Automatentheorie und formale Sprachen

Prof. W. Lippe – WWU Münster, Wintersemester 2008/09
Vorlesungsmitschrift von Christian Schulte zu Berge
8. März 2009

Inhaltsverzeichnis

Vorwort

Dieses Skript entstand als Mitschrift in der der Vorlesung „Automatentheorie und formale Sprachen“, gelesen im Wintersemester 2008/2009 von Prof. W. Lippe an der Universität Münster.

Es besteht keine Garantie auf Richtigkeit oder Vollständigkeit des Skriptes. Diese Version der Mitschrift ist zur Veröffentlichung bestimmt und darf unverändert im Original-pdf gerne weiterverbreitet werden. Diese Mitschrift ist noch in der Erstellungsphase und so gibt es noch laufend Ergänzungen und Korrekturen. Dies stets aktuelle Version ist auf meiner Homepage `www.cszb.net` zu finden.

An dieser Stelle möchte ich mich bei Philipp Hanraths bedanken, der in den Vorlesungen wo ich verhindert war fleißig mitgeschrieben und so zu einer möglichst lückenlosen Mitschrift beigetragen hat.

Falls Fehler gefunden werden oder Fragen auftauchen, bitte einfach eine kurze Mail an `skript@cszb.net` schreiben.

Christian Schulte zu Berge

Kapitel 0

Einführung

Wie ist ein Automat aufgebaut:

- Kontrolleinheit
  - kann Zustände annehmen
  - enthält das Programm ≈ Regeln für Zustandsübergänge
- Eingabemedium
  - enthält Eingabezeichenreihe, die verarbeitet werden muss (= lesen)
  - reiner Lesekopf (keine Schreibmöglichkeit)
- Speicher
  - in ihm können Symbole gespeichert, gelesen und geändert werden
  - Schreib- / Lesekopf

Arbeitsweise:

1. Konfiguration ist gegeben durch:
   - Inhalt des Eingabemediums
   - Position des Lesekopfes auf der Eingabe
   - aktueller Zustand der Kontrolleinheit
   - Inhalt des Speichers
   - Position des Schreib- / Lesekopfes
2. Übergangsverhalten ist gegeben durch:
   - aktueller Zustand
   - das gelesene Symbol auf der Eingabe
   - das gelesene Symbol im Speicher
3. Der Automat kann:
   - den Zustand ändern
   - den Lesekopf der Eingabe um *eine* Position verändern
   - das aktuelle Symbol im Speicher lesen/ändern/löschen
   - den Schreib- / Lesekopf des Speichers um *eine* Position verändern

Definition: (Automat)
Ein Automat A ist ein Tupel $A = (Q, I, S, q_0, F, \delta)$ mit:

- Q: endliche Menge von Zuständen
- I: endliches Eingabealphabet
- S: endliches Speicheralphabet
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: Menge von Endzuständen
- δ: Übergangsfunktion

Definition: (Wort/Sprache)
Eine Zeichenreihe, für die A eine Endkonfiguration erreicht, heißt akzeptiertes Wort. Die Menge aller akzeptierten Wörter heißt akzeptierte Sprache $L(A)$.

Variationsmöglichkeiten:

1. deterministisch vs. nichtdeterministisch
2. 1-Weg-Automat vs. 2-Wege-Automat (wie kann der Lesekopf bewegt werden)
3. Speicher
   - kein Speicher
   - der Schreib- / Lesekopf steht stets am oberen Ende (Kellerautomat)
   - der Schreiblesekopf darf überall lesen aber nur oben schreiben/ändern (Stapelautomat)
   - keine Einschränkung (Turingmaschine)
   - beschränkt vs. unbeschränkt
   - strukturierter Speicher (z.B. Baumstruktur)
4. Automaten mit Ausgabe

Die verschiedenen Varianten sind gekennzeichnet durch unterschiedliche Sprachen.

Definition: (Grammatik)
Eine Grammatik G ist ein Tupel $G = (N, T, P, S)$ mit

- N: endliche Menge von nichtterminalen Zeichen
- T: endliche Menge von terminalen Zeichen
- P: endliche Menge von Regeln (Produktionen)
- $S \in N$: Startsymbol

Arbeitsweisen:

1. Top-Down (reduzieren)
   man versucht die gegebene Zeichenreihe zum Startsymbol zu reduzieren
2. Bottom-Up (ableiten)
   ausgehend vom Startsymbol versucht man die Zeichenreihe abzuleiten

Variationen:

1. deterministisch vs. nichtdeterministisch
2. Arten der Regeln
   - linke Seite *nur* nichtterminale Zeichen
   - Länge der Zeichenreihe beschränkt
   - spezielle Formen der Zeichenhreihe

Definition: (Wort/Sprache)
Eine Zeichenreihe aus terminalen Zeichen, die mit Hilfe der Regeln aus P auf das Axiom S reduziert werden kann, heißt Wort (von G erzeugt). Die Menge aller von G erzeugten Wörter heißt Sprache von G, $L(G)$.

Zusammenhänge zwischen Automaten und Grammatiken:
Ein Automat A ist äquivalent zu einer Grammatik G, genau dann wenn $L(A) = L(G)$.

Kapitel 1

Mathematische Grundlagen

1.1 Mengen

An dieser Stelle wurden zunächst sehr grundlegende Dinge zur Mengenlehre eingeführt, welche eigentlich für einen Mathe- oder Informatikstudenten selbstverständlich sein dürften.

1.2 Relationen

Definition: (Relation)
Eine n-stellige Relation über den Mengen $A_1, \ldots, A_n$ ist eine Teilmenge

$$R \subseteq A_1 \times \ldots \times A_n$$

Definition: (Eigenschaften von Relationen)
Eine binäre Relation R auf einer Menge M heißt:

$$\begin{aligned}
\text{symmetrisch} &\Leftrightarrow \forall x, y \in M : (xRy \Rightarrow yRx) \\
\text{antisymmetrisch} &\Leftrightarrow \forall x, y \in M : (xRy \wedge yRx \Rightarrow x = y) \\
\text{asymmetrisch} &\Leftrightarrow \forall x, y \in M : (xRy \Rightarrow \neg yRx) \\
\text{reflexiv} &\Leftrightarrow \forall x \in M : xRx \\
\text{irreflexiv} &\Leftrightarrow \forall x \in M : \neg xRx \\
\text{transitiv} &\Leftrightarrow \forall x, y, z \in M : (xRy \wedge yRz \Rightarrow xRz)
\end{aligned}$$

1.3 Funktionen

Definition: (Totalität und Eindeutigkeit)
Eine binäre Relation $R \subseteq M \times N$ heißt:

$$\begin{aligned}
\text{linkstotal} &\Leftrightarrow \forall x \in M : [\ \exists y \in N : (x, y) \in R\ ] \\
\text{rechtstotal} &\Leftrightarrow \forall y \in N : [\ \exists x \in M : (x, y) \in R\ ] \\
\text{rechtseindeutig} &\Leftrightarrow \forall x \in M \forall y, z \in N : [\ (x, y) \in R \wedge (x, z) \in R \Rightarrow y = z\ ] \\
\text{linkseindeutig} &\Leftrightarrow \forall x, y \in M \forall z \in N : [\ (x, z) \in R \wedge (y, z) \in R \Rightarrow x = y\ ]
\end{aligned}$$

Definition: (Funktion)
Eine rechtseindeutige Relation $F \subseteq M \times N$ heißt partielle Funktion von M nach N. Ist F zusätzlich linkstotal, so heißt F eine total definierte Funktion.

1.4 Halbgruppen und Monoide

Mengen mit auf ihnen definierte Verknüpfungen stellen wichtige algebraische Strukturen dar. Von besonderer Bedeutung sind Halbgruppen und Monoide.

Definition: (Halbgruppe, Monoid)
Eine *Halbgruppe* H besteht aus einer Menge A und einer auf A abgeschlossenen, assoziativen Verknüpfung $\circ$, in Zeichen $[A, \circ]$.

Exisitiert in A zusätzlich ein neutrales Element, so nennt man H *Monoid*.

1.5 Zeichenreihen

Definition: (Zeichenvorrate und Alphabete)

(i) Ein *Zeichenvorrat* ist eine nichtleere endliche Menge A.

(ii) Ist ein Zeichenvorrat A durch eine Relation $\prec$ total geordnet, so nennen wir A *Alphabet*.

Definition: (Zeichenreihe)
Eine total definierte Funktion $Z : [1, \ldots, n] \to A$ heißt *Zeichenreihe* oder *Wort*. Da bei definieren dessen Länge durch n. Das leere Wort wird mit ε bezeichnet.

Definition: (Mengen über Alphabete)
Sei ein Alphabet A gegeben, dann defieren wir:

- A^* als die Menge aller Worte über A einschließlich ε.
- A^+ als die Menge aller Worte über A mit Länge ≥ 1.
- A^n als die Menge aller Worte über A mit Länge $= n$

Definition: (Konkatenation)
Wir definieren die Konkatenation zweier Worte $a = a_1 \cdots a_n$ und $b = b_1 \cdots b_m$ durch die Verknüpfung:

$$a_1 \cdots a_n \circ b_1 \cdots b_m := a_1 \cdots a_n b_1 \cdots b_m$$

Bemerkung:
$[A^*, \circ]$ bildet ein freies Monoid.

1.6 Aussagenlogik

Definition: (Syntax der Aussagenlogik)
Ein zulässiger *aussagenlogischer Ausdruck* (Formel) ist eine endliche Zeichenreihe, die induktiv über dem Alphabet $A = \{a, \ldots, z, \wedge, \vee, \neg, \to, \leftrightarrow, (,), W, F\}$ definiert ist:

(i) Kleine Buchstaben sind ein logischer Ausdruck.

(ii) W, F sind ein logischer Ausdruck.

(iii) Sind A, B zulässige logische Ausdrücke, so auch $(A), \neg A, (A \wedge B), (A \vee B), (A \to B), (A \leftrightarrow B)$.

Notation:
Zur Vereinfachung der Schreibweise werden folgende Prioritäten für die Verknüpfung von zulässigen Ausdrücken vereinbart: $\neg$ vor $\wedge, \vee$, vor $\to$, vor $\leftrightarrow$. Die damit entbehrlichen Klammern können also weggelassen werden, jedoch muss gleichzeitiges Auftreten von $\wedge$ und $\vee$ durch Klammerung geregelt werden.

Definition: (Semantik der Aussagenlogik)

(i) W ist stets wahr, F stets falsch.

(ii) Ein kleiner Buchstabe bezeichnet eine Aussagenvariable, welcher durch eine Wahrheitsfunktion δ ein Wahrheitswert (*Belegung*) $\delta(a) \in \{W, F\}$ zugewiesen wird.

(iii) Für die unären und binären Operatoren gilt die folgende Bedeutung:

		Negation	Konjunktion	Disjunktion	Subjunktion	Bijunktion
$\delta(a)$	$\delta(b)$	$\delta(\neg a)$	$\delta(a \wedge b)$	$\delta(a \vee b)$	$\delta(a \rightarrow b)$	$\delta(a \leftrightarrow b)$
W	W	F	W	W	W	W
W	F	F	F	W	F	F
F	W	W	F	W	W	F
F	F	W	F	F	W	W

(iv) Für einen n-stelligen zulässigen Ausdruck $A(x_1, \ldots, x_n)$ gilt:

$$\delta\left(A(x_1, \ldots, x_n)\right) = A\left(\delta(x_1), \ldots, \delta(x_n)\right)$$

Definition: (Konsistenz, Tautologie)
Sei $A(x_1, \ldots, x_n)$ ein n-stelliger zulässiger Ausdruck

(i) $A(x_1, \ldots, x_n)$ heißt *erfüllbar* oder *konsistent*, wenn es eine Belegung für $x_1, \ldots x_n$ gibt, sodass $\delta(A(x_1, \ldots, x_n)) = W$ gilt. Gibt es keine solche Belegung, so heißt $A(x_1, \ldots, x_n)$ *unerfüllbar* oder *inkonsistent*.

(ii) Ist $A(x_1, \ldots, x_n)$ für jede der 2^n möglichen Belegungen wahr, so heißt die Aussage *allgemeingültig* oder *Tautologie*.

Definition: (Äquivalenz von logischen Ausdrücken)
Zwei logische Ausdrücke heißen äquivalent, wenn sich für jede mögliche Belegung der gleiche Wahrheitswert ergibt, Schreibweise: $A \Leftrightarrow B$.

Definition: (Folgerichtigkeit)
Seien $A_1, \ldots, A_n, B_1, \ldots B_n$ n-stellige logische Ausdrücke. Wenn für jede Belegung der x_i für die der Wahrheitswert der $A_i = W$ auch die $B_i = W$ sind, so heißen die B_i folgerichtig aus $A_1 \ldots, A_n$.

1.7 Graphen

Bemerkung:
Graphen sind wichtig zur Modellierung von Prozessen.

Definition: (Graph)
Ein Graph G ist eine zweistellige Relation auf einer Menge V. V ist dabei die endliche Menge der Knoten.

Ein Graph G kann als Menge von Paaren (x, y) dargestellt werden. Kanten (die Verbindung zwischen zwei Knoten) können benannt werden, dann spricht man von einem gewichteten Graphen.

Alternativ kann man auch die Matrixdarstellung verwenden: Zeilen und Spalten sind Knoten, eine 1 (oder der Name) bedeutet eine Kante, eine 0 bedeutet keine Kante.

Definition: (ungerichteter Graph)
Ein Graph $G \subseteq V \times V$ heißt ungerichteter Graph, wenn G symmetrisch ist, also alle Verbindungen in beide Richtungen laufen.

Definition: (Weg)
Eine Folge von Kanten $w = K_1, K_2, \ldots, K_n = v$ mit $\forall i = 1, \ldots, n-1 : (K_i, K_{i+1}) \in G$ heißt Weg. Dabei gibt es zyklenfreie Wege und Wege mit Zyklen.

Ein Weg heißt zusammenhängend, wenn es zwischen je zwei Knoten auch einen Weg gibt.

Definition: (Baum)
Ein Baum ist ein zusammenhängender, zyklenfreier und ungerichteter Graph.

Kapitel 2

Endliche Automaten

Bemerkung:
Endliche Automaten sind unter anderem Grundlage aller formaler Spezifikationssprachen (Prozessmodellierung), lexikalischer Analyse bei Compilern.

2.1 Endliche Automaten

Definition: (endlicher Automat)
Ein endlicher Automat EA ist ein Tupel $EA = (Q, \Sigma, \delta, q_0, F)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- $\delta : Q \times \Sigma \to Q$: Übergangsfunktion (Menge von Regeln)
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: endliche Menge von Endzuständen

Darstellungen:

1. Mengendarstellung
2. *Matrixdarstellung:*
   Die Zeilen entsprechen den Zuständen, die Spalten sind die Eingabesymbole, in der Matrix steht der neue Zustand.
3. *graphische Darstellung:*
   Die Knoten sind die Zustände. Existiert eine Regel $\delta(q, a) = q'$, dann ex. eine gerichtete Kante zwischen q und q' mit der Benennung a. Endzustände werden durch Doppelkreise dargestellt. Der Startzustand q_0 wird durch einen Kreis mit eingehender unbenannter Kante gekennzeichnet.

Definition: (Wort/Sprache)
Ein von einem endlichen Automaten akzeptiertes Wort ist diejenige Zeichenreihe, die auf der Eingabe gelesen wurde, bis der Automat einen Endzustand erreicht hat. Die Menge aller akzeptierten Wörter nennt man die akzeptierte Sprache eines Automaten.

Definition: (Erweiterung der Definition von δ auf mehrere Schritte)
Die erweiterte Übergangsfunktion $\hat{\delta}$ ist definiert durch:

$$\hat{\delta} : Q \times \Sigma^* \to Q$$
$$\hat{\delta}(q, \varepsilon) = q, \ q \in \Sigma$$
$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a), \ a \in \Sigma, w \in \Sigma^*$$

Definition: (akzeptierte Sprache)
Eine Zeichenreihe $w \in \Sigma^*$ wird in einem endlichen Automaten A akzeptiert, falls $\hat{\delta}(q_0, w) \in F$

Definition: (reguläre Sprachen)
Die in einem endlichen Automaten akzeptierten Sprachen heißen reguläre Sprachen.

