Formale Sprachen und Berechenbarkeit

Vorlesung
von
Peter Schroeder-Heister

Skript
Reinhard Katzmann

Revision
Stefan Laible

2004

Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik

Vorwort

Die Vorlesung „Formale Sprachen und Berechenbarkeit“ wurde von mir bisher dreimal an der Universität Tübingen gehalten. Das vorliegende Skript wurde von Reinhard Katzmann im Wintersemester 1992/93 angefertigt. Eine Revision und Erweiterung erfolgte im Wintersemester 2003/04 mit der Hilfe von Stefan Laible.

Die Vorlesung hat an vielen Stellen, ohne daß dies jeweils im einzelnen angegeben ist, Materialien aus anderen Quellen aufbereitet und verwendet: im Teil über formale Sprachen inbesondere den Klassiker von Hopcroft & Ullman (1990) sowie die Vorlesungsskripten von Rüdiger Loos (1989) und Klaus Schulz (1991), im Teil über Berechenbarkeit vor allem die ebenfalls klassischen Lehrbücher von Boolos & Jeffrey (1989), Hermes (1978) und Lewis & Papadimitriou (1981). Im letzten Jahrzehnt hat sich die Anzahl der Lehrbücher zum Themengebiet der Vorlesung enorm vergrößert. Einige dieser Lehrbücher, die ich zum Studium empfehle, sind im Literaturverzeichnis vermerkt.

Ich danke meinen Hörerinnen und Hörern für zahlreiche Verbesserungsvorschläge, die in diese Revision eigegangen sind. Namentlich erwähnen möchte ich dabei René Gazzari und Ingmar Schuster. Besonderer Dank gebührt Reinhard Katzmann und Stefan Laible für ihr großes Engagement bei der Erstellung bzw. Revision des Skripts.

Peter Schroeder-Heister

Inhalt

Teil I: Formale Sprachen

1 Grundlagen

Formale Sprachen sind Mengen von Zeichenketten. In der Theorie der formalen Sprachen, deren elementare Grundlagen in Kapitel 1–9 dargestellt werden interessiert man sich für Methoden, Sprachen zu erkennen oder zu erzeugen.

Definition 1.1 (Alphabet, Wort, Länge, Sprache)

Ein Alphabet ist eine nichtleere endliche Menge von Objekten. Diese Objekte werden „Zeichen“ genannt. Variablen: Σ für Alphabete, a,b,c,d,e für Zeichen.

Ein n-Tupel $(a_1,\ldots,a_n)$ von Zeichen eines Alphabets Σ heißt Wort oder Zeichenreihe über Σ. Als Grenzfall $(n = 0)$ lassen wir das leere Wort ε zu. Variablen für Wörter: u, v, w, x, y, z.

Ist w das Wort $(a_1,\ldots,a_n)$, dann heißt n die Länge von w (Notation: $|w|$). Das leere Wort hat die Länge 0.

Ist w das Wort $(a_1,\ldots,a_n)$, dann sei $(w)_i$ für $1 \leq i \leq n$ das Zeichen a_i, d.h. die i-te Komponente von w. [Bsp.: $((\mathrm{a,n,t,o,n}))_3{=}t$] Wir identifizieren das Wort (a) mit dem Zeichen a.

Σ^n bezeichnet die Menge der Wörter der Länge n

Σ^* bezeichnet die Menge aller Wörter über Σ

$\Sigma^* = \bigcup\limits_{n\in\mathbb{N}} \Sigma^n$

$\Sigma^+ = \Sigma^* \backslash \{\varepsilon\}$

Jede Teilmenge von Σ^* heißt *Sprache* über Σ. (Variable für Sprachen: L)

Definition 1.2 (Komposition)

Sind u und v die Wörter $(a_1,\ldots,a_n)$ und $(b_1,\ldots,b_m)$, dann sei $u \circ v$ das Wort $(a_1,\ldots,a_n,b_1,\ldots,b_m)$. Insbesondere sei $\varepsilon \circ u = u \circ \varepsilon = u$. Statt $u \circ v$ schreibt man meist uv.

Bemerkungen (Schreibweisen):

1. Da $\circ$ offensichtlich assoziativ ist, ist uvw und $u_1u_2 \ldots u_n$ erklärt.
2. Durch die Identifizierung von a mit (a) sind auch Schreibweisen wie $uaxb$ erklärt. Insbesondere können wir ein Wort $(a_1,\ldots,a_n)$ als $a_1a_2\ldots a_n$ schreiben.
3. $\langle\Sigma^*, \circ\rangle$ ist eine Halbgruppe, $\langle\Sigma^*, \circ, \varepsilon\rangle$ ist ein Monoid (Halbgruppe mit neutralem Element)

Definition 1.3 (Operationen auf Wörtern)

u heißt Teilwort von v, falls v sich als xuy schreiben läßt. u heißt dabei echtes Teilwort von v, falls *nicht* $x = y = \varepsilon$.

u heißt Präfix von v, falls v sich als ux schreiben läßt. u heißt dabei echtes Präfix von v, falls $x \neq \varepsilon$.

u heißt Suffix von v, falls v sich als xu schreiben läßt. u heißt dabei echtes Suffix von v, falls $x \neq \varepsilon$.

Die Umkehrung u^R von u ist wie folgt definiert:
$\varepsilon^R = \varepsilon \quad (wa)^R = aw^R$

u heißt Palindrom, falls $u = u^R$

u^n bedeutet $\underbrace{u \dots u}_{n-\mathrm{mal}}$, d.h. formal $u^0 = \varepsilon,\ u^{n+1} = u^n u$

Definition 1.4 (Operationen auf Sprachen)

$L_1 \circ L_2 := \{w_1 w_2 | w_1 \in L_1 \text{ und } w_2 \in L_2\}$ (Verkettung, auch $L_1 L_2$ geschrieben)

$L^n = \underbrace{L \dots L}_{n-\mathrm{mal}}$, d.h. formal: $L^0 := \{\varepsilon\}$. $L^{n+1} := L^n L$ (Potenzierung)

$L^* := \bigcup_{i \in \mathbb{N}} L^i$ (Kleene-Stern)

$L^+ := L^* \backslash \{\varepsilon\}$

Beispiel:

L_1: alle 0-Folgen $0 \dots 0$
L_2: alle 1-Folgen $1 \dots 1$

$L_1 \circ L_2$: alle Folgen $0 \dots 01 \dots 1$
$(L_1 \cup L_2)^*$: Alle Folgen $0 \dots 01 \dots 1 \dots 0 \dots 01 \dots 1$, darunter z.B.
00111010000011

Bemerkungen:

1. Die üblichen mengentheoretischen Operationen ($L_1 \cup L_2,\ L_1 \cap L_2,\ L_1 \backslash L_2$) sind natürlich auch definiert.

2. Aufgefaßt als Menge aller Wörter der Länge 1 ist Σ eine Sprache über Σ, d.h. $\Sigma \subseteq \Sigma^*$, Σ^* im Sinne von Definition 1.4 stimmt dann mit Σ^* im Sinne von Definition 1.1 überein.

3. Es gelten verschiedene Gesetze über die Operationen auf Sprachen, z.B.

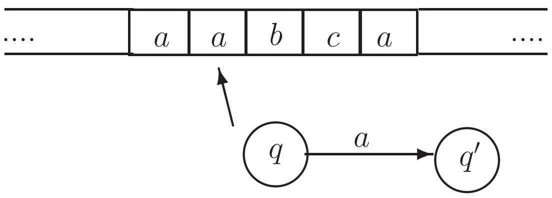
   $L_1 \circ (L_2 \cup L_3) = (L_1 \circ L_2) \cup (L_1 \circ L_3)$
   $L^* \circ L^* = L^*$
   $L \circ \{\varepsilon\} = \{\varepsilon\} \circ L = L$
   $L \circ \emptyset = \emptyset \circ L = \emptyset$

4. Die Menge aller Sprachen über Σ ist überabzählbar. Die Menge aller Wörter über Σ jedoch abzählbar. Man kann also nicht jede Sprache über Σ durch ein Wort über einem festem Alphabet Σ' charakterisieren. Bei einer bestimmten Teilklasse von Sprachen, den regulären Sprachen, ist dies jedoch möglich (siehe Kapitel 3).

2 Endliche Automaten

Automaten sind Modelle für Verfahren, die Zugehörigkeit gegebener Zeichenketten (Eingabewörter) zu einer Sprache festzustellen. Die Bezeichnung „Automat“ deutet an, daß man auch an die Realisierung solcher Verfahren durch konkrete Maschinen denkt. Die theoretischen Überlegungen sind jedoch von solchen Realisierungen unabhängig. Endliche Automaten lassen sich realisieren, kompliziertere Automaten jedoch nur mit Einschränkungen. Insofern sind Automaten ideale Maschinen.

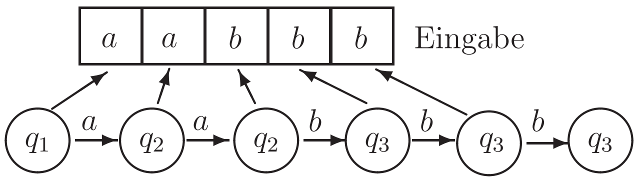
Idee: Ein endlicher Automat liest in einem bestimmten Zustand q ein Eingabezeichen a, und geht dann in Abhängigkeit vom gelesenen Zeichen in einen neuen Zustand q' über.

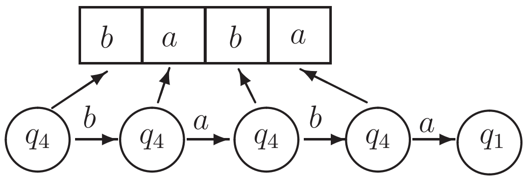


Beispiel:

Zustände, die Automat annehmen kann: q_1, q_2, q_3, q_4
Eingabealphabet: $\{a, b\}$
Übergangsfunktion:

δ	a	b
q_1	q_2	q_4
q_2	q_2	q_3
q_3	q_4	q_3
q_4	q_4	q_4





Definition 2.1 (Deterministischer endlicher Automat)

Ein deterministischer endlicher Automat (DEA) $\mathcal{A}$ über dem Alphabet Σ ist ein Quintupel $\langle Q, \Sigma, \delta, s, F \rangle$, wobei

Q : nichtleere endliche Menge von Zuständen

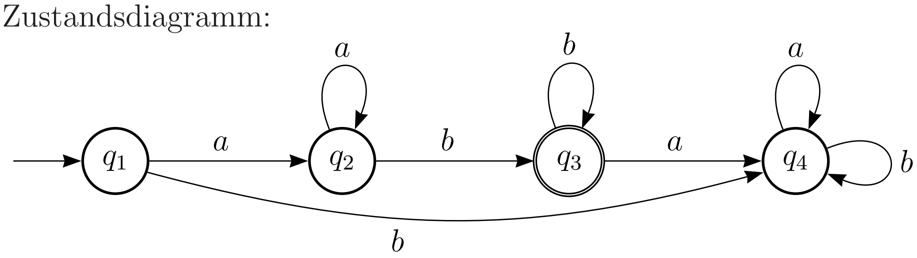
$\delta : Q \times \Sigma \longrightarrow Q$ (Übergangsfunktion)

s : $s \in Q$ Startzustand

$F \subseteq Q$: ausgezeichnete Final- oder Endzustände

Beispiel:

$Q = \{q_1, q_2, q_3, q_4\}$
$\Sigma = \{a, b\}$
δ siehe Matrix oben (Tabelle Übergangsfunktion)
$s = q_1$
$F = \{q_3\}$
Zustandsdiagramm:



Der linke, zu q_1 führende Pfeil charakterisiert q_1 als Startzustand. Endzustände erhalten Doppelringe.

Definition 2.2 (Konfiguration, Akzeptanz)

Ein Element von $Q \times \Sigma^*$ heißt Konfiguration von $\mathcal{A}$. Der sich aus einer Konfiguration (q, w) ergebende Endzustand q' ist durch eine Antwortfunktion $\delta^* : Q \times \Sigma^* \longrightarrow Q$ gegeben, die wie folgt definiert ist:

$\delta^*(q, \varepsilon) = q$

$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

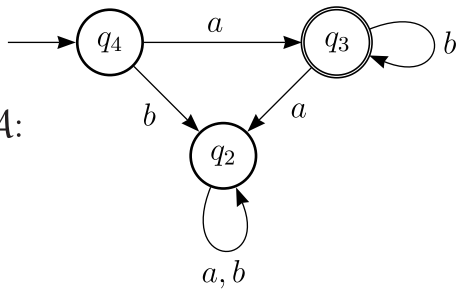
$\mathcal{A}$ akzeptiert w, falls $\delta^*(s, w) \in F$.

$\mathcal{A}$ verwirft w, falls $\delta^*(s, w) \notin F$

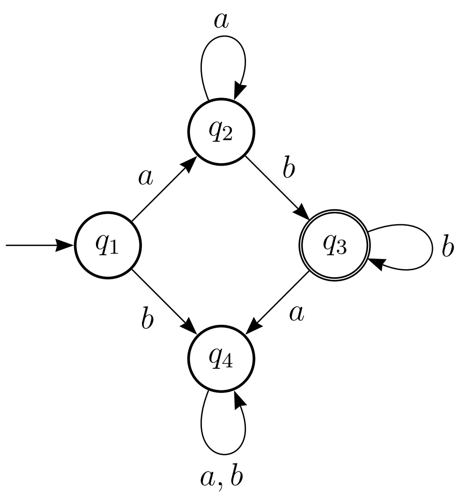
Die von $\mathcal{A}$ akzeptierte Sprache $L(\mathcal{A})$ ist die Menge aller von $\mathcal{A}$ akzeptierten Wörter. Intuitiv ist bei einer Konfiguration (q, w) q der gegenwärtige Zustand des Automaten und w das noch zu lesende (Teil-)Wort. $\delta^*(q, w)$ ist der Zustand, der von (q, w) aus erreicht wird nach vollständiger Lektüre von w.

Beispiele für DEAen, notiert als Graphen:

$\mathcal{A}$:



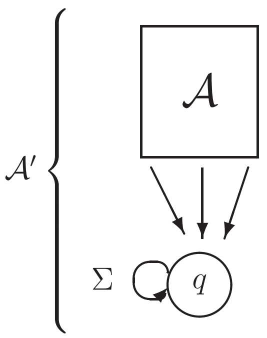
$\Sigma = \{a, b\}$
$L(\mathcal{A}) = \{ab^n : n > 0\}$



$\Sigma = \{a, b\}$
$L(\mathcal{A}) = \{a^n b^m : n, m > 0\}$

Bemerkung zur Notation von DEAen als Graphen:

Da $\delta(q, a)$ für jedes $q \in Q$ und jedes $a \in \Sigma$ definiert ist, muß an sich aus jedem q für jedes a ein mit a markierter Pfeil herausführen. Wenn in der Notation von $\mathcal{A}$ an einer oder mehreren Stellen ein solcher Pfeil fehlt, ist das als Abkürzung für einen Automaten $\mathcal{A}'$ wie folgt verstanden:



Hier ist q ein nicht in $\mathcal{A}$ vorkommender „toter“ Zustand, in den alle in $\mathcal{A}$ „fehlenden“ Pfeile hineinführen.
Dies ist als *informelle* Konvention zu verstehen, die dann greift, wenn man nur die Graphennotation verwendet und nicht schon von vornherein definitiv durch Angabe eines Quintupels festgelegt hat, welche Zustände der Automat umfaßt. Die Konvention ließe sich auch *formal* explizieren, wenn man die Übergangsfunktion δ eines Automaten nicht als totale, sondern als partielle (d.h. nicht überall definierte) Funktion auffaßt. Dann wäre $\mathcal{A}'$ der zu einem Automaten mir partieller Übergangsfunktion gehörende gleichwertige Automat mit totaler Übergangsfunktion.

Definition 2.3 (DEA-Akzeptierbarkeit)

Eine Sprache L heißt DEA-akzeptierbar, wenn es einen DEA $\mathcal{A}$ gibt , so daß $L = L(\mathcal{A})$.

Bemerkung (Eindeutigkeit):

$\mathcal{A}$ ist nicht eindeutig, wie folgendes Beispiel zeigt:

Sei L Sprache über Σ.
Sei $\Sigma' \supseteq \Sigma$. Dann gilt:
Es gibt DEA $\mathcal{A}$ über Σ mit $L = L(\mathcal{A})$ g.d.w. es $\mathcal{A}'$ über Σ' mit $L = L(\mathcal{A}')$ gibt.

Definition 2.4 (Äquivalenz und Erreichbarkeit)

Zwei Zustände q_1 und q_2 eines DEA $\mathcal{A} = \langle Q, \Sigma, \delta, s, F \rangle$ heißen äquivalent, falls gilt, daß $\{w : \delta^*(q_1, w) \in F\} = \{w : \delta^*(q_2, w) \in F\}$.

Der Zustand q_2 heißt von q_1 aus erreichbar, falls es $w \in \Sigma$ gibt, so daß $\delta^*(q_1, w) = q_2$.

Der Zustand q heißt erreichbar, falls q von s aus erreichbar ist.

$\mathcal{A}$ heißt reduziert, falls jeder Zustand aus Q erreichbar ist und keine zwei Zustände äquivalent sind.

Zwei DEAen $\mathcal{A}_1$ und $\mathcal{A}_2$ heißen äquivalent, falls $L(\mathcal{A}_1) = L(\mathcal{A}_2)$

Lemma 2.5

Zu jedem DEA $\mathcal{A}$ gibt es einen äquivalenten reduzierten DEA $\mathcal{A}'$.

Beweis:
Sei $\mathcal{A} = \langle Q, \Sigma, \delta, s, F\rangle$

1. Seien Q_1 die erreichbaren, $Q_2 = Q \backslash Q_1$ die nicht erreichbaren Zustände von $\mathcal{A}$. Dann gibt es keinen Übergang von Q_1 nach Q_2. Wir können also Q_2 mit allen davon ausgehenden Übergängen streichen, d.h. $\langle Q_1, \Sigma, \delta|_{Q_1\times\Sigma}, s, F \cap Q_1\rangle$ ist mit $\mathcal{A}$ äquivalent.

2. Falls q_1 und q_2 äquivalent, eliminieren wir q_2 und setzen

   $\delta'(q, a) = q_1$ falls $\delta(q, a) = q_2$,

   $\delta'|_{Q\backslash\{q_2\}\times\Sigma} = \delta$ sonst.

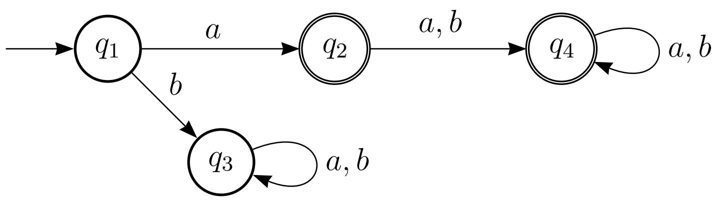
   Dann gilt:

   $\langle Q\backslash\{q_2\}, \Sigma, \delta', s, F\backslash\{q_2\}\rangle$ ist mit $\mathcal{A}$ äquivalent.

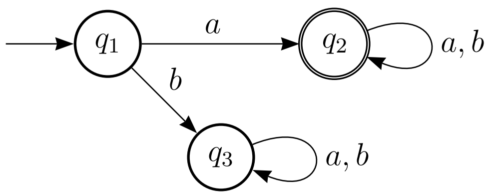
   Falls s einer der beiden äquivalenten Zustände ist, haben wir o.B.d.A. angenommen, daß es sich um q_1 handelt, der von s verschiedene Zustand also gestrichen wird.

Da es nur endlich viele Zustände gibt, kann man so durch iterierte Anwendung von 1. und 2. alle Paare äquivalenter Zustände eliminieren.

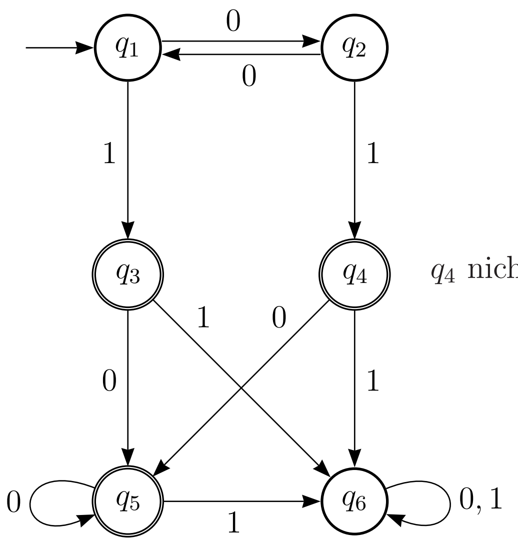
Beispiel:



Hier sind q_2 und q_4 äquivalent. Der Automat läßt sich reduzieren zu

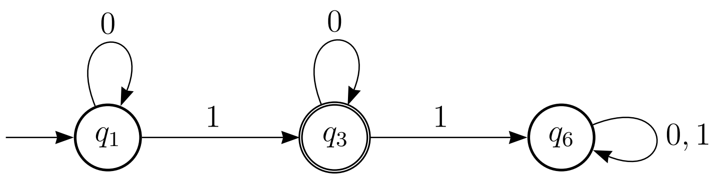


Anderes Beispiel:



q_4 nicht mehr erreichbar nach Elimination von q_2

Dieser Automat läßt sich schrittweise reduzieren zu:



Der Beweis des Lemmas liefert keinen Algorithmus, da nicht gesagt ist, wie man die Äquivalenz von Zuständen feststellen kann. Einen Algorithmus liefert folgendes Verfahren zur Konstruktion eines *Äquivalenzklassenautomaten* durch Definition sukzessiv verfeinerter Äquivalenzrelationen.

Definition 2.6 (Äquivalenzklassenautomat)

Gegeben sei ein DEA $\mathcal{A} = \langle Q, \Sigma, \delta, s, F \rangle$. Wir nehmen an, daß $\mathcal{A}$ keine nichterreichbaren Zustände enthält (d.h. daß diese in einem ersten Schritt eliminiert worden sind).

Eine Folge sukzessiv verfeinerter Äquivalenzrelationen auf Q ist wie folgt definiert:
$q \sim_0 q'$ g.d.w. $q \in F \Leftrightarrow q' \in F$
$q \sim_{n+1} q'$ g.d.w. $q \sim_n q'$ und für alle $a \in \Sigma : \delta(q, a) \sim_n \delta(q', a)$
Die Äquivalenzrelation $\sim$ sei die feinste so erhaltene Äquivalenzrelation:
$q \sim q'$ g.d.w. $q \sim_n q'$ für das kleinste n mit $\sim_n = \sim_{n+1}$.
Da es nur *endlich* viele Zustände gibt, ist $\sim$ wohldefiniert. Da offenbar aus $\sim_n = \sim_{n+1}$ folgt:

$\sim_n = \sim_{n+k}$ für beliebiges $k > 1$, ist $\sim$ in der Tat die feinste erhaltene Äquivalenzrelation. Sei $[q]$ die zu einem Zustand $q \in Q$ gehörende Äquivalenzklasse bezüglich $\sim$. Dann ist der *Äquivalenzklassenautomat* $\widetilde{\mathcal{A}} = \langle \widetilde{Q}, \Sigma, \widetilde{\delta}, \widetilde{s}, \widetilde{F} \rangle$ wie folgt definiert:

$\widetilde{Q} := Q/\sim$
$\widetilde{\delta}([q], a) := [\delta(q, a)]$
$\widetilde{s} := [s]$
$\widetilde{F} := \{[q] : q \in F\}$

Lemma 2.7

$\widetilde{\mathcal{A}}$ ist ein zu $\mathcal{A}$ äquivalenter reduzierter DEA.

