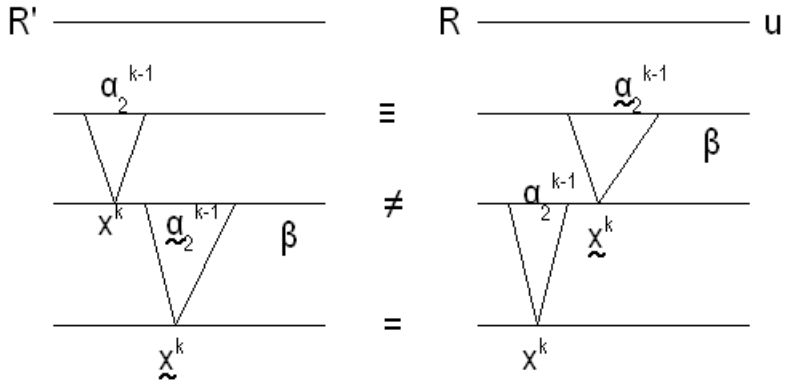


Ableitungs- bzw. Reduktionsbäume: sukzessive Anwendung jeweils einer einzigen Regel.
u läßt sich auf zwei verschiedene Arten auf v reduzieren.
Frage: Wann ist eine derartige Situation Kritisch bzw. harmlos.
Ab jetzt: nur reduzieren(ableiten analog)
Definition: Ein Reduktionsschritt ist ein Tripel $(\alpha_1, X \to \alpha_2, \alpha_3), \alpha_1, \alpha_2 \in A^*, (X, \alpha_2) \in P$.
Eine Reduktionsfolge von u nach v ist eine nichtleere Folge von Reduktionsschritten.
$(\alpha_1^0, X^1 \to \alpha_2^0\alpha_3^0), ..., (\alpha_1^{n-1}, X^{n-1} \to \alpha_2^{n-1}\alpha_3^{n-1}), n \geq 1$
$u = u_0 = \alpha_1^0\alpha_2^0\alpha_3^0, v = u_n = \alpha_1^{n-1}X_1^{n-1}\alpha_3^{n-1}$ und $u_i = \alpha_1^i\alpha^i\alpha_3^i, u_{i+1} = \alpha_1^iX^i\alpha_3^i$

Reduktionsfolgen beschreiben Redunktionsbäume:
Beispiel: $P : S \to V, S \to (S + S), S \to (S * S), S \to (S \uparrow S)$
$N = S, T = \{V, +, *, \uparrow, (,)\}$
$w : (1) : (V + ((V \uparrow V) + V)), (2) : (V + ((v \uparrow V) + V))$

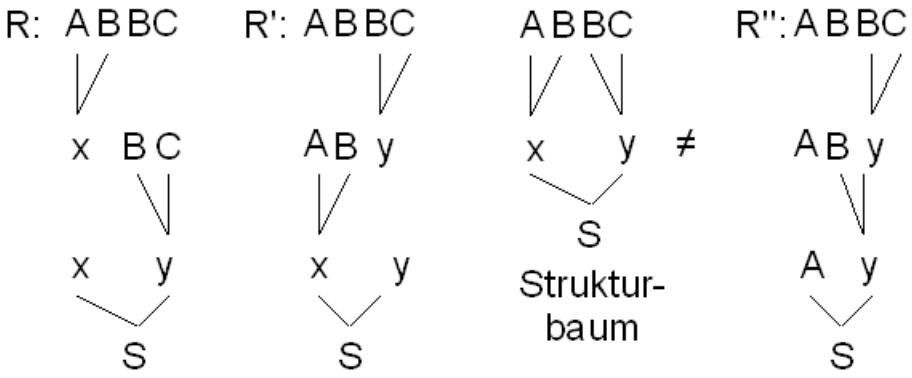


2 unterschiedliche Reduktionsfolgen $\approx$ 2 unterschiedliche Reduktionsbäume, aber lediglich die Reihenfolge der Regeln wurde verändert.
Definition: Seine R und R' zwei Reduktionsfolgen von u nach v (Länge n)
$R^+ \dot{<} R$ heißt unmittelbar kanonischer $\Leftrightarrow$ R' und R sind identisch bis auf den K und k+1-ten Reduktionsschritt, die such durch folgende Situation unterscheiden.