2.2 Nichtdeterministische endliche Automaten

Definition: (Nichtdeterministische endliche Automaten)
Ein NEA ist ein Tupel $(Q, \Sigma, \delta, q_0, F)$, wobei

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- $\delta : Q \times \Sigma \to \mathcal{P}(Q) \equiv 2^q$: Übergangsfunktion (Menge von Regeln)
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: endliche Menge von Endzuständen

Die Erweiterung von δ auf $\hat{\delta}$, ist dementsprechend definiert durch:

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$$
$$\hat{\delta}(q, \varepsilon) = \{q\}$$
$$\hat{\delta}(q, w, a) = \{p \mid \text{es gibt einen Zustand } r \text{ in } \hat{\delta}(q, w), \text{ für den } p \text{ in } \delta(r, a) \text{ ist}\}$$

Die Definition der Sprache ist analog zum deterministischen EA.

Satz:
Sei ein nichtdeterministischer endlicher Automat NEA gegeben. Dann lässt sich effektiv ein äquivalenter deterministischer endlicher Automat EA konstruieren mit $L(EA) = L(NEA)$.

Beweis:
Sei $NEA = (q, \Sigma, \delta, q_0, F)$ gegeben. Der äquivalente EA ist gegeben durch $EA = (Q', \Sigma, \delta', q_0', F')$ mit

1. $Q' = \mathcal{P}(Q)$. Die Elemente in Q' sind bezeichnet durch $[q_1, q_2, \ldots, q_n]$.
2. F' ist die Menge aller Zustände aus Q', die mindestens einen Zustand aus F enthalten.
3. $q_0' = [q_0]$.
4. $$\delta'([q_1, \ldots, q_i], a) = [p_1, \ldots, p_j] \;\Leftrightarrow\; \{p_1, \ldots, p_j\} = \bigcup_{l=1,\ldots,i} \delta(q_l, a)$$

□

2.3 Endliche Automaten mit ε-Übergang

Definition: (endliche Automaten mit ε-Übergang)
Ein endlicher Automat mit ε-Übergang ist ein Tupel $\varepsilon NEA = (Q, \Sigma, \delta, q_0, F)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to Q$: Übergangsfunktion (Menge von Regeln)
- $q_0 \in Q$: Startzustand
- $F \subseteq Q$: endliche Menge von Endzuständen

Eine Zeichenreihe w wird akzeptiert, wenn es einen mit w markierten Pfad gibt, der auch ε Kanten enthalten kann und von q_0 zu einem Endzustand führt.

Definition: (ε-Hülle)
Die ε-Hülle $(q), q \in Q$, ist die Menge aller Knoten p, für die es einen Weg von q nach p gibt, dessen Kanten ausschließlich mit ε markiert sind (einschließlich q selbst).

Definition: (erweiterte Übergangsfunktion $\hat{\delta}$)
$\hat{\delta}$ für einen ε-NEA ist induktiv gegeben durch

$$\hat{\delta}(q, \varepsilon) = \varepsilon\text{-Hülle}(q)$$
$$\hat{\delta}(q, wa) = \varepsilon\text{-Hülle}(P) \text{ mit } w \in \Sigma^*, a \in \Sigma \text{ und } P = \{p \mid \text{es ex. ein } r \text{ in } \hat{\delta}(q, w) \text{ und } p \in \delta(r, a)\}$$

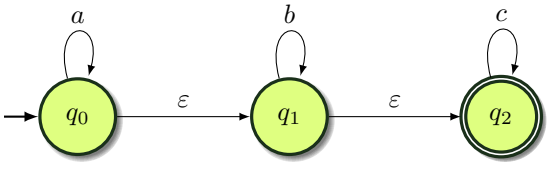
Erweiterung auf Zustandsmengen R

$$\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$$
$$\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$$

Definition: (Sprache eines ε-NEA)
Die von einem ε-NEA akzeptierte Sprache $L(\varepsilon NEA)$ ist gegeben durch

$$L(\varepsilon NEA) = \{w \mid \hat{\delta}(q_0, w) \in F\}$$

Beispiel:
Betrachten wir den folgenden Automaten:



Dann sind die ε-Hüllen gegeben durch:

$$q_0 := \{q_0, q_1, q_2\}$$
$$q_1 := \{q_1, q_2\}$$
$$q_2 := \{q_2\}$$

Und damit gilt:

$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= \varepsilon\text{-Hülle}(q_0) \\
\hat{\delta}(q_0, a) &= \varepsilon\text{-Hülle}(\hat{\delta}(q_0, \varepsilon), a) \\
&= \varepsilon\text{-Hülle}(\delta(\{q_0, q_1, q_2\}, a)) \\
&= \varepsilon\text{-Hülle}(\delta(q_0, a) \cup \delta(q_1, a) \cup \delta(q_2, a)) \\
&= \varepsilon\text{-Hülle}(\{q_0\}) = \{q_0, q_1, q_2\} \\
\hat{\delta}(q_0, ab) &= \{q_1, q_2\} \\
L(\varepsilon NEA) &= a^* b^* c^*
\end{aligned}$$

Satz:
Zu jedem ε-NEA lässt sich effektiv ein NEA konstruieren mit $L(\varepsilon NEA) = L(NEA)$.

Beweis:
Sei $\varepsilon NEA = (Q, \Sigma, \delta, q_0, F)$ gegeben, dann ist $NEA = (Q, \Sigma, \delta', q_0 F')$ mit

$$\delta'(q, a) = \hat{\delta}(q, a), q \in Q, a \in \Sigma$$

$$F' = \begin{cases} F \cup \varepsilon\text{-Hülle}(q_0) & \text{falls die } \varepsilon\text{-Hülle von } q_0 \text{ einen Zustand aus } F \text{ enthält} \\ F & \text{sonst} \end{cases}$$

Durch Induktion über $|w|, w \in \Sigma^*$ müsste man nun noch zeigen, dass $\hat{\delta}'(q_0, w) = \hat{\delta}(q_0, w)$. □

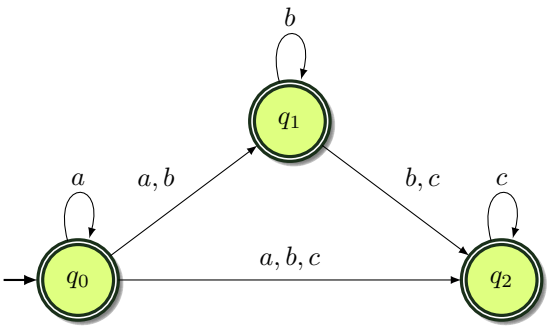
Beispiel:
Wir betrachten das Beispiel von oben.

1. ε-Hüllen bestimmen [...]
2. $\hat{\delta}$ bestimmen

$$\begin{aligned}
\hat{\delta}(q_0, \varepsilon) &= \varepsilon\text{-Hülle}(q_0) = \{q_0, q_1, q_2\} \\
\hat{\delta}(q_0, a) &= \{q_0, q_1, q_2\} \\
\hat{\delta}(q_0, b) &= \hat{\delta}(q_0, \varepsilon b) \\
&= \varepsilon\text{-Hülle}(\delta(\hat{\delta}(q_0, \varepsilon), b)) \\
&= \varepsilon\text{-Hülle}(\delta(q_0, b) \cup \delta(q_1, b) \cup \delta(q_2, b)) \\
&= \varepsilon\text{-Hülle}(\emptyset \cup \{q_1\} \cup \emptyset) = \{q_1, q_2\} \\
\hat{\delta}(q_0, c) &= \ldots = \{q_2\}
\end{aligned}$$

$$\hat{\delta}(q_1, \varepsilon) = \{q_1, q_2\}, \quad \hat{\delta}(q_1, a) = \emptyset, \quad \hat{\delta}(q_1, b) = \{q_1, q_2\}, \quad \hat{\delta}(q_1, c) = \{q_2\},$$

Der Automat von oben kann auch so dargestellt werden:



Bemerkung:
Bisher steht fest: NEA mit ε-Übergängen $\equiv$ NEA ohne ε-Übergänge $\equiv$ DEA

Aber wie können wir die Sprachklassen füe EAs charakterisieren?

Kapitel 3

Reguläre Ausdrücke

Erinnerung Zeichenreihen:
Sie Σ eine endliche Menge von Symbolen, $L, L_1, L_2 \approx$ Mengen von Zeichenreihen aus Σ^*. Konkatenation von L_1, L_2 ist gegeben durch:

$$(L_1, L_2) = \{xy \mid x \in L_1, y \in L_2\}$$
$$L^0 = \{\varepsilon\},\ L^i = LL^{i-1}, i \geq 1$$

Definition: (Hüllen)
Die transitiv reflexive Hülle (Kleenesche Hülle) von L ist gegeben durch:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Die transitive Hülle (positive Hülle) von L ist gegeben durch:

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Beispiel:
Sei $\Sigma = \{0, 1\}, L_1 = \{10, 1\}, L_2 = \{011, 11\}$, dann ist $(L_1 L_2) = \{10011, 1011, 111\}$.
Sei $L = \{10, 11\}$, dann ist $L^* = \{\varepsilon, 10, 11, 1010, 1011, 1111, 1110, \ldots\}$

Definition: (Reguläre Ausdrücke)
Sei Σ ein Alphabet (endliche Menge von Symbolen). Die *regulären Ausrücke* über Σ und die von ihnen beschriebenen Mengen sind rekursiv definiert durch:

(i) $\emptyset$ ist ein regulärer Ausdruck und bezeichnet die leere Menge

(ii) ε ist ein regulärer Ausdruck und bezeichnet die Menge $\{\varepsilon\}$

(iii) Für $a \in \Sigma$ ist a ein regulärer Ausdruck und bezeichnet die Menge $\{a\}$

(iv) Seien r, s reguläre Ausdrücke, die die Mengen (Sprachen) R, S bezeichnen.

- (a) Dann ist $(r + s)$ ein regulärer Ausdruck und bezeichnet die Menge $R \cup S$.
- (b) Dann ist (rs) ein regulärer Ausdruck und bezeichnet die Menge RS (Konkatenation).
- (c) Dann ist (r^*) ein regulärer Ausdruck und bezeichnet die Menge R^*.

Es gelten die folgenden Prioritäten: $* \geq$ Konkatenation $\geq +$.

Notation:
Sofern keine Komplikationen auftreten wird zwischen regulärem Ausdruck r und der Sprache (Menge) die r bezeichnet nicht unterschieden.

Beispiel:

regulärer Ausdruck	Menge
00	$\{00\}$
$(0+1)^*$	Menge aller Zeichenreihen über 0,1 einschließlich ε
$(0+1)^*00(0+1)^*$	Menge aller Zeichenreihen über 0,1 die mindestens 00 enthalten

Satz: *Äquivalenz zwischen endlichen Automaten und regulären Ausdrücken*
Sei r ein beliebiger regulärer Ausdruck. Dann gibt es einen ε-NEA, der $L(r)$ akzeptiert, d.h. $L(r) = L(\varepsilon NEA)$.

Beweis:
Durch Induktion über Anzahl der Operatoren in r:

(i) $EA = (\{q_0\}, \emptyset, \emptyset, q_0, \{q_0\})$

(ii) $EA = (\{q_0, q_1\}, \emptyset, \emptyset, q_0, \{q_1\})$

(iii) $EA = (\{q_0\}, \{a\}, \delta(q_0, a) = q_1, q_0, \{q_1\})$

(iv) Induktionsschritt: Der Satz gelte für alle regulären Ausdrücke mit weniger als i Operatoren. Wir haben also schon zwei gültige endliche Automaten $EA_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\}), EA_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$, wobei ohne Einschränkung der Allgemeinheit $Q_1 \neq Q_2$.

(a) $r = r_1 + r_2$:
Dies realisieren wir durch eine Parallelschaltung von EA_1, EA_2. Formal:

$$EA = (Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\})$$

mit

$$\begin{aligned}
\delta(q_0, \varepsilon) &= \{q_1, q_2\} \\
\delta(q, a) &= \delta_1(q, a) \quad , q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \{\varepsilon\} \\
\delta(q, a) &= \delta_2(q, a) \quad , q \in Q_2 - \{f_2\}, a \in \Sigma_2 \cup \{\varepsilon\} \\
\delta(f_1, \varepsilon) &= \delta(f_2, \varepsilon) = \{f_0\}
\end{aligned}$$

Durch scharfes Hingucken sehen wir, dass der neue Automat genau das macht, was er soll.

(b) $r = r_1 r_2$:
Die Konkatenation realisieren wir durch eine Reihenschaltung von EA_1, EA_2. Formal:

$$EA = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\})$$

mit

$$\begin{aligned}
\delta(q, a) &= \delta_1(q, a) \quad , q \in Q_1 - \{f_1\}, a \in \Sigma_1 \cup \varepsilon \\
\delta(f_1, \varepsilon) &= \{q_2\} \\
\delta(q, a) &= \delta_2(q, a) \quad , q \in Q_2, a \in \Sigma_2 \cup \varepsilon
\end{aligned}$$

(c) $r = r_1^*$:
realisieren wir durch eine Schleife, die EA_1 beliebig oft wiederholen lässt. Formal:

$$EA = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\})$$

mit

$$\begin{aligned}\delta(q_0, \varepsilon) &= \{q_1, f_0\} \\ \delta(f_1, \varepsilon) &= \{q_1, f_0\} \\ \delta(q, a) &= \delta_1(q, a) \quad , q \in Q_1, a \in \Sigma_1 \cup \varepsilon\end{aligned}$$

□

Satz: *Äquivalenz zwischen endlichen Automaten und regulären Ausdrücken*
Wenn L von einem DEA akzeptiert wird, dann wird L durch einen regulären Ausdruck beschrieben

Beweis:
Sei M ein DEA mit

$$M = (\{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F)$$

Sei ferner R_{ij}^K die Menge aller Zeichenreihen, die DEA vom Zustand q_i in den Zustand q_j bringt, ohne dabei einen Zustand zu erreichen, der mit etwas echt größerem als K nummeriert ist. Demnach ist R_{ij}^n die Menge *aller* 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 Teilmengen zusammen:
   (a) einer Zeichenreihe aus R_{iK}^{K-1} (bringt M ausgehend von q_i zum ersten Mal nach q_K)
   (b) beliebig vielen Zeichenhreihen aus R_{KK}^{K-1} (bringt M von q_K nach q_K)
   (c) einer Zeichenreihe aus R_{Kj}^{K-1} (bringt M von q_K nach q_j)

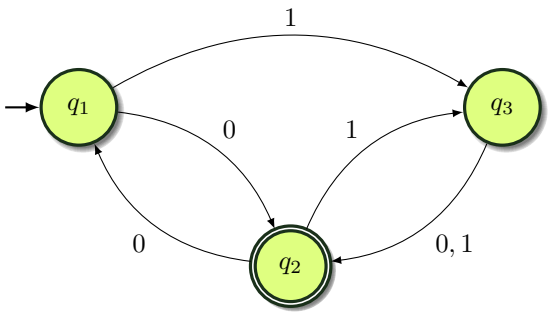
Daraus können wir die rekursive Definition für R_{ij}^K ableiten durch:

$$R_{ij}^K = R_{ij}^{K-1} \cup \left(R_{KK}^{K-1}\right)^* \cup R_{Kj}^{K-1}$$

$$R_{ij}^0 = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{für } i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{für } i = j \end{cases}$$

Zu zeigen durch Induktion: Für alle i, j, K gibt es einen regulären Ausdruck r_{ij}^K, der die Sprache R_{ij}^K repräsentiert. Durch scharfen Hingucken sehen wir jedoch, dass alle Operationen in der Definition der R_{ij}^K den Operationen aus den regulären Ausdrücken entsprechen. □

Beispiel:
Sei der folgende EA gegeben:



Wir möchten jetzt systematisch $L(EA)$ bestimmen:

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

1. Bestimme maximalen Index der Zustände $n = 3$
2. Bestimmung des Anfangszustandes (q_1) und der Endzustände $(\{q_2, q_3\})$
3. Bestimme alle $r_{AnfEnd}^n \approx r_{12}^3, r_{13}^n$
4. Bestimme alle diese Ausdrücke mit $+$

So kommen wir auf:

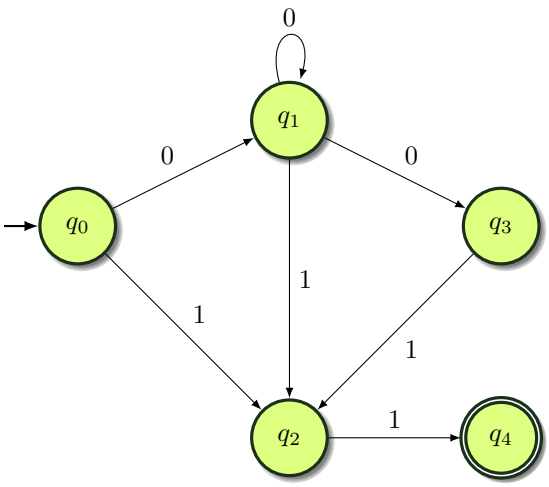
$$\begin{aligned} r_{12}^3 &= r_{13}^2(r_{33}^2)^*r_{r2}^2 + r_{12}^2 \\ &= 0^*1(\varepsilon + (0+1)0^*1)^*(0+1)(00)^* + 0(00)^* \\ &= 0^*1((0+1)0^*1)^*(0+1)(00)^* + 0(00) \\ r_{13}^3 &= 0^*1((0+1)0^*1)^* \\ L(EA) &= r_{12}^3 + r_{13}^3 \end{aligned}$$

Bemerkung:
Die von endlichen Automaten akzeptierten Sprachen heißen auch reguläre Sprachen.