Beweis:

1. $\widetilde{\mathcal{A}}$ ist wohldefiniert als DEA, d.h. $\widetilde{\delta}([q], a)$ ist unabhängig von der Auswahl des Repräsentanten q des ersten Arguments, d.h.

   $q \sim q' \Rightarrow \delta(q, a) \sim \delta(q', a)$

   Sei $\sim_n = \sim_{n+1}$
   Dann gilt:

   $$\begin{aligned} q \sim q' &\iff q \sim_n q' \\ &\iff q \sim_{n+1} q' \\ &\Longleftarrow \delta(q, b) \sim_n \delta(q', b) \text{ für alle } b \in \Sigma \\ &\iff \delta(q, a) \sim \delta(q', a) \end{aligned}$$

2. $\widetilde{\mathcal{A}}$ ist reduziert.

   a) Nach Voraussetzung ist jeder Zustand von $\mathcal{A}$ und damit auch jeder Zustand von $\widetilde{\mathcal{A}}$ erreichbar.

   b) Keine zwei Zustände von $\mathcal{A}$ sind äquivalent:
   Wir benutzen als leicht beweisbare Hilfssätze:

   $\otimes$: $q \sim_n q'$ g.d.w. q und q' sind in $\mathcal{A}$ durch Wörter der Länge $\leq n$ nicht unterscheidbar, d.h. ($\forall w$ mit $|w| \leq n$) $\delta^*(q, w) \in F \Leftrightarrow \delta^*(q', w) \in F$

   $\otimes\otimes$: $\widetilde{\delta}^*([q], w) = [\delta^*(q, w)]$

Aus $\otimes$ ergibt sich:
$q \sim q'$ g.d.w. q und q' sind äquivalente Zustände von $\mathcal{A}$.
Aus $\otimes\otimes$ ergibt sich:
q und q' sind äquivalente Zustände von $\mathcal{A}$
g.d.w. $[q]$ und $[q']$ sind äquivalente Zustände von $\widetilde{\mathcal{A}}$.
Aus diesen beiden Resultaten ergibt sich sofort:
$[q]$ und $[q']$ sind äquivalente Zustände von $\widetilde{\mathcal{A}}$ g.d.w. $[q] = [q']$.

Anwendung auf das vorige Beispiel:

δ	0	1
q_1	q_2	q_3
q_2	q_1	q_4
q_3	q_5	q_6
q_4	q_5	q_6
q_5	q_5	q_6
q_6	q_6	q_6

$s = q_1$
$F = \{q_3, q_4, q_5\}$

$\sim_0$: Äquivalenzklassen $\{q_1, q_2, q_6\}$ $[= Q\backslash F]$
$\{q_3, q_4, q_5\}$ $[= F]$

$\sim_1$: $q_1 \sim_1 q_2 : \checkmark$
$q_1 \sim_1 q_6 : \times \quad \delta(q_1, 1) = q_3 \nsim_0 q_6 = \delta(q_6, 1)$
$q_3 \sim_1 q_4 : \checkmark$
$q_3 \sim_1 q_5 : \checkmark$
Äquivalenzklassen $\{q_1, q_2\}, \{q_6\}, \{q_3, q_4, q_5\}$

$\sim_2$: $q_1 \sim_1 q_2 : \checkmark$

Damit ist $\sim_1 = \sim_2$, d.h. $\sim = \sim_1$.

Äquivalenzklassenautomat:

$\widetilde{\delta}$	0	1
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_3, q_4, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_6\}$
$\{q_3, q_4, q_5\}$	$\{q_3, q_4, q_5\}$	$\{q_6\}$

$s = \{q_1, q_2\}$
$F = \{\{q_3, q_4, q_5\}\}$

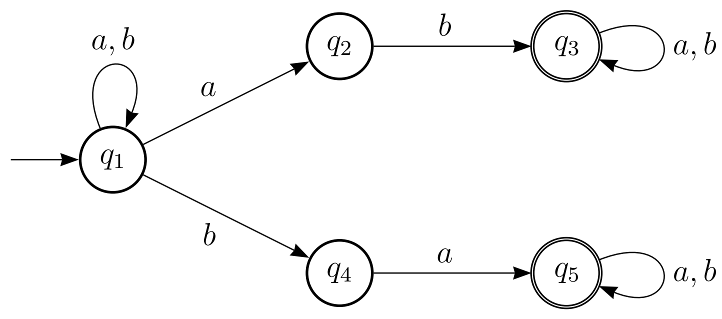
Das Verfahren der sukzessiven Verfeinerung von Äquivalenzklassen läßt sich auch so beschreiben: Bei jedem Schritt werden Paare $\langle q_i, q_j \rangle$ von Zuständen als *nicht-äquivalent* verworfen, und zwar so lange, bis sich nichts mehr ändert. Für alle verbleibenden Paare $\langle q, q' \rangle$ gilt dann $q \sim q'$. Dies läßt sich leicht in Form einer (symmetrischen) Matrix veranschaulichen, in der schrittweise für nichtäquivalente Zustände q_i und q_j das Feld $\langle i, j \rangle$ markiert wird.
Man verwechsle die Konstruktion des Äquivalenzklassenautomaten nicht mit derjenigen des Potenzmengenautomaten bei der Konstruktion eines deterministischen aus einem nichtdeterministischen endlichen Automaten (siehe Theorem 2.9). Beim Äquivalenzklassenautomaten bedeutet $\widetilde{\delta}(M, a) = M'$ *nicht*, daß $\delta(M, a) = M'$.

Nichtdeterministische endliche Automaten (NDEA)

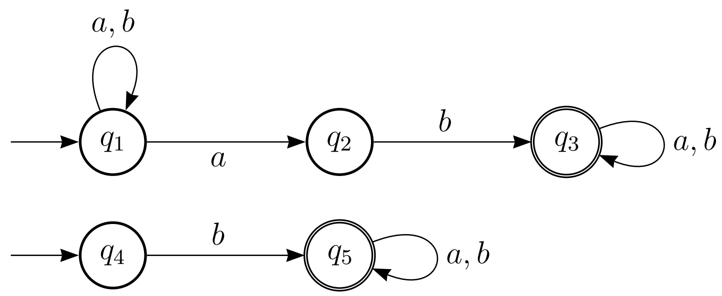
Motivation durch 2 Beispiele:

Beispiel 1:

$\Sigma = \{a, b\}$.
Es sollen alle Wörter der Form $\ldots ab \ldots$ oder $\ldots ba \ldots$ akzeptiert werden, d.h. die Sprachen $\{uabv : u, v \in \Sigma^*\} \cup \{ubav : u, v \in \Sigma^*\}$



Beispiel 2: $\{uabv\} \cup \{bu\}$



Dies ist *ein* Automat, in dem q_1 und q_2 alternative Startzustände sind.

Definition 2.8 (NDEA)

Ein NDEA ist ein Quintupel $\mathcal{A} = \langle Q, \Sigma, \delta, S, F \rangle$ wobei Q, Σ, F wie bei DEAen definiert sind, S ist eine Menge ausgezeichneter Startzustände und $\delta : Q \times \Sigma \to \mathcal{P}(Q)$.

Die Antwortfunktion $\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$ ist wie folgt definiert:

$\delta^*(q, \varepsilon) = \{q\}$

$\delta^*(q, wa) = \bigcup_{q' \in \delta^*(q,w)} \delta(q', a)$

$\mathcal{A}$ akzeptiert w, falls $\bigcup_{s \in S} \delta^*(s, w) \cap F \neq \emptyset$.

$L(\mathcal{A})$ ist die Menge der von $\mathcal{A}$ akzeptierten Wörter.

Bemerkung (Besonderheiten von NDEAen):

1. Intuitiv: Ein NDEA $\mathcal{A}$ akzeptiert w, wenn man aus einem ausgezeichnetem Startzustand einen ausgezeichneten Endzustand erreichen kann bei Lektüre von w und geeigneter Wahl der bei jedem Zeichen anstehenden Alternativen.

2. Ein DEA $\mathcal{A} = \langle Q, \Sigma, \delta, s, F \rangle$ kann als NDEA $\mathcal{A}' = \langle Q, \Sigma, \delta', S, F \rangle$ aufgefaßt werden mit $S = \{s\}; \delta'(q, a) = \{\delta(q, a)\}$. Offenbar gilt: $L(\mathcal{A}) = L(\mathcal{A}')$

3. $\delta(q, a) = \emptyset$ ist möglich bei NDEAen.

4. Manche Autoren erlauben ε-Übergänge ($\delta(q, \varepsilon) = \ldots$) oder das Konsumieren ganzer Wörter in einem Schritt ($\delta(q, w) = \ldots$). Bei uns ist das nicht definiert.

Theorem 2.9 (Äquivalenz DEA - NDEA)

Für jede Sprache L gilt: L ist DEA-akzeptabel g.d.w. L NDEA-akzeptabel ist, d.h.:
Zu jedem DEA $\mathcal{A}$ läßt sich ein NDEA $\mathcal{A}'$ konstruieren, so daß $L(\mathcal{A}) = L(\mathcal{A}')$.
Zu jedem NDEA $\mathcal{A}$ läßt sich ein DEA $\widetilde{\mathcal{A}}$ konstruieren, so daß $L(\mathcal{A}) = L(\widetilde{\mathcal{A}})$.

Beweis:
„$\Rightarrow$“: siehe Bemerkung 2 hinter Definition 2.8

„$\Leftarrow$“: Sei NDEA $\mathcal{A} = \langle Q, \Sigma, \delta, s, F \rangle$ gegeben.
Wir konstruieren DEA $\widetilde{\mathcal{A}} = \langle \widetilde{Q}, \widetilde{\Sigma}, \widetilde{\delta}, \widetilde{s}, \widetilde{F} \rangle$ mit $L(\mathcal{A}) = L(\widetilde{\mathcal{A}})$.
$\widetilde{\Sigma} := \Sigma \qquad \widetilde{Q} := \mathcal{P}(Q) \qquad \widetilde{s} := S$
$\widetilde{F} := \{\widetilde{q} : \widetilde{q} \in \widetilde{Q} \wedge \widetilde{q} \cap F \neq \emptyset\}$

$\widetilde{\delta}(\widetilde{q}, a) := \bigcup_{q \in \widetilde{q}} \delta(q, a)$

$$\begin{aligned} w \in L(\mathcal{A}) & \text{ g.d.w. } \bigcup_{s \in S} \delta^*(s, w) \cap F \neq \emptyset \\ & \text{ g.d.w. } \widetilde{\delta}^*(S, w) \cap F \neq \emptyset \qquad \text{Lemma 2.10 (unten)} \\ & \text{ g.d.w. } \widetilde{\delta}^*(\widetilde{s}, w) \in \widetilde{F} \text{ (Definition von } \widetilde{F}) \\ & \text{ g.d.w. } w \in L(\widetilde{\mathcal{A}}) \end{aligned}$$

Statt des NDEA über Q betrachtet man also einen DEA über der Potenzmenge $\mathcal{P}(Q)$. Man faßt so die *Menge* der Zustände, in die ein nichtdeterministischer Automat übergehen kann, als *den* deterministisch nächsten Zustand auf.

Lemma 2.10

$\widetilde{\delta}^*(\widetilde{q}, w) = \bigcup_{q \in \widetilde{q}} \delta^*(q, w)$

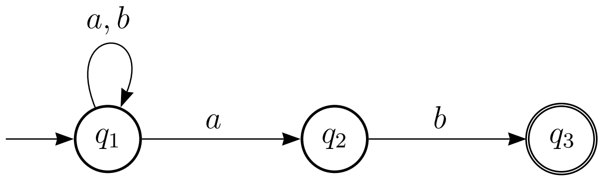
Beweis: Durch Induktion über Wortlänge

I.A.: $\widetilde{\delta}^*(\widetilde{q}, \varepsilon) = \widetilde{q} = \bigcup_{q \in \widetilde{q}} \delta^*(q, \varepsilon)$

$$\begin{aligned} \text{I.S.: } \widetilde{\delta}^*(\widetilde{q}, wa) & = \widetilde{\delta}(\widetilde{\delta}^*(\widetilde{q}, w), a) \\ & = \bigcup_{q' \in \widetilde{\delta}^*(\widetilde{q}, w)} \delta(q', a) \text{ (Def. von } \widetilde{\delta}) \\ & = \bigcup_{q' \in \bigcup_{q \in \widetilde{q}} \delta^*(q, w)} \delta(q', a) \text{ (I.V. bzgl. } w) \\ & = \bigcup_{q \in \widetilde{q}} \underbrace{\bigcup_{q' \in \delta^*(q, w)} \delta(q', a)}_{\substack{\delta^*(q, wa) \\ (\text{Def. von } \delta^*)}} \end{aligned}$$

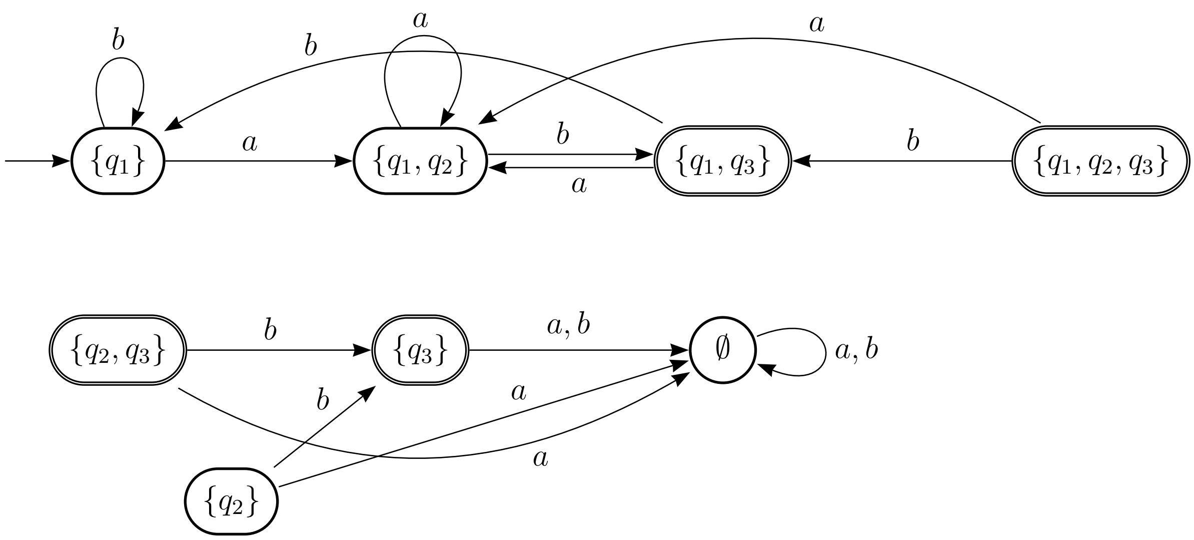
Beispiel:

$\Sigma = \{a, b\}$

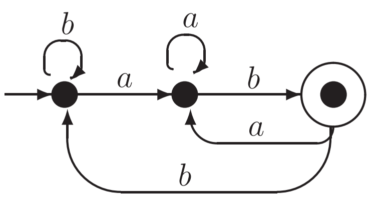


$L(\mathcal{A}) = \{wab : w \in \Sigma^*\}$

$\widetilde{\delta}$	a	b
$\emptyset$	$\emptyset$	$\emptyset$
$\{q_1\}$	$\{q_1, q_2\}$	$\{q_1\}$
$\{q_2\}$	$\emptyset$	$\{q_3\}$
$\{q_3\}$	$\emptyset$	$\emptyset$
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1, q_3\}$
$\{q_1, q_3\}$	$\{q_1, q_2\}$	$\{q_1\}$
$\{q_2, q_3\}$	$\emptyset$	$\{q_3\}$
$\{q_1, q_2, q_3\}$	$\{q_1, q_2\}$	$\{q_1, q_3\}$



Reduziert (ergibt sich durch Streichen nichterreichbarer Zustände):

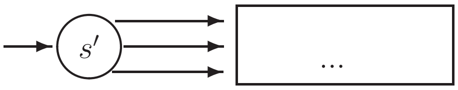


Lemma 2.11

Zu jedem NDEA $\mathcal{A} = \langle Q, \Sigma, \delta, S, F \rangle$ gibt es einen äquivalenten NDEA $\mathcal{A}' = \langle Q', \Sigma, \delta', S', F' \rangle$, so daß gilt:

a) $|S'| = 1$, d.h. $S' = \{s'\}$ für ein $s' \in Q'$

b) $s' \notin \delta'(q, a)$ für alle $q \in Q', a \in \Sigma$, d.h. in s' führen keine Übergänge hinein.

$\mathcal{A}$ soll also so aussehen: 

Beweis:

$\mathcal{A}'$ sei wie folgt definiert:

$Q' := Q \cup \{s'\}$ für neues s'
$S' := \{s'\}$
$\delta'(s', a) := \bigcup_{s \in S} \delta(s, a)$, sonst: $\delta'(q, a) = \delta(q, a)$
$F' := F \cup \{s'\}$, falls $S \cap F \neq \emptyset$; sonst: $F' := F$

a) und b) sind offenbar erfüllt. Es ist noch zu zeigen, daß $\mathcal{A}$ und $\mathcal{A}'$ äquivalent sind.

Es gilt: ${\delta'}^*(s', \varepsilon) = s' \in F' \Leftrightarrow S \cap F \neq \emptyset \Leftrightarrow \bigcup_{s \in S} \delta^*(s, \varepsilon) \cap F \neq \emptyset$,
d.h.: $\varepsilon \in L(\mathcal{A}') \Leftrightarrow \varepsilon \in L(\mathcal{A})$

Zeigen noch: $\otimes$: $\quad {\delta'}^*(s', w) = \bigcup_{s \in S} \delta^*(s, w)$ für $w \neq \varepsilon$

Mit $s' \notin {\delta'}^*(s', w)$ ergibt sich daraus ${\delta'}^*(s', w) \cap F' = \bigcup_{s \in S} \delta^*(s, w) \cap F$ für alle $w \neq \varepsilon$, d.h. zusammen mit $\varepsilon \in L(\mathcal{A}') \Leftrightarrow \varepsilon \in L(\mathcal{A})$ die Äquivalenz von $\mathcal{A}$ und $\mathcal{A}'$.

Beweis von $\otimes$: Induktion über Länge von w.

Für $w = a$ folgt die Behauptung aus der Definition von δ'.

$$
\begin{aligned}
{\delta'}^*(s', wa) \quad &= \quad \bigcup_{q \in {\delta'}^*(s', w)} \delta'(q, a) \\
&\overset{\text{I.V.}}{=} \quad \bigcup_{q \in \bigcup_{s \in S} \delta^*(s, w)} \delta'(q, a) \\
&\overset{\text{q}\neq\text{s}'}{=} \quad \bigcup_{q \in \bigcup_{s \in S} \delta^*(s, w)} \delta(q, a) \\
&= \quad \bigcup_{s \in S} \bigcup_{q \in \delta^*(s, w)} \delta(q, a) \\
&= \quad \bigcup_{s \in S} \delta^*(s, wa)
\end{aligned}
$$

Bemerkung: Diese Argumentation gilt auch für DEAen.

Lemma 2.12

Zu jedem NDEA $\mathcal{A} = \langle Q, \Sigma, \delta, S, F \rangle$ gibt es einen

NDEA $\mathcal{A}' = \langle Q', \Sigma, \delta', S', F' \rangle$ so daß gilt:

a) $|F'| = 1$, d.h. $F' = \{f'\}$ für ein $f' \in Q'$

b) $\delta'(f', a) = \emptyset$ für alle $a \in \Sigma$, d.h. aus f' führen keine Übergänge heraus.

c) $L(\mathcal{A}') = L(\mathcal{A}) \backslash \{\varepsilon\}$, d.h. $\mathcal{A}$ und $\mathcal{A}'$ sind äquivalent bis auf die Akzeptanz des leeren Wortes.

Beweis:

$Q' := Q \cup \{f'\} \quad f'$ neu

$S' := S$

$\delta'(q, a) := \delta(q, a) \cup \{f'\}$ falls $\delta(q, a) \cap F \neq \emptyset$

$\delta'(q, a) := \delta(q, a)$ sonst

$F' := \{f'\}$

Rest des Beweises läuft ähnlich wie Beweis zu Lemma 2.11

Bemerkungen:

1. $\delta^*(s, \varepsilon) \in F \Rightarrow s \in F$ für $s \in S$. Jedoch: $s \notin F'$.

2. Für DEAen gilt das Lemma nicht.

Definition 2.13 (Normalisierter NDEA)

Ein NDEA heißt normalisiert, falls er genau einen ausgezeichneten Anfangszustand hat, in den kein Übergang hineinführt, und genau einen ausgezeichneten Endzustand, aus dem kein Übergang herausführt.

Lemma 2.14 (Normalisierung für NDEA)

Zu jedem NDEA $\mathcal{A}$ gibt es einen NDEA $\mathcal{A}'$, der normalisiert ist, wobei $\mathcal{A}'$ bis auf höchstens das leere Wort äquivalent zu $\mathcal{A}$ ist, d.h. $L(\mathcal{A}') = L(\mathcal{A}) \backslash \{\varepsilon\}$.

Beweis: Folgt aus Lemma 2.11 und 2.12

Schema eines normalisierten NDEA:

Lemma 2.15

Seien $\mathcal{A}_1$ und $\mathcal{A}_2$ NDEAen über Σ, jedoch mit disjunkten Zustandsmengen. In jedem der folgenden Fälle gibt es einen NDEA $\mathcal{A}$ mit der gewünschten Eigenschaft.

a) $L(\mathcal{A}) = L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$

b) $L(\mathcal{A}) = \Sigma^* \backslash L(\mathcal{A}_1)$

c) $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$

d) $L(\mathcal{A}) = L(\mathcal{A}_1) \circ L(\mathcal{A}_2)$

e) $L(\mathcal{A}) = (L(\mathcal{A}_1))^*$

D.h. die Menge der NDEA-akzeptierbaren Sprachen ist abgeschlossen unter $\cup$,$\backslash$,$\cap$,$\circ$,$*$.