Definition: Eine Reduktionsfolge ist genau dann Kanonisch, wenn es keine Reduktionsfolge R' gibt mit $R' \dot{<} R$.
Definition: R und R' heißen unwesentlich verschieden, wenn eine Serie von Reduktionsfolgen existiert mit $R = R_1, R_2, .., R_m = R'$ mit $R_i < R_{i+1}$ oder $R_{i+1} < R_i$.
Definition: Ein Programm $w \in L(G)$ heißt eindeutig $\Leftrightarrow$ alle Reduktionsfolgen von w nach S sind nur unwesentlich verschieden.
Eine Grammatik G heißt eindeutig $\Leftrightarrow$ alle Sätze(Programme) aus L(G) sind eindeutig.
Definition: Führt man beim Reduzieren in jedem Schritt alle möglichen Reduktionen

parallel aus, so erhält man den Strukturbaum.
Satz: Zwei Reduktionsfolgen sind genau dann unwesentlich verschieden, wenn ihre Strukturbäume identisch sind.
Ein Programm $w \in L(G)$ ist eindeutig.
$\Leftrightarrow$ w besitzt nur einen einzigen Strukturbaum.
Beispiel:
1) $P : S \to XY|AY; Y \to BY|AC; X \to AB; T = \{A, B, C\}, N = \{S, X, Y\}$
$w = ABBC$



R und R’ sind unwesentlich verschieden
Grammatik ist nicht eindeutig
Beispiel bei Programmiersprachen(eindeutig)


```
if <boolean> then for .. do if <boolean> then <us> else <s>
```

!!Ab dieser Vorlesung ist der Stoff nicht mehr klausurrelevant.!!

Am letzten Vorlesungstermin vor der Klausur wird es eine Fragestunden geben. Wer also noch Fragen hat, sollte sich auf diesen Termin gut vorbereiten.

Kombinatorische Logik
Schönfinkel 1924;Curry 1930
Definition: V Menge von Variablen
„C“ Menge von Konstanten, spezielle Konstanten $\overbrace{K, S}^{\text{Basiskonfiguration}}$, $A = V \cup C$
Die Menge T der kombinatorischen Terme ist gegeben durch
1. $A \subset CT$
2. $X, Y \in CT \Rightarrow (XY) \in CT$
Die Semantik der Kombinatoren :
K ax = a
S fgx = fx(gx)
Identität: $I \equiv SKK$
$If \equiv SKKf = Kf(Kf) = f$

Alle „intuitiv“ berechenbaren Funktionen sind von der kombinatorischen Logik darstellbar.
J.W. Backus: „Vater“ von Fortran
1977: FP-Systeme

```
programm p
var a : int
function f(x int): int
begin a:= x +1m f:= a end;
function g(x : int): int
  begin a:= x; + 2; g:= a; end
begin
a:= 0; print(f(a) + g(a));       --->4
a:= 0, print(g(a) + f(a));       --->5
end
```

Ein FP-System (A,F,$\mathbb{F}$,:) besteht aus:
A: Menge vom atomaren Objekten mit $\perp \in A$, true, False $\in A, N|L \in A$
Die Menge O der Objekte ist
$O := A \cup \{< x_1, .., x_n > | n \geq 1, x_i \in O \ \{\perp\}\}$ Sequenzen von Folgen

$N|L \approx <>$ leere Folge
$\perp \approx$ undefiniert $\Rightarrow < ..., \perp, .. > = \perp$
F: Menge von Basisfunktionen : $f|O \to O$
$\mathbb{F}$ Menge von Funktionalen $\begin{cases} f : [O \to O] \to [O \to O] \\ f : O \to [O \to O] \end{cases}$
Programm
def $f_1 \equiv \tau_1$
..
def $f_n \equiv \tau_n$
$f_i : a$
N_0: Objekte
Konstanten Funktion: $\overline{X}$
$\overline{X} : Y := y = \perp \to \perp; X, \epsilon \in 0$
apply_to_all $\alpha : \alpha : f$
$\alpha f : X := c \equiv NIL \to NIL;\ x \equiv < x_1, .., x_n > \wedge f : x_1 \neq \perp \wedge .. \wedge f : x_n \neq \perp$
$\to < f : x_1, .., f : x_n >; \perp$
Komposition: $\circ$
$(f \circ g)x := f : (g : x)$
Reduktion: /
$/f : x := x < x_1 > \to X_1$
$x = < x_1, .., x_n > \wedge n \geq l \to f :< x_1, /f :< x_2, .., x_n >; \perp$