Kapitel 4

Minimierung von endlichen Automaten

Beispiel:
Betrachten wir den folgenden Automaten:



dann gilt $L(EA) = \{11\} \cup \{00^*11\} \cup \{00^*011\}$. Offensichtlich ist der Zustand q_3 überflüssig

Bemerkung:
Aufgrund der Äquivalenz beschränken wir uns in Zukunft nur auf deterministische endliche Automaten.

Definition: (Äquivalenz von Zuständen)
Zwei Zustände p, q heißen äquivalent, wenn für *alle* Zeichenreihen $w \in \Sigma^*$ gilt:

$$\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$$

(Dabei muss es *nicht* der gleiche Endzustand sein).

Zwei Zustände p, q heißen unterscheidbar, falls mindestens eine Zeichenreihe $w \in \Sigma$ existiert, sodass $\hat{\delta}(p, w) \in F$ aber $\hat{\delta}(q, w) \notin F$.

Definition: (Äquivalenz von endlichen Automaten)
Zwei endliche Automaten heißen äquivalent, wenn ihre Anfangszustände äquivalent sind.

Algorithmus: *Table-Filling-Algoritmus*
Konstruiere eine Tabelle $T : Q \times Q \to \{0, 1\}$ durch:

$$T(q, q') := \begin{cases} 0 & \text{falls } (q \in F \wedge q' \notin F) \vee (q \notin F \wedge q' \in F) \\ 1 & \text{sonst} \end{cases}$$

Führe nun die folge Schleife aus:

$$T(q, q') = 0, \text{ falls } \exists a \in \Sigma : T\left(\delta(q, a), \delta(q', a)\right) = 0$$

bis „keine Veränderung mehr“.

Algorithmus: *Bestimmung eines minimalen Automaten*
Bestimung des minimalen Automaten $EA' = (Q', \Sigma, \delta', q_0', F')$ zum Ausgangsautomaten $EA = (Q, \Sigma, \delta, q_0, F)$ durch:

(i) Eliminiere alle Zustände, die von q_0 nicht erreichbar sind.

(ii) Ermittle mit dem Table-Filling-Algorithmus alle Paare äquivalenter Zustände (mit 1 markiert).

(iii) Partitioniere Q in Blöcke (sind (p, q) und (q, r) äuqivalent, dann ist auch (p, r) äquivalent).

(iv) Die Blöcke sind die Zustandsmenge Q' des Minimalautomaten

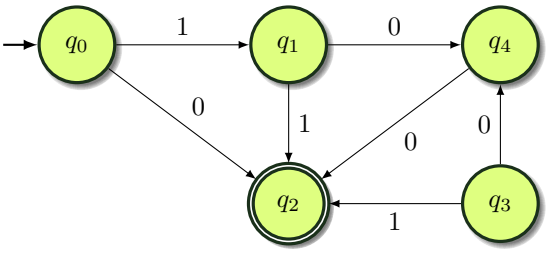
(v) δ' des Minimalautomaten ist gegeben durch: Seien $S, T \in Q'$ Blöcke und $a \in \Sigma$ beliebig

$$[\ \delta'(S, a) = T\ ] \quad \Leftrightarrow \quad [\ \forall q \in S : (\delta(q, a) \in T)\ ]$$

(vi) Der Anfangszustand ist derjenige Block, in dem q_0 liegt

(vii) Alle Blöcke die einen alten Endzustand enthalten sind neue Endzustände

Beispiel:
Betrachten wir den folgenden Automaten



Wir sparen uns Schritt 1 und initialisieren direkt unsere Table-Filling-Tabelle:

T	q_0	q_1	q_2	q_3	q_4
q_0		1	0	1	1
q_1			0	1	1
q_2				0	0
q_3					1
q_4					

Nun iterieren wir die Schleife:

$$[\ \delta(q_0, 0) = q_2, \delta(q_1, 0) = q_4\ ] \quad \Rightarrow \quad [\ T(q_2, q_4) = 0 \Rightarrow T(q_0, q_1) = 0\ ]$$

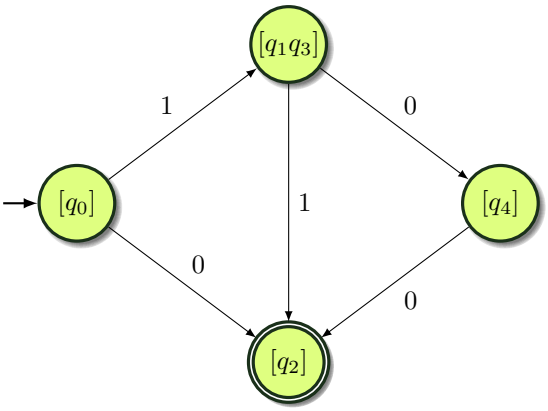
$$\vdots$$

Nach dem ersten Durchlauf erhalten wir die Tabelle:

T	q_0	q_1	q_2	q_3	q_4
q_0		0	0	0	0
q_1			0	1	0
q_2				0	0
q_3					0
q_4					

Ein zweiter Durchlauf bringt keine Veränderung mehr.

Der Algorithmus erstellt „Blöcke“ (Mengen) die aus einem Zustand q selbst und allen zu q äquivalenten Zuständen besteht. Die neuen Zustände sind: $[q_0], [q_1, q_3], [q_2], [q_4]$. Heraus kommt dieser schicke Automat:



Kapitel 5

Endliche Automaten mit Ausgabe

Bemerkung:
Wir haben zwei verschiedene Möglichkeiten die Ausgabe zu modellieren: Entweder die Ausgabe an einen Zustand zu binden (*Moore-Automat*), oder die Ausgabe an eine Kante zu binden (*Mealy-Automat*).

Definition: (Moore-Maschine)
Eine Moore-Maschine ist ein Tupel $MO = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- Δ: endliches Ausgabealphabet
- $\delta : Q \times \Sigma \to Q$: Übergangsfunktion (Menge von Regeln)
- $\lambda : Q \to \Delta$: Ausgabefunktion
- $q_0 \in Q$: Startzustand

Bemerkung:
Die Ausgabe von einem Moore-EA als Antwort auf die Eingabe $a_1, \ldots, a_n, n \geq 0$, ist gegeben durch

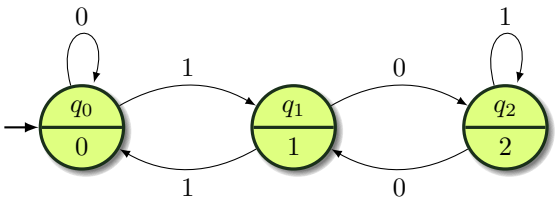
$$\lambda(q_0), \ldots, \lambda(q_{n-1})$$

wobei $q_0, \ldots, q_n$ diejenige Zustandsfolge ist, für die $\delta(q_{i-1}, a_i) = q_i$.

Es gibt keine Endzustandsmenge. Ein klassischer EA ist „äquivalent“ zu einer Moore-Maschine, für die $\Delta = \{0, 1\}$ und ein Zustand als Endzustand interpretiert wird, wenn seine Ausgabe 1 ist.

Ausgabe erfolgt bereits im Startzustand, d.h. für die Eingabe ε wird $\lambda(q_0)$ ausgegeben.

Beispiel:
Gesucht ist eine Moore-Maschine, die für eine binäre Zeichenreihe (interpretiert als ganze Zahl) den Rest modulo 3 ausgibt:



Definition: (Mealy-Maschine)
Eine Mealy-Maschine ist ein Tupel $ME = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- Δ: endliches Ausgabealphabet
- $\delta : Q \times \Sigma \to Q$: Übergangsfunktion (Menge von Regeln)
- $\lambda : Q \times \Sigma \to \Delta$: Ausgabefunktion
- $q_0 \in Q$: Startzustand

Bemerkung:
Die Ausgabe von einem Mealy-EA als Antwort auf die Eingabe $a_1, \ldots, a_n, n \geq 0$, ist gegeben durch

$$\lambda(q_0, a_1), \ldots, \lambda(q_{n-1}, a_n)$$

wobei $q_0, \ldots, q_n$ diejenige Zustandsfolge ist, für die $\delta(q_{i-1}, a_i) = q_i$.

Vergleich von Moore- und Mealy-Maschinen:

1. Für die Eingabe ε liefert eine Mealy-Maschine stets ε, die Moore-Maschine ein $\lambda(q_0) \in \Delta$.
2. Zu einem gegebenen Weg mit n Knoten, ist die Länge der Ausgabe einer Moore-Maschine $n+1$ und die einer Mealy-Maschine nur n.

„De Facto“ *(sic!)* liegt der Unterschied in der ersten Ausgabe.

Definition: (Ausgabe einer Maschine)
Sei M eine Mealy- oder Moore-Maschine. Wir definieren $T_M(w), w \in \Sigma^*$ als die Ausgabe, die von M bei Abarbeitung von w ausgegeben wird.

Bemerkung:
T_{ME} und T_{MO} erzeugen für ein $w \in \Sigma^*$ nie die gleiche Ausgabe.

Definition: (Äquivalenz von Maschinen mit Ausgabe)
Ein Mealy-EA und ein Moore-EA sind äquivalent, wenn für $b := \lambda_{MO}(q_0)$ gilt:

$$\forall w \in \Sigma^* \ : \ b\, T_{ME}(w) = T_{MO}(w)$$

Satz: *Überführung von Moore-Maschinen in Mealy-Maschinen und umgekehrt*
$MO = (Q_{MO}, \Sigma, \Delta, \delta, \lambda_{MO}, q_0)$ lässt sich effektiv in eine äquiv. Maschine $ME = (Q_{ME}, \Sigma, \Delta, \delta, \lambda_{ME}, q_0)$ überführen durch:

$$\begin{aligned} Q_{ME} &= Q_{MO} \\ q_{0_{ME}} &= q_{0_{MO}} \\ \lambda_{ME}(q, a) &= \lambda_{MO}(\delta(q, a)), q \in Q, a \in \Sigma \end{aligned}$$

Umgekehrt funktioniert dies durch:

$$\begin{aligned} Q_{MO} &= \{[q, b] \mid q \in Q_{ME}, b \in \Delta\} \\ q_{0_{MO}} &= [q_{0_{ME}}, b_0], b_0 \text{ beliebig aus } \Delta \\ \lambda_{MO}(q) &= ([q, b]) = b \end{aligned}$$

Kapitel 6

Eigenschaften von regulären Mengen

6.1 Pumping-Lemma

Bemerkung:
Alles was für reguläre Mengen gilt, gilt auch für endliche Automaten.

Das Pumping-Lemma informal:
Sei eine reguläre Sprache R gegeben. Dann existiert ein DEA $M = (Q, \Sigma, \delta, q_0, F)$ mit $R = L(M)$.
Annahme: $|Q| = n$
Betrachte Eingabe, die n oder mehr Symbole umfasst: $a_1, \ldots, a_m, m \geq n$. Sei $\delta(q, a_1, \ldots, a_i) = q_i$. Dann nimmt M $m{+}1$ Zustände an, also muss mindestens ein Zustand doppelt vorkommen, zum Beispiel $q_j = q_k$.

Also existiert irgendwo mindestens ein Zyklus $a_{j+1}, \ldots, a_k$, der mindestens die Länge 1 hat und nicht länger als n ist.

Sei $q_m \in F$, dann gilt:

$$\Rightarrow a_1 \cdots a_j a_{k+1} \cdots a_m \in L(M)$$
$$\Rightarrow a_1 \cdots a_j \left(a_{j+1} \cdots a_k\right)^* a_{k+1} \cdots a_m \in L(M)$$

„Dies ist quasi das Pumping-Lemma."

Satz: *Pumping-Lemma für reguläre Mengen*
Sei L eine reguläre Menge. Dann gibt es eine Konstante n, sodass sich ein Wort $z \in L$ mit $|z| \geq n$ schreiben lässt in $z = uvw$ mit $v \neq \varepsilon \wedge |uv| \leq n$. Weiter gilt:

$$\forall i \in \mathbb{N} : uv^i w \in L$$

6.2 Anwendungen des Pumping-Lemmas

Beispiel:
Gegeben sei die Sprache $L = a^n b^n$. Zu zeigen ist, dass L *keine* reguläre Sprache ist:

Wähle das n aus dem Pumping-Lemma und Betrachte alle möglichen Aufspaltungen von $z \in L$

$$z = a^n b^n = uvw$$

mit den Randbedingungen aus dem Pumping-Lemma, also $v \neq \varepsilon, |uv| \leq n$. Also kann uv nur aus a bestehen.

Mit dem Pumping-Lemma folgt nun dass $\forall i \in \mathbb{N} : uv^i w \in L$, insbesondere auch für $i = 2$. Da aber v mindestens ein a enthält, enthält $uv^2 w$ mindestens ein a mehr als b.

Widerspruch!

Prinzipielle Anwendung des Pumping-Lemmas:

1. Gegeben eine Sprache L von der gezeigt werden soll, dass sie *nicht* regulär ist
2. Wähle *eine* konkrete und feste Zeichenreihe aus $z \in L$ (genügend lang, darf vom n aus dem Pumping-Lemma abhängen)
3. Zeige, dass für *jede* mögliche Aufspaltung von z in uvw mit den Randbedingungen es eine Zeichenreihe $uv^i w, i \in \mathbb{N}$ gibt, die *nicht* in L liegt.
4. Dann folgt, dass L nicht regulär ist.

6.3 Abschlusseigenschaften von regulären Mengen

Definition: (Abgeschlossenheit regulärer Mengen)
Eine reguläre Menge heißt abgeschlossen bezüglich einer Operation, wenn die Anwendung dieser Operation wieder eine reguläre Menge liefert.

Satz:
Die regulären Mengen sind abgeschlossen unter den Operationen Vereinigung, Konkatenation, und Kleen'scher Hüllenbildung.

Beweis:
Folgt direkt aus der induktiven Definition regulärer Mengen. □

Satz:
Die regulären Mengen sind abgeschlossen unter Komplementbildung. Also sei $L \subseteq \Sigma^*$ eine reguläre Menge, dann ist auch $\Sigma^* \setminus L$ eine reguläre Menge.