Bemerkung: Wegen Theorem 2.9 gilt die Behauptung natürlich auch für DEAen. D.h. die Menge der DEA-akzeptierbaren Sprachen ist ebenfalls abgeschlossen unter den genannten Operationen.

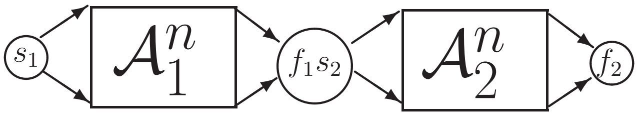
Beweis:

a) Sei $\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, S_1, F_1 \rangle$
$\mathcal{A}_2 = \langle Q_2, \Sigma, \delta_2, S_2, F_2 \rangle$
Wir definieren: $\mathcal{A} = \langle Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, S_1 \cup S_2, F_1 \cup F_2 \rangle$

b) Sei $\mathcal{A}_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1 \rangle$ DEA (können wir wegen Theorem 2.9 annehmen). Damit ist ausgeschlossen, daß $\mathcal{A}$ nach Lesen eines akzeptierten Wortes auch in einem Nicht-Finalzustand sein kann.
Wir definieren: $\mathcal{A} = \langle Q_1, \Sigma, \delta_1, s_1, Q_1 \backslash F_1 \rangle$

c) Nach den de Morganschen Gesetzen gilt:
$L(\mathcal{A}_1) \cap L(\mathcal{A}_2) = \Sigma^* \backslash ((\Sigma^* \backslash L(\mathcal{A}_1)) \cup (\Sigma^* \backslash L(\mathcal{A}_2)))$.
Aus a) und b) ergibt sich die Behauptung.

d) Seien $\mathcal{A}_1^n = \langle Q_1, \Sigma, \delta_1, \{s_1\}, \{f_1\}\rangle$
$\mathcal{A}_2^n = \langle Q_2, \Sigma, \delta_2, \{s_2\}, \{f_2\}\rangle$ nach Lemma 2.14 normalisierte Versionen von $\mathcal{A}_1$ und $\mathcal{A}_2$. Die Konstruktion von $\mathcal{A}$ ist durch folgendes Schema gegeben:

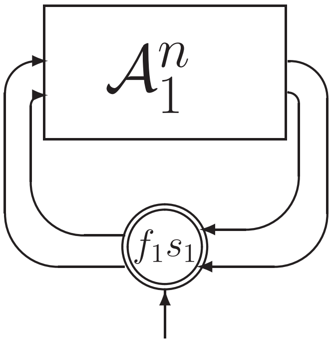


Hier sei f_1s_2 ein durch Identifizierung von f_1 und s_2 entstandener Zustand. Die Mengen ausgezeichneter Start- und Finalzustände sind wie folgt definiert:

$$S = \begin{cases} \{s_1\} \cup \{f_1s_2\} & \text{falls } \varepsilon \in L(\mathcal{A}_1) \\ \{s_1\} & \text{sonst} \end{cases}$$

$$F = \begin{cases} \{f_2\} \cup \{f_1s_2\} & \text{falls } \varepsilon \in L(\mathcal{A}_2) \\ \{f_2\} & \text{sonst} \end{cases}$$

e) Sei $\mathcal{A}_1^n = \langle Q_1, \Sigma, \delta_1, \{s_1\}, \{f_1\}\rangle$ normalisierte Version von $\mathcal{A}_1$. Die Konstruktion von $\mathcal{A}$ ist durch folgendes Schema gegben:



Hier sei f_1s_1 ein durch Identifizierung von f_1 und s_1 entstandener Zustand.

Bemerkung:

Wegen Theorem 2.9 gilt die Behauptung natürlich auch für DEAen, d.h. die Menge der DEA-akzeptierten Sprachen ist ebenfalls unter $\cup$,$\setminus$,$\cap$,$\circ$,$*$ abgeschlossen.

3 Reguläre Sprachen und reguläre Ausdrücke

Reguläre Sprachen sind Sprachen, die unabhängig von automatentheoretischen Konzeptionen durch bestimmte Abschlußeigenschaften charakterisiert sind und sich durch die Wörter einer speziellen (nichtregulären) Sprache, die regulären Ausdrücke, beschreiben lassen. Die regulären Sprachen sind genau die von endlichen Automaten akzeptierbaren Sprachen.

Definition 3.1 (Reguläre Sprachen)

Die Menge aller regulären Sprachen über Σ ist induktiv wie folgt definiert:

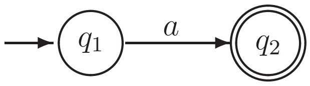
1. $\emptyset$ ist eine reguläre Sprache
2. Für jedes $a \in \Sigma$ ist $\{a\}$ eine reguläre Sprache
3. Sind L_1 und L_2 reguläre Sprachen, dann auch $L_1 \cup L_2, L_1 \circ L_2, L_1^*$
4. Nichts sonst ist eine reguläre Sprache

Theorem 3.2 (Kleene/Myhill)

L ist DEA-akzeptabel $\Leftrightarrow$ L ist regulär

Beweis:

„$\Leftarrow$“: $\emptyset$ wird akzeptiert durch einen DEA mit $F = \emptyset$.

$\{a\}$ wird durch  akzeptiert. Der Rest folgt aus Lemma 2.15.

„$\Rightarrow$“: Sei $\mathcal{A} = \langle Q, \Sigma, \delta, s, F\rangle$ DEA, $Q = \{q_1, \ldots, q_n\}$, $s = q_1$.
Sei $P_{i,j}^k := \{w \in \Sigma^* : (q_i, w) = q_j$, wobei beim Weg von q_i nach q_j kein q_l mit $l > k$ berührt wird$\}$
Formell: $P_{i,j}^k := \{w \in \Sigma^* : \delta^*(q_i, w) = q_j$, und für kein echtes Präfix u von w gilt $\delta^*(q_i, u) = q_l$ für $l > k\}$

Wir zeigen:

$\otimes$ $P_{i,j}^k$ ist regulär

Daraus folgt dann: $L(\mathcal{A})$ ist regulär, da $L(\mathcal{A}) = \bigcup_{q_j \in F} P_{1,j}^n$ (da n die Anzahl der Zustände ist, gibt es kein q_l mit $l > n$).

<u>Beweis von $\otimes$:</u>
$R_{i,j}^0 := \{a \in \Sigma : \delta(q_i, a) = q_j\} \cup \{\varepsilon : i = j\}$

$R_{i,j}^{k+1} := R_{i,j}^k \cup (R_{i,k+1}^k \circ (R_{k+1,k+1}^k)^* \circ R_{k+1,j}^k)$

Offenbar gilt: $R^k_{i,j}$ regulär

Behauptung: $P^k_{i,j} = R^k_{i,j}$

<u>Beweis:</u>
$\supseteq$: $k=0$: Bei unmittelbaren Übergängen gibt es keine echten Präfixe.
Sei nun $w = w_1 \circ (w_{21} \circ \cdots \circ w_{2m}) \circ w_3$ mit $w_1 \in R^k_{i,k+1},\ w_{2p} \in R^k_{k+1,k+1},\ w_3 \in R^k_{k+1,j}$.
Es gilt $\delta^*(q_i, w_1) = \delta^*(q_1, w_1 w_{2p}) = q_{k+1}$ für $1 \leq p \leq m$
Für alle anderen Präfixe u von w gilt $\delta^*(q_i, u) = q_r,\ r \leq k$ nach I.V. Also ist $w \in P^{k+1}_{i,j}$.

$\subseteq$: $k=0$: wie vorhin
Betrachte nun alle Durchläufe von q_{k+1} bei Lektüre von w. Wenn q_{k+1} gar nicht durchlaufen wird, ist $w \in P^k_{i,j}$ und damit nach I.V. $w \in R^k_{i,j}$. Sonst hat w die obige Form $w = w_1 w_{21} \ldots w_{2m} w_3$, wobei nur an den Endpunkten von $w_1, w_{21}, \ldots, w_{2m}$ der Zustand q_{k+1} durchlaufen wird, dazwischen höchstens q_k.
D.h. es gilt $w_1 \in P^k_{i,k+1}$, $w_{2p} \in P^k_{k+1,k+1}$ $(1 \leq p \leq m)$, $w_3 \in P^k_{k+1,j}$. Mit Anwendung der I.V. auf w_1, w_{2p} und w_3 ergibt sich $w \in R^k_{i,k+1} \circ (R^k_{k+1,k+1})^* \circ R^k_{k+1,j}$.

Ein Beispiel für diese Konstruktion wird unten hinter Korollar 3.6 gegeben.

Definition 3.3 (Reguläre Ausdrücke)

Die Menge R der regulären Ausdrücke (RA) über Σ ist eine Sprache über $\Sigma \cup \{\emptyset, \cup, \circ, ^*, (,)\}$, die wie folgt definiert ist:

1. $\emptyset$ ist ein RA
2. Jedes $a \in \Sigma$ ist ein RA
3. Mit α, β sind auch $(\alpha \cup \beta)$ und $(\alpha \circ \beta)$ RAe
4. Mit α ist auch α^* ein RA
5. Nichts sonst ist ein RA

Variablen für RA: α, β, γ

Bemerkung:

1. Alternative Notation: „+“ statt „$\cup$“
   „$\cdot$“ statt „$\circ$“
2. In Hilfsmitteln wie `grep`, `egrep`, `lex`, die der Erkennung von Ausdrücken dienen, wird eine erweiterte Syntax regulärer Ausdrücke geboten. In neueren Musterekennern, etwa innerhalb Perl und Qt, führt eine solche Erweiterung sogar zu einer Verwendung des Terminus „regulärer Ausdruck“, die nicht nur hinsichtlich der Syntax, sondern

auch hinsichtlich der prinzipiellen Leistungsfähigkeit über das hinausgeht, was durch reguläre Ausdrücke im strikten Sinne beschreibbar ist (nämlich reguläre Sprachen), z.B. durch die Verwendung von Rückverweisen auf Zeichenketten unbeschränkter Länge. Es ist also eine gewisse Vorsicht geboten bei der Begegnung mit dem Terminus „regulärer Ausdruck“: Praktiker verstehen darunter häufig etwas anderes als Theoretiker.

Lemma 3.4 (Darstellung regulärer Sprachen)

R stellt die Menge der regulären Sprache dar, d.h. es gibt eine surjektive Funktion

$\langle\cdot\rangle : R \to \{L : \ L \text{ regulär } \}$

Genauer ist $\langle\cdot\rangle$ ein Homormophismus bezüglich der Operationen in R und der Abschlußoperationen auf der Menge der regulären Sprachen.

Beweis: Setze

$$\begin{aligned}
\langle\emptyset\rangle &:= \emptyset \\
\langle a\rangle &:= \{a\} \text{ für jedes } a \in \Sigma \\
\langle(\alpha \cup \beta)\rangle &:= \langle\alpha\rangle \cup \langle\beta\rangle \\
\langle(\alpha \circ \beta)\rangle &:= \langle\alpha\rangle \circ \langle\beta\rangle \\
\langle\alpha^*\rangle &:= \langle\alpha\rangle^*
\end{aligned}$$

Beispiel:

$\langle(ab)^*\rangle = \langle ab\rangle^* = (\langle a\rangle\langle b\rangle)^* = (\{a\}\{b\})^* = (\{ab\})^* = \{(ab)^n, n \geq 0\}$

Bemerkungen:

1. Im Zusammenhang mit regulären Ausdrücken sind $\emptyset, \cup, \circ, ^*$ bloße Zeichen, keine Operationen auf Wörtern oder Wortmengen. Insofern bedeuten sie auf der linken und rechten Seite obiger Gleichungen etwas verschiedenes.

2. Die Verknüpfung $\circ$ bindet stärker als $\cup$, kann also weggelassen werden. Außenklammern können weggelassen werden. Man schreibt ε für $\emptyset^*$ (ε stellt also die Sprache $\{\varepsilon\}$ dar).

3. R ist selbst keine reguläre Sprache (siehe unten Beispiel 2 zu Theorem 3.16).

4. Wir identifizieren häufig einen RA α mit der durch ihn dargestellten regulären Sprache $\langle\alpha\rangle$, falls sich aus dem Kontext ergibt, was gemeint ist.

Definition 3.5

Sind α und β RAe, dann bedeute $\alpha = \beta$, daß $\langle\alpha\rangle = \langle\beta\rangle$.

Beispiele:

$\alpha \cup \alpha = \alpha : \langle\alpha \cup \alpha\rangle = \langle\alpha\rangle \cup \langle\alpha\rangle = \langle\alpha\rangle$.

$\emptyset \circ \alpha = \alpha \circ \emptyset = \emptyset : \langle\emptyset \circ \alpha\rangle = \langle\emptyset\rangle \circ \langle\alpha\rangle = \emptyset \circ \langle\alpha\rangle = \emptyset = \langle\emptyset\rangle$.

$\varepsilon \circ \alpha = \alpha$ analog.

$(\alpha \cup \beta) \cup \gamma = \alpha \cup (\beta \cup \gamma),\ (\alpha \circ \beta) \circ \gamma = \alpha \circ (\beta \circ \gamma)$ (d.h. bei iteriertem $\circ$ und $\cup$ können Klammern entfallen).

$\alpha \cup \beta = \beta \cup \alpha$

$(\alpha \cup \varepsilon)^* = (\alpha \cup \varepsilon)(\alpha \cup \varepsilon)^* = (\alpha \cup \varepsilon)(\alpha \cup \varepsilon)^*(\alpha \cup \varepsilon) = \alpha^*$

Korollar 3.6

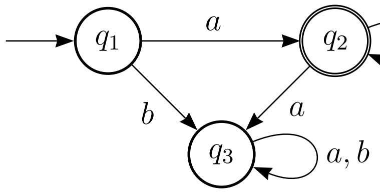
Zu jedem DEA gibt es einen RA α, so daß $L(\mathcal{A}) = \langle\alpha\rangle$.

Beweis:

Fasse im Beweis von Theorem 3.2 die $R^k_{i,j}$ als RAe auf, wobei $R^0_{i,j} = a_1 \cup \cdots \cup a_n$, falls im alten Sinne $R^0_{i,j} = \{a_1, \ldots, a_n\}$ (hier kann ein a_l das leere Wort ε sein, falls nämlich $i = j$);

$R^{k+1}_{i,j}$= wie vorher.

Beispiel: Kleene-Myhill-Konstruktion für folgenden DEA $\mathcal{A}$ über $\{a, b\}$:



$R^0_{1,1} = \varepsilon$
$R^0_{1,2} = a$
$R^0_{1,3} = b$
$R^0_{2,1} = \emptyset$
$R^0_{2,2} = b \cup \varepsilon$
$R^0_{2,3} = a$
$R^0_{3,1} = \emptyset$
$R^0_{3,2} = \emptyset$
$R^0_{3,3} = a \cup b \cup \varepsilon$
$R^1_{1,1} = R^0_{1,1} \cup (R^0_{1,1} \circ (R^0_{1,1})^* \circ R^0_{1,1}) = \varepsilon \cup \varepsilon = \varepsilon$
$R^1_{1,2} = R^0_{1,2} \cup (R^0_{1,1} \circ (R^0_{1,1})^* \circ R^0_{1,2}) = a \cup (\varepsilon \circ a) = a$
$R^1_{1,3} = R^0_{1,3} \cup (R^0_{1,1} \circ (R^0_{1,1})^* \circ R^0_{1,3}) = b \cup (\varepsilon \circ b) = b \cup b = b$
$R^1_{2,1} = R^0_{2,1} \cup (R^0_{2,1} \circ (R^0_{1,1})^* \circ R^0_{1,1}) = \emptyset \cup (\emptyset \circ \varepsilon) = \emptyset$
$R^1_{2,2} = R^0_{2,2} \cup (R^0_{2,1} \circ (R^0_{1,1})^* \circ R^0_{1,2}) = b \cup \varepsilon \cup (\emptyset \circ \varepsilon \circ a) = b \cup \varepsilon \cup \emptyset = b \cup \varepsilon$
$R^1_{2,3} = R^0_{2,3} \cup (R^0_{2,1} \circ (R^0_{1,1})^* \circ R^0_{1,3}) = a \cup (\emptyset \circ \varepsilon \circ b) = a \cup \emptyset = a$
$R^1_{3,1} = R^0_{3,1} \cup (R^0_{3,1} \circ (R^0_{1,1})^* \circ R^0_{1,1}) = \emptyset \cup (\emptyset \circ \varepsilon) = \emptyset$
$R^1_{3,2} = R^0_{3,2} \cup (R^0_{3,1} \circ (R^0_{1,1})^* \circ R^0_{1,2}) = \emptyset \cup (\emptyset \circ \varepsilon \circ a) = \emptyset$
$R^1_{3,3} = R^0_{3,3} \cup (R^0_{3,1} \circ (R^0_{1,1})^* \circ R^0_{1,3}) = a \cup b \cup \varepsilon \cup (\emptyset \circ \varepsilon \circ b) = a \cup b \cup \varepsilon \cup \emptyset = a \cup b \cup \varepsilon$
$R^2_{1,1} = R^1_{1,1} \cup (R^1_{1,2} \circ (R^1_{2,2})^* \circ R^1_{2,1}) = \varepsilon \cup (a \circ (b \cup \varepsilon)^* \circ \emptyset) = \varepsilon$
$R^2_{1,2} = R^1_{1,2} \cup (R^1_{1,2} \circ (R^1_{2,2})^* \circ R^1_{2,2}) = a \cup a(b \cup \varepsilon)^*(b \cup \varepsilon) = ab^*$
$R^2_{1,3} = R^1_{1,3} \cup (R^1_{1,2} \circ (R^1_{2,2})^* \circ R^1_{2,3}) = b \cup a(b \cup \varepsilon)^* a = b \cup ab^* a$
$R^2_{2,1} = R^1_{2,1} \cup (R^1_{2,2} \circ (R^1_{2,2})^* \circ R^1_{2,1}) = \emptyset \cup (b \cup \varepsilon)(b \cup \varepsilon)^* \emptyset = \emptyset$
$R^2_{2,2} = R^1_{2,2} \cup (R^1_{2,2} \circ (R^1_{2,2})^* \circ R^1_{2,2}) = (b \cup \varepsilon) \cup (b \cup \varepsilon)(b \cup \varepsilon)^*(b \cup \varepsilon) = b^*$
$R^2_{2,3} = R^1_{2,3} \cup (R^1_{2,2} \circ (R^1_{2,2})^* \circ R^1_{2,3}) = a \cup (b \cup \varepsilon)(b \cup \varepsilon)^* a = b^* a$
$R^2_{3,1} = R^1_{3,1} \cup (R^1_{3,2} \circ (R^1_{2,2})^* \circ R^1_{2,1}) = \emptyset \cup \emptyset(b \cup \varepsilon)^* \emptyset = \emptyset$
$R^2_{3,2} = R^1_{3,2} \cup (R^1_{3,2} \circ (R^1_{2,2})^* \circ R^1_{2,2}) = \emptyset \cup \emptyset(b \cup \varepsilon)^* b = \emptyset$
$R^2_{3,3} = R^1_{3,3} \cup (R^1_{3,2} \circ (R^1_{2,2})^* \circ R^1_{2,3}) = a \cup b \cup \varepsilon \cup \emptyset(b \cup \varepsilon)^* a = a \cup b \cup \varepsilon$
$L(\mathcal{A}) = R^3_{1,2} = R^2_{1,2} \cup (R^2_{1,3} \circ (R^2_{3,3})^* \circ R^2_{3,2}) = ab^* \cup ((b \cup ab^* a)(a \cup b \cup \varepsilon)^* \emptyset) = ab^* \cup \emptyset = ab^* = \{ab^n : n \geq 0\}$

Die Kleene-Myhill-Konstruktion ist ein effektives (wenn auch nicht sehr effizientes) Verfahren, einen RA zu einem DEA zu erzeugen. Ein anderes Verfahren besteht in der Lösung von Gleichungssystemen. Dem letzten Beispiel entspricht z.B. das Gleichungssystem:

$$
\begin{aligned}
X_1 &= aX_2 \;\cup\; bX_3 \\
X_2 &= aX_3 \;\cup\; bX_2 \;\cup\; \varepsilon \\
X_3 &= aX_3 \;\cup\; bX_3
\end{aligned}
$$

Die Lösung für X_1 (Anfangszustand) ergibt den gesuchten RA.
Man benutzt Ardens Regel:
$X = uX \cup v$ hat die Lösung $X = u^*v$, denn $u^*v = u(u^*v) \cup v$.
X ist eindeutig, falls ε nicht in der Sprache $\langle u\rangle$ enthalten ist (siehe Aufgabe).

Im vorliegenden Fall:

$$
X_3 = (a \cup b)X_3
$$

$$
\begin{aligned}
\Longrightarrow\; X_3 &= \emptyset, \text{ da } \emptyset = w \circ \emptyset \\
\Longrightarrow\; X_2 &= bX_2 \cup \varepsilon \\
\Longrightarrow\; X_2 &= b^*\varepsilon \;=\; b^* \\
\Longrightarrow\; X_1 &= ab^* \cup \emptyset \\
&= ab^*
\end{aligned}
$$

Man kann zeigen, daß durch sukzessive Auflösung mit Ardens Regel jedes solche Gleichungssystem einen RA als Lösung hat.
Eine gut lesbare systematische Darstellung des hier nur exemplarisch beschriebenen Verfahrens findet sich bei Kozen (1997, Suppl. Lecture A, pp. 55–60).

Anderes Beispiel:

Zweites Beispiel hinter Def. 2.2:

$$
\begin{aligned}
X_1 &= aX_2 \;\cup\; bX_4 \\
X_2 &= aX_2 \;\cup\; bX_3 \\
X_3 &= bX_3 \;\cup\; aX_4 \;\cup\; \varepsilon \\
X_4 &= aX_4 \;\cup\; bX_4
\end{aligned}
$$

Schematisches Verfahren (von oben nach unten Auflösen und Einsetzen):

1.Schritt: Rekursivität mit Arden beseitigen:

$$
\begin{array}{lcll}
X_1 & = & aX_2 \;\cup\; bX_4 & \text{(unverändert)} \\
X_2 & = & a^*bX_3 & \text{(Arden)} \\
X_3 & = & b^*(aX_4 \cup \varepsilon) & \text{(Arden)} \\
X_4 & = & \emptyset &
\end{array}
$$

2.Schritt: Einsetzung:

$$
\begin{array}{lcl}
X_1 & = & aa^*bX_3 \cup \emptyset \\
 & = & aa^*bb^*(a\emptyset \cup \varepsilon) \\
 & = & aa^*bb^*\varepsilon \\
 & = & aa^*bb^*
\end{array}
$$

Umgekehrt gibt es neben dem im Beweis von Theorem 3.2 enthaltenen, auf den Konstruktionen von Lemma 2.10-2.12 und Lemma 2.14 aufbauenden Verfahren weitere effektive Methoden, zu einem RA einen korrespondierenden DEA zu erzeugen. Das sog. Verfahren von Gluschkow ist bei Winter (2002, Kap. 3.4.1, S. 88ff.) erläutert.