Gegeben: Programm, das die Länge einer Folge bestimmt
def. Folge $\equiv (/+) \circ (\alpha\overline{1})$
Start des Programms:
Länge: $< 1, 2, 3 >$
$\equiv (/+) \circ (\alpha\overline{1}) :< 1, 2, 3 >$
$\equiv (/+) : ((\alpha\overline{1}) :< 1, 2, 3 >)$
$\equiv (/+) :< \overline{1} : 1, \overline{1} : 2, \overline{1} : 3 >$
$\equiv (/+)_< 1, 1, 1 >$
$\equiv + :< 1, /+ :< 1, 1 >>$
$\equiv + :< 1, + :< 1, + < 1 >>>$
$\equiv + :< 1, + :< 1, 1 >>$
$\equiv + :< 1, 2 >$
$\equiv 3$
Programm zur Multiplikation zweier Matrizen
def $MM \equiv (\alpha(\alpha IP)) \circ distr \circ [S1, trans \circ S2]$
def $IP \equiv (/+) \circ (\alpha *) \circ trans$

λ-Kalkül (klassisch ungetypt)
(Church 1932)
Bildung von Funktionen ausschließlich unter dem Aspekt der Rechenvorschrift.
$\Rightarrow$ 1. Funktionen können Argumente sein
2. Funktionen können Ergebnisse sein.

Einführung
Programm $\approx$ Ausdruck $\approx$ Term
1. Ein λ-Term: $\lambda x.M \approx f(\underbrace{x}_{\substack{\text{Variable}\\(\text{Parameter})}}) = \underbrace{M}_{\text{Rumpf}}$
2. Ein λ-Term (M N) entspricht der Anwendung von M auf das Argument N (Funktions-Applikation)
3. Ein λ-Term $((\lambda \times Y)\text{A})$, in dem x in Y auftritt kann reduziert (ausgewertet) werden, in dem man (unter Berücksichtigung von Gültigkeitsbereichen) jedes Vorkommen von x in Y durch A ersetzt.
4. Es gibt 3 Typen von λ-Termen:
Konstanten + Variablen $\approx$ Atome
$\lambda \times M \approx$ Abstraktion
(M N) $\approx$ Applikation
Bsp 1: $\lambda x(xy) \approx$ Anwendung von x auf Y
$(\lambda x(xy)F) \rightarrow^{3.} (Fy)$ (für ein beliebiges F)
2. $\lambda xY \approx$ konstante Funktion mit dem Wert Y
3. $\lambda x(xx) \approx$ Selbstapproximation
$(\lambda x(xx)F) \rightarrow^{3.} (FF)$

Jede n-stellige Funktion kann durch „Curryfizieren“ in eine einstellige überführt werden.
Beispiel:
$h(x,y) = x - y$
$\rightarrow \lambda x(\lambda y(x-y)) \approx \lambda$-Term(Funktion), die eine Funktion als Ergebnis liefert:
$((\lambda x(\lambda y(x-y))5)3) \rightarrow^{3.} ((\lambda y(5-y))3) \rightarrow^{3.} 5-3$
Formale Definition:
Sei V eine Menge von Variablen, C eine Menge von Konstanten, $V \cap C = \emptyset$
$(C = \emptyset \approx$ reiner λ-Kalkül)
$G_\lambda = T_\lambda, N_\lambda, P_\lambda, L_\lambda$
$T_\lambda = (\lambda, C,) \cup V \cup C$
$N_\lambda = L_\lambda, V_\lambda, C_\lambda$
$P_\lambda : L_\lambda \rightarrow V_\lambda | C_\lambda | (\lambda V_\lambda L_\lambda) | (L_\lambda L_\lambda)$
$V_\lambda \rightarrow x|y|...; x, y \in V$

$C_\lambda \to a|b|..; a, b \in C$

	x	$(\lambda$	x	y)	$(\lambda x(xy))$
	V_λ			V_λ	
Beispiel:	L_λ			L_λ	L_λ
			V_λ		
			L_λ		

1. kleine Buchstaben vom Anfang des Alphabets: Konstanten
   kleine Buchstaben vom Ende des Alphabets : Variablen
   Große Buchstaben: „dicke“ Ausdrücke
2. Klammerregeln:
   - Äußere Klammerpaare dürfen fehlen
   - M $N_1...N_n$ bedeutet $(...((MN_1)N_2)...N_n)$ (Linksklammerung)
3. Mehrstellige Funktionen können durch:
   $\lambda x_1..x_n.M$ dargestellt werden.