Beweis:
Sei $L \subseteq \Sigma^*$ die akzeptierte Sprache für den DEA

$$M = (Q, \Sigma_1, \delta, q_0, F)$$

Setze ohne Beschränkung der Allgemeinheit $\Sigma = \Sigma_1$ (Angenommen es existieren Symbol in Σ_1, die nicht in Σ sind, dann sind diese Symbole „Sackgassen" und können gelöscht werden. Umgekehrt: existieren Symbole in Σ, die nicht in Σ_1 sind, so treten sie in keinem Wort aus $L(M)$ auf.)
Wir konstruieren nun einen endlichen Automaten mit $L(M') = \Sigma^* \setminus L$:

$$M' = (Q, \Sigma, \delta, q_0, Q \setminus F)$$

Es gilt nun:

$$\begin{aligned} w \in L(M') &\Leftrightarrow \delta(q_0, w) \in Q \setminus F \\ &\Rightarrow w \in \Sigma^* \setminus L \end{aligned}$$

Anm. d. Verf.: Die Aussage des Satzes stimmt freilich auf jeden Fall. Jedoch habe ich gewisse Zweifel, dass dieser Beweis dies auch wirklich zeigt. □

Satz:
Die regulären Mengen sind unter Schnittbildung abgeschlossen.

Beweis:

$$L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$$

□

6.4 Substitution und Homomorphismen

Beispiel:
Seien zwei Alphabete Σ, Δ gegeben mit $\Sigma = \{0,1\}, \Delta = \{a,b\}$. Dann ist eine Substitution f gegeben durch $f(0) = a, f(1) = b^*$.

f lässt sich natürlich auch auf Zeichenreihen erweitern, so ist z.B. $f(010) = ab^*a$.

Analog ist die Erweiterung auf Sprachen (Mengen von Zeichenreihen). Sei z.B. $L = 0^*(0+1)1^*$, so gilt $f(L) = a^*(a+b^*)\,(b^*)^* = a^*b^*$

Definition: (Substitution)
Seien Σ, Δ zwei Alphabete. Eine Substitution f ist eine Abbildung von Σ auf Teilmengen von Δ^*.

f lässt sich auf Zeichenketten erweitern durch:

$$f(\varepsilon) = \varepsilon$$
$$f(xa) = f(x)f(a)$$

f lässt sich auf Sprachen erweitern durch:

$$f(L) = \bigcup_{x \in L} f(x)$$

Satz:
Die Klasse der regulären Mengen ist unter Substitution abgeschlossen.

Beweis:
Sei $R \subseteq \Sigma^*$ ein reguläre Menge. Sei ferner für jedes $a \in \Sigma$ $R_a \subseteq \Delta^*$ eine reguläre Menge. Sei $f : \Sigma \to \Delta^*$ eine Substitution gegeben durch $f(a) = R_a$.

Betrachte die regulären Ausdrücke, die R und die R_a repräsentieren:
Es gilt: Die Substitution einer Vereinigung eines Produktes oder einer Hülle ist gleich der Vereinigung des Produktes bzw. der Hülle der Substitution, d.h. $f(L_1 \cup L_2) = f(L_1) \cup f(L_2)$ und $f(L^*) = f(L)^*$.
Dies gilt es noch durch Induktion über die Anzahl der Operatoren zu beweisen, doch Prof. Lippe verzichtet darauf.

Allgmein: Gegeben ein regulärer Ausdruck. Durch die Ersetzung von einzelnen Symbolen durch reguläre Ausdrücke entstehen Zeichenreihen, die nur die für reguläre Ausdrücke erlaubten Operationen enthalten. □

Definition: (Homomorphismus)
Ein Homomorphismus h ist eine Substitution, bei der $h(a)$ eine *einzige* Zeichenreihe für jedes a enthält (*Anm. d. Verf.*: Prof. Lippe meint damit wohl Rechtseindeutigkeit). Erweiterung analog zur Substitution.

Beispiel:
$h(0) = aa,\ h(1) = aba \Rightarrow h(010) = aaabaaa$

Bemerkung:
Ein Homomorphismus ist also ein Spezialfall der Substitution. Damit gilt der Satz über die Abgeschlossenheit auch für Homomorphismen.

Definition: (Inverser Homomorphismus)
Sei h ein Homomorphismus, dann ist der inverse Homomorphismus definiert durch:

$$h^{-1}(L) = \{x \mid h(x) \in L\}$$

Der inverse Homomorphismus ist also gerade das Urbild.

Beispiel:
Sei $L_2 = (ab + ba)^* a \Rightarrow h^{-1}(L_2) = \{1\}$.

Satz: *Abgeschlossenheit*
Reguläre Mengen sind auch unter inversen Homomorphismen abgeschlossen.

Beweis:
ohne Beweis. □

Satz: *Kardinalität der Sprache von endlichen Automaten*
Die Menge von Sätzen, die von einem EA aus n Zuständen akzeptiert wird ist

(i) genau dann nicht leer, wenn der EA einen Satz der Länge $< n$ akzeptiert

(ii) genau dann unendlich, wenn der EA Sätze der Länge l mit $n \leq l \leq z_n$ akzeptiert

Beweis:

1. „$\Leftarrow$“: trivial

   „$\Rightarrow$“: Es existiere eine nicht leere Menge von Sätzen und sei w das kürzeste aller anderen akzeptierten Wörter.
   Angenommen es sei $|w| \leq n$, so kann man w nach dem Pumping-Lemma zerlegen in xyz mit $|y| \geq 1$.
   Aber da xz auch ein Wort der Sprache ist, kann w nicht das kürzeste Wort der Sprache sein.
   Also muss $|w| < n$ gelten.

2. „$\Leftarrow$“: Sei $w \in L(\text{EA})$ mit $n \leq |w| \leq z_n$. Auf Grund des Pumping-Lemmas ist dann $xy^i z \in L(\text{EA})$.

   „$\Rightarrow$“: Sei $L(\text{EA})$ unendlich und es ex. *kein* Wort der Länge zwischen n und $2n - 1$. Dann hat w mindestens die Länge $2n$. Auf Grund des Pumping-Lemmas gilt aber, dass auch $yz \in L(\text{EA}) \Rightarrow$ Widerspruch!

□

Kapitel 7

Reguläre Grammatiken

Definition: (Grammatik)
Eine (allgemeine) Grammatik G ist eine Tupel $G = (N, T, P, S)$ mit

- N: endliche Menge von nichtterminalen Symbolen
- T: endliche Menge von terminalen Symbolen
- P: endliche Menge von Regeln (Produktionen)
- $S \in N$: Startsymbol

Die durch G erzeugte Sprache ist gegeben durch

$$L(G) = \left\{ w \in T^* \mid S \overset{P}{\Longrightarrow} w \right\}$$

Definition: (rechts-lineare Grammatiken)
Eine rechts-lineare Grammatik G ist eine allgemeine Grammatik mit den Einschränkungen, dass alle Regeln die folgende Form haben:

$$A \to wB$$
$$A \to w$$

wobei $A, B \in N, w \in T^*$.

Definition: (links-lineare Grammatiken)
Eine links-lineare Grammatik G ist eine allgemeine Grammatik mit den Einschränkungen, dass alle Regeln die folgende Form haben:

$$A \to Bw$$
$$A \to w$$

wobei $A, B \in N, w \in T^*$.

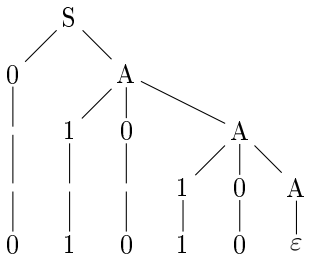
Bemerkung:
Eine rechts- oder links-lineare Grammatik wird auch reguläre Grammatik genannt.

Beispiel:
Sei die Sprache $L = 0(10)^*$ gegeben.

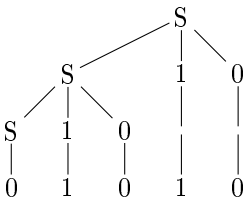
1. Dann wird L durch die rechtslineare Grammatik $G = (\{0,1\}, \{A, S\}, P, S\}$ mit

$$P = \{S \to 0A,\ A \to 10A,\ A \to \varepsilon\}$$

erzeugt. Der Ableitungsbaum von $w = 01010 \in L$ schaut so aus:



2. Eine linkslineare Grammatik ist gegeben durch $P = \{S \to S10,\ S \to 0\}$. Der dazu gehörige Ableitungsbaum von $w = 01010$:



7.1 Äquivalenz regulärer Grammatiken und endlicher Automaten

Satz:
Wenn L eine reguläre Grammatik besitzt, so ist L eine reguläre Menge.

Beweis:
Sei $L = L(G)$ einer rechts-linearen Grammatik $G = (T, N, P, S)$. Wir konstruieren uns einen ε-NEA $M = (Q, T, \delta, [S], [\varepsilon])$, der die Ableitungen von G simuliert:

Q ist gegeben aus den Symbolen $[a]$ mit

1. $a = S$
2. a ist ein (nicht notwendigerweise echte) rechte Seite einer Produktion aus P

δ ist gegeben durch:

1. $A \in N \Rightarrow \delta([A], \varepsilon) = \{[\alpha] \mid A \to \alpha \in P\}$
2. $a \in T, \alpha \in T^*N \to \delta([a\alpha], a) = \{[\alpha]\}$

Durch Induktion über eine Ableitungsfolge lässt sich zeigen:
Hier folgen nun leider einige Dinge, die nicht mehr zu entziffern waren und die auch im Nachhinein irgendwie keinen Sinn ergeben.

□

Bemerkung:
Spiegelt man eine links-lineare Grammatik, so erhält man eine rechts-lineare Grammatik und umgekehrt

Beispiel:

$$L_1 = 0(10)^*$$
$$\left\{\begin{matrix} S & \to S10 \\ S & \to 0 \end{matrix}\right\} \overset{\text{spiegeln}}{\Longrightarrow} \left\{\begin{matrix} S & \to 01S \\ S & \to 0 \end{matrix}\right\}$$
$$L_2 = (01)^*0$$

Dies ist leider ein Beispiel, wo beide Varianten ausnahmsweise identisch sind.

Satz:
Wenn L eine reguläre Menge ist, so wird L von einer links-linearen Grammatik und von einer rechts-linearen Grammatik erzeugt.

Beweis:
Sei $L = L(M)$ für einen DEA $M = (Q, \Sigma, \delta, q_0, F)$.

Fall 1: $q_0 \notin F \Rightarrow L = L(G)$ für die rechtslineare Grammatik $G = (Q, \Sigma, P, q_0)$ mit

$$P := \begin{cases} p \to aq & \text{falls } \delta(p,a) = q \\ p \to a & \text{falls } \delta(p,a) \in F \end{cases}$$

Dann gilt $\delta(p, w) = a \;\Leftrightarrow\; p \overset{x}{\Rightarrow} wa$, denn:

„$\Rightarrow$“: Falls also wa von M akzeptiert wird, $(wa \in L(M))$, dann existiert ein p mit $\delta(q_0, w) = p$ und $\delta(p, a) \in F$.

„$\Leftarrow$“: Gelte $q_0 \overset{x}{\Rightarrow} x$, so folgt $q_0 \overset{x}{\Rightarrow} wp \;\Rightarrow\; \delta(q_0, w) = p \wedge \delta(p, a) \in F$. Also ist $x \in L(M)$.

Also haben wir $L(M) = L(G) = L$.

Fall 2: $q_0 \in F \Rightarrow \varepsilon \in L(M)$. In diesem Fall erweitere die Grammatik G um das nichtterminale Symbol S und die Regeln $S \to q_0$ und $S \to \varepsilon$. □

Bemerkung:

Kapitel 8

Kellerautomaten

Definition: (nichtdeterministischer 1-Weg-Kellerautomat)
Ein nichtdeterministischer 1-Weg-Kellerautomat ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliche Menge von Eingabesymbolen
- Γ: endliche Menge von Kellersymbolen
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$: Übergangsfunktion
- $q_0 \in Q$: Startzustand
- $z_0 \in \Gamma$: Kellerstartsymbol
- $F \subseteq Q$: Menge von Endzuständen

Definition: (Konfiguration eines Kellerautomaten)

(i) Eine Konfiguration K_M eines Kellerautomaten M ist ein Tupel

$$(q, w, p) \in (Q \times \Sigma^* \times \Gamma^*)$$

(ii) Für eine Konfiguration M heißt

$$\vdash_M \subseteq (Q \times \Sigma^* \times \Gamma^*) \times (Q \times \Sigma^* \times \Gamma^*)$$

Konfigurationsübergang mit

$$[\ (q, aw, z\tilde{\gamma}) \vdash (p, w, \gamma'\tilde{\gamma})\ ] \ \Leftrightarrow\ [\ (p, \gamma') \in \delta(q, a, z)\ ]$$

Bemerkung:
$\vdash^+$ ist die transitive-, $\vdash^*$ die transitiv-reflexive Fortsetzung von $\vdash$.

Darstellungen:

1. Tabellenform:

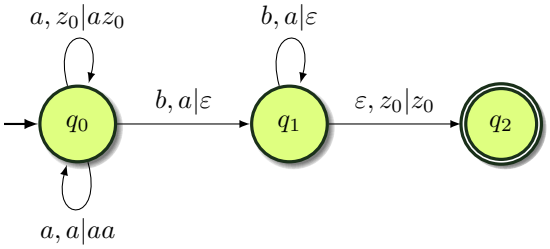
δ	(a_1, z_1)	$\cdots$	(ε, z_x)
q_0	$(p_1, \gamma_1)(p_2, \gamma_2)$	$\cdots$	
$\vdots$	$\vdots$	$\ddots$	
q_n			

2. Graphisch:
   - Knoten sind die Zustände
   - Kanten verbinden Quell und Zielzustand
   - Markierungen beinhalten <Eingabe>, <alter Keller> | < neuer Keller>

Beispiel:
Wir betrachten die folgende Tabelle:

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

Dies sollte äquivalent zum folgenden Kellerautomaten sein:



Hier ein Beispielkonfigurationsübergang:

$$(q_0, aabb, z_0) \vdash (q_0, abb, az_0) \vdash (q_0, bb, aaz_0) \vdash (q_1, b, az_0) \vdash (q_1, \varepsilon, z_0) \vdash (q_2, \varepsilon, z_0)$$

Definition: (Sprache von Kellerautomaten)

(i) Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ ein nichtdeterministischer 1-Weg-Kellerautomat. Dann heißt $L(M) := \{w \in \Sigma^* \exists p \in F, \gamma \in \Gamma^* \ : \ (q_o, w, z_o) \vdash_\mu^* (p, \varepsilon, \gamma)\}$ die von M unter erreichen eines Endzustandes akzeptierte Sprache.

(ii) $N(M) := \{w \in \Sigma^* \mid (q_0, w, z_0) \vdash^* (p, \varepsilon, \varepsilon)\}$ heißt die von M unter Leeren des Kellers akzeptierte Sprache.

Satz:
Seien M_1 und M_2 jeweils nichtdeterministische 1-Weg-Kellerautomaten.

(i) Für jede Sprache $L = L(M_2)$ exisitiert ein M_1 mit $L = N(M_1)$.

(ii) Für jede Sprache $L = N(M_1)$ exisitiert ein M_2 mit $L = L(M_2)$.