Theorem 3.7 (Pumping Lemma)

Sei $L = L(\mathcal{A})$ für DEA $\mathcal{A}$ mit n Zuständen. Sei $w \in L$ mit $|w| \geq n$. Dann läßt sich w schreiben als $w = uxv$ mit $x \neq \varepsilon$, wobei gilt: $ux^kv \in L$ für alle $k \in \mathbb{N}$ (einschließlich $k = 0$).

Beweis:

Sei $\mathcal{A} = \langle Q, \Sigma, \delta, s, F \rangle$ mit $|Q| = n$
Da $|w| \geq n$, gibt es mindestens einen Zustand q, der bei Lektüre von w ausgehend von s mehr als einmal durchlaufen wird.
D.h.: Es gibt u, x, v mit $w = uxv$ und $x \neq \varepsilon$, so daß gilt:
$\delta^*(s, u) = q$; $\delta^*(q, x) = q$; $\delta^*(q, v) = f \in F$ (wobei $u = \varepsilon$ und/oder $v = \varepsilon$ sein kann). Die durch x bestimmte Schleife von q nach q kann nun beliebig oft (einschließlich überhaupt nicht) durchlaufen werden, da $\delta^*(q, x^k) = q$. Also wird ux^kv für beliebiges k akzeptiert.

Bemerkung:

Man benutzt folgendes Prinzip: Wenn mehr als n Objekte auf n Mengen aufgeteilt werden, enthält eine Menge mindestens zwei Elemente („Dedekindsches Schubfächerprinzip“, „pigeon-hole principle“). Beweise für dieses Prinzip sind komplexitätstheoretisch von Interesse.

Anwendungsbeispiel:

$\{a^n b^n : n \geq 0\}$ ist nicht regulär:
Für geeignet großes n stelle man $a^n b^n$ als uxv im Sinne des Pumping Lemmas dar.
Für x bestehen drei Möglichkeiten: $x = a^i$, $x = b^i$ oder $x = a^i b^j$
In allen Fällen ist schon ux^2v nicht mehr von der Form $a^m b^m$.

Korollar 3.8

Ist L unendliche reguläre Sprache, dann gibt es $uxw \in L$ mit $x \neq \varepsilon$, so daß $ux^k w \in L$ für alle $k \in \mathbb{N}$.

Beweis:

Da L unendlich, gibt es Wörter beliebig großer Länge, so daß das Pumping Lemma auf jeden Fall angewendet werden kann.

Korollar 3.9

Sei $L = L(\mathcal{A})$, wobei $\mathcal{A}$ n Zustände hat.
Falls $L \neq \emptyset$, gibt es ein $w \in L$ mit $|w| < n$.

Beweis:

Sei w ein Wort kleinster Länge in L. Falls $|w| \geq n$, ist $w = uxv$ mit $x \neq \varepsilon$ und $ux^k v \in L$, d.h. $uv \in L$ (für k=0). Da $|uv| < n$, Widerspruch zur Wahl von w. Also gilt $|w| < n$.

Korollar 3.10

$L(\mathcal{A}) = \emptyset$ ist entscheidbar, d.h. es gibt einen Algorithmus, der für DEA $\mathcal{A}$ entscheidet, ob $L(\mathcal{A}) = \emptyset$.

Beweis:

Sei n die Anzahl der Zustände von $\mathcal{A}$. Wenn $\mathcal{A}$ kein Wort der Länge $< n$ akzeptiert, dann ist nach Korollar 3.9 $L(\mathcal{A}) = \emptyset$. (Die Umkehrung ist trivial.) Also muß man nur von allen Wörtern der Länge $< n$ prüfen, ob sie akzeptiert werden.

Korollar 3.11

Sei $L = L(\mathcal{A})$, wobei $\mathcal{A}$ n Zustände hat.
L unendlich gdw $\exists w \in L : n \leq |w| < 2n$.

Beweis:

„$\Leftarrow$“: Pumping Lemma: Nur $n \leq |w|$ nötig.

„$\Rightarrow$“: Sei $u \in L$ von kleinster Länge $\geq 2n$, d.h. $|u| \geq 2n$ und es gibt kein $v \in L$ mit $2n \leq |v| \leq |u|$. Wegen der Unendlichkeit von L existiert ein solches u. Schreibe u als $u = u_1 u_2$ mit $|u_1| = n,\ |u_2| \geq n$. Analog zum Beweis des Pumping Lemmas läßt sich u_1 schreiben als $u_1 = v_1 x v_2$ mit $x \neq \varepsilon$ und $v_1 x v_2 u_2 \in L$, d.h. inbesondere $v_1 v_2 u_2 \in L$. (Der Unterschied zum Beweis des Pumping Lemmas ist einzig, daß nach Durchlaufen von u_1 nicht notwendig ein Endzustand erreicht wird.) Es gilt $|v_1 v_2 u_2| < 2n$, da $|v_1 v_2 u_2| < |u|$ und $|u|$ *minimal* $\geq 2n$. Ferner $|v_1 v_2 u_2| \geq |u_2| \geq n$, d.h. $n \leq |v_1 v_2 u| < 2n$.

Korollar 3.12

Für reguläre Sprachen L ist entscheidbar, ob L unendlich ist.

Beweis:

Entsprechend Korollar 3.11 prüft man der Reihe nach bei allen Wörtern, deren Länge zwischen n und $2n$ liegt, ob sie zu L gehören. Falls man auf ein solches Wort stößt, ist L unendlich.

Definition 3.13 (Syntaktische Kongruenz)

Sei $L \subseteq \Sigma^*$

Wörter $u, v \in \Sigma^*$ heißen syntaktisch kongruent bezüglich L ($u \equiv_L v$), falls für alle $x \in \Sigma^*$ gilt: $ux \in L \Leftrightarrow vx \in L$

Lemma 3.14 (Eigenschaften von $\equiv_L$)

a) $\equiv_L$ ist eine Äquivalenzrelation

b) $\forall a \in \Sigma :\ u \equiv_L v \ \Rightarrow ua \equiv_L va$

Beweis:

a) Leicht zu prüfen

b) Ersetze x durch ax als Suffix von u und v gemäß Definition 3.13.

Definition 3.15 (Endlicher Index)

Die Relation $\equiv_L$ der syntaktischen Kongruenz hat endlichen Index, falls sie nur endlich viele Äquivalenzklassen erzeugt.

Theorem 3.16 (Myhill/Nerode)

L ist regulär $\Leftrightarrow$ $\equiv_L$ hat endlichen Index

Beweis:

„$\Rightarrow$": Sei $L = L(\mathcal{A})$ mit DEA $\mathcal{A} = \langle Q, \Sigma, \delta, s, F\rangle$. Es gilt:

$u \equiv_L v$ g.d.w. $(\forall x)\delta^*(s, ux) \in F \Leftrightarrow (\forall x)\delta^*(s, vx) \in F$

g.d.w. $(\forall x)\delta^*(\delta^*(s, u), x) \Leftrightarrow (\forall x)\delta^*(\delta^*(s, v), x) \in F$

g.d.w. $\delta^*(s, u)$ und $\delta^*(s, v)$ sind äquivalent.

Da wir annehmen können, daß $\mathcal{A}$ reduziert ist, gilt damit: $u \equiv_L v \Leftrightarrow \delta^*(s, u) = \delta^*(s, v)$.
Also ist $\equiv_L$ von endlichem Index, da die Anzahl der Zustände von $\mathcal{A}$ endlich ist.

„$\Leftarrow$": Wir definieren einen DEA $\mathcal{A}$.
Mit $[w]$ bezeichnen wir die Äquivalenzklasse von w bezüglich $\equiv_L$, d.h. ein Element von $\Sigma^*/\equiv_L$.
$Q := \Sigma^*/\equiv_L$ (Q ist endlich, da $\equiv_L$ von endlichem Index ist)
$s := [\varepsilon]$
$F := \{[w] | w \in L\}$
$\delta([w], a) := [wa]$
Es gilt: $\delta^*([w], u) = [wu]$ (leichte Induktion). Damit gilt:

$$
\begin{aligned}
w \in L(\mathcal{A}) \quad & \text{gdw } \delta^*(s, w) \in F \\
& \text{gdw } \delta^*([\varepsilon], w) \in F \\
& \text{gdw } [\varepsilon w] \in F \\
& \text{gdw } w \in L, \\
& \text{d.h. } L = L(\mathcal{A}).
\end{aligned}
$$

Bemerkung:

Das Theorem von Myhill/Nerode dient insbesondere dem Nachweis der Nicht-Regularität, wie die folgenden beiden Beispiele demonstrieren.

Beispiel 1: $L = \{a^n b^n : n \geq 0\}$

$$
\left.\begin{array}{lll} a & bb & \notin L \\ aa & bb & \in L \end{array}\right\} a \not\equiv_L aa, \text{ d.h. } [a] \neq [aa]
$$

$$\left.\begin{array}{lll} aa & bb & \in L \\ aaa & bb & \notin L \end{array}\right\} aa \not\equiv_L aaa, \text{ d.h. } [aa] \neq [aaa]$$

$$\left.\begin{array}{lll} a & b & \in L \\ aaa & b & \notin L \end{array}\right\} a \not\equiv_L aaa, \text{ d.h. } [a] \neq [aaa]$$

$$\vdots$$

Es gibt unendlich viele Äquivalenzklassen bezüglich $\equiv_L$, d.h. L ist nicht regulär.

Beispiel 2:

Die Menge aller regulären Ausdrücke ist nicht regulär.

$$\left.\begin{array}{lll} (& \emptyset \circ \emptyset) \circ \emptyset) & \notin \text{RA} \\ ((& \emptyset \circ \emptyset) \circ \emptyset) & \in \text{RA} \end{array}\right\}$$

$$\left.\begin{array}{lll} ((& \emptyset \circ \emptyset) \circ \emptyset) & \in \text{RA} \\ (((& \emptyset \circ \emptyset) \circ \emptyset) & \notin \text{RA} \end{array}\right\}$$

$$\left.\begin{array}{lll} (& \emptyset \circ \emptyset) & \in \text{RA} \\ (((& \emptyset \circ \emptyset) & \notin \text{RA} \end{array}\right\}$$

$$\vdots$$

(Argument analog zu Beispiel 1)

Bemerkungen:

1. Wir haben hier eine weitere Charakterisierung der regulären Sprachen vorliegen.

2. Man kann $\equiv_L$ auch so charakterisieren:
   $\equiv_L$ ist die gröbste Äquivalenzrelation auf Σ^* mit den Eigenschaften:

   (i) $u \sim v \Rightarrow ua \sim va$ für alle $a \in \Sigma$ (Rechtskongruenz)

   (ii) $u \sim v \Rightarrow (u \in L \Leftrightarrow v \in L)$

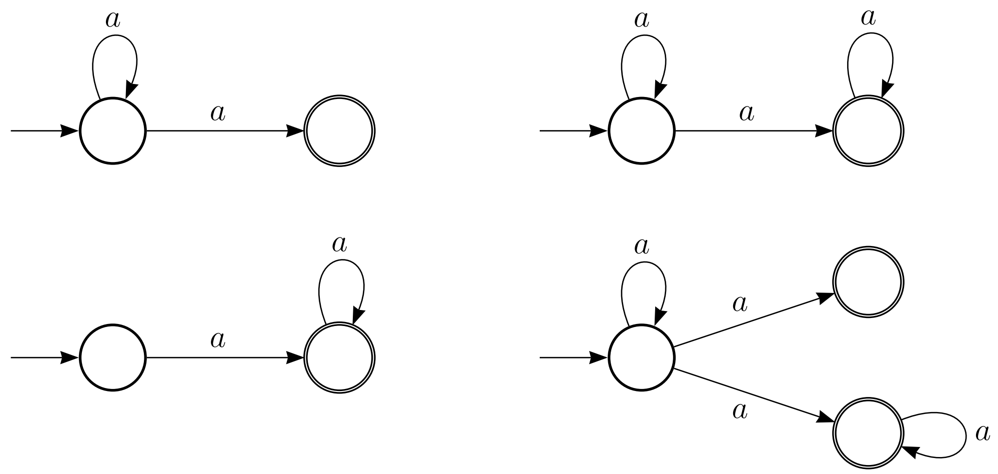
   (iii) $\sim$ hat endlichen Index.

   Eine Relation $\sim$ mit den Eigenschaften (i)-(iii) heißt auch Myhill-Nerode-Relation. (Daß $\equiv_L$ die gröbste Myhill-Nerode-Relation ist, bedeutet: $u \sim v \Rightarrow u \equiv_L v$.)

3. Die Richtung „$\Leftarrow$" des Beweises von Theorem 3.16 konstruiert einen *reduzierten* DEA, dessen Zustände die Äquivalenzklassen bzgl. $\equiv_L$ sind. Die Richtung „$\Rightarrow$" zeigt, daß die Zustände eines *beliebigen reduzierten* DEA eineindeutig den Äquivalenzklassen bzgl. $\equiv_L$ entsprechen. Zusammengenommen impliziert dies, daß alle *reduzierten* DEAen, die eine Sprache L akzeptieren, isomorph sind. Damit erhalten wir als *fundamentale Eigenschaft regulärer Sprachen*, daß sie durch (bis auf Isomorphie) *eindeutig bestimmte reduzierte DEAen* beschrieben werden können. Für die Darstellung dieser Zusammenhänge sei auf Kozen (1997, pp. 89–99) verwiesen.

Zum Äquivalenzbegriff für NDEAen

Zwei DEAen $\mathcal{A}_1$ und $\mathcal{A}_2$, die äquivalent sind, also dieselbe Sprache akzeptieren, ist ein (bis auf Isormophie) eindeutig bestimmter reduzierter DEA zugeordnet. Für NDEAen gilt dies nicht.
Die NDEAen



akzeptieren allesamt dieselbe Sprache (nämlich $a^*a = aa^*$) und sind allesamt reduziert, keine zwei von ihnen sind jedoch isomorph. Der Äquivalenz im Sinne von Akzeptanz derselben Sprache entspricht nicht mehr die eindeutige Bestimmtheit eines reduzierten Automaten.

Mann kann sich daher fragen, ob es einen *feineren* Äquivalenzbegriff und einen *feineren* Begriff eines reduzierten Automaten gibt derart, daß zwei äquivalenten NDEAen ein eindeutig bestimmter reduzierter NDEA entspricht. Dies läßt sich in der Tat bewerkstelligen, wenn man Äquivalenz von NDEAen nicht über Akzeptanz derselben Sprache, sondern über „Verhaltensäquivalenz“ definiert in dem Sinne, daß ein Automat den anderen *simulieren* kann und umgekehrt — man spricht von „Bisimulation“. Es läßt sich dann zu einem gegebenen NDEA $\mathcal{A}$ ein reduzierter NDEA A' konstruieren, reduziert in dem Sinne, daß er der bis auf Isomorphie eindeutig bestimmte minimale zu $\mathcal{A}$ bisimulare NDEA ist. $\mathcal{A}'$ ist damit eine eindeutige Charakterisierung aller zu A bisimularen (d.h. „verhaltensäquivalenten“ Automaten. Dies wird bei Kozen (1997, Suppl. Lecture B, pp. 100–107) genauer erläutert.

Folgendes von R. Milner (1999, p. 14f.) vorgeschlagene Beispiel motiviert, daß bei NDEAen Akzeptanz derselben Sprache ein zu grober Äquivalenzbegriff ist.

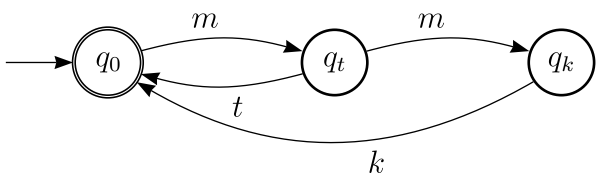
Ein Kaffee-/Tee-Automat soll bei Einwurf einer 50ct-Münze nach Betätigung einer Tee-Taste Tee ausgeben, bei Einwurf von zwei 50ct-Münzen nach Betätigung einer Kaffee-Taste Kaffee.

Sei

$$\begin{aligned} m &:= \text{Eingabe von 50ct-Münze} \\ t &:= \text{Drücken der Tee-Taste} \\ k &:= \text{Drücken der Kaffee-Taste} \end{aligned}$$

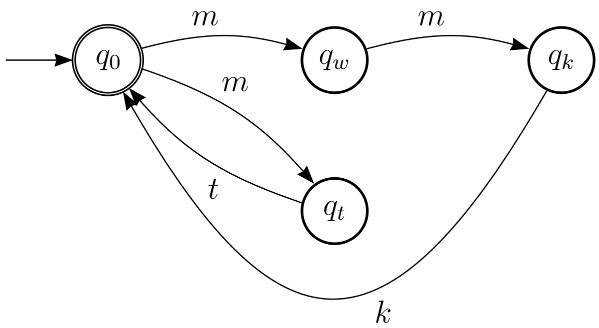
$$\begin{aligned} q_0 &:= \text{Getränkeausgabe und Zurücksetzen} \\ q_t &:= \text{Freigabe der Tee-Taste} \\ q_k &:= \text{Freigabe der Kaffee-Taste} \\ q_w &:= \text{Warten auf Eingabe von zweiter Münze} \end{aligned}$$

Dann würde der durch folgendes Diagramm gegebene NDEA



das Gewünschte leisten.

Folgender NDEA würde jedoch dieselbe Sprache akzeptieren:



nämlich die Sprache $(mt \cup m^2k)^*$
(„für eine Münze Tee oder für zwei Münzen Kaffee").

Trotz dieser Äquivalenz zum ersten NDEA ist er, was sein „Verhalten" angeht, als Kaffee-/Tee-Automat unbrauchbar: Wenn ich Kaffee erhalten möchte, der Automat aber nach Einwurf der ersten Münze in den Zustand q_t übergegangen ist, ist für mich die Münze verloren, es sei denn, ich entschiede mich nachträglich, Tee zu trinken.

Der Begriff der „Bisimulation" spielt in Theorien paralleler Informationsverarbeitung eine wesentliche Rolle.

4 Reguläre Grammatiken

Wir haben eine Klasse von Sprachen beschrieben durch Automaten, die diese Sprachen akzeptieren (DEAs bzw. NDEAs) sowie durch Abschlußoperationen ausgehend von Grundsprachen (reguläre Sprachen, dargestellt durch reguläre Ausdrücke). Wir charakterisieren dieselbe Sprachklasse jetzt durch die Art und Weise, wie Wörter „generiert" oder „produziert" werden.

Definition 4.1 (Grammatik)

Eine Grammatik Γ besteht aus zwei disjunktiven Teilalphabeten $\mathcal{V}$, $\mathcal{T}$, die zusammen das Alphabet Σ ergeben, d.h. $\Sigma = \mathcal{T} \cup \mathcal{V}$ mit $\mathcal{T} \cap \mathcal{V} = \emptyset$, einer endlichen Menge Π von Wortpaaren (u, v) mit $u, v \in \Sigma^*$, die auch „Produktionen" genannt werden und $u \to v$ geschrieben werden, sowie einem ausgezeichneten Startsymbol $S \in \mathcal{V}$, d.h.
$\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$. Die Elemente von $\mathcal{V}$ heißen „Variablen" (=„Nichtterminale"), die von $\mathcal{T}$ „Terminale".

Metaprachliche Variablen:	A, B, C, D, E	für Elemente von $\mathcal{V}$
	a, b, c, d, e	für Elemente von $\mathcal{T}$
	u, v, w, x, y, z	für Wörter aus Σ^*

Eine Produktion der Form $u \to \varepsilon$ heißt ε-Produktion.

Bemerkungen:

1. Statt „Produktion" sagt man auch „Regel".

2. Die Terminologie „Nichtterminale" statt „Variablen" ist zur Zeit noch (?) die Gebräuchlichere. Ich schließe mich der von Logikern vorgeschlagene Verwendung von „Variablen" an. (vgl. z.B. Davis & Weyuker 1983, Schöning 2001)

3. Grammatiken sind Spezialfälle von allgemeinen „Produktionssystemen" (oder „Semi-Thue-Systemen"), in denen man keine Variablen auszeichnet.

Definition 4.2 (Ableitung, erzeugte Sprache, Äquivalenz)

Sei Γ Grammatik. Falls $u \to v$ eine Produktion von Γ, ist für alle x, y das Wort xvy aus dem Wort xuy unmittelbar ableitbar in Γ; symbolisch: $xuy \overset{1}{\underset{\Gamma}{\Rightarrow}} xvy$.

Eine Ableitung in Γ ist eine Folge von Wörtern $w_0, \ldots, w_n$ $(n \geq 0)$, für die gilt: $w_i \overset{1}{\underset{\Gamma}{\Rightarrow}} w_{i+1}$ für alle i mit $0 \leq i < n$. Die Folge $w_0, \ldots, w_n$ heißt auch Ableitung von w_n aus w_0 in Γ.

Dabei heißt n die *Länge* der Ableitung. v ist aus u in Γ in n Schritten ableitbar, symbolisch $u \overset{n}{\underset{\Gamma}{\Rightarrow}} v$, falls es eine Ableitung der Länge n von v aus u in Γ gibt.

v ist aus u in Γ ableitbar, symbolisch: $u \overset{*}{\underset{\Gamma}{\Rightarrow}} v$, falls es eine Ableitung von v aus u in Γ gibt.

v ist in Γ ableitbar, falls v aus S in Γ ableitbar ist, d.h. falls $S \overset{*}{\underset{\Gamma}{\Rightarrow}} v$.

Die von Γ erzeugte Sprache $L(\Gamma)$ ist die Menge der in Γ ableitbaren Wörter über dem Terminalalphabet, d.h. $L(\Gamma) := \{w \in \mathcal{T}^* : S \overset{*}{\underset{\Gamma}{\Rightarrow}} w\}$.

Zwei Grammatiken Γ_1 und Γ_2 heißen äquivalent, falls $L(\Gamma_1) = L(\Gamma_2)$.

Bemerkungen:

1. Γ unter $\Rightarrow$ wird weggelassen, falls aus dem Kontext klar ist, um welches Γ es sich handelt.

2. „$u \overset{1}{\underset{\Gamma}{\Rightarrow}} v$“ ist nicht mehrdeutig. v ist aus u *unmittelbar* ableitbar g.d.w. es eine Ableitung *der Länge 1* von v aus u gibt.

3. Der Ausdruck „Terminal“ wird durch die Definition von $L(\Gamma)$ gerechtfertigt: Variablen haben nur Hilfsfunktion. Am *Ende* einer Ableitung eines Wortes von $L(\Gamma)$ kann nur ein Wort stehen, das aus Terminalen besteht. Welche Funktionen Variablen genau erfüllen, wird aus der Definition spezieller Grammatiktypen klar werden.