Beispiel:
$\lambda xy.yx(\lambda z.z) \approx (\lambda x(\lambda y((yx)(\lambda zz))))$

Definition: Λ (Menge alle λ-Terme)
Für $M \in \Lambda$ ist die Menge FV(M) (Menge der freien Variablen von M) und die Menge BV(M) (Menge gebundenen Variablen von M) induktiv durch:
$FV(x) = \{x\}, FV(a) = \emptyset.$
$FV(MN) = FV(M) \cup FV(N)$
$FV(\lambda xM) = FV(M)\ \{x\}$
$BV(x) = \emptyset, BV(a) = \emptyset$
$BV(MN) = BV(M) \cup BV(N)$
$BV(\lambda xM) = BV(M) \cup \{x\}$
Beispiel: $\lambda \underbrace{x}_{geb.}\underbrace{x}_{geb.} : \lambda x(\underbrace{y}_{frei}(\lambda y(y\underbrace{x}_{geb}))) \underbrace{x}_{frei}$
Terme ohne freie Variable sind Kombinatoren(geschlossene Terme)
Prinzip: Reduziere (ausrechnen) alles was möglich ist. Das Endergebnis ist die Bedeutung des Terms.
Formalisierung der Reduktion:
Definition: Die Substitution $\text{Sub}_N^X[M]$ des λ-Terms N(Argumente) für alle freien Vorkommen von x(Parameter) ein λ-Term M(Rumpf f) ist induktiv definiert durch 1a. $\text{Sub}_N^X[x] = N$; 1b. $\text{Sub}_N^x[a] = a$, $a \in (V \cup C)\ \{x\}$

2. $\mathrm{Sub}_N^X[(M_1 M_2)] = (\mathrm{Sub}_N^x[M_1] : \mathrm{Sub}_N^x[M_2])$
3. $\mathrm{Sub}_N^X[\lambda x M] = \lambda x M$
4. $\mathrm{Sub}_N^X[\lambda y M] = \begin{cases} \lambda y\, \mathrm{Sub}_N^X[M] & , \text{ falls } y \notin FV(N) \text{ oder } x \notin FV(M) \\ \lambda z\, \mathrm{Sub}_N^X[\mathrm{Sub}_Z^Y[M]] & , \text{ falls } y \in FV(N) \text{ oder } x \in FV(M) \text{ und } z \\ & \text{ist eine neue Variable, die nicht in N bzw. M vorkommt.} \end{cases}$

Beispiel: Annahme: keine Umbenennung („naive“ Substitution)
5. $\overline{\mathrm{Sub}}_N^x[\lambda y M] = \lambda y \overline{\mathrm{Sub}}_N^X[M]$
Betrachte konstante Funktion $\lambda y x$
Umbenennung in konstante Funktion $\overline{\mathrm{Sub}}_W^X[\lambda y x] = \lambda y w$ konstante Funktion
$\overline{\mathrm{Sub}}_Y^X[\lambda y x] = \lambda y y$ Identität
mit Umbenennung: static scoping
ohne Umbenennung: dynamic scoping

Definition der „Ausrechnung“: Reduzieren
Ein λ-Term P wird zu einem λ-Term P’ reduziert ($P \to P'$), wenn einer der folgenden Reduktionsschritte entweder auf P oder einen Teilterm von P umgewandelt wird.
1. α-Reduktion: $\lambda x M \to_\alpha \lambda y\, \mathrm{Sub}_y^X[M]$ (Umbenennung)
2. β-Reduktion $(\lambda x M) N \to_\beta \mathrm{Sub}_N^X[M]$
Beispiel: I. $\equiv \lambda x x$, $K \equiv \lambda x y.x; S \equiv \lambda x y z.xz(yz)$
SMNL $\equiv (((\lambda x(\lambda y(\lambda z(xz(yz))))M)N)L) \to ((\lambda y(\lambda z(Mz(yz)))N)L)$
$\to_{beta} ((\lambda z(Mz(Nz)))L)$
$\to_\beta ML(NL)$

Frage: Ist die Reihenfolge der Reduktion wichtig?

3. Situation beim Auswerten

1. „Eindeutiges“ Ergebnis: $\lambda x(\lambda y z)zt \to (\lambda y z)t \to z$
2. kein Abbruch beim Reduzieren : $\lambda x(xx)\lambda x(xx) \to \cdots \to \lambda x(xx)\lambda x(xx) \to \cdots$
3. Auswahl verschiedener Alternativen: $\lambda x(\lambda y(yx)z)v$
   (a) $\lambda y(yv)z$
   (b) $\lambda x(zx)v$

   beide kommen zu zv