Beweis:

(i) Sei $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$. Seien weiter $x_0 \notin \Gamma$ ein neues Kellersymbol und $q_0', q_E \notin Q$ neue Zustäde. Dann ist M_1 gegeben durch:

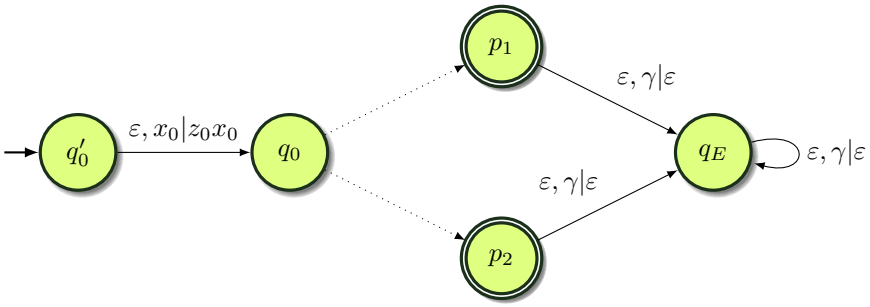
$$M_1 := (Q \cup \{q_0', q_E\}, \Sigma, \Gamma \cup \{x_0\}, \delta', q_0', x_0, \emptyset)$$

wobei $\delta' = \delta$ bis auf

$$\delta'(q_0', \varepsilon, x_0) = \{(q_0, z_0 x_0)\}$$

und für alle $p \in F, \gamma \in \Gamma$:

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



Nun gilt noch zu zeigen, dass auch tatsächlich $L(M_2) = N(M_1)$:
Sei also $w \in L(M_2)$, dann gilt:

$$(q_0', w, x_0) \vdash_{\mu_1} (q_0, w, z_0x_0) \vdash_{\mu_1\mu_2} (p, \varepsilon, \gamma, x_0) \vdash (q_E, \varepsilon, \gamma x_0) \vdash_{\mu_1}^* (q_E, \varepsilon, \varepsilon)$$

Also ist $w \in N(M_1)$. Die Rückrichtung verläuft analog also gilt wohl Gleichheit.

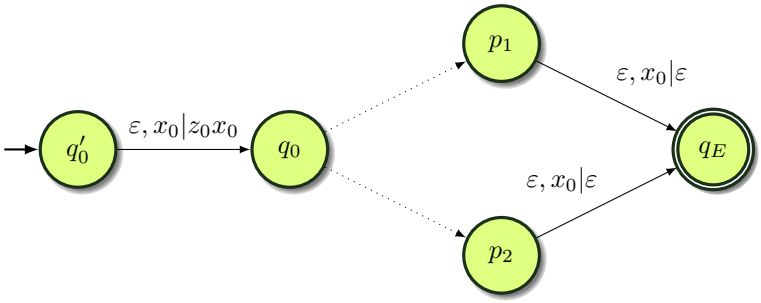
(ii) Sei $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, dann wählen wir uns M_2 durch

$$M_2 := (Q \cup \{q_0', q_E\}, \Sigma, \Gamma \cup \{x_0\}, \delta', q_0', x_0, \{q_E\})$$

wobei wieder δ' wie δ ist, bis auf:

$$\delta'(q_0', \varepsilon, x_0) = \{(q_0, z_0x_0)\}$$
$$\delta'(p, \varepsilon, x_0) = \{(q_E, \varepsilon)\}$$

mit $p \in \mathbb{Q}$.



Hier verzichteten wir nun auf den technischen Beweis der Gleichheit der beiden Sprachen

□

Definition: (deterministische 1-Weg Kellerautomaten)
Ein deterministischer 1-Weg-Kellerautomat ist bis auf das δ ein nichtdeterministischer 1-Weg-Kellerautomat. Es gilt:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$$

sowie für alle $q \in Q, z \in \Gamma$ mit $\delta(q, \varepsilon, z) \neq \emptyset$ muss gelten:

$$\forall a \in \Sigma : \delta(q, a, z) \uparrow \text{ (ist nicht definiert)}$$

Nach dieser Definition ist also unser Beispielautomat deterministisch.

Bemerkung:
$\text{Sprachen}_{EA} \subsetneq \text{Sprachen}_{\text{1-N-KA}}$.

Beweis:
Ist klar...

□

8.1 Kontextfreie Grammatiken

Definition: (Kontextfreie Grammatik)
Eine kontextfreie Grammatik G ist eine allgemeine Grammatik $G = (N, T, P, S)$ mit der Einschränkung, dass für alle Regeln $p \in P$ gilt: p hat die Form

$$A \to \alpha,\ A \in N, \alpha \in (N \cup T)^*$$

8.1.1 Normalformen

Satz: *Chomsky-Normalform*
Zu jeden kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ lässt sich effektiv eine äquivalente kontextfreie Grammatik G' erzeugen, bei der alle Produktionen aus P die Form

$$A \to BC,\ A, B, C \in N$$
$$A \to a,\ A \in N, a \in T$$

haben. Diese Art der Grammatik nennen wir Chomsky-Normalform.

Satz: *Greibach-Normalform*
Zu jeden kontextfreien Grammatik G mit $\varepsilon \notin L(G)$ lässt sich effektiv eine äquivalente kontextfreie Grammatik G' erzeugen, bei der alle Produktionen aus P die Form

$$A \to a\alpha,\ A \in N, a \in T, \alpha \in N^*$$

haben. Diese Art der Grammatik nennen wir Greibach-Normalform.

8.1.2 Pumping-Lemma für kontextfreie Sprachen

Satz: *Pumping-Lemma für kontextfreie Sprachen*
Sei L eine kontextfreie Sprache. Dann existiert eine von L unabhängige Konstante n, sodass für *jedes* Wort $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ existiert, wobei $|vx| \geq 1$ und $|vwx| \leq n$ gilt, sowie

$$\forall i \in \mathbb{N} : uv^i wx^i y \in L$$

Lemma:
Sei $L = L(G)$ mit G in Chomsky-Normalform. Exisitiert in einem Ableitungsbaum für ein Wort $w \in L(G)$ *kein* Pfad P mit der Eigenschaft $|P| > i$, dann folgt $|w| \leq 2^{i-1}$.

Beweis:
Die Grammatik ist in Chomsky-Normalform, also ist der Ableitungsbaum ein binärer Baum, wo jeder Knoten 2,1 oder 0 Nachfolger haben.
Auf Grund der Beschränkung der Tiefe, lässt sich die Anzahl der Blätter abschätzen.

Nun Induktion über i:
Induktionsanfang $i = 1$*:*

$$|w| = 1 = 2^0 = 2^{1-1}$$

Induktionsschritt: $i > 1$*:* Nach Induktionsvoraussetzung gilt die Behauptung bereits für die zwei Teilbäume T_1 und T_2 mit den Anfangsknoten A und B. Nun erweitern wir um die Regel $S \to AB$ und damit folgt

$$w = w_1 w_2, |P| = |P_{1,2}| + 1 \Rightarrow w \leq 2 \cdot 2^{i-2} = 2^{i-1}$$

□

Beweis: *(des Pumping Lemmas)*
Wir betrachten zunächst zwei Spezialfälle:
$L(G) = \emptyset$*:* Der Satz gilt nicht, da $L(G)$ keine Zeichenreihe enthält.

$L(G) = \varepsilon$*:* Der Satz gilt nicht, da $L(G)$ keine Zeichenreihe enthält mit $|z| \geq n > 0$.

Allgemeinfall: Sei $|N| = K$ und $n = 2^K$ und $z \in L(G)$ mit $|z| \geq n = 2^K$. Dann enthält der Ableitungsbaum für z einen Pfad P mit $|P| \geq K + 1$, es existieren also mindestens 2 Knoten mit dem gleichen Nichtterminalzeichen.

Sei P der längste Pfad, dann exisitieren in P zwei Knoten v_1, v_2 mit:

1. v_1, v_2 repräsentieren gleiches Nichtterminalsymbol A.
2. v_1 sei näher an der Wurzel als v_2
3. v_1, v_2 seien vom Blatt (a) aus gesehen die nächsten Knoten mit A.
4. Dann gilt die Länge des Pfades von v_1 bis zum Blatt a kleiner gleich $K + 1$ (es kann keine weiteren doppelten Nichtterminalsymbole geben).

Es gilt:

1. Sei z_1 die Front (*Anm. des Verf.:* Blätter?) von T_1
2. Sei z_2 die Front von T_2

[...]
Anm. des Verf.: Leider wurde dieser Beweis äußerst unstrukturiert rübergebracht. Für Interessierte empfehle ich die Lektüre des Logik 1 Skriptes von Prof. Pohlers Seite 96 ff. □

Beispiel:
Wir möchten durch Widerspruch zeigen, dass die Sprache $L = \{a^i b^i c^i \mid i \geq 1\}$ nicht kontextfrei ist:

Angenommen, L sei kontextfrei. Dann sei n die Konstante aus dem Pumping-Lemma. Sei $z = a^n b^n c^n \in L$ und $z = uvwxy$ eine beliebige Zerlegung, die die Randbedingungen des Pumping-Lemmas erfüllt. Aus $|vwx| \leq n$ folgt, dass vx nicht gleichzeitig a und c enthalten kann.

1. Fall: v und x bestehen nur aus a, dann folgt $uv^0wx^0y \notin L$, da uwy weniger a enthält als b und c.

2. Fall: v und x bestehen aus a und b, dann folgt $uv^0wx^0y \notin L$, da uwy weniger a und b enthält als c.

Restliche Fälle analog. Also kann L nicht kontextfrei sein.

8.2 Zusammenhänge

Bemerkung:
Kontextfreie Grammatiken sind äquivalent zu nichtdeterministischen 1-Weg-Kellerautomaten. Ein nichtdeterministischer 1-Weg-Kellerautomat akzeptiert duch Leeren des Kellers. Dazu die folgende Idee:

1. Ersetze z_0 (Startsymbol des Kellers) durch das Axiom S der Grammatik
2. zwei Möglichkeiten:
   (a) oberstes Kellersymbol $\in N \rightarrow$ wähle Produktion P, sodass linke Seite gleich oberstes Kellersymbol $\rightarrow$ ersetze oberstes Kellersymbol durch rechte Seite der Regel
   (b) oberstes Kellersymbol $\in T \rightarrow$ vergleiche oberstes Kellersymbol mit gelesenem Symbol auf der Eingabe $\rightarrow$ bei Übereinstimmung oberstes Kellersymbol löschen und auf der Eingabe einen Schritt weitergehen. Bei keiner Übereinstimmung Abbruch.
3. mit Schritt 2 weitermachen

Satz: *Automaten für kontextfreie Sprachen*
Sei L eine kontextfreie Sprache, dann lässt sich effektiv ein nichtdeterministischer 1-Weg-Kellerautomat M konstruieren mit $L = L(M)$.

Beweis:
Sei $G = (N, T, P, S)$ kontextfrei. Der äquivalente nichtdeterministische 1-Weg-Kellerautomat ist gegeben durch $M = (\{q\}, T, N \cup T, \delta, q, z_0, \emptyset\}$, mit:

$$\begin{aligned}\delta(q, \varepsilon, A) &= \{(q, \beta) \mid A \to \beta \in P\}, \quad A \in N, \beta \in N \cup T \\ \delta(q, a, a) &= \{(q, \varepsilon)\}, \quad a \in T \\ \delta(q, \varepsilon, z_0) &= \{(q, S)\}\end{aligned}$$

□

Satz: *(ohne Beweis)*
Ist $L(M) = L$ für einen nichtdeterministischen 1-Weg-Kellerautomat M, dann ist L kontextfrei.

8.3 Eigenschaften von kontextfreien Grammatiken

Satz: *Abschlusseigenschaften*
Die Menge der kontextfreien Sprachen sind unter Vereinigung, Konkatenation und Hüllenbildung abgeschlossen.

Beweis:
Seien L_1, L_2 kontextfreie Sprachen, die durch die Grammatiken $G_i = (N_i, T_i, P_i, S_i), i = 1, 2$ erzeugt werden. Ohne Einschränkung der Allgemeinheit können wir annehmen, dass $N_1 \cap N_2 = \emptyset$ gilt und wir unsere neuen Startsymbole $S_3, S_4, S_5 \notin N_1 \cap N_2$ definieren können.

Vereinigung:

$$\begin{aligned}G_3 &= (N_1 \cup N_2 \cup \{S_3\}, T_1 \cup T_2, P_3, S_3) \\ P_3 &= P_1 \cup P_2 \cup \{S_3 \to S_1, S_3 \to S_2\}\end{aligned}$$

Konkatenation:

$$\begin{aligned}G_4 &= (N_1 \cup N_2 \cup \{S_4\}, T_1 \cup T_2, P_4, S_4) \\ P_4 &= P_1 \cup P_2 \cup \{S_4 \to S_1 S_2\}\end{aligned}$$

Hüllenbildung:

$$\begin{aligned}G_5 &= (N_1 \cup \{S_5\}, T_1, P_5, S_5) \\ P_5 &= P_1 \cup \{S_5 \to S_1 S_5, S_5 \to \varepsilon\}\end{aligned}$$

□

sonstige Eigenschaften:

	nichtdeterministisch	**deterministisch**
Substitution	abgeschlossen	abgeschlossen
Homomorphismus	abgeschlossen	nicht abgeschlossen
Inverser Homomorphismus	abgeschlossen	abgeschlossen
Schnitt	nicht abgeschlossen	
Komplement	nicht abgeschlossen	
Schnitt mit regulären Mengen	abgeschlossen	abgeschlossen
Vereinigung, Konkatenation	abgeschlossen	nicht abgeschlossen

Beispiel:
$a^n b^n$ ist kontextfrei und deterministisch.
$a^n b^{2n}$ ist kontextfrei und deterministisch.
$a^n b^n \cup a^n b^{2n}$ ist kontextfrei aber nicht deterministisch

8.4 Vereinfachung von kontextfreien Grammatiken

Definition: (nützliche Symbole einer Grammatik)
Ein Symbol $X \in N \cup T$ einer Grammatik $G = (N, T, P, S)$ heißt nützlich, wenn es eine Ableitung der Form

$$S \overset{\pi}{\Rightarrow} \alpha X \beta, \quad X \overset{*}{\Rightarrow} w, w \in T^*$$

gibt. Nutzlose Symbole besitzen *keinen* Einfluss auf die Sprache.

Bemerkung:
Nutzlose Symbole können dann wohl entfernt werden.

Definition: (erzeugende Symbole)
Ein Symbol X heißt erzeugend, wenn es eine Zeichenreihe $w \in T^*$ gibt, mit $X \overset{*}{\Rightarrow} w$. Das heißt aber noch nicht, dass w Bestandteil der Sprache ist!

Algorithmus: *Konstruktion der Menge der erzeugenden Symbole*
Induktion:

(i) Jedes Symbol aus T ist erzeugend.

(ii) Falls $A \to \varepsilon \in P$, dann ist A erzeugend

(iii) Gegeben sei eine Produktion $A \to \alpha_1 \cdots \alpha_n$, wobei jedes der α_i erzeugend sei. Dann ist auch das A erzeugend.

Definition: (erreichbare Symbole)
Ein Symbol $X \in N \cup T$ heißt erreichbar, falls $\alpha, \beta \in (N \cup T)^*$ existieren, sodass es eine Ableitung $S \overset{*}{\Rightarrow} \alpha X \beta$ gibt.

Algorithmus: *Konstruktion der Menge der erreichbaren Symbole*
Induktion:

(i) S ist erreichbar

(ii) Sei eine Produktion $A \to \alpha_1 \cdot \alpha_n$ gegeben und A erreichbar. Dann sind auch die α_i erreichbar.

Beispiel:
Sei die Grammatik $G = (N, T, \{S \to AB | a, A \to b\}, S)$ gegeben.

Nach Regel 1 sind a, b erzeugend. Nach Regel 3 sind auch A, S erzeugend. Also sind die erzeugenden Symbole $\{a, b, A, S\}$.

Nach Regel 1 ist S erreichbar. Nach Regel 2 sind auch A, B, a erreichbar. Nach erneuter Anwendung von Regel 2 ist auch b erreichbar. Also sind die 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 $G' = (N', T', P', S)$ mit $L(G) = L(G')$ erzeugen, die keine nutzlosen Symbole enthält.

Algorithmus:

Schritt 1:

(i) Bestimme alle Symbole, die nichts erzeugen und eliminiere sie

(ii) Eliminiere alle Produktionen, die mindestens ein nichterzeugendes Symbol enthalten

Wir erhalten so eine Grammatik $G_1 = (N_1, T_1, P_1, S)$.

Schritt 2:

Eliminiere alle Symbole, die in G_1 nicht erreichbar sind (einschließlich zugehöriger Produktionen) und erhalte so G'.

Beispiel:
Gleiches Beispiel von oben, die erzeugnenden Symbole sind $\{a, b, A, S\}$. Also wird nach 1.1 B eliminiert. Nach 1.2 wird die Regel $S \to AB$ eliminiert. Also ist $P_1 = \{S \to a, A \to b\}$.