4. Grammatiken sind ihrer Intention nach Erzeugungssysteme für Sprachen. Man kann aber auch Akzeptanz definieren, wenn man eine Klasse F von Wörtern auszeichnet. Γ akzeptiert dann ein Wort w, falls $w \overset{*}{\underset{\Gamma}{\Rightarrow}} u$ für ein $u \in F$.

5. Grammatiken können auf zwei Weisen nichtdeterministisch sein:
   a) Zu w gibt es mehrere unmittelbare Nachfolger (d.h. u mit $w \overset{1}{\Rightarrow} u$),
   b) Zu w gibt es mehrere unmittelbare Vorgänger (d.h. u mit $u \overset{1}{\Rightarrow} w$)

Beispiel: $\Gamma_1 =< \{A, B, S\}, \{a, b\}, \Pi, S >$

Π umfasse folgende Produktionen:

1.	$S \to aB$	5.	$A \to bAA$
2.	$S \to bA$	6.	$B \to b$
3.	$A \to a$	7.	$B \to bS$
4.	$A \to aS$	8.	$B \to aBB$

Folgende Folge von Wörtern ist eine Ableitung in Γ_1:
$\underset{2}{S,}\ \underset{4}{bA,}\ \underset{1}{baS,}\ \underset{7}{baaB,}\ \underset{2}{baabS,}\ \underset{3}{baabbA,}\ baabba$
(Die Ziffern geben jeweils an, welche Produktion angewendet wurde.)

Definition 4.3 (Reguläre Grammatik)

Eine Grammatik Γ heißt regulär, falls jede ihrer Produktionen eine der folgenden Formen hat:

$A \to aB,\ A \to a,\ A \to \varepsilon \quad$ (A und B müssen nicht verschieden sein)

Bemerkungen:

1. Man spricht auch von rechtsregulären oder rechtslinearen Grammatiken. Bei $A \to Ba$ statt $A \to aB$ spricht man von linksregulären oder linkslinearen Grammatiken.

2. Offensichtlich gilt: In einer Ableitung aus S in Γ hat das letzte Wort die Form uA oder u für $u \in \mathcal{T}^*$. Jedes frühere Wort hat die Form uA für $u \in \mathcal{T}^*$.

3. Man benutzt die eingeführte Terminologie („regulär“, „rechtslinear“, „linkslinear“) auch für Grammatiken mit Produktionen der Art $A \to wB$, $A \to Bw$ und $A \to w$, wobei $w \in \mathcal{T}^*$ ein *ganzes Wort* ist. Es ist offensichtlich, daß solche Grammatiken mit den hier definierten gleichwertig sind (Aufgabe).

Beispiele:

$\Gamma_2 = \langle \{S\}, \{a, b\}, \{S \to a, S \to ab, b \to bb\}, S\rangle$

$\quad L(\Gamma_2) = \langle ab^* \rangle$

$\Gamma_3 = \langle \{S\}, \{a, b\}, \{S \to a, S \to Sb\}, S\rangle$

$\quad L(\Gamma_3) = L(\Gamma_2) = \langle ab^* \rangle$

$\Gamma_4 = \langle \{S\}, \{a, b\}, \{S \to aA, A \to bA, A \to \varepsilon\}, S\rangle$

$\quad L(\Gamma_4) = L(\Gamma_3) = L(\Gamma_2) = \langle ab^* \rangle$

$\Gamma_5 = \langle \{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\}, S\rangle$

$\quad L(\Gamma_5) = \{a^n b^n : n \geq 0\}$

$\Gamma_2, \Gamma_3, \Gamma_4$ sind äquivalent. Γ_4 ist regulär (oder rechtsregulär), Γ_3 ist linksregulär.

Lemma 4.4 (ε-Produktionen)

Zu jeder regulären Grammatik Γ gibt es eine reguläre Grammatik Γ' ohne ε-Produktion, so daß $L(\Gamma') = L(\Gamma)\backslash\{\varepsilon\}$.

Beweis: Γ' entsteht aus Γ wie folgt:
Streiche jedes $A \to \varepsilon$ und füge stattdessen zu jedem $B \to aA$ die Produktion $B \to a$ hinzu.
Ableitungen in Γ und Γ' werden dann wie folgt ineinander übersetzt:

$$\overbrace{\ldots wB, \underset{B\to a}{\overset{\uparrow}{}}wa}^{\text{Ableitung in }\Gamma'} \quad \leftrightsquigarrow \quad \overbrace{\ldots wB, \underset{B\to aA}{\overset{\uparrow}{}}waA, \underset{A\to\varepsilon}{\overset{\uparrow}{}}wa}^{\text{Ableitung in }\Gamma}$$

ε ist in Γ' nicht ableitbar.
Die Ableitung $S, \underset{S\to\varepsilon}{\overset{\uparrow}{}}\varepsilon$ ist nicht in Γ' übersetzbar.

Beispiel:

$\Gamma_6 = \langle\{S\}, \{a, b\}, \{S \to aA, S \to a, A \to bA, A \to b\}, S\rangle$ enthält keine ε-Produktion und entsteht aus Γ_4 gemäß dem im Beweis beschriebenen Verfahren.

Bemerkung:

Offensichtlich ist $L(\Gamma') = L(\Gamma)$, falls $S \to \varepsilon$ nicht zu den Produktionen von Γ gehört.
Fügt man zu Γ' für eine neue Variable S' die Produktionen $S' \to \varepsilon$ und $S' \to S$ hinzu und erklärt S' zum Startsymbol, dann gilt $L(\Gamma') = L(\Gamma)$.
D.h., es gibt zu jeder regulären Gramatik Γ eine äquivalente reguläre Grammatik Γ', die höchstens für das Startsymbol S' die ε-Produktion $S' \to \varepsilon$ enthält, wobei S' auf keiner rechten Seite einer Produktion vorkommt. Man sagt auch, Γ' sei ε-frei.
ε-Produktionen sind also nicht wesentlich. Manche Autoren lassen sie definitorisch weg.

Theorem 4.5 (Reguläre Sprachen und reguläre Grammatiken)

Die regulären Sprachen sind genau die von regulären Grammatiken erzeugten Sprachen.

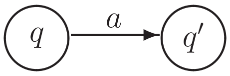
Beweis:

„$\Rightarrow$“: Sei $L = L(\mathcal{A})$, wobei $\mathcal{A} =< Q, \Sigma, \delta, s, F >$ DEA.

Sei $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$ definiert durch:

$\mathcal{V} := Q \quad \mathcal{T} := \Sigma \quad S := s \quad \Pi := \{q \to aq' : \delta(q, a) = q'\} \cup \{q \to \varepsilon : q \in F\}$

Das heißt:

 wird zu $q \to aq'$

 wird zu $q' \to \varepsilon$

Wir zeigen:
$\otimes \quad S \overset{*}{\Rightarrow} wq$ gdw $q = \delta^*(s, w)$

$$
\begin{array}{rcl}
\text{Daraus folgt dann: } w \in L(\mathcal{A}) & \Rightarrow & q = \delta^*(s,w) \in F \\
& \Rightarrow & S \overset{*}{\Rightarrow} wq \text{ (mit } \otimes) \text{ und } q \to \varepsilon \in \Pi \\
& \Rightarrow & w \in L(\Gamma) \\
w \in L(\Gamma) & \Rightarrow & S \overset{*}{\Rightarrow} w \\
& \Rightarrow & S \overset{*}{\Rightarrow} wq \text{ für ein } q \text{ mit } q \to \varepsilon \in \Pi \\
& & \text{(Definition von } \Gamma, \text{ da } w \in \mathcal{T}^*) \\
& \Rightarrow & q = \delta^*(s,w) \text{ (mit } \otimes) \\
& \Rightarrow & w \in L(\mathcal{A})
\end{array}
$$

Beweis von $\otimes$: Induktion über $|w|$

$$
\begin{array}{llcl}
w = \varepsilon: & \text{Gilt wegen } S = s & & \\
w = ua: & q = \delta^*(s,ua) & \Rightarrow & \delta(\delta^*(s,u),a) = q \\
& & \Rightarrow & \delta^*(s,u) \to aq \in \Pi \\
& & \Rightarrow & S \overset{*}{\Rightarrow} uaq, \text{ da } S \overset{*}{\Rightarrow} u\delta^*(s,u) \text{ nach I.V.} \\
& \text{Umgekehrt} & & \\
& S \overset{*}{\Rightarrow} uaq & \Rightarrow & S \overset{*}{\Rightarrow} uq' \text{ mit } \delta(q',a) = q \\
& & \Rightarrow & q' = \delta^*(s,u) \text{ nach I.V.} \\
& & \Rightarrow & q = \delta(\delta^*(s,u),a) \\
& & \Rightarrow & q = \delta^*(s,ua)
\end{array}
$$

„$\Leftarrow$“: Sei $L = L(\Gamma)$ mit $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$
Wir setzen voraus, daß Γ im Sinne von Lemma 4.4 keine ε-Produktion enthält.
Der NDEA $\mathcal{A} =< Q, \Sigma, \delta, S, F >$ sei definiert durch

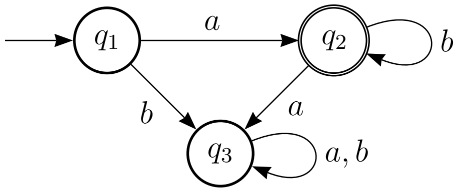
$Q := \mathcal{V} \cup \{f\}$, wobei f neu; $\Sigma = \mathcal{T}$; $\quad S := \{S\} \quad F := \{f\}$

$$
\delta(A,a) := \begin{cases} \{B : A \to aB \in \Pi\} \cup \{f\} & \text{falls } A \to a \in \Pi \\ \{B : A \to aB \in \Pi\} & \text{sonst} \end{cases}
$$

Jeder Ableitung aus S in Γ entspricht ein Pfad durch den Automaten und umgekehrt. Falls $\varepsilon \in L(\Gamma)$, fügen wir S zu F hinzu.
(Formal durchgeführter Beweis von „$\Leftarrow$“ als Übung.)

Beispiel:

$\mathcal{A}$:



$\Sigma = \{a, b\}$; $L(\mathcal{A}) = \{ab^n | n \geq 0\} =< ab^* >$ (siehe Beispiel hinter Korollar 3.6).

Zugehöriges Γ:

$S := q_1$, $\mathcal{T} := \Sigma = \{a, b\}$, $\mathcal{V} := \{q_1, q_2, q_3\}$

$$\begin{array}{llll} \Pi: & q_1 \to aq_2 & q_1 \to bq_3 & \\ & q_2 \to bq_2 & q_2 \to aq_3 & q_2 \to \varepsilon \\ & q_3 \to aq_3 & q_3 \to bq_3 & \end{array}$$

Γ ist äquivalent zu Γ_2, Γ_3 und Γ_4 aus obigen Beispielen.

Umgekehrt konstruieren wir einen DEA aus Γ:

Elimination von ε-Produktionen:

$$\left.\begin{array}{l} q_1 \to a \\ q_2 \to b \end{array}\right\} \text{ersetzen } q_2 \to \varepsilon.$$

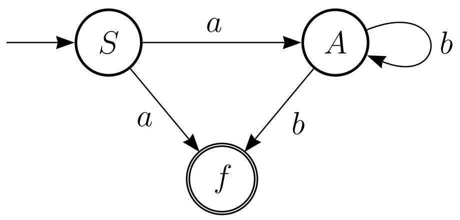
Produktionen mit q_3 können weggelassen werden (q_3 ist nicht terminalisierbar, vgl. unten Definition 5.4 und Lemma 5.6).

Ergebnis: $\boxed{\begin{array}{ll} q_1 & \to aq_2 \\ q_2 & \to bq_2 \\ q_1 & \to a \\ q_2 & \to b \end{array}}$ oder in anderer Notation $\boxed{\begin{array}{ll} S & \to aA \\ A & \to bA \\ S & \to a \\ A & \to b \end{array}}$

Dies ist genau Γ_6 aus obigem Beispiel.

Wir konstruieren hieraus einen NDEA

δ	a	b
S	$\{A, f\}$	$\emptyset$
A	$\emptyset$	$\{A, f\}$
f	$\emptyset$	$\emptyset$



Formt man diesen NDEA in einen DEA um und reduziert den DEA, so erhält man einen DEA, der zu $\mathcal{A}$ isomorph ist.

Lemma 4.6

Zu jeder regulären (=rechtsregulären=rechtslinearen) Grammatik Γ gibt es eine linksreguläre (=linkslineare) Grammatik Γ' mit $L(\Gamma) = L(\Gamma')$; ebenso gilt die Umkehrung.

Beweis: Aufgabe

5 Kontextfreie Sprachen

Die durch kontextfreie Grammatiken erzeugten kontextfreien Sprachen spielen sowohl bei der Definition künstlicher Sprachen (z.B. Programmiersprachen) wie auch bei der Syntaxanalyse natürlicher Sprachen eine ausgezeichnete Rolle.

Definition 5.1 (Kontextfreiheit)

Eine Grammatik $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$ heißt kontextfrei (kf), wenn alle ihre Produktionen die Form $A \to w$ haben für $A \in \mathcal{V}, w \in (\mathcal{V} \cup \mathcal{T})^*$. $L(\Gamma)$ heißt kontextfreie Sprache.

Bemerkungen:

1. „Kontextfrei“ steht im Gegensatz zu „kontextsensitiv“, wo Produktionen die Form $uAv \to uwv$ haben $(u, v \in (\mathcal{V} \cup \mathcal{T})^*)$.
2. Mehrere Produktionen $A \to u_1, \ldots, A \to u_n$ können als $A \to u_1|u_2|\ldots|u_n$ geschrieben werden (vgl. BNF).

Lemma 5.2

Jede reguläre Grammatik ist kf.

Beispiele:

$\Gamma_1 :$ $\quad \mathcal{V} := \{S\} \;\; \mathcal{T} := \{a, b\}$

Π: $S \to \varepsilon$
$\quad S \to aSb$

abgekürzt: $S \to \varepsilon | aSb$

$L(\Gamma_1) = \{a^n b^n : n \geq 0\}$

$\Gamma_2 :$ $\quad \mathcal{V} = \{S\} \;\; \mathcal{T} = \{\emptyset, \circ, \cup, *, (,), a, b, c\}$
Π: $S \to \emptyset|a|b|c|(S \circ S)|(S \cup S)|S^*$

$L(\Gamma_2)$=Menge der regulären Ausdrücke über $\{a, b, c\}$

$\Gamma_3 :$ $\quad \mathcal{V} = \{S\} \;\; \mathcal{T} = \{a, b, c\}$
Π: $S \to \varepsilon|a|b|c|aSa|bSb|cSc$

$L(\Gamma_3)$= Menge der Palindrome über $\{a, b, c\}$

$\Gamma_4 :$ $\quad \mathcal{V} := \{S, A, B\} \quad \mathcal{T} := \{a, b\}$

$$\begin{aligned} \Pi\text{:}\ S &\to aB|bA \\ A &\to a|aS|bAA \\ B &\to b|bS|aBB \end{aligned}$$

$L(\Gamma_4) = \{w \in \mathcal{T}^* : w$ enthält gleich viele a's wie b's$\}$

Beispiel:

Backus-Naur-Form (BNF) als Notation für kfGen, illustriert an der Definition von Implikationsformeln, d.h. Formeln wie p_{27} oder $(p_{010} \supset (p_{137} \supset p_{300}))$ etc.

Grammatik:
$\mathcal{V} = \{\langle\text{Grundziffer}\rangle, \langle\text{Ziffer}\rangle, \langle\text{Aussagenvariable}\rangle, \langle\text{Implikationsformel}\rangle\}$
$\mathcal{T} = \{_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, p, (,), \supset\}$
Startsymbol: $\langle$Implikationsformel$\rangle$
„::=“ ist als Regelpfeil „→“ zu lesen.

$\langle\text{Grundziffer}\rangle ::= {}_0|{}_1|{}_2|{}_3|{}_4|{}_5|{}_6|{}_7|{}_8|{}_9$

$\langle\text{Ziffer}\rangle ::= \langle\text{Grundziffer}\rangle|\langle\text{Grundziffer}\rangle\langle\text{Ziffer}\rangle$

$\langle\text{Aussagenvariable}\rangle ::= p\langle\text{Ziffer}\rangle$

$\langle\text{Implikationsformel}\rangle ::=$
$\langle\text{Aussagenvariable}\rangle|(\langle\text{Implikationsformel}\rangle \supset \langle\text{Implikationsformel}\rangle)$

Erweiterte Backus-Naur-Formen (EBNF) lassen sich leicht in kfGen übersetzen. Entsprechendes gilt für Syntaxdiagramme, denen ebenfalls kfGen korrespondieren. (Übung)

Lemma 5.3

Sei Γ kfG. Seien $u_1, \ldots, u_{n+1} \in \mathcal{T}^*, w \in (\mathcal{V} \cup \mathcal{T})^*$. Dann gilt:
$u_1 A_1 u_2 A_2 u_3 \ldots u_n A_n u_{n+1} \underset{\Gamma}{\overset{k}{\Longrightarrow}} w$ impliziert
$w = u_1 v_1 u_2 v_2 u_3 \ldots u_n v_n u_{n+1}$ mit $A_i \underset{\Gamma}{\overset{\leq k}{\Longrightarrow}} v_i$ $(1 \leq i \leq n)$ für gewisse $v_i \in (\mathcal{V} \cup \mathcal{T})^*$.
$A_i \overset{k_i}{\Longrightarrow} v_i$ für alle i $(1 \leq i \leq n)$ impliziert
$u_1 A_1 u_2 A_2 u_3 \ldots u_n A_n u_{n+1} \overset{k_1+\cdots+k_n}{\Longrightarrow} w$ mit $w = u_1 v_1 u_2 v_2 u_3 \ldots u_n v_n u_{n+1}$

Beweis: Übung.

Korollar zu Lemma 5.3

Sei Γ kfG. Seien $u_1, \ldots, u_{n+1} \in \mathcal{T}^*, w \in (\mathcal{V} \cup \mathcal{T})^*$. Dann gilt:
$u_1 A_1 u_2 A_2 \ldots u_n A_n u_{n+1} \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ g.d.w.
$w = u_1 v_2 u_2 v_2 \ldots u_n v_n u_{n+1}$ mit $A_i \overset{*}{\underset{\Gamma}{\Longrightarrow}} v_i$ $(= 1, \ldots, n)$ für gewisse $v_i \in (\mathcal{V} \cup \mathcal{T})^*$

Definition 5.4 (Erreichbarkeit, Terminalisierbarkeit, Reduziertheit)

Sei $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$ kfG. Seien $A, B \in \mathcal{V}$.

B heißt von A aus erreichbar, falls $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} uBv$ für $u, v \in (\mathcal{V} \cup T)^*$

B heißt erreichbar, wenn B von S aus erreichbar ist.

A heißt terminalisierbar, falls $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ für ein $w \in \mathcal{T}^*$

A heißt nutzlos, falls es keine $u, v \in (\mathcal{V} \cup \mathcal{T})^*, w \in \mathcal{T}^*$ gibt, so daß $S \overset{*}{\underset{\Gamma}{\Longrightarrow}} uAv \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$

Γ heißt reduziert, falls Γ keine nutzlosen Variablen enthält oder $\Pi = \emptyset$

Bemerkung:

Intuitiv: A nutzlos $\Leftrightarrow$ es gibt keine Ableitung eines Terminalwortes, in der A benutzt wird.

Satz 5.5

Zu jeder kfG Γ läßt sich eine äquivalente reduzierte kfG Γ' konstruieren.

Beweis:

folgt unten nach dem Beweis von Lemma 5.6 und Lemma 5.7

Lemma 5.6

Zu jeder kfG Γ läßt sich eine äquivalente kfG Γ' konstruieren, in der jede Variable terminalisierbar ist.

Beweis:

Sei $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$.
Sei $\Gamma' = \langle \mathcal{V}', \mathcal{T}, \Pi', S \rangle$ wie folgt definiert:

Wir definieren zunächst	$\mathcal{V}'_i$ induktiv wie folgt:
	$\mathcal{V}'_0 := \emptyset$
	$\mathcal{V}'_{n+1} := \mathcal{V}'_n \cup \{A \in \mathcal{V} :\ A \to w \in \Pi \text{ mit } w \in (\mathcal{V}'_n \cup \mathcal{T})^*\}$.
Insbesondere ist	$\mathcal{V}'_1 = \{A \in \mathcal{V} :\ A \to w \in \Pi \text{ mit } w \in \mathcal{T}^*\}$.
Dann sei:	$\mathcal{V}' := \bigcup_{n \geq 0} \mathcal{V}'_n$
	$\Pi' := \{A \to w \in \Pi :\ A \in \mathcal{V}' \text{ mit } w \in (\mathcal{V}' \cup \mathcal{T})^*\}$
Offenbar gilt:	$\mathcal{V}'_i \subseteq \mathcal{V}'_{i+1}$ für jedes $i \geq 0$
	$\mathcal{V}'_i = \mathcal{V}'_{i+1}$ impliziert $\mathcal{V}'_i = \mathcal{V}_{i+m}$ für alle $i, m \geq 0$
	$\mathcal{V}' = \mathcal{V}'_{\lvert\mathcal{V}\rvert}$ (da im ungünstigstem Fall bei der induktiven Konstruktion von $\mathcal{V}'$ in jedem Schritt nur eine Variable hinzukommt).

Damit liegt ein effektives Verfahren zur Konstruktion von $\mathcal{V}'$ vor.

Wir zeigen jetzt:

1. Jede Variable aus $\mathcal{V}'$ ist in Γ' terminalisierbar
   Induktion über Konstruktion von $\mathcal{V}'$.
   $\mathcal{V}'_0$: ✓
   $\mathcal{V}'_{n+1}$: Sei $A \overset{1}{\underset{\Gamma'}{\Longrightarrow}} w \in (\mathcal{V}'_n \cup \mathcal{T})^*$ mit $A \to w \in \Pi'$
   Nach Induktionsvorraussetzung gilt: Jede Variable in w ist terminalisierbar. Also ist nach Lemma 5.3 A terminalisierbar.

2. $L(\Gamma') = L(\Gamma)$
   „$\subseteq$": Klar, da $\mathcal{V}' \subseteq \mathcal{V}$ und $\Pi' \subseteq \Pi$.
   „$\supseteq$": Durch Induktion über der Länge von Ableitungen zeigen wir:
   Sei $u \in (\mathcal{V} \cup \mathcal{T})^*,\ w \in \mathcal{T}^*$.
   Falls $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$, dann $u \in (\mathcal{V}' \cup \mathcal{T})^*$ und $u \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$.
   Beweis:
   $u \overset{0}{\underset{\Gamma}{\Longrightarrow}} u$ ist trivial, da u dann keine Variablen enthält
   Betrachte $u = u_1 A u_2 \overset{1}{\underset{\Gamma}{\Longrightarrow}} u_1 v u_2 \overset{n}{\underset{\Gamma}{\Longrightarrow}} w$ mit $A \to v \in \Pi$.
   Nach I.V gilt $u_1 v u_2 \in (\mathcal{V}' \cup \mathcal{T})^*$ und $u_1 v u_2 \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$. Damit
   $u_1 A u_2 \in (\mathcal{V}' \cup \mathcal{T})^*$ und $A \to v \in \Pi'$. Damit $u_1 A u_2 \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$.