In G_1 sind nun noch S und a erreichbar, also wird in Schritt 2 noch A, $A \to b$ und b eliminiert.

Bemerkung:
Ganz wichtig ist das Einhalten der richtigen Reihenfolge dieser beiden Schritte!!! (sic!) Sonst klappt das Verfahren nicht.

8.5 Schlussbemerkung

Bemerkung:
Die Kellerautomaten sind alle unterschiedlich:

- 1-ND-KA $\approx$ kontextfreie Sprachen $\approx$ kontextfreie Grammatiken
- Der 1-D-KA kann sowohl $a^n b^n$ als auch $a^n b^{2n}$ erkennen aber nicht deren Vereinigung
- 2-ND-KA und 2-D-KA können auch $a^n b^n c^n$ erkennen. Es ist aber nach wie vor ein ungelöstes Problem wie die Sprachen der 1-Weg Kellerautomaten und der 2-Weg-Kellerautomaten zueinander stehen.

Kapitel 9

Turingmaschinen

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

1. Das Symbol im Speicher verändern (ersetzen)
2. Den Schreib-/Lesekopf für den Speicher um eine Position in eine beliebige Richtung bewegen
3. Der Lesekopf auf der Eingabe um eine Position weiterschieben
4. Den Zustand der Kontrolleinheit ändern

Definition: (Turingmaschine)
Eine Turingmaschine TM ist ein Tupel $TM = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit:

- Q: endliche Menge von Zuständen
- Σ: endliches Eingabealphabet
- Γ: endliches Bandalphabet
- $\delta : Q \times \Gamma \times \Sigma \to Q \times \Gamma \times \{l, r, n\}$: Übergangsfunktion
- $q_0 \in Q$: Startzustand
- $B \in \Gamma - \Sigma$: Blanksymbol
- $F \subseteq Q$: Menge von Endzuständen

Die Startkonfiguration ist gegeben durch den Startzustand q_0 in der Kontrolleinheit gegeben und die Schreib-/Leseköpfe stehen jeweils auf dem untersten Element ihrer Bänder. Das Speicherband ist komplett mit Blanks initialisiert.

Variationen:

Variante 1:
Eine Turingmaschine ohne separate Eingabe ist ein Tupel $TM = (\cdots)$ mit

- $\Sigma \subseteq \Gamma - \{B\}$
- $\delta Q \times \Gamma \to Q \times \Gamma \times \{l, r\}$

Die Startkonfiguration ist gegeben durch das Speicherband was mit der Eingabe intialisiert ist. Der Schreib-/Lesekopf ist am Fuß der Eingabe positioniert.

Variante 2:
Auf das Sondersymbol B wird verzichtet werden.

Satz: *(ohne Beweis)*
Jede Turingmaschine mit speratem Eingabeband läßt sich durch eine Turingmaschine ohne Eingabeband (Variante 1) simulieren.

Notation:

Zustandsbeschreibung:
Wir beschreiben ab sofort den Zustand von Turingmaschinen durch

$$\alpha_1\ q\ \alpha_2$$

Dabei sind α_1, α_2 die relevanten Inhalte des Bandes vor und hinter dem Zeichen auf dem der Schreib-/Lesekopf gerade steht. Per Definition soll das am weitesten links stehende Symbol von α_2 gerade gelesen werden (sic!). Also steht der Schreib-/Lesekopf stets auf dem ersten Zeichen von α_2.

Zustandsübergang:
Sei $X_1X_2\cdots X_{i-1}\ q\ X_i\cdots X_n$ gegeben und $\delta(q, X_i) = (p, Y, l)$, dann gilt

$$X_1X_2\cdots X_{i-1}\ q\ X_i\cdots X_n \quad \vdash_M \quad X_1X_2\cdots X_{i-2}\ p\ X_{i-1}YX_{i+1}\cdots X_n$$

transitiv reflexive Hülle:
$\vdash_M^*$ ist wie üblich die transitiv reflexive Hülle von $\vdash_M$

Definition: (akzeptierte Sprache von Turingmaschinen)
Die von einer Turingmaschine TM akzeptierte Sprache $L(TM)$ ist gegeben durch

$$L(TM) = \{w \mid w \in \Sigma^* \wedge q_0w \vdash_{TM}^* \alpha_1p\alpha_2 \wedge p \in F, \alpha_1, \alpha_2 \in \Gamma^*\}$$

9.1 Erweiterungen von Turingmaschinen

In diesem Abschnitt werden wir zeigen, dass sämtliche denkbaren Erweiterungen von Turingmaschinen auch auf der Ur-Turingmaschine zu simulieren sind und somit zu ihr äquivalent sind.

9.1.1 Mehrspurige Turingmaschinen

Die Turingmaschine kann mit ihrem Schreib-/Lesekopf mehrere Spuren auf einem Band gleichzeitig adressieren.

Satz:
Jede n-spurige Turingmaschine (eine Turingmaschine, die mit einem Schreib-/Lesekopf n Spuren auf einem Band gleichzeitig adressiert) lässt sich auf einer normalen Turingmaschine simulieren.

Beweis:
Wir erweitern unser Speicherbandalphabet um alle möglichen Kombinationen der Symbole der n Spuren:

$$\Gamma' = \{(\alpha_1, \ldots, \alpha_n) \mid \alpha_i \in \Gamma\}$$

□

9.1.2 Turingmaschine mit beidseitig unendlichem Speicherband

Das Speicherband der Turingmaschine ist nicht nur in eine Richtung unbegrenzt, sondern in beide Richtungen unbegrenzt beschreibbar.

Satz:
Jede Turingmaschine M mit einem beidseitig unendlichem Speicherband kann durch eine Turing-Maschine M' auf einem nur einseitig unbegrenztem Speicherband simuliert werden.

Beweis:
Wir erweitern das Bandalphabet von M' um ein eindeutig identifizierbares Trennelement \$ und erweitern das Band von M auf zwei Spuren. So arbeitet M' auf einem Band mit zwei Spuren, wobei jeweils nur eine relevant ist.
Die Information welches der beiden Bänder gerade relevant ist, wird im Zustand vermerkt. Jedes Mal wenn der Schreib-/Lesekopf über das Trennzeichen läuft, wechselt sich die Relevanz. □

9.1.3 Mehrbändige Turingmaschinen

Die Turingmaschine mit n Bändern kann mit n Schreib-/Leseköpfen n unterschiedliche Bänder unabhängig voneinander adressieren. In Abhängigkeit vom Zustand q und den n gelesenen Symbolen kann die Turingmaschine M:

1. Den Zustand ändern
2. Die gelesenen Bandsymbole ändern
3. Jeden Kopf unabhängig von einander um eine Position bewegen

Satz:
Jede mehrbändige Turingmaschine M kann durch eine n-bändige Turingmaschine M' simuliert werden.

Beweis:
M' ist eine Turingmaschine mit einem Band mit $2n$ Spuren. Jedes Band von M wird in zwei Spuren von M' gespeichert: Eine Spur speichert den Inhalt, die zweite Spur ist komplett leer und ist lediglich an der aktuellen Schreib-/Lesekopfposition von M markiert.
Mit etwas Arbeit kann nun M auf M' simuliert werden. □

9.1.4 Nichtdeterministische Turingmaschinen

Nichtdeterministische Turingmaschinen können für jeden Zustand und jedes Bandsymbol eine feste (endliche) Anzahl von möglichen Alternativen für die Übergangsfunktion δ haben.

Satz:
Jede nichtdeterministische Turingmaschine M kann durch eine deterministische Turingmaschine M' simuliert werden.

Beweis:
Sei r die Maximalanzahl der Alternativen. Dann kann jede Berechnungsfolge durch eine Folge von Ziffern aus $\{1, \ldots, r\}$ dargestellt werden, *aber* nicht jede derartige Ziffernfolge muss eine gültige Berechnungsfolge darstellen.

Nun schauen wir uns die Konstruktion von M' an: M' hat 3 Bänder:

- Band 1 enthält die Eingabe
- M' generiert auf Band 2 systematisch Ziffernfolgen aus $1, \ldots, r$ (kürzeste Folge zuerst).
  Also: $1, 2, \ldots, r, 1|1, 1|2, \ldots, 1|r, 2|1, 2|2, \ldots$
- Band 3 ist das Arbeitsband

M' kopiert zunächst die Eingabe von Band 1 auf Band 3 und arbeitet dann gemäß der Ziffernfolge von Band 2. Ist die Eingabe auf Band 3 abgearbeitet, beginnt M' von vorn: Kopiert also die Eingabe von Band 1 auf Band 3 und generiert die nächste Ziffernfolge auf Band 2. Erreicht M' irgendwann einen Endzustand, so ist M' natürlich fertig und bricht ab.
Existiert keine Regel gemäß der Ziffernfolge, so macht M' natürlich gleich von vorn mit der nächsten Ziffernfolge weiter.

Dies ist deterministisch, da das r fest ist und vorher feststeht. □

9.2 Church'sche These

Alle Varianten der Turingmaschinen bringen keine neuen Möglichkeiten. Ab 1930 haben sich viele verschiedene Menschen Gedanken über den Begriff „Berechenbarkeit“ gemacht. Dabei sind dann Sachen wie Turingmaschinen, Registermaschinen, das λ-Kalkül, die Theorie der rekursiven Funktionen, die kombinatorische Logik, Post-Systeme und vieles mehr entwickelt worden.

Irgendwann haben sie festgestellt, dass alle diese Systeme die gleichte Mächtigkeit besitzen.

Church'sche These:
Alle „intuitiv berechenbaren Funktionen“ sind Turing-berechenbar.

9.3 Klassen von Sprachen

Definition: (rekursive aufzählbare Sprache)
Die Sprache, die von einer (beliebigen) Turingmaschine erkannt wird, heißt *rekursiv aufzählbar*.

Bemerkung:
Eine Turingmaschine kann durchaus bei einer Eingabe *nicht* anhalten.

Definition: (rekursive Sprache)
Eine Sprache heißt *rekursiv*, wenn es mindestens eine Turingmaschine gibt, die die Sprache akzeptiert und bei jeder beliebigen Eingabe anhält.

Bemerkung:
Die Klasse der rekursiven Sprache ist eine Unterklasse der rekursiv aufzählbaren Sprachen.

9.3.1 Eigenschaften

Satz: *Vereinigung von Sprachenklassen*

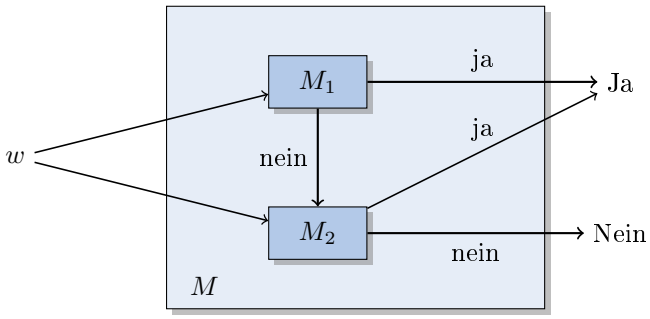
(i) Die Vereinigung zweier rekursiver Sprachen ist rekursiv.

(ii) Die Vereinigung zweier rekursiv aufzählbaren Sprachen ist rekursiv aufzählbar.

Beweis:

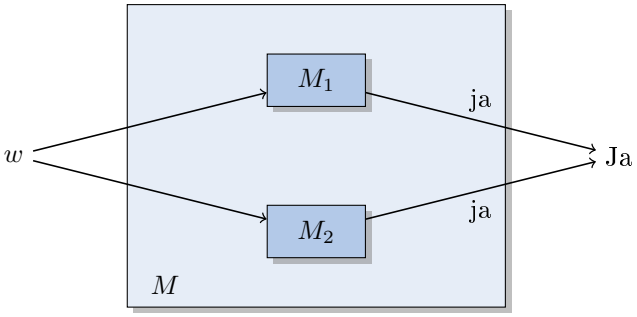
(i) Seien L_1, L_2 rekursive Sprachen und M_1, M_2 Turingmaschinen die für jede beliebige Eingabe halten und für die $L(M_1) = L_1$ und $L(M_2) = L_2$ gilt. Wir konstruieren eine dritte Turingmaschine M mit den Eigenschaften:

1. M simuliert zunächst M_1.
2. akzeptiert M_1 die Eingabe, dann auch M.
3. falls nicht, dann simuliert M die Maschine M_2
   akzeptiert M_2 die Eingabe, dann auch M, falls nicht, dann nicht.



(ii) Seien L_1, L_2 rekursiv aufzählbare Sprachen und M_1, M_2 Turingmaschinen für die $L(M_1) = L_1$ und $L(M_2) = L_2$ gilt. Wir konstruieren eine dritte Turingmaschine M mit den Eigenschaften:

1. M simuliert parallel M_1 und M_2
2. falls eine der beiden Maschinen die Eingabe akzeptiert, dann akzeptiert auch M die Eingabe

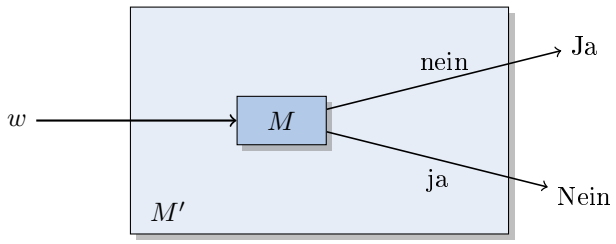


□

Satz: *Komplement rekursiver Sprachen*
Das Komplement einer rekursiven Sprache ist rekursiv.

Beweis:
Sei M eine Turingmaschine, die bei jeder Eingabe hält mit rekursiver Sprache $L = L(M)$. Dann konstruieren wir eine Turingmaschine M' mit den 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.



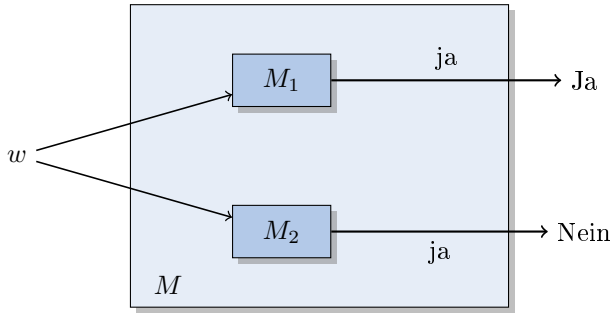
□

Satz:
Wenn eine Sprache L und ihr Komplement $\overline{L}$ beide rekursiv aufzählbar sind, dann ist L und somit auch $\overline{L}$ rekursiv.

Beweis:
Seien M_1, M_2 Turingmaschinen mit $L(M_1) = L_1$ und $L(M_2) = \overline{L}$. Wir konstruieren eine Maschine M mit den Eigenschaften

1. M akzeptiert die Eingabe, wenn sie von M_1 akzeptiert wird.

2. M stoppt ohne zu akzeptieren, wenn die Eingabe von M_2 akzeptiert wird.



□

Korollar: *Konsequenz aus den Sätzen*
Seien L und $\overline{L}$ ein Paar von komplementären Sprachen. Dann gilt eine der folgenden Aussagen:

(i) L und $\overline{L}$ sind rekursiv.

(ii) Weder L noch $\overline{L}$ sind rekursiv aufzählbar.

(iii) Entweder L oder $\overline{L}$ sind rekursiv aufzählbar, die jeweils andere ist nicht rekursiv aufzählbar.

9.4 nicht turingberechenbare Funktionen

Ziel:
Wir wollen eine Funktion konstruieren, die nicht auf einer Turingmaschine berechenbar ist.