Bemerkung (Grenzfall):

Falls $L(\Gamma) = \emptyset$, ist S in Γ nicht terminalisierbar und damit $S \notin \mathcal{V}'$. Dies sei hier als Grenzfall zugelassen.

Beispiel:

$S \to AB|a$
$A \to a|aB$
$C \to Aa$

Konstruktion von $\mathcal{V}'$:
$\mathcal{V}'_0 = \emptyset$
$\mathcal{V}'_1 = \emptyset \cup \{S, A\}$
$\mathcal{V}'_2 = \{S, A\} \cup \{C\}$
$\mathcal{V}'_3 = \{S, A, C\} \cup \emptyset$
$\mathcal{V}' = \{S, A, C\}$

$$\Pi' = \{ \quad \begin{array}{l} S \to a \\ A \to a \\ C \to Aa \ \} \end{array}$$

Lemma 5.7

Zu jeder kfG $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$ läßt sich eine kfG $\Gamma' = \langle \mathcal{V}', \mathcal{T}, \Pi', S \rangle$ konstruieren, in der jede Variable erreichbar ist, so daß gilt: $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ g.d.w. $A \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$ für alle $A \in \mathcal{V}'$ und $w \in (\mathcal{V}' \cup \mathcal{T})^*$

Beweis:

Wir definieren zunächst	$\mathcal{V}'_i$ induktiv wie folgt:
	$\mathcal{V}'_0 := \{S\}$
	$\mathcal{V}'_{n+1} := \mathcal{V}'_n \cup \{A \in \mathcal{V} :\ B \to uAv \in \Pi \text{ mit } B \in \mathcal{V}'_n\}$
	$\mathcal{V}' := \bigcup_{n \geq 0} \mathcal{V}'_n$
	$\Pi' := \{A \to w \in \Pi :\ A \in \mathcal{V}'\}$
Es gilt wieder:	$\mathcal{V}'_i \subseteq \mathcal{V}'_{i+1}$ für jedes $i \geq 0$
	$\mathcal{V}'_i = \mathcal{V}'_{i+1}$ impliziert $\mathcal{V}'_i = \mathcal{V}_{i+m}$ für alle $i, m \geq 0$
	$\mathcal{V}' = \mathcal{V}'_{\lvert\mathcal{V}\rvert}$
Ferner gilt:	$A \in \mathcal{V}'_n$ g.d.w. $S \overset{\leq n}{\underset{\Gamma}{\Longrightarrow}} uAv$ für $u, v \in (\mathcal{V} \cup \mathcal{T})^*$.
Insbesondere ist	$A \in \mathcal{V}'$ g.d.w. A ist in Γ von S aus erreichbar.

Wir zeigen jetzt:

1. $A \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$ impliziert $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ für $A \in \mathcal{V}',\ w \in (\mathcal{V}' \cup \mathcal{T})^*$.
   Dies ist klar, da $\mathcal{V}' \subseteq \mathcal{V}$ und $\Pi' \subseteq \Pi$.

2. $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ impliziert $A \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$ für $A \in \mathcal{V}',\ w \in (\mathcal{V}' \cup \mathcal{T})^*$.
   In einer Ableitung von w aus A in Γ können nur Produktionen benutzt sein für von S aus erreichbare Variablen, da $A \in \mathcal{V}'$ selbst von S aus erreichbar ist. Diese Produktionen sind in Π' vorhanden.

Bemerkung:

Wir haben mehr gezeigt als nur Äquivalenz von Γ und Γ': Sogar die Ableitbarkeit von Wörtern *mit Variablen* aus beliebigen von S aus erreichbaren Variablen bleibt erhalten.

Beispiel:
Grammatik wie im vorigen Beispiel
$\mathcal{V}_0' = \{S\}$
$\mathcal{V}_1' = \{S, A, B\}$
$\mathcal{V}_2' = \{S, A, B\} = \mathcal{V}'$
$\Pi' = \{S \to AB|a, A \to a|aB\}$

Beweis von Satz 5.5:

Ausgehend von Γ wenden wir Lemma 5.6 an, resultierend in Γ_1. Auf Γ_1 wenden wir Lemma 5.7 an, resultierend in Γ_2. Wir setzen $\Gamma' := \Gamma_2$. Wir müssen zeigen, daß keine Variable aus Γ' nutzlos ist (falls Γ' überhaupt Produktionen enthält). Für beliebige Variablen A aus Γ' gilt offenbar, daß sie in Γ' erreichbar sind, d.h. $S \underset{\Gamma_2}{\overset{*}{\Longrightarrow}} uAv$ und damit auch $S \underset{\Gamma_1}{\overset{*}{\Longrightarrow}} uAv$.
Damit gilt, da alle Variablen aus uAv in Γ_1 terminalisierbar sind, $S \underset{\Gamma_1}{\overset{*}{\Longrightarrow}} uAv \underset{\Gamma_1}{\overset{*}{\Longrightarrow}} w$ für $w \in \mathcal{T}^*$.
Daraus mit Lemma 5.7: $S \underset{\Gamma_2}{\overset{*}{\Longrightarrow}} uAv \underset{\Gamma_2}{\overset{*}{\Longrightarrow}} w$

Bemerkungen:

1. Die Reihenfolge der Anwendungen von Lemma 5.6 und 5.7 ist relevant. Die Anwendung in umgekehrter Reihenfolge führt nicht zum gewünschten Resultat. Man mache sich dies an der in den vorigen beiden Beispielen benutzten Grammatik klar.
   Der Grund dafür liegt darin, daß der Übergang von $S \underset{\Gamma_1}{\overset{*}{\Longrightarrow}} uAv \underset{\Gamma_1}{\overset{*}{\Longrightarrow}} w$ zu $S \underset{\Gamma_2}{\overset{*}{\Longrightarrow}} uAv \underset{\Gamma_2}{\overset{*}{\Longrightarrow}} w$ (letzter Schritt des Beweises) nicht gedeckt ist, falls Γ_1 durch Lemma 5.7 und Γ_2 anschließend durch Lemma 5.6 entsteht, da uAv kein Terminalwort ist (vgl. Bemerkung hinter Lemma 5.7).
   Wir betrachten folgendes vereinfachte Beispiel einer Grammatik $\langle \{S, A, B\}, \{a\}, \Pi, S\rangle$ mit den Produktionen:

   $S \to AB|a$
   $A \to a$

Anwendung von Lemma 5.6 ergibt:
$S \to a$
$A \to a$

Darauf folgende Anwendung von Lemma 5.7 ergibt:
$S \to a$

Anwendung von Lemma 5.7 ergibt:
$S \to AB|a$
$A \to a$

Darauf folgende Anwendung von Lemma 5.6 ergibt:
$S \to a$
$A \to a$

A ist nutzlos.

2. Im Spezialfall $L(\Gamma) = \emptyset$ gibt es nach Anwendung von Lemma 5.6 keine Produktion $S \to w$ für S. Das bedeutet, daß nach Anwendung von Lemma 5.7 die Menge der Produktionen leer ist. Damit ist die resultierende Grammatik reduziert.

Lemma 5.8

Zu jeder kfG $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S\rangle$ läßt sich eine kfG $\Gamma' = \langle \mathcal{V}, \mathcal{T}, \Pi', S\rangle$ ohne ε-Produktionen konstruieren derart, daß $L(\Gamma') = L(\Gamma)\backslash\{\varepsilon\}$.

Beweis:

$$\begin{array}{ll} \mathcal{V}'_0 := & \{A \in \mathcal{V} : A \to \varepsilon \in \Pi\} \\ \mathcal{V}'_{n+1} := & \mathcal{V}'_n \cup \{A \in \mathcal{V} : A \to w \in \Pi \text{ für } w \in \mathcal{V}'^*_n\} \\ \mathcal{V}' := & \bigcup_{n\geq 0} \mathcal{V}'_n \end{array}$$

Es gilt:

$$\begin{array}{ll} \mathcal{V}'_i \subseteq & \mathcal{V}'_{i+1} \ (i \geq 0) \\ \mathcal{V}'_i = & \mathcal{V}'_{i+1} \text{ impliziert } \mathcal{V}'_i = \mathcal{V}'_{i+m} \\ \mathcal{V}' = & \mathcal{V}'_{|\mathcal{V}|} \end{array}$$

Ferner gilt:
$\mathcal{V}'$ ist die Menge der in Γ „nullierbaren“ (d.h. zu ε führenden) Variablen,
d.h $\mathcal{V}' = \{A : A \overset{*}{\underset{\Gamma}{\Longrightarrow}} \varepsilon\}$

$$\begin{array}{ll} \Pi' := & \{A \to a_1 \dots a_n : a_1, \dots, a_n \in (\mathcal{V} \cup \mathcal{T}) \text{ und} \\ & A \to u_1 a_1 u_2 a_2 \dots u_n a_n u_{n+1} \in \Pi \text{ für gewisse } u_1, \dots, u_{n+1} \in \mathcal{V}'^*\} \end{array}$$

d.h. es werden alle Produktionen betrachtet, deren rechte Seite nicht ε ist und bei der keine, einige oder alle nullierbaren Variablen weggelassen werden. (Insbesondere enthält Π' alle Produktionen aus Π, die nicht ε-Produktionen sind.)
Achtung: Γ' hat dieselben Variablen wie Γ. $\mathcal{V}'$ geht nur in die Definition von Π' ein.

Bleibt zu zeigen: $L(\Gamma') = L(\Gamma)\backslash\{\varepsilon\}$

Beweis:

„$\subseteq$“:
Jede in Γ' gegenüber Γ neue Produktion kann in Γ simuliert werden durch Anwendung von ε-Produktionen. Ferner ist ε in Γ' nicht ableitbar.

„$\supseteq$“:
Wir zeigen: $A \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$ für $A \in \mathcal{V},\ w \in \mathcal{T}^*,\ w \neq \varepsilon$ impliziert $A \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$
Induktion über Ableitungslänge:
0: tritt nicht auf, da $w \neq A$
Sei $A \overset{1}{\underset{\Gamma}{\Longrightarrow}} u_1A_1u_2A_2 \dots u_2A_nu_{n+1} \overset{k}{\underset{\Gamma}{\Longrightarrow}} w,\ u_i \in \mathcal{T}^*$ für $A \to u_1A_1 \dots u_nA_nu_{n+1} \in \Pi$.
Dann gilt nach Lemma 5.3:
$w = u_1v_1u_2v_2 \dots u_nv_nu_{n+1}$ mit $A_i \overset{\leq k}{\underset{\Gamma}{\Longrightarrow}} v_i$. Nach I.V. gilt: $A_i \overset{*}{\underset{\Gamma'}{\Longrightarrow}} v_i$, falls $v_i \neq \varepsilon$.

Sei $w_i = \begin{cases} A_i & \text{falls } v_i \neq \varepsilon \\ \varepsilon & \text{falls } v_i = \varepsilon \end{cases}$

Dann ist $A \to u_1w_1u_2w_2 \dots u_nw_nu_{n+1} = \Pi'$, es gilt also:
$A \overset{1}{\underset{\Gamma'}{\Longrightarrow}} u_1w_1u_2w_2 \dots u_nw_nu_{n+1} \overset{k}{\underset{\Gamma'}{\Longrightarrow}} u_1v_1 \dots u_nv_nu_{n+1}$.

Bemerkung:

Der Induktionsschritt schließt den Fall der Ableitungslänge 1 ein. Separat wird dieser Fall so behandelt:
Sei $A \overset{1}{\underset{\Gamma}{\Longrightarrow}} w$ mit $A \to w \in \Pi$. Da $w \neq \varepsilon$, ist $A \to w \in \Pi'$. Also gilt $A \overset{1}{\underset{\Gamma'}{\Longrightarrow}} w$.

Beispiel:

$$\Pi = \{ \quad \begin{aligned} S &\to AaSb|aa \\ A &\to BaB|\varepsilon \\ B &\to cA|AA \quad \} \end{aligned}$$

$\mathcal{V}_0' = \{A\};\ V_1' = \{A, B\};\ \mathcal{V}_2' = \{A, B\}$ (nullierbare Variablen sind also A und B)

$$\Pi' = \{ \quad \begin{aligned} S &\to AaSb|aSb|aa \\ A &\to BaB|aB|Ba|a \\ B &\to cA|c|AA|A \quad \} \end{aligned}$$

Definition 5.9 (Kettenproduktion)

Eine Produktion der Form $A \to B$ heißt Kettenproduktion.

Lemma 5.10

Zu jeder kfG $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$ läßt sich eine äquivalente kfG $\Gamma' = \langle \mathcal{V}, \mathcal{T}, \Pi', S \rangle$ ohne Kettenproduktion konstruieren.

Beweis:

Wir definieren eine Funktion $\mathcal{V}'$, die Variablen Mengen von Variablen zuordnet.

$$\begin{array}{ll} \mathcal{V}'_0(A) & := \{A\} \\ \mathcal{V}'_{n+1}(A) & := \mathcal{V}'_n(A) \cup \{B \in \mathcal{V} : C \to B \in \Pi \text{ mit } C \in \mathcal{V}'_n(A)\} \\ \mathcal{V}'(A) & := \bigcup_{i \geq 0} \mathcal{V}'_n(A) \end{array}$$

Es gilt:

$$\begin{array}{ll} \mathcal{V}'(A) & = \mathcal{V}'_{|\mathcal{V}|}(A) \\ \mathcal{V}'(A) & = \{B \in \mathcal{V} : A \overset{*}{\underset{\Gamma}{\Longrightarrow}} B\} \\ \mathcal{V}'_n(A) & = \{B \in \mathcal{V} : A \overset{\leq n}{\underset{\Gamma}{\Longrightarrow}} B\} \\ \Pi' & := \{A \to w : w \notin \mathcal{V}, B \to w \in \Pi \text{ mit } B \in \mathcal{V}'(A)\} \end{array}$$

Intuitiv:

Ersetze alle Produktionsfolgen

$$\left\{ \begin{array}{ll} A_1 & \to A_2 \\ A_2 & \to A_3 \\ & \vdots \\ A_{n-1} & \to A_n \\ A_n & \to w \notin \mathcal{V} \end{array} \right.$$

durch

$$\left\{ \begin{array}{ll} A_1 & \to w \\ A_2 & \to w \\ & \vdots \\ A_{n-1} & \to w \\ A_n & \to w. \end{array} \right.$$

$L(\Gamma) \subseteq L(\Gamma')$:
Ersetze jedes $A_1 \overset{1}{\underset{\Gamma}{\Longrightarrow}} A_2 \ldots \overset{1}{\underset{\Gamma}{\Longrightarrow}} A_n \overset{1}{\underset{\Gamma}{\Longrightarrow}} w \notin \mathcal{V}$ durch $A_1 \overset{1}{\underset{\Gamma'}{\Longrightarrow}} w$

$L(\Gamma') \subseteq L(\Gamma)$:
Ersetze Anwendung $u_1 A u_2 \overset{1}{\underset{\Gamma'}{\Longrightarrow}} u_1 w u_2$ einer neuen Produktion $A \to w$ mit $B \in \mathcal{V}'(A)$ und $B \to w \in \Pi$ durch $u_1 A u_2 \overset{*}{\underset{\Gamma}{\Longrightarrow}} u_1 B u_2 \overset{1}{\underset{\Gamma}{\Longrightarrow}} u_1 w u_2$

Beispiel:

$$\begin{array}{lll}\Pi = \{ & S \to aSb|A & \\ & A \to B|Ba & \\ & B \to bB|\varepsilon \ \ \} & \end{array}$$

$$\begin{array}{lll}\Pi' = \{ & S \to aSb & \\ & S \to Ba|bB|\varepsilon & \text{neue Produktionen} \\ & A \to Ba & \\ & A \to bB|\varepsilon & \text{neue Produktionen} \\ & B \to bB|\varepsilon \ \ \} & \end{array}$$

Definition 5.11 (Chomsky-Normalform)

Eine kfG ist in Chomsky-Normalform (CNF), falls jede Produktion die Form $A \to BC$ oder $A \to a$ hat.

Theorem 5.12

Zu jeder kfG Γ mit $\varepsilon \notin L(\Gamma)$ läßt sich eine äquivalente kfG Γ' in CNF konstruieren.

Beweis:
Aufgrund von Lemma 5.8 und 5.10 können wir annehmen, daß Γ keine ε-Produktionen und keine Kettenproduktionen enthält. (Achtung: Die Reihenfolge der Anwendungen dieser Lemmas ist wichtig. Anwendung von Lemma 5.10 vor Lemma 5.8 führt nicht notwendigerweise zum Ziel, da die Elimination von ε-Produktionen Kettenproduktionen erzeugen kann, aber nicht umgekehrt.)
Sei $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$.
Setze $\mathcal{V}' := \mathcal{V} \cup \{B_a : a \in \mathcal{T}\}$.
Ferner sei für jedes Wort $w \in (\mathcal{V} \cup \mathcal{T})^*$:

$$w' := \begin{cases} w & \text{falls} \quad |w| = 1 \\ \text{Resultat der Ersetzung aller } a \in \mathcal{T} \text{ durch} \\ B_a \text{ in } w & \text{sonst.} \end{cases}$$

$\widetilde{\Pi} := \{A \to w' : A \to w \in \Pi\} \cup \{B_a \to a : a \in \mathcal{T}\}$
$\widetilde{\Gamma} := \langle \mathcal{V}', \mathcal{T}, \widetilde{\Pi}, S \rangle$.

Es gilt: Γ und $\widetilde{\Gamma}$ sind äquivalent (Übung).
Alle Produktionen von $\widetilde{\Gamma}$ haben die Form $A \to A_1 \ldots A_n$ oder $A \to a$
Falls $n > 2$, ersetzen wir $A \to A_1 \ldots A_n$ durch
$A \to A_1 B_1$
$B_1 \to A_2 B_2$
$\quad\vdots$

$B_{n-3} \to A_{n-2}B_{n-2}$
$B_{n-2} \to A_{n-1}A_n$
wobei $B_1, \ldots, B_{n-2}$ geeignete neue Variablen seien.
Das Resultat sei Π'
$\Gamma' := \langle \mathcal{V}'', \mathcal{T}, \Pi', S \rangle$, wobei $\mathcal{V}'' := \mathcal{V}' \cup \{B_1, \ldots, B_{n-2}\}$, ist offenbar äquivalent zu $\widetilde{\Gamma}$ und damit zu Γ.

Beispiel:

$\Gamma = \langle \{S, A, B\}, \{a, b\}, \Pi, S \rangle$

$$\Pi = \{ \begin{array}{l} S \to aAbB \\ A \to aA|a \\ B \to bB|b \quad \} \end{array}$$

1. Terminale-Ersetzen ergibt:
   $S \to B_aAB_bB$
   $A \to B_aA|a$
   $B \to B_bB|b$
   $B_a \to a$
   $B_b \to b$

2. Ersetzen von $S \to B_a \underbrace{A \overbrace{B_bB}^{B_2}}_{B_1}$ ergibt:

   $S \to B_aB_1$
   $B_1 \to AB_2$
   $B_2 \to B_bB$

Π' hat also die Produktionen:

$S \to B_aB_1$
$B_1 \to AB_2$
$B_2 \to B_bB$
$A \to B_aA|a$
$B \to B_bB|b$
$B_a \to a$
$B_b \to b$

Bemerkungen:

1. Da der Algorithmus zur Konstruktion einer reduzierten kfG nur Produktionen ausläßt, jedoch nicht deren Form verändert, kann man auch verlangen, daß als CNFen nur reduzierte kfGen zugelassen sind.

2. Eine andere Normalform ist die *Greibach-Normalform*. Hier haben alle Produktionen die Form $A \to a$ oder $A \to aB_1 \dots B_n$ für $a \in \mathcal{T}$. Es gilt, daß sich zu jeder kfG Γ mit $\varepsilon \notin L(\Gamma)$ eine äquivalente kfG in Greibach-Normalform konstruieren läßt.

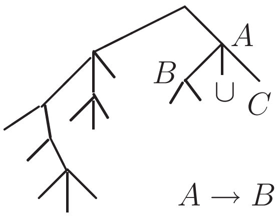
Definition 5.13 (Ableitungsbaum)

Sei $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S \rangle$ kfG ohne ε-Produktion.

Ein Ableitungsbaum in Γ ist ein Baum, dessen Blätter mit Variablen oder Terminalen markiert sind, wobei gelten muß:
Falls $a_1, \dots, a_n \in \mathcal{V} \cup \mathcal{T}$ die unmittelbaren Nachfolger (Kinder) von A sind (gelesen von links nach rechts), dann ist $A \to a_1 \dots a_n \in \Pi$. Ein Ableitungsbaum für w aus $a \in \mathcal{V} \cup \mathcal{T}$ in Γ ist ein Ableitungsbaum, dessen Wurzel mit a und dessen Blätter von links nach rechts mit den Zeichen aus w markiert sind. Ein Ableitungsbaum für w in Γ ist ein Ableitungsbaum für w aus S in Γ. Man nennt ihn auch Strukturbaum (parse tree) für w in Γ.

Ein Zweig ist eine Folge aufeinanderfolgender Knoten, beginnend mit der Wurzel und endend mit einem Blatt. Die Anzahl der Knoten einschließlich der Blätter heißt Größe des Baumes, die Anzahl der Knoten (einschließlich Blatt) in seinem längsten Zweig seine Höhe. (Auf eine formale mathematische Definition des Begriffs „Baum“ wird hier verzichtet.)