1. Schritt:
Wieviele Turingmaschinen gibt es auf dieser Welt.

Vorbereitungen:
Ohne Beschränkung der Allgemeinheit gilt: $\Sigma = \{0, 1, B\}, \Gamma = \{0, 1\}, Q = \{q_1, \ldots, q_n\}$ und unsere Turingmaschine ist gegeben durch:

$$TM = (Q, \Sigma, \Gamma, \delta, q_1, B, q_2)$$

Zudem führen wir die folgenden Abkürzungen (Nummerierungen) ein:

$$0 = X_1, 1 = X_2, B = X_3, L = R_1, R = R_2$$

Gödelisierung:
Wir kodieren nun jede Regel der Form

$$\delta(q_i, X_j) = (q_k, X_l, R_m)$$

durch

$$0^i\ 1\ 0^j\ 1\ 0^k\ 1\ 0^l\ 1\ 0^m$$

Offensichtlich tritt in einer Regel nie 11 auf, wir können also 11 als Regeltrenner nutzen. Somit können wir die Menge δ aller Regeln kodieren durch die binäre Ziffernfolge

$$\underbrace{111}_{\text{Anfang}}\ [\text{Regel } 1]\ 11\ [\text{Regel } 2]\ 11\ \cdots\ 11\ [\text{Regel } n]\ \underbrace{111}_{\text{Ende}}$$

Dabei ist die Reihenfolge der Regeln irrelevant, also kann ein und die selbe Turingmaschine durch unterschiedliche Ziffernfolgen codiert werden.

Jede binäre Zeichenreihe beschreibt eine ganze Zahl, also kann jeder Turingmaschine mindestens eine ganze Zahl zugeordnet werden, aber es gibt keine ganze Zahl die zwei unterschiedliche Turingmaschinen repräsentiert (linkseindeutigkeit).

Mächtigkeit der ganzen Zahlen: abzählbar unendlich.
Mächtigkeit der Funktionen: überabzählbar unendlich.

Also muss es Funktionen geben, die *nicht* auf einer Turingmaschine berechenbar sind.

Effektive Konstruktion einer nicht rekursiv aufzählbaren Sprache:
Wir wollen eine Sprache konstruieren, die nicht turing-berechenbar ist, also nicht rekursiv aufzählbar. Methode: Diagonalisierungsverfahren.

Sei die (unendliche) Liste aller Zeichenreihen (Worte) über $(0+1)^*$ in kanonischer Odnung

$$0, 1, 00, 01, 10, 11, 000, 001, \ldots$$

Sei ferner w_i das i-te Wort in dieser Liste und M_j die Turingmaschine, deren Codierung die ganze Zahl j in Binärdarstellung ist.
Bilde eine Matrix $A = (a_{ij})_{i=1,2,\ldots j=1,2,\ldots}$ mit den Einträgen

$$a_{ij} = \begin{cases} 0 & \text{falls } w_i \notin L(M_j) \\ 1 & \text{falls } w_i \in L(M_j) \end{cases}$$

Betrachte nun die Diagonale: Die Sprache L_D ist folgendermaßen definiert. L_D ist die Menge der Zeichenreihen w_i, sodass $w_i \notin L(M_i)$. Es gilt

$$w_i \in L_D \Leftrightarrow a_{ii} = 0$$

Angenommen: L_D wird von der Turingmaschine M_j akzeptiert

1. $\Rightarrow w_j \in L_D = L(M_j) \Rightarrow a_{jj} = 0 \Rightarrow w_j \notin L(M_j) \Rightarrow$ Widerspruch!
2. $\Rightarrow w_j \notin L_D = L(M_j) \Rightarrow a_{jj} = 1 \Rightarrow w_j \in L(M_j) \Rightarrow$ Widerspruch!

w_j ist also weder in L_D noch nicht in L_D, also muss die Annahme (für beliebiges j) falsch sein. Es gibt also keine Turingmaschine in A, die L_D akzeptiert, also kann L_D nicht rekursiv aufzählbar sein.

9.5 Nicht eingeschränkte Grammatiken

Definition: (nicht eingeschränkte Grammatiken)
Eine Grammatik $G = (N, T, P, S)$, bei der die Regeln aus P die Form

$$\alpha \to \beta, \alpha, \beta \in (N \cup T)^*, \alpha \neq \varepsilon$$

haben, heißen nicht eingeschränkte Grammatiken. Man nennt sie auch „Typ-0-Grammatiken“, „allgemeine Grammatiken“ oder „Semi-Thue-Systeme“.

Satz:
Wenn $L = L(M)$ die Sprache einer Typ-0-Grammatik G ist, dann ist L eine rekursiv aufzählbare Sprache.

Beweis:
Wir konstruieren eine nichtdeterministische zweibändige Turingmaschine, die L akzeptiert:

- Das erste Band speichert die Eingabe und wird nicht verändert.
- Das zweite Band ist das Arbeitsband und speichert die Satzformen (alle möglichen Zeichenreihen, die aus $(N \cup T)$ gebildet werden und von S aus abgeleitet werden können) α von G.

Die Turingmaschine simuliert Ableitungen:

Initialisierung: Die Turingmaschine speichert S auf Band 2.

Rekursives Vorgehen:

1. Wähle beliebige Position i in α (α ist Zeichenreihe *ohne* B).
2. Wähle nicht deterministisch eine Produktion $\beta \to \gamma \in P$ aus.
3. Überprüfe, ob ab der Position i die Zeichenreihe β in α vorkommt.
   Falls ja, ersetze β durch γ (ggf. durch „Platz machen“ bzw. „zusammenschieben“).
4. Vergleiche den Inhalt von Band 2 mit dem von Band 1.
   Sind die Inhalte identisch, dann akzeptiere, falls nicht, wähle neues i, neue Produktion oder starte neu.

Anm. des Verf.: Bei Schritt 1 währe es evtl. sinnvoller die verschiedenen möglichen i systematisch durchzugehen, anstatt jedes Mal zu raten. Nach dem Gesetz der großen Zahlen schreibt zwar auch ein Affe mal Shakespeare, aber irgendwie finde ich Determinismus bei Algorithmen schöner. □

Satz: *(ohne Beweis)*
Sei L eine rekursiv aufzählbare Sprache, so gilt $L = L(G)$ für eine Typ-0-Grammatik G.

9.6 Zusammenfassung

Hierarchie der verschiedenen Sprachenklassen:

$$L(\text{EA}) \subset L(\text{1-ND-KA}) \subset L(\text{TM})$$

9.7 Linear beschränkte Automaten

Definition: (linear beschränkter Automat)
Ein linear beschränkter Automat (LBA) ist eine nichtdeterministische Turingmaschine mit den beiden Bedingungen:

(i) Das Eingabealphabet enthält zwei Sonderzeichen: $¢, \$$, welche den linken und rechten Rand markieren.

(ii) Der LBA kann sich nicht über die Randmarkierungen hinwegbewegen bzw. sie überschreiten.

Ein LBA ist also ein Tupel $LBA = (Q, \Sigma, \Gamma, \delta, q_0, F, ¢, \$), ¢, \$ \in \Sigma$. Die Sprache eines LBA ist gegeben durch

$$L(LBA) = \{w \mid w \in (\Sigma - \{¢, \$\})^* \ \wedge \ q_0 ¢\, w \$ \vdash^* \alpha q \beta, q \in F\}$$

Bemerkung:
Jeder LBA gemäß der obigen Definition, kann einen LBA simulieren, bei dem der Arbeitsbereich auf $K \cdot n$ beschränkt ist, wobei n die Länge der Eingabe ist.

9.8 Kontextsensitive Sprachen / Grammatiken

Definition: (kontextsensitive Grammatik)
Eine kontextsensitive Grammatik KsG ist eine allgemeine Grammatik $KsG = (N, T, P, S)$, wobei für die Produktionen aus P gilt:

$$\alpha \to \beta,\ \alpha, \beta \in (N \cup T)^*, |\alpha| \leq |\beta|$$

Die davon erzeugte Sprache heißt kontextsensitive Sprache.

Satz: *Normalformen*
Jede kontextsensitive Grammatik G lässt sich durch eine kontextsensitive Grammatik G' beschreiben, bei der alle Produktionen die Form

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2,\ \beta \neq \varepsilon, a \in N, \alpha \in (N \cup T)^*, \beta \in (N \cup T)^+$$

Satz:
Ist L eine kontextsensitive Sprache, dann wird L von einem LBA erkannt.

Beweis:
Analog zum Beweis der Äquivalenz von rekursiv aufzählbaren Sprachen zu Turingmaschinen:

Wir konstruieren unseren LBA mit 2 Spuren:

1. Spur 1 speichert die Eingabe
2. Spur 2 ist das Arbeitsband

Spur 1 wird mit $\not\in w_1, \ldots, w_n \not\$$ initialisiert, Spur 2 wird mit $\not\in SB \cdots B \not\$$ mit $|SB \cdots B| = n$ initialisiert. Anschließend wird auf Spur 2 sukzessive:

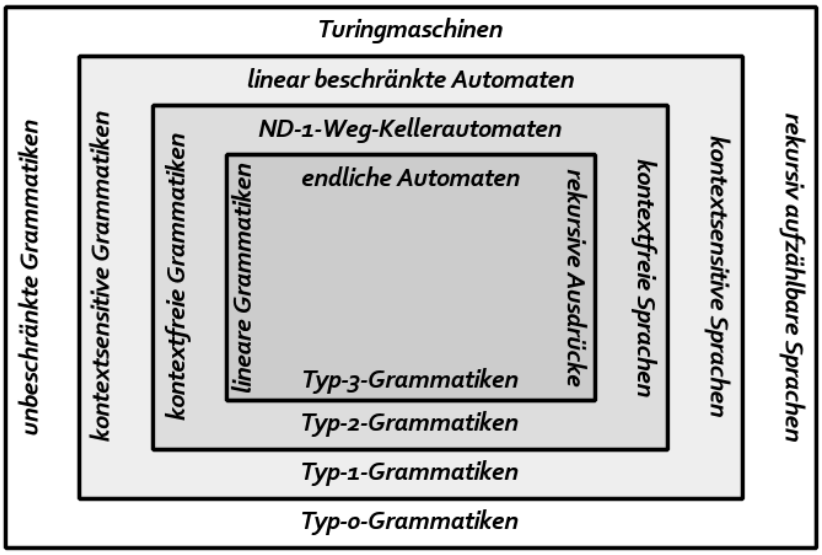
1. eine beliebige (*Anm. d. Verf.:* Mit „beliebig“ meint Prof. Lippe nicht wirrkürlich, sondern eher „systematisch nichtdeterministisch“) Position gesucht
2. auf dieser Position eine *mögliche* Regel (Ableitung) angewandt
3. musste hierzu $\not\$$ überschritten werden, so bleibt der LBA (*Anm. d. Verf.:* die nichtdet. Instanz des LBA?) stehen ohne zu akzeptieren

Damit folgt, das der LBA $w_1, \ldots, w_n$ akzeptiert, wenn es eine Ableitung $S \to^* w$ gibt, sodass keine Zwischensatzform länger als w ist.

Andererseits, kann keine es Ableitung $S \to^* \alpha \to^x w$ geben mit $|\alpha| > |\beta|$. □

Bemerkung:
Jede kontextfreie Grammatik ist auch kontextsensitiv (mit leerem Kontext).

9.9 Chomsky Hierarchie



Bemerkung:
Die deterministischen 1-Weg-Kellerautomaten liegen noch echt zwischen den Typ-3 und Typ-2 Grammatiken, gehören aber nicht zur „klassischen“ Chomsky-Hierarchie.

Kapitel 10

Weitere Automatenmodelle

10.1 Stapelautomaten

Notation:
Kellerautomat $\approx$ Pushdown Automaton
Stapelautomat $\approx$ Stack Automaton

Definition: (Stapelautomat)
Ein Stapelautomat (SA) ist ein Kellerautomat KA, mit den zusätzlichen Eigenschaften:

(i) Die Eingabe ist eine 2-Weg-Eingabe mit Anfangs- und Endmarkierung

(ii) Der Schreib-/Lesekopf kann zusätzlich zu den Schreib- und Löschoperationen am oberen Ende Leseoperationen im gesamten Keller durchführen.

Das Übergangsverhalten ist bestimmt durch den aktuellen Zustand q, dem gelesenen Symbol auf der Eingabe und dem gelesenen Symbol im Stack. In *einem* Schritt kann der SA:

(i) den Zustand ändern

(ii) den Lesekopf der Eingabe um ein Symbol (beide Richtungen) bewegen

(iii) auf dem Stack eine der folgenden Operationen vornehmen:
  (a) Hinzufügen eines Symbols auf dem oberen Rand des Stacks (*push*)
  (b) Entfernen des obersten Symbols des Stacks (*pop*)
  (c) Bewegen um eine Position (beliebige Richtung) ohne ein Symbol zu verändern, zu löschen oder hinzuzufügen.

Bemerkung:
Die Operationen (iii.a) und (iii.b) dürfen nur durchgeführt werden, wenn der Schreib-/Lesekopf an der richtigen (= obersten) Stelle steht.

Variationen:
Mögliche Variationen des Stapelautomaten sind:

- deterministischer SA
- nichtdeterministiescher SA
- Nur-Löschen SA
- 1-Weg SA

Beispiel:
Sei $L = \{0^n 1^n 2^n \mid n \geq 1\}$. Der diese Sprache akzetierende Stapelautomat:

1. schreibt zunächst sämtliche Nullen in den Keller
2. geht dann für jede Eins die er liest einen Eintrag im Keller nach unten

3. wenn er die erste 2 liest, muss er am Boden vom Keller angekommen sein
4. dann geht er wieder ganz nach vorne und macht mit Schritt 2 für die Zweien weiter.

Offensichtlich muss nicht gelöscht werden, L kann also durch einen 1-NL-D-SA akzeptiert werden.

Kapitel 11

Zusammenhang zwischen kontextfreien Grammatiken und Programmiersprachen

Kontextfreie Grammatiken finden Anwendung bei der Definition der Syntax von Programmiersprachen. Generell liegt jedoch der Aufwand für die Syntaxanalyse eines Programmes der Länge n bei $\mathcal{O}(n^3)$. In der Praxis beschränkt man sich daher auf determistische Grammatiken, die nur noch den Aufwand $\mathcal{O}(n)$ haben.

Definition: (unmittelbar reduzierbar/ableitbar)
Sei $u = \alpha_1\alpha_2\alpha_3 \in (N \cup T)^*, X \in N$ und $v = \alpha_1 X \alpha_3$. Sei ferner $(X, \alpha_2) \in P$, also $X \to_P \alpha_2$. Dann sagt man „u ist unmittelbar reduzierbar auf v“ bzw. „u ist unmittelbar aus v ableitbar“.

Notation: $v \Rightarrow u$ oder $(v \to u)$.

Bei der Syntaxanalyse muss nun überprüft werden, ob eine Zeichenreihe (Programm) auf das Axiom S reduzierbar ist, bzw. ob umgekehrt die Zeichenreihe aus S abgeleitet werden kann.

Definition: (Satzform)
Die Zeichenreihen $v \in (N \cup T)^*$ mit $S \Rightarrow^* v$ heißen *Satzformen*.

Jedes Element der Sprache ist eine Satzform, aber nicht jede Satzform ist ein Element von $L(G)$.

Definition: (Reduktionsschritt, Reduktionsfolge)
Ein *Reduktionsschritt* ist ein Tripel $(\alpha_1, X \to \alpha_2, \alpha_3)$ mit $\alpha_1, \alpha_2, \alpha_3 \in (N \cup T)^*, X \in N, (X, \alpha_1) \in P$.

Eine *Reduktionsfolge* von u nach v ist eine nicht leere Folge von Reduktionsschritten

$$(\alpha_1^0, X^0 \to \alpha_2^0, \alpha_3^0), \ldots, (\alpha_1^{n-1}, X^{n-1} \to \alpha_2^{n-1}, \alpha_3^{n-1}), n \geq 1$$

mit

$$u = u_0 = \alpha_1^0\alpha_2^0\alpha_3^0,\ v = u_n = \alpha_1^{n-1}X^{n-1}\alpha_3^{n-1}$$

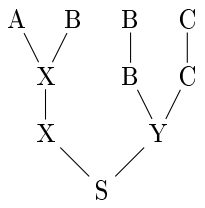
Reduktionsfolgen beschreiben Reduktionsbäume.