Beispiel:

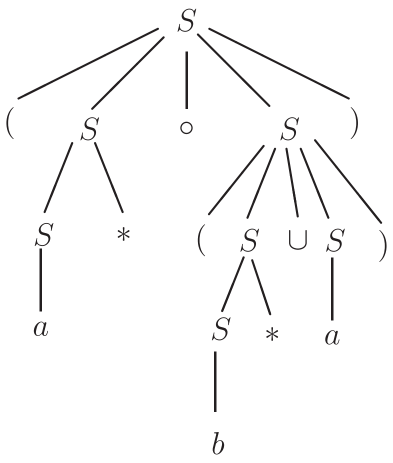


Größe 21

Höhe 6

$A \to B \cup C \in \Pi$

Beispiel:
Γ_2 (reguläre Ausdrücke über {a,b,c})
$\Pi : S \to \emptyset | (S \circ S) | (S \cup S) | S^* | a | b | c$
Ableitungsbaum für $(a^* \circ (b^* \cup a))$:



Verschiedene Ableitungen:

$$
\begin{aligned}
S &\overset{1}{\Rightarrow} (S \circ S) \overset{1}{\Rightarrow} (S^* \circ S) \overset{1}{\Rightarrow} (S^* \circ (S \cup S)) \overset{1}{\Rightarrow} (a^* \circ (S \cup S)) \\
&\overset{1}{\Rightarrow} (a^* \circ (S \cup a)) \overset{1}{\Rightarrow} (a^* \circ (S^* \cup a)) \overset{1}{\Rightarrow} (a^* \circ (b^* \cup a)) \\
S &\overset{1}{\Rightarrow} (S \circ S) \overset{1}{\Rightarrow} (S^* \circ S) \overset{1}{\Rightarrow} (a^* \circ S) \overset{1}{\Rightarrow} (a^* \circ (S \cup S)) \\
&\overset{1}{\Rightarrow} (a^* \circ (S^* \cup S)) \overset{1}{\Rightarrow} (a^* \circ (b^* \cup S)) \overset{1}{\Rightarrow} (a^* \circ (b^* \cup a))
\end{aligned}
$$

Definition 5.14 (Rechtsableitung, Linksableitung)

Eine Rechtsableitung ist eine Ableitung, bei der jeder Schritt die Form $uAv \overset{1}{\Rightarrow} uwv$ für eine Produktion $A \to w$ hat, wobei $v \in \mathcal{T}^*$, $u \in (\mathcal{V} \cup \mathcal{T})^*$. Bei einer Linksableitung gilt stattdessen $u \in \mathcal{T}^*$, $v \in (\mathcal{V} \cup \mathcal{T}^*)$.
D.h., bei einer Rechtsableitung wird in jedem Schritt die rechteste, bei einer Linksableitung die linkeste Variable ersetzt.

Satz 5.15

Sei Γ ohne ε-Produktion, dann ist folgendes gleichwertig:

a) $w \in L(\Gamma)$

b) Es gibt einen Ableitungsbaum für w

c) Es gibt eine Linksableitung für w

d) Es gibt eine Rechtsableitung für w

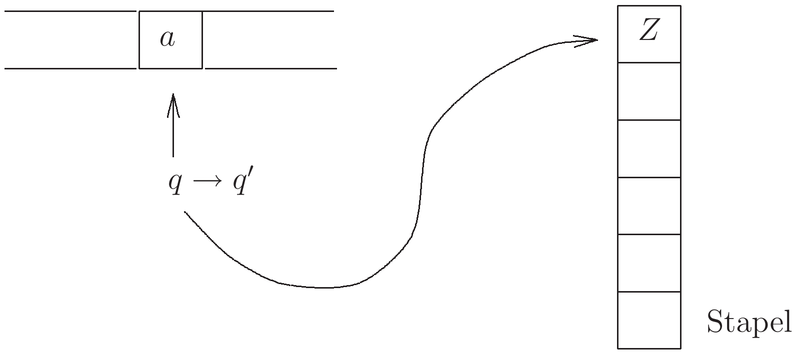
Beweis: Übung

Bemerkung:

Es gilt allgemeiner: Sei f eine Auswahlfunktion, die aus jeder Menge von Variablenvorkommen eines auswählt. Dann ist $w \in L(\Gamma)$ g.d.w. es eine Ableitung für w gibt, bei der in jedem Schritt gerade das durch f ausgewählte Variablenvorkommen ersetzt wird.

6 Kellerautomaten

Kellerautomaten haben einen Speicher von der Struktur eines Stapels von Zeichen. In Abhängigkeit vom Zustand des Automaten q, vom gelesenen Eingabezeichen a und vom obersten Zeichen Z des Stapels wird beim Übergang in einen neuen Zustand q' das oberste Zeichen Z des Stapels weggenommen; der verbleibende Stapel kann dann durch ein oder mehrere Zeichen neu aufgestockt werden. (Wenn Z durch Z ersetzt wird, bleibt der Stapel unverändert, wenn Z durch ε ersetzt wird, verliert er das oberste Element.) Anders als bei endlichen Automaten (genauer NDEAs) lassen wir ε-Übergänge zu, d.h. der Stapel soll beim Übergang von q zu q' modifiziert werden können, ohne ein (echtes) Zeichen zu lesen. Wir betrachten nichtdeterministische Kellerautomaten, d.h. q, a und Z determinieren nicht eindeutig die auszuführende Operation.



Definition 6.1 (Kellerautomat)

Ein Kellerautomat (KA) ist ein Septupel $\mathcal{A} =< Q, \Sigma, \Phi, \delta, q_0, Z_0, F >$ mit

Q ... nichtleere endliche Zustandsmenge

Σ ... Eingabealphabet

Φ ... Kelleralphabet

$q_0 \in Q$ ... ausgezeichneter Anfangszustand

$Z_0 \in \Phi$ ... Anfangssymbol des Kellerspeichers

$F \subseteq Q$ ... Menge ausgezeichneter Endzustände

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Phi \to \mathcal{P}(Q \times \Phi^*)$ Übergangsfunktion

$\mathcal{A}$ heißt deterministisch, falls $|\delta(q, a, Z)| \leq 1$ für $q \in Q, a \in \Sigma \cup \{\varepsilon\}, Z \in \Phi$,
sowie $\delta(q, \varepsilon, Z) \neq \emptyset \;\Rightarrow \delta(q, a, Z) = \emptyset$ für alle $a \in \Sigma$

Metasprachliche Variablen:
X, Y, Z für Elemente von Φ; α, β, γ für Wörter über Φ

Bemerkungen:

1. Wenn Kellerspeicher leer, dann kein Übergang definiert.
2. ε-Übergänge können grundsätzlich eliminiert werden (Zeigen wir hier nicht. Vgl. Harrison (1978) 5.51 - 5.53)
3. Deterministische und nichtdeterministische Kellerautomaten sind *nicht* gleichwertig. (Obzwar fundamental für bestimmte Anwendungen, insbesondere den Compilerbau, kann dies hier nicht behandelt werden. Vgl. Hopcroft & Ullman (1990) Kap. 10.)

Definition 6.2 (Konfiguration)

Sei ein KA wie vorher gegeben. Eine Konfiguration ist ein Tripel $(q, w, \alpha) \in Q \times \Sigma^* \times \Phi^*$.
Die Relation $\overset{1}{\underset{\mathcal{A}}{\vdash}}$ zwischen Konfigurationen ist wie folgt definiert:

$(q, w, Z\alpha) \overset{1}{\underset{\mathcal{A}}{\vdash}} (q', w, \beta\alpha)$ falls $(q', \beta) \in \delta(q, \varepsilon, Z)$
$(q, aw, Z\alpha) \overset{1}{\underset{\mathcal{A}}{\vdash}} (q', w, \beta\alpha)$ falls $(q', \beta) \in \delta(q, a, Z)$
Für Konfiguration $K_0, \ldots, K_n$ ist definiert

$$K_0 \overset{n}{\underset{\mathcal{A}}{\vdash}} K_n :\Leftrightarrow \begin{cases} K_0 = K_n & \text{falls} \quad n = 0 \\ K_0 \overset{1}{\underset{\mathcal{A}}{\vdash}} K_1 \overset{1}{\underset{\mathcal{A}}{\vdash}} K_2 \ldots \overset{1}{\underset{\mathcal{A}}{\vdash}} K_n & \text{falls} \quad n > 0 \end{cases}$$

$K \overset{*}{\underset{\mathcal{A}}{\vdash}} K' \Leftrightarrow K \overset{n}{\underset{\mathcal{A}}{\vdash}} K'$ für ein $n \in \mathbb{N}$.

$\mathcal{A}$ akzeptiert w durch einen Endzustand, falls $(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q, \varepsilon, \alpha)$ für ein $q \in F,\ \alpha \in \Phi^*$.

$\mathcal{A}$ akzeptiert w durch den Nullkeller, falls $(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q, \varepsilon, \varepsilon)$ für q beliebig.
$L(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A}$ akzeptiert W durch Endzustand$\}$
$N(\mathcal{A}) = \{w \in \Sigma^* : \mathcal{A}$ akzeptiert W durch den Nullkeller$\}$

Beispiel:
$\mathcal{A} =< \{q_0, q_1\}, \{a, b, c\}, \{Z_0, Z_a, Z_b\}, \delta, q_0, Z_0, \emptyset >$

$$\left.\begin{array}{ll} \delta(q_0, a, Z) = & \{(q_0, Z_a Z)\} \\ \delta(q_0, b, Z) = & \{(q_0, Z_b Z)\} \\ \delta(q_0, c, Z) = & \{(q_1, Z)\} \end{array}\right\} \forall Z \in \{Z_0, Z_a, Z_b\}$$

$$\begin{array}{ll} \delta(q_1, a, Z_a) = & \{(q_1, \varepsilon)\} \\ \delta(q_1, b, Z_b) = & \{(q_1, \varepsilon)\} \\ \delta(q_1, \varepsilon, Z_0) = & \{(q_1, \varepsilon)\} \end{array}$$

Für alle anderen Argumente hat δ den Wert $\emptyset$.
$N(\mathcal{A}) = \{wcw^R :\ w \in (a \cup b)^*\}$

Schrittfolge

$$\begin{aligned}(q_0, abcba, Z_0) &\overset{1}{\vdash} (q_0, bcba, Z_aZ_0)\\ &\overset{1}{\vdash} (q_0, cba, Z_bZ_aZ_0)\\ &\overset{1}{\vdash} (q_1, ba, Z_bZ_aZ_0)\\ &\overset{1}{\vdash} (q_1, a, Z_aZ_0)\\ &\overset{1}{\vdash} (q_1, \varepsilon, Z_0)\\ &\overset{1}{\vdash} (q_1, \varepsilon, \varepsilon)\end{aligned}$$

(Dieser KA ist deterministisch, da $\delta(q_1, a, Z_0) = \delta(q_1, b, Z_0) = \delta(q_1, c, Z_0) = \emptyset$.)

Ein *echt* nichtdeterministisches Beispiel ist das folgende:
$\mathcal{A} =< \{q_0, q_1\}, \{a, b\}, \{Z_0, Z_a, Z_b\}, \delta, q_0, Z_0, \emptyset >$

$$\begin{aligned}\delta(q_0, \varepsilon, Z_0) &= \{(q_0, \varepsilon)\}\\ \delta(q_0, a, Z_0) &= \{(q_0, Z_aZ_0)\}\\ \delta(q_0, b, Z_0) &= \{(q_0, Z_bZ_0)\}\\ \delta(q_0, a, Z_a) &= \{(q_0, Z_aZ_a), (q_1, \varepsilon)\}\\ \delta(q_0, b, Z_b) &= \{(q_0, Z_bZ_b), (q_1, \varepsilon)\}\\ \delta(q_0, a, Z_b) &= \{(q_1, Z_aZ_b)\}\\ \delta(q_0, b, Z_a) &= \{(q_1, Z_bZ_a)\}\\ \delta(q_1, a, Z_a) &= \{(q_1, \varepsilon)\}\\ \delta(q_1, b, Z_b) &= \{(q_1, \varepsilon)\}\\ \delta(q_1, \varepsilon, Z_0) &= \{(q_1, \varepsilon)\}\end{aligned}$$

Für alle anderen Argumente hat δ den Wert $\emptyset$.
$N(\mathcal{A}) = \{ww^R: \; w \in (a \cup b)^*\}$

In den beiden Fällen, in denen der Wert von δ eine Zweiermenge ist, besteht (nichtdeterministisch) die Option, entweder den Stapel weiter aufzustocken oder mit dem Abbau des Stapels zu beginnen (Übergang von w zu w^R). Man kann zeigen, daß im zweiten Beispiel kein äquivalenter deterministischer KA zur Verfügung steht.

Satz 6.3

Zu jedem KA $\mathcal{A}$ läßt sich ein KA $\mathcal{A}'$ konstruieren, für den $L(\mathcal{A}) = N(\mathcal{A}')$.

Beweis: Sei $\mathcal{A} =< Q, \Sigma, \Phi, \delta, q_0, Z_0, F >$
$\mathcal{A}' :=< Q \cup \{q_e, q_0'\}, \Sigma, \Phi \cup \{Z_0'\}, \delta', q_0', Z_0', \emptyset >$ $\quad$ (q_e, q_0', Z_0' neu)
δ' entsteht aud δ durch Hinzunahme folgender Übergänge:

$\delta'(q_0', \varepsilon, Z_0') = \{(q_0, Z_0Z_0')\}$ (Stapel wird unterlegt)
$\delta'(q, \varepsilon, Z) \ni (q_e, \varepsilon)$ falls $q \in F, Z \in \Phi \cup \{Z_0'\}$

$\delta'(q_e, \varepsilon, Z) = \{(q_e, \varepsilon)\}$, falls $Z \in \Phi \cup \{Z'_0\}$ (falls Finalzustand erreicht, wird Stapel abgebaut.)

Sei $w \in L(\mathcal{A})$, d.h. $(q_0, w, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q, \varepsilon, \gamma)$ mit $q \in F$. Offenbar gilt dann:

$$(q'_0, w, Z'_0) \overset{1}{\underset{\mathcal{A}'}{\vdash}} \underbrace{(q_0, w, Z_0 Z'_0) \overset{*}{\underset{\mathcal{A}'}{\vdash}} (q, \varepsilon, \gamma Z'_0)}_{\text{Simulation von } \mathcal{A} \text{ in } \mathcal{A}'} \overset{*}{\underset{\mathcal{A}'}{\vdash}} (q_e, \varepsilon, \varepsilon), \text{ d.h. } w \in N(\mathcal{A}')$$

Sei umgekehrt $(q'_0, w, Z'_0) \overset{*}{\underset{\mathcal{A}'}{\vdash}} (q, \varepsilon, \varepsilon)$ (d.h. $w \in N(\mathcal{A}')$). Dann $q = q_e$ und die Ableitung hat die oben erwähnte Form.

Satz 6.4

Zu jedem KA $\mathcal{A}$ Läßt sich ein KA $\mathcal{A}'$ konstruieren, für den $N(\mathcal{A}) = L(\mathcal{A}')$

Beweis: Sei $\mathcal{A} =< Q, \Sigma, \Phi, \delta, q_0, Z_0, \emptyset >$
Setze $\mathcal{A}' :=< Q \cup \{q'_0, q_f\}, \Sigma, \Phi \cup \{Z'_0\}, q'_0, Z'_0, \{q_f\} >$ (q'_0, q_f, Z'_0 neu)
δ' entsteht aus δ durch Hinzufügen von $\delta'(q'_0, \varepsilon, Z'_0) = \{(q_0, Z_0 Z'_0)\}$ (Unterlegen des Stapels) und $\delta'(q, \varepsilon, Z'_0) = \{(q_f, \varepsilon)\}$ für alle $q \in Q$. Wenn $\mathcal{A}$ Keller geleert hat, geht $\mathcal{A}'$ in Finalzustand über.

Theorem 6.5

Zu jeder kontextfreien Sprache L gibt es einen KA $\mathcal{A}$, so daß
$L = N(\mathcal{A})$

Beweis: Sei $L = L(\Gamma)$ mit $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$.
Sei $\mathcal{A} :=< \{q\}, \mathcal{T}, \mathcal{V} \cup \mathcal{T}, \delta, q, S, \emptyset >$

$\delta(q, \varepsilon, A) := \{(q, w) : A \to w \in \Pi\}$ „Vorhersage“
$\delta(q, a, a) := \{(q, \varepsilon)\}$ für alle $a \in \mathcal{T}$ „Eingabevergleich“

D.h. es werden Linksableitungen vorhergesagt (versucht), die dann mit dem Gelesenen verglichen werden („Top-Down-Parse“).
Wir beweisen $L(\Gamma) \subseteq N(\mathcal{A})$ und $N(\mathcal{A}) \subseteq L(\Gamma)$ jeweils unter Verwendung einer Hilfsbehauptung.

1. Hilfsbehauptung

Falls $S \overset{*}{\underset{\Gamma}{\Rightarrow}}{}^{\text{links}}\, uv$ mit $u \in \mathcal{T}^*$ für $v \in (\mathcal{V} \circ (\mathcal{V} \cup \mathcal{T})^*) \cup \{\varepsilon\}$ (d.h. v beginnt mit einer Variablen oder ist das leere Wort), dann gilt
$(q, uw, S) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q, w, v)$ für alle $w \in \mathcal{T}^*$

Beweis der Hilfsbehauptung (Induktion über *)

$S \overset{0}{\Rightarrow} S$ klar, da $u = \varepsilon$ und $v = S$

Sei $S \underset{\Gamma}{\overset{n}{\Rightarrow}}^{\text{links}} uBv \underset{\Gamma}{\overset{1}{\Rightarrow}} uv_1v_2v$ mit $B \to v_1v_2 \in \Pi,\ v_1 \in \mathcal{T}^*,\ v_2 \in (\mathcal{V} \circ (\mathcal{V} \cup \mathcal{T})^*) \cup \{\varepsilon\}$.

$$\begin{aligned} \text{Damit:} \quad (q, uv_1w, S) &\underset{\mathcal{A}}{\overset{*}{\vdash}} (q, v_1w, Bv) \text{ (nach I.V.)} \\ &\underset{\mathcal{A}}{\overset{1}{\vdash}} (q, v_1w, v_1v_2v) \text{ (Vorhersage)} \\ &\underset{\mathcal{A}}{\overset{*}{\vdash}} (q, w, v_2v) \text{ (Eingabevergleich).} \end{aligned}$$

Aus dieser Hilfsbehauptung ergibt sich: Falls $u \in L(\Gamma)$, d.h. $S \underset{\Gamma}{\overset{*}{\Rightarrow}} u \in \mathcal{T}^*$, dann gilt:

$(q, u, S) \underset{\mathcal{A}}{\overset{*}{\vdash}} (q, \varepsilon, \varepsilon)$, d.h. $u \in N(\mathcal{A})$.

2. Hilfsbehauptung

Falls $(q, uv, S) \underset{\mathcal{A}}{\overset{*}{\vdash}} (q, v, w)$ für $u, v \in \mathcal{T}^*,\ w \in (\mathcal{V} \cup \mathcal{T})^*$, dann gilt $S \underset{\Gamma}{\overset{*}{\Rightarrow}}^{\text{links}} uw$.

Beweis: (Induktion über die Länge der Schrittfolge)

Länge 0: $u = \varepsilon,\ w = S$ ergibt $S \underset{\Gamma}{\overset{*}{\Rightarrow}}^{\text{links}} S$

Länge > 0: 2 Fälle gemäß Definition von δ:

1. $(q, uav, S) \overset{n}{\vdash} (q, av, aw) \overset{1}{\vdash} (q, v, w)$.
   Nach I.V.: $S \underset{\Gamma}{\overset{*}{\Rightarrow}}^{\text{links}} uaw$.

2. $(q, uv, S) \overset{n}{\vdash} (q, v, Bw) \overset{1}{\vdash} (q, v, xw)$ mit $B \to x \in \Pi$.
   Nach I.V.: $S \overset{*}{\Rightarrow}^{\text{links}} uBw$
   $\overset{*}{\Rightarrow} uxw$

Aus dieser Hilfsbehauptung ergibt sich: Falls $u \in N(\mathcal{A})$, d.h $(q, u\varepsilon, S) \underset{\mathcal{A}}{\overset{*}{\vdash}} (q, \varepsilon, \varepsilon)$, dann gilt $S \underset{\Gamma}{\overset{*}{\Rightarrow}}^{\text{links}} u$, d.h. $u \in L(\Gamma)$

Beispiel: $\Gamma = \langle \{S\}, \{a, b\}, \Pi, S \rangle$

$\Pi :\ S \to aSb | \varepsilon$

$\delta(q, \varepsilon, S) = \{(q, aSb), (q, \varepsilon)\}$
$\delta(q, a, a) = \delta(q, b, b) = \{(q, \varepsilon)\}$

$$
\begin{array}{llll}
\text{Schrittfolge:} & (q, aabb, S) & \overset{1}{\vdash} (q, aabb, aSb) & \text{(Vorhersage)} \\
& & \overset{1}{\vdash} (q, abb, Sb) & \text{(Vergleich)} \\
& & \overset{1}{\vdash} (q, abb, aSbb) & \text{(Vorhersage)} \\
& & \overset{1}{\vdash} (q, bb, Sbb) & \text{(Vergleich)} \\
& & \overset{1}{\vdash} (q, bb, bb) & \text{(Vorhersage)} \\
& & \overset{1}{\vdash} (q, b, b) & \text{(Vergleich)} \\
& & \overset{1}{\vdash} (q, \varepsilon, \varepsilon) & \text{(Vergleich)}
\end{array}
$$

Bemerkung: Der konstruierte Automat ist nichtdeterministisch.

Theorem 6.6

Ist $\mathcal{A}$ KA, dann ist $N(\mathcal{A})$ kf Sprache.

Beweis: Sei $\mathcal{A} =< Q, \Sigma, \Phi, \delta, q_0, Z_0, F >$

Setze $\Gamma :=< \mathcal{V}, \mathcal{T}, \Pi, S >$
$\mathcal{T} := \Sigma$
$\mathcal{V} := (Q \times \Phi \times Q) \cup \{S\}$ (S neu)
Π enthält folgende Produktionen:

$$
\begin{array}{rcll}
S & \rightarrow & (q_0, Z_0, q) & \text{für alle } q \in Q \\
(q, Z, q') & \rightarrow & a, & \text{falls } \delta(q, a, Z) \ni (q', \varepsilon) \\
(q, Z, q_k) & \rightarrow & a(q', Z_1, q_1)(q_1, Z_2, q_2) \cdots (q_{k-1}, Z_k, q_k) & \\
& & \quad \text{für alle } q_1, \ldots, q_{k-1} \in Q \text{ für } k > 0, & \text{falls } \delta(q, a, Z) \ni (q', Z_1 \cdots Z_k)
\end{array}
$$

Dabei kann $a = \varepsilon$ sein.

Intuitiv: Die Variable (q, Z, p) erlaubt die Produktion derjenigen Wörter, die beim Übergang von q zu p das Stapelsymbol Z zu eliminieren erlauben. Dies besagt folgende Hilfsbehauptung:

Hilfsbehauptung

Für alle $x \in \mathcal{T}^*$: $(q, Z, p) \overset{*}{\underset{\Gamma}{\Rightarrow}} x$ g.d.w. $(q, x, Z) \overset{*}{\underset{\mathcal{A}}{\vdash}} (p, \varepsilon, \varepsilon)$

Aus der Hilfsbehauptung ergibt sich der Beweis des Theorems wie folgt:

$$
\begin{array}{ll}
x \in N(\mathcal{A}) & \text{g.d.w. } (q_0, x, Z_0) \overset{*}{\underset{\mathcal{A}}{\vdash}} (p, \varepsilon, \varepsilon) \text{ für ein } p \\
& \text{g.d.w. } (q_0, Z_0, p) \overset{*}{\underset{\Gamma}{\Longrightarrow}} x, \text{ d.h. } S \overset{*}{\underset{\Gamma}{\Longrightarrow}} x.
\end{array}
$$

Beweis der Hilfsbehauptung

„$\Leftarrow$“: Induktion über i in $\overset{i}{\underset{\mathcal{A}}{\vdash}}$

$i=0$: tritt nicht auf, da $Z \neq \varepsilon$

$i = 1$: $x = a$ mit $a \in \mathcal{T} \cup \{\varepsilon\}$ mit $\delta(q,a,Z) \ni (p,\varepsilon)$. Also $(q,Z,p) \to a \in \Pi$, d.h. $(q,Z,p) \overset{1}{\underset{\Gamma}{\Rightarrow}} a$.