Beispiel:
Wir betrachten die Grammatik:

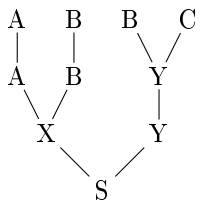
$$N := \{S, X, Y\}, T := \{A, B, C\}, P := \{S \to XY,\ Y \to BC,\ X \to AB\}$$

Für $ABBC \in L(G)$ gibt es die folgenden Reduktionsbäume:

R_1:



R_2:



Offensichtlich wurde die Reihenfolge der Reduktion/Ableitung vertauscht. Also müssen wir die Reduktion näher definieren.

Definition: (kanonische Reduktion)
Seien R, R' zwei Reduktionsfolgen der Länge n von u nach v mit. R' heißt *unmittelbar kanonischer* als R ($R' \lessdot R$), genau dann wenn R' und R sind identisch sind, bis auf dass der k und $(k+1)$-te Reduktionsschritt vertauscht sind. Dabei ist R' kanonischer als R, wenn R' die „linkeste mögliche" Regel zuerst anwendet. (sic!)

Eine Reduktionsfolge R ist genau dann *kanonisch*, wenn es keine Reduktionsfolge R' gibt, mit $R' \lessdot R$.

Definition: (unwesentlich verschieden)
Zwei Reduktionsbäume R, R' heißen unwesentlich verschieden, wenn eine Serie von Reduktionsfolgen existiert mir $R = R_1, R_2, \ldots, R_m = R'$ und $R_i \lessdot R_{i+1}$ oder $R_{i+1} \lessdot R_i$.

Definition: (Eindeutigkeit)
Ein Programm $w \in L(G)$ heißt *eindeutig*, wenn alle Reduktionsfolgen von w nach S unwesentlich verschieden sind.

Eine Grammatik G heißt *eindeutig*, wenn alle Sätze aus $L(G)$ eindeutig sind.

Definition: (Strukturbaum)
Führt man beim Reduzieren eines Wortes $w \in L(G)$ in jedem Schritt alle möglichen Reduktionen parallel aus, so erhält man den *Strukturbaum* von w.

Satz: *(ohne Beweis)*
Zwei Reduktionsfolgen sind genau dann unwesentlich verschieden, wenn ihre Sturkturbäume identisch sind.

Korollar:
Ein Programm $w \in L(G)$ ist eindeutig, genau dann wenn w nur einen einzigen Strukturbaum besitzt.

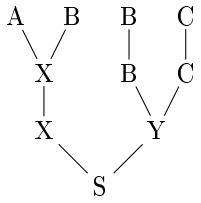
Beispiel:

1. Die Grammatik

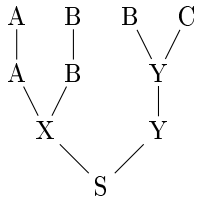
$$N := \{S, X, Y\}, T := \{A, B, C\}, P := \{S \to XY|AY,\ Y \to BY|BC,\ X \to AB\}$$

ist nicht eindeutig, da es für das Wort $ABBC$ die folgenden verschiedenen Reduktionsmöglichkeiten gibt:

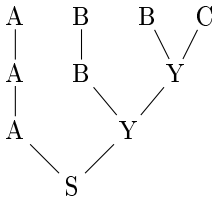
R_1:



R_2:

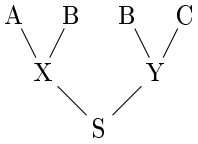


R_3:



R_1 und R_2 sind unwesentlich verschieden, was jedoch für R_1 und R_3 nicht gilt.

Der Strukturbaum von R_1 und R_2 ist gegeben durch:



Kapitel 12

Kombinatorische Logik

Definition: (kombinatorische Logik)
Sei V eine Menge von Variablen, C eine Menge von Konstanten. Spezielle Konstanten seien die Basisfunktionen K und S

Sei $A = V \cup C$. Die Menge CT der kombinatorischen Terme ist gegeben durch:

(i) $A \subset CT$

(ii) $X, Y \in CT \Rightarrow (XY) \in CT$

Die Semantik der Basiskombinatoren ist gegeben durch:

$$Kax = a$$
$$Sfgx = fx(gx)$$

Beispiel:
Die Identitätsfunktion ist gegeben durch: $I \equiv SKK$:

$$If \equiv SKKf = Kf(Kf) = f$$

Bemerkung:
Alle „intuitiv“ berechenbaren Funktionen sind in der kombinatorischen Logik darstellbar.

Beispiel:
Dieses Beispiel soll ein Problem von Programmiersprachen mit Variablen verdeutlichen:

```
1  program p;
2  var a : int;
3  function f(x: int) : int
4  begin
5      a := x+1;
6      f := a;
7  end
8  function g(x: int) : int
9  begin
10     a := x+2;
11     g := a;
12 end;
13
14 a := 0; print(f(a) + g(a));
15 a := 0; print(g(a) + f(a));
```

Dieses Programm gibt zunächst 4 und dann 5 aus. f und g verhalten sich also nicht kommutativ.

Definition: (FB-System)
Ein FB-System $(A, F, \mathbb{F}, :)$ besteht aus:

- A : Menge an atomaren Objekten mit $\perp, true, false, NIL \in A$.
- Die Menge O der Objekte ist

$$\begin{aligned} O &:= A \cup \{< x_1, \ldots, x_n > |\ n \geq 1, x \in O - \{\perp\}\} \\ NIL &\approx < > \text{ leere Folge} \\ \perp &\approx \text{ undefiniert } \Rightarrow < \cdots \perp \cdots > = \perp \end{aligned}$$

- F : Menge von Basisfunktionen $f : O \to O$
- $\mathbb{F}$: Menge von Funktionalen $f : [O \to O] \to [O \to O]$ oder $f : O \to [O \to O]$.

Definition: (Grundfunktionen)
Einige Grundfunktionen für FB-Systeme sind die Folgenden:

Konstante Funktion $\overline{x}$:

$$\overline{x} : y \quad := \quad \begin{cases} \perp & \text{falls } y = \perp \\ x & \text{falls } y \in O \end{cases}$$

apply to all α:

$$\alpha f : x \quad := \quad \begin{cases} NIL & \text{falls } x = NIL \\ < f : x_1, \ldots, f : x_n > & \text{falls } [x =< x_1, \ldots, x_n >] \wedge [f : x_1 \neq \perp \wedge \cdots \wedge f : x_n \neq \perp] \end{cases}$$

Komposition $\circ$:

$$(f \circ g) : x \quad := \quad f : (g : x)$$

Reduktion /:

$$/f : x \quad := \quad \begin{cases} x_1 & \text{falls } x =< x_1 > \\ f :< x_1,\ /f :< x_2, \ldots, x_n >> & \text{falls } [x =< x_1, \ldots, x_n >] \wedge [n \geq 2] \end{cases}$$

Anm. des Verf.: Da die hier verwendete Syntax ist leider nicht besonders intuitiv ist: Der Doppelpunkt bedeutet nichts anderes als „angewendet auf“, also statt $f : x$ würde man normalerweise eher $f(x)$ schreiben.

Beispiel:
Gesucht ist ein Programm, dass die Länge einer Folge bestimmt:

$$\text{def länge} \equiv (/+) \circ (\alpha\overline{1})$$

Start des Programmes angewandt auf $< 1, 2, 3 >$:

$$\begin{aligned} \text{länge} :< 1, 2, 3 > &\equiv (/+) \circ (\alpha\overline{1}) :< 1, 2, 3 > \\ &\equiv (/+) : \big((\alpha\overline{1}) :< 1, 2, 3 >\big) \\ &\equiv (/+) :< \overline{1} : 1, \overline{1} : 2, \overline{1} : 3 > \\ &\equiv (/+) :< 1, 1, 1 > \\ &\equiv + :< 1, /+ :< 1, 1 >> \\ &\equiv + :< 1, + :< 1, /+ < 1 >>> \\ &\equiv + :< 1, + :< 1, 1 >> \\ &\equiv + :< 1, 2 >= 3 \end{aligned}$$

Beispiel:
Hier nun noch ein Programm zur Matrixmultiplikation:

$$\text{def MM} \equiv (\alpha(\alpha IP)) \circ distr \circ [S1, trans \circ S2]$$

$$\text{IP} \equiv (/+) \circ (\alpha *) \circ trans$$

Bemerkung:
Offensichtlich lassen sich Funktionen auch völlig ohne Parameter, Variablen und Typisierungen programmieren. Diese Art von Programmierung schaut zwar anders aus, wie herkömmliche Programmierung, stellt aber keine Einschränkung dar – man muss nur „etwas anders denken".

Sie hat sogar den Vorteil, dass mathematische Beweise über die Korrektheit von Programmen dieser Art möglich sind.

Kapitel 13

λ-Kalkül

13.1 informale Version

1932 überlegte sich Church wie es möglich ist Funktionen ausschließlich unter dem Aspekt der Rechenvorschrift zu bilden. Dadurch können Funktionen sowohl Argumente als auch Ergebnisse sein.

Einführung:
Programme $\approx$ Ausdrücke $\approx$ Terme

1. Ein λ-Term $\lambda x\ M \approx \underbrace{f(x)}_{\text{Variable}} = \underbrace{M}_{\text{Rumpf}}$
2. Ein λ-Term $(M\ N)$ entspricht der Anwendung von M auf das Argument N (Funktion-Applikation)
3. Ein λ-Term $((\lambda x\ Y)\underbrace{A}_{N})$, 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
   - Konstanden + Variable $\approx$ Atome
   - $\lambda x\ M \approx$ Abstraktion
   - $(M\ N) \approx$ Applikation

Beispiel:

1. $\lambda x(x\ y) \approx$ Anwendung von x auf y
$$(\lambda x(x\ y)F) \xrightarrow{3.} (F\ y)$$
2. $\lambda x\ Y \approx$ Konstante Funktion mit dem Wert Y
$$((\lambda x\ Y)F) \to Y$$
3. $\lambda x(x\ x) \approx$ Selbstapplikation
$$(\lambda x(x\ x)F) \xrightarrow{3.} (F\ F)$$

Bemerkung:
Jede mehrstellige Funktion kann durch „Curryfizieren“ in eine einstellige Funktion überführt werden.

Beispiel:
Subtraktion: $h(x, y) = x - y$

$$\lambda x(\lambda y(x - y)) \approx \lambda\text{-Term(Funktion), die eine Funktion als Ergebnis liefert.}$$

Anwendung:

$$((\lambda x(\lambda y(x-y))5)3) \xrightarrow{3.} ((\lambda y(5-y))3)$$
$$\xrightarrow{3.} 5-3$$

13.2 Formale Version

Definition: (λ-Kalkül)
Sei V eine Menge von Variablen, C eine Menge von Konstrukten, $V \cap C = \emptyset$. (Ist $C = \emptyset$, so spricht man von einem reinen λ-Kalkül).

Die folgende kontextfreie Grammatik beschreibt das λ-Kalkül:

$$G_\lambda = \{T_\lambda, N_\lambda, P_\lambda, L_\lambda\}$$
$$T_\lambda = \{\lambda, (,)\} \cup V \cup C$$
$$N_\lambda = \{L_\lambda, V_\lambda, C_\lambda\}$$

Das Regelwerk P_λ ist gegeben durch:

$$L_\lambda \to V_\lambda \mid C_\lambda \mid (\lambda V_\lambda L_\lambda) \mid (L_\lambda L_\lambda)$$
$$V_\lambda \to x \mid y \mid \cdots, x, y \in V$$
$$C_\lambda \to a \mid b \mid \cdots, a, b \in C$$

Konvention zur Übersichtlichkeit:

1.
   - Konstanten: Kleine Buchstaben vom Anfang des Alphabetes
   - Variablen: Kleine Buchstaben vom Ende des Alphabetes
   - *dicke Ausdrücke* (sic!): Große Buchstaben
2.
   - Äußerste Klammerpaare dürfen fehlen
   - $M\ N_1 \cdots N_n$ bedeutet $(\cdots((M\ N_1)N_2)\cdots N_n)$ (Linksklammerung)
3. Mehrstellige Funktionen können auch durch $\lambda x_1 \cdots x_n.M$ dargestllt werden

Beispiel:

$$\lambda xy.xy(\lambda z.z) \quad \approx \quad (\lambda x(\lambda y((x\ y)(\lambda z\ z))))$$

13.3 Gültigkeitsbereiche

Definition: (freie und gebundene Variablen)
Sei Λ die Menge aller λ-Terme. Für $M \in \Lambda$ ist $FV(M)$ die Menge der *freien Variablen* und $BV(M)$ die Menge der *gebundenen Variablen*, induktiv gegeben durch:

- $FV(x) = \{x\}, \quad BV(x) = \emptyset$
- $FV(a) = \emptyset, \quad BV(a) = \emptyset$
- $FV(M\ N) = FV(M) \cup FV(N), \quad BV(M\ N) = BV(M) \cup BV(N)$
- $FV(\lambda x\ M) = FV(M) - \{x\}, \quad BV(\lambda x\ M) = BV(M) \cup \{x\}$

Terme ohne freie Variablen nennen wir Kombinatoren oder geschlossene Terme.

klassische Semantik (Reduktionssemantik):
Prinzip: Reduziere ($\approx$ errechne) alles was möglich ist. Das Endergebnis ist die Bedeutung des Terms.

Definition: (Substitution)
Die Substitution $\mathrm{Sub}_N^x[M]$ des λ-Termes N (Argument) für alle freien Vorkommen von x (Parameter) im λ-Term M (Rumpf) ist induktiv definiert durch:

(i) $\mathrm{Sub}_N^x[x] = N, \mathrm{Sub}_N^x[a] = a, a \in (V \cup C) - \{x\}$

(ii) $\mathrm{Sub}_N^x[(M_1\ M_2)] = (\mathrm{Sub}_N^x[M_1]\ \mathrm{Sub}_N^x[M_2])$

(iii) $\mathrm{Sub}_N^x[\lambda x\ M] = \lambda x\ M$

(iv)

$$\mathrm{Sub}_N^x[\lambda y\ M] = \begin{cases} \lambda y\ \mathrm{Sub}_N^x[M] & \text{falls } y \notin FV(N) \wedge x \notin FV(M) \\ \lambda z\ \mathrm{Sub}_N^x[\mathrm{Sub}_Z^y[M]] & \text{falls } y \in FV(N) \vee x \in FV(M) \\ & \text{dabei ist } z \text{ eine } \textit{neue} \text{ Variable, } \textit{nicht} \text{ in } N, M \text{ vorkommend} \end{cases}$$

Beispiel:
Annahme: *keine* Umbenennung („naive Substitution")

$$\overline{\mathrm{Sub}}_N^x[\lambda y\ M] = \lambda y\ \overline{\mathrm{Sub}}_N^x[M]$$

Betrachte konstante Funktion $\lambda x\ x$ – Umbenennung in Konstante Funktion:

$$\overline{\mathrm{Sub}}_W^x[\lambda y\ x] = \lambda y\ w$$

Aber: $\overline{\mathrm{Sub}}_y^x[\lambda y\ x] = \lambda y\ y$ ist die Identitätsfunktion

Bemerkung:
Mit Umbenennung: *static scoping*
Ohne Umbenennung: *dynamic scoping*

Definition: (Reduktion)
Ein λ-Term P wird zu einem λ-Term P' reduziert, wenn einer der folgenden Reduktionsschritte entweder auf P oder auf einen Teilterm von P angewandt wird:

(i) α-Reduktion:

$$\lambda x\ M \xrightarrow{\alpha} \lambda y\ \mathrm{Sub}_y^x[M]$$

(ii) β-Reduktion:

$$(\lambda x\ M)N \xrightarrow{\beta} \mathrm{Sub}_N^x[M]$$

Beispiel:

$$I \equiv \lambda x.x$$

$$K \equiv \lambda xy.x$$

$$S \equiv \lambda xyz.xz(yz)$$

$$\begin{aligned} SMNL &\equiv (((\lambda x(\lambda y(\lambda z(xz(yz))))M)N)L) \\ &\to ((\lambda y(\lambda z(Mz(yz)))N)L) \\ &\xrightarrow{\beta} ((\lambda z(Mz(Nz)))L) \\ &\xrightarrow{\beta} ML(NL) \end{aligned}$$

Bemerkung:
Frage: Ist die Reihenfolge der Reduktionen wichtig? Nein.