$i > 1$: $(q,ay,Z) \overset{1}{\vdash} (q_1,y,Z_1\dots Z_n) \overset{i}{\vdash} (p,\varepsilon,\varepsilon)$. Wir schreiben: $y = y_1 \dots y_n$ wie folgt: y_1 ist das Präfix von y, nach dessen Lektüre der Stapel $n-1$ Symbole hat und Z_2 oben liegt, $y_1 y_2$ das Präfix von y, nach dessen Lektüre der Stapel $n-2$ Symbole hat und Z_3 oben liegt, usw.

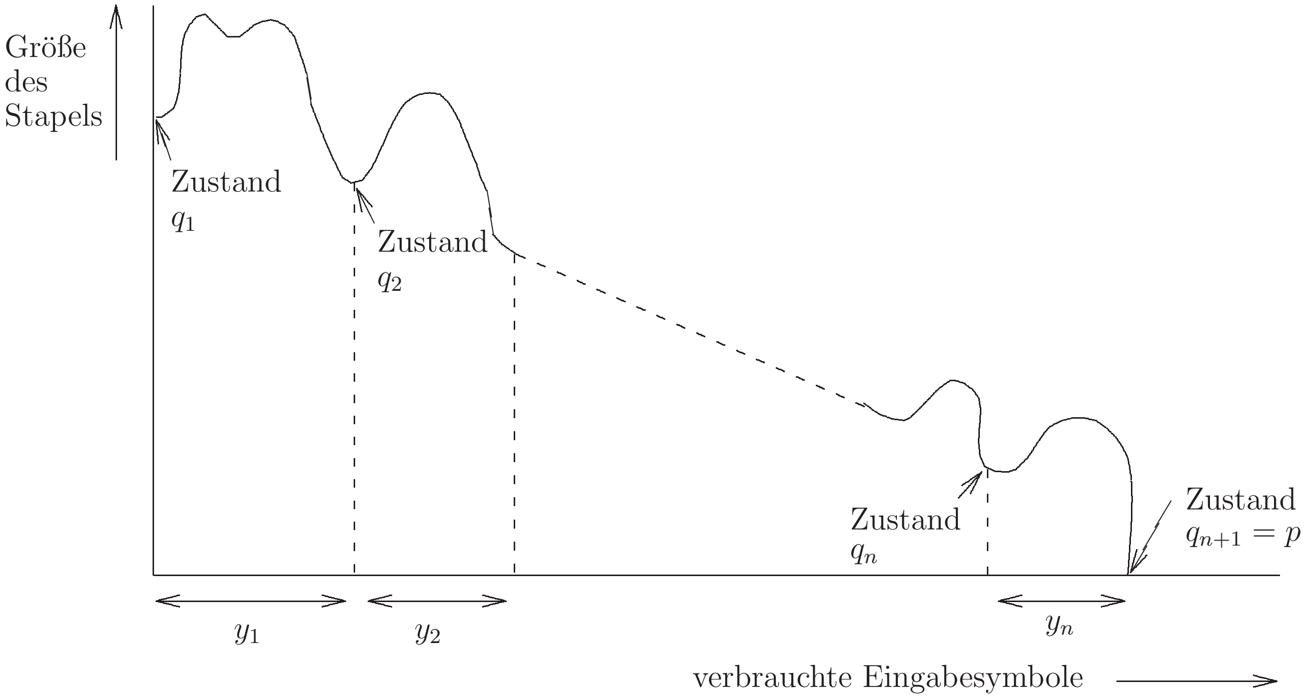


Abb.: Stapelgröße als Funktion der verbrauchten Eingabe (Skizze nach Hopcroft & Ullman (1990))

Es existieren Zustände $q_2,\dots,q_{n+1} = p$ mit $(q_j,y_j,Z_j) \overset{\leq i}{\vdash} (q_{j+1},\varepsilon,\varepsilon)$ $(1 \leq j \leq n)$.
Nach I.V.: $(q_j,Z_j,q_{j+1}) \overset{*}{\underset{\Gamma}{\Rightarrow}} y_j$.

Dem ersten Übergang von Konfigurationen entspricht:
$(q,Z,p) \to a(q_1,Z_1,q_2)(q_2,Z_2,q_3)\dots(q_n,Z_n,p)$. Daraus: $(q,Z,p) \overset{*}{\underset{\Gamma}{\Rightarrow}} \underbrace{ay_1\dots y_n}_{ay}$.

”$\Rightarrow$”: Sei $(q,Z,p) \overset{i}{\underset{\Gamma}{\Rightarrow}} x$. Induktion über i:

$i=0$: tritt nicht auf, da $x \in \mathcal{T}^*$, d.h. $x \notin \mathcal{V}$

$i=1$: $(q,Z,p) \to x \in \Pi$ und $x \in \mathcal{T} \cup \{\varepsilon\}$, d.h. $\delta(q,x,Z) \ni (p,\varepsilon)$, d.h. $(q,x,Z) \overset{1}{\underset{\mathcal{A}}{\vdash}} (p,\varepsilon,\varepsilon)$

$i > 1$: $(q, Z, p) \overset{1}{\underset{\Gamma}{\Rightarrow}} a(q_1, Z_1, q_2)(q_2, Z_2, q_3) \cdots (q_n, Z_n, p) \overset{i}{\Rightarrow} x$ mit $\delta(q, a, Z) \ni (q_1, Z_1 \dots Z_n)$.
Dann nach Lemma 5.3 $x = ax_1 \dots x_n$ mit $(q_j, Z_j, q_{j+1}) \overset{\leqq i}{\Rightarrow} x_j$.
Mit I.V.: $(q_j, x_j, Z_j) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q_{j+1}, \varepsilon, \varepsilon)$.
Damit: $(q_j, x_j x_{j+1} \dots x_n, Z_j Z_{j+1} \dots Z_n) \overset{*}{\underset{\mathcal{A}}{\vdash}} (q_{j+1}, x_{j+1} \dots x_n, Z_{j+1} \dots Z_n)$
Dem ersten Schritt der Ableitung in Γ entspricht

$$(q, \underset{\substack{\| \\ ax_1 \dots x_n}}{x}, Z) \overset{1}{\underset{\mathcal{A}}{\vdash}} (q_1, x_1 \dots x_n, Z_1 \dots Z_n)$$

Durch Hintereinanderschaltung der Übergänge zur Entfernung von Z_j für jedes j ergibt sich $(q, x, Z) \overset{*}{\vdash} (p, \varepsilon, \varepsilon)$

Beispiel:

$\mathcal{A} =< \{q_0, q_1\}, \{0, 1\}, \{Z_0, Z_1\}, \delta, q_0, Z_0, \emptyset >$

$\delta(q_0, 0, Z_0) = \{(q_0, Z_1 Z_0)\}$ $(*)$
$\delta(q_0, 0, Z_1) = \{(q_0, Z_1 Z_1)\}$ $(**)$
$\delta(q_0, 1, Z_1) = \{(q_1, \varepsilon)\}$
$\delta(q_1, 1, Z_1) = \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, Z_1) = \{(q_1, \varepsilon)\}$
$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$

Es gilt: $N(\mathcal{A}) = \{0^n 1^m :\ 0 < n \geq m > 0\}\}$

$\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$ mit $\mathcal{T} = \{0, 1\}$
$\mathcal{V} = \{S, (q_0, Z_0, q_0), (q_0, Z_0, q_1), (q_0, Z_1, q_0), (q_0, Z_1, q_1),$
$\quad (q_1, Z_0, q_0), (q_1, Z_0, q_1), (q_1, Z_1, q_0), (q_1, Z_1, q_1)\}$

Produktionen:

(1) $S \longrightarrow (q_0, Z_0, q_0) | (q_0, Z_0, q_1)$

$$(2)\ \begin{cases} (q_0, Z_1, q_1) \to 1 \\ (q_1, Z_1, q_1) \to 1 | \varepsilon \\ (q_1, Z_0, q_1) \to \varepsilon \end{cases}$$

$$(3) \text{ mit } (*) \begin{cases} (q_0, Z_0, q_0) \to 0(q_0, Z_1, q_0)\ (q_0, Z_0, q_0) \\ (q_0, Z_0, q_0) \to 0(q_0, Z_1, q_1)\ (q_1, Z_0, q_0) \\ (q_0, Z_0, q_1) \to 0(q_0, Z_1, q_0)\ (q_0, Z_0, q_1) \\ (q_0, Z_0, q_1) \to 0(q_0, Z_1, q_1)\ (q_1, Z_0, q_1) \end{cases}$$

$$(3) \text{ mit } (**) \begin{cases} (q_0, Z_1, q_0) \to 0(q_0, Z_1, q_0)\ (q_0, Z_1, q_0) \\ (q_0, Z_1, q_0) \to 0(q_0, Z_1, q_1)\ (q_1, Z_1, q_0) \\ (q_0, Z_1, q_1) \to 0(q_0, Z_1, q_0)\ (q_0, Z_1, q_1) \\ (q_0, Z_1, q_1) \to 0(q_0, Z_1, q_1)\ (q_1, Z_1, q_1) \end{cases}$$

Es läßt sich durch Umformung zeigen, daß dies gleichwertig ist zur Grammatik mit den Produktionen:
$S \to 0D \quad D \to 0D|0D1|1$
Offenbar erzeugt diese Grammatik die Sprache $\{0^n1^m : \ 0 < n \geq m > 0\}$.

7 Eigenschaften kontextfreier Sprachen

Wir beweisen das Pumping Lemma für kontextfreie Sprachen und zeigen einige seiner Konsequenzen. Ferner zeigen wir den Zusammenhang zwischen dem Postschen Korrespondenzproblem und der Entscheidbarkeit gewisser Eigenschaften kontextfreier Sprachen.

Theorem 7.1 (Pumping Lemma)

Zu jeder kontextfreien Sprache L läßt sich eine Zahl n angeben, so daß sich jedes $w \in L$ mit $|w| \geq n$ als $w = u_1 v_1 x v_2 u_2$ schreiben läßt, wobei gilt:

1. $v_1 v_2 \neq \varepsilon$
2. $|v_1 x v_2| \leq n$
3. für alle $i \geq 0$ gilt $u_1 v_1^i x v_2^i u_2 \in L$

Beweis: Sei Γ kfG in Chomsky-Normalform mit $L(\Gamma) = L \backslash \{\varepsilon\}$

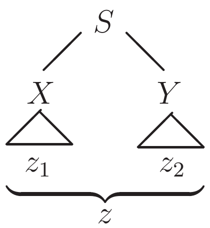
Hilfsbehauptung

Sei i die Höhe eines Ableitungsbaumes für ein z in Γ. Dann ist $|z| \leq 2^{i-2}$.

Beweis: Induktion über i:

$i = 2$: trivial, da dann der Baum so aussieht: $\begin{array}{c} S \\ | \\ a \end{array}$ $\qquad |a| \leq 2^{(2-2)} = 1$

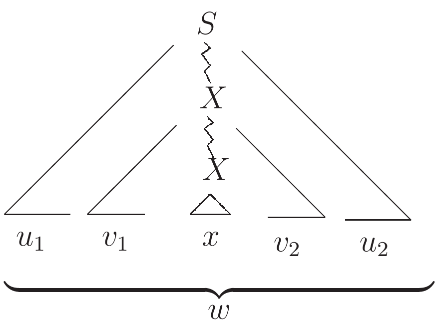
$i > 2$: Baum sieht so aus:



Nach I.V. gilt: $|z_1| \leq 2^{i-3}$, $|z_2| \leq 2^{i-3}$, d.h. $|z| = |z_1| + |z_2| \leq 2^{i-3} + 2^{i-3} = 2^{i-2}$.

Fortsetzung des Beweises des Theorems.

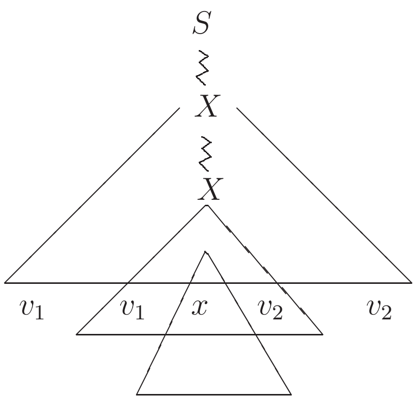
Setze $n := 2^k$, wobei k Zahl der Variablen in Γ: Wegen $|w| \geq 2^k > 2^{k-1}$ hat nach der Hilfsbehauptung jeder Ableitungsbaum für w die Höhe $> k + 1$, d.h. es gibt mindestens einen Zweig, der mindestens die Länge $k + 2$ hat, d.h. $k + 1$ Knoten, die mit Variablen markiert sind. Also taucht in einem solchen Zweig eine Variable X mindestens zweimal auf. Wir betrachten die Teilbäume, die an diesen beiden Vorkommen von X „hängen“:



1. Wir können annehmen, daß $v_1 v_2 \neq \varepsilon$. Andernfalls kann man den oberen X-Baum durch den unteren X-Baum ersetzen („pruning“), so daß der ausgewählte Zweig eine kürzere Länge erhält. Dieses Verfahren muß irgendwann bei einem Zweig der Länge $\geq k+2$ abbrechen.

2. Wir können ferner annehmen, daß unter dem oberen X außer X keine Variable im gewählten Zweig doppelt vorkommt. Ansonsten wählen wir eine entsprechende andere Variable aus. Also hat der obere X-Baum eine Höhe $\leq k+2$.
   Damit gilt nach der Hilfsbehauptung: $|v_1 x v_2| \leq 2^{(k+2)-2} = 2^k = n$

3. Einen Ableitungsbaum für $u_1 x u_2$ erhält man durch Ersetzen des oberen X-Baum durch den unteren X-Baum. Einen Ableitungsbaum für $u_1 v_1^i x v_2^i u_2$ für $i > 0$ erhält man durch $(i-1)$-faches Ersetzen des unteren X-Baumes durch den oberen („splicing“):



Beispiel: $L = \{a^m b^m c^m : m \geq 0\}$ ist nicht kontextfrei.

Beweis: Sei n wie im Theorem. Dann:
$a^n b^n c^n = u_1 v_1 x v_2 u_2$

Da $|v_1 x v_2| \leq n$, sind in $v_1 x v_2$ höchstens zwei verschiedene Buchstaben vertreten.
Wegen $v_1 v_2 \neq \varepsilon$ ist beim Aufblasen von $v_1 x v_2$ zu $v_1^2 x v_2^2$ mindestens ein Buchstabe nicht betroffen.

Korollar 7.2

Die Menge der kf Sprachen ist nicht unter Durchschnitt abgeschlossen.

Beweis:

$L_1 := \{a^m b^n c^m :\ m, n \geq 0\}$ und $L_2 := \{a^n b^m c^m :\ m, n \geq 0\}$ sind kontextfrei.
$L_1 \cap L_2 = \{a^m b^m c^m :\ m \geq 0\}$ jedoch nicht.

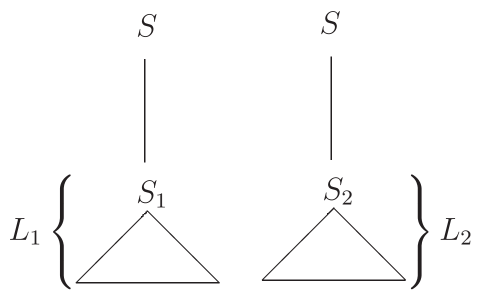
Lemma 7.3

Seien L_1 und L_2 kontextfrei. Dann ist auch

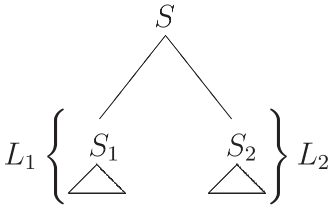
1. $L_1 \cup L_2$ kontextfrei
2. $L_1 \circ L_2$ kontextfrei
3. L_1^* kontextfrei

Beweis:
Sei $L_1 = L(\Gamma_1)$ mit $\Gamma_1 = \langle \mathcal{V}_1, \mathcal{T}_1, \Pi_1, S_1 \rangle$
$L_2 = L(\Gamma_2)$ mit $\Gamma_2 = \langle \mathcal{V}_2, \mathcal{T}_2, \Pi_2, S_2 \rangle$
Wir nehmen an, daß $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$

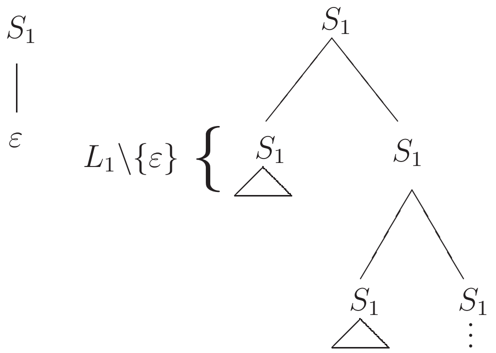
1. $L_1 \cup L_2 := L(\Gamma)$ mit $\Gamma = \langle \mathcal{V}_1 \cup \mathcal{V}_2 \cup \{S\}, \mathcal{T}_1 \cup \mathcal{T}_2, \Pi_1 \cup \Pi_2 \cup \{S \to S_1 | S_2\}, S \rangle$ (S neu).



2. $L_1 \circ L_2 := L(\Gamma)$ mit $\Gamma = \langle \mathcal{V}_1 \cup \mathcal{V}_2 \cup \{S\}, \mathcal{T}_1 \cup \mathcal{T}_2, \Pi_1 \cup \Pi_2 \cup \{S \to S_1 S_2\}, S\rangle$ (S neu).



3. $L_1^* := L(\Gamma)$ mit $\Gamma = \langle \mathcal{V}_1, \mathcal{T}_1, \Pi_1 \cup \{S_1 \to S_1 S_1 | \varepsilon\}, S_1\rangle$



Lemma 7.4

Wenn $L \subseteq \Sigma^*$ kf ist, dann nicht notwendigerweise $\Sigma^* \backslash L$.

Beweis:
Ansonsten wäre mit L_1 und L_2 auch $L_1 \cap L_2 = \Sigma^* \backslash ((\Sigma^* \backslash L_1) \cup (\Sigma^* \backslash L_2))$ kf, da nach Lemma 7.3 kfe Sprachen unter Vereinigung „$\cup$“ abgeschlossen sind. Kfe Sprachen sind jedoch nach Korollar 7.2 nicht unter Durchschnitt „$\cap$“ abgeschlossen.

Satz 7.5

Sei L kf Sprache und R reguläre Sprache. Dann ist $L \cap R$ kf Sprache.

Beweis:
Sei $L = L(A_L)$ und $R = L(A_R)$, wobei $\mathcal{A}_L = \langle Q_L, \Sigma, \Phi, \delta_L, q_L, Z_0, F_L\rangle$ KA,
$\mathcal{A}_R = \langle Q_R, \Sigma, \delta_R, q_R, F_R\rangle$ DEA.
Setze $\mathcal{A}_{LR} := \langle Q_L \times Q_R, \Sigma, \Phi, \delta_{LR}, (q_L, q_R), Z_0, F_L \times F_R\rangle$ mit
$\delta_{LR}((q,p), a, Z) \ni ((q',p'), \alpha)$ g.d.w. $\delta_L(q, a, Z) \ni (q', \alpha)$ und $\delta_R(p, a) = p'$,
wobei $\delta_R(p, \varepsilon) = p$ festgesetzt sei. Es gilt:
$((q_L, q_R), w, Z_0) \ \underset{\mathcal{A}_{LR}}{\overset{*}{\vdash}} \ ((q,p), \varepsilon, \alpha)$ g.d.w. $(q_L, w, Z_0) \ \underset{\mathcal{A}_L}{\overset{*}{\vdash}} \ (q, \varepsilon, \alpha)$ und $\delta_R^*(q_R, w) = p$
Damit $w \in L(\mathcal{A}_{LR})$ g.d.w. $w \in L(\mathcal{A}_L) \cap L(\mathcal{A}_R)$ da $F_{LR} = F_L \times F_R$.

Lemma 7.6

Sei $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S\rangle$ kfG in Chomsky-Normalform. Sei $A \underset{\Gamma}{\overset{*}{\Rightarrow}} w$ mit $A \in \mathcal{V}, w \in \mathcal{T}^*$. Dann: $A \underset{\Gamma}{\overset{2|w|-1}{\Longrightarrow}} w$

Beweis:
$|w| = 1: \quad A \to w \in \Pi.$
$|w| > 1: \quad A \overset{1}{\Rightarrow} BC \overset{*}{\Rightarrow} \overbrace{u_1 u_2}^{w}.$
Wir wissen: $B \overset{*}{\Rightarrow} u_1,\ C \overset{*}{\Rightarrow} u_2$ (Vgl. Lemma 5.3).
Nach I.V. $B \overset{2|u_1|-1}{\Longrightarrow} u_1,\ C \overset{2|u_2|-1}{\Longrightarrow} u_2$
Damit erhält man: $A \overset{1+2|u_1|-1+2|u_2|-1}{\Longrightarrow} u_1 u_2$, d.h. $A \overset{2|w|-1}{\Longrightarrow} w$

Satz 7.7

Es gibt einen Algorithmus, der, angewendet auf eine kfG $\Gamma = \langle \mathcal{V}, \mathcal{T}, \Pi, S\rangle$ und ein Wort $w \in \mathcal{T}^*$ entscheidet, ob $w \in L(\Gamma)$

Beweis:
Falls $w = \varepsilon$, berechnet man gemäß dem Algorithmus von Lemma 5.8 $\{A \in \mathcal{V} : A \underset{\Gamma}{\overset{*}{\Rightarrow}} \varepsilon\}$ und prüft, ob S in dieser Menge ist. Ansonsten transformiert man Γ in CNF Γ' und prüft bei allen Ableitungen der Länge $2|w| - 1$ in Γ', ob sie mit w enden.

Bemerkung:

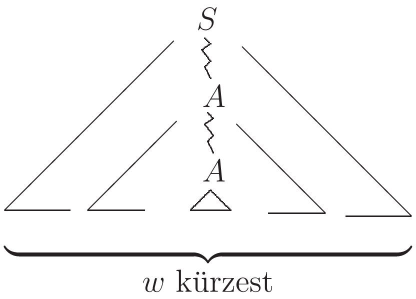
Wenn keine Variable aus Γ' mehr als k Produktionen hat, dann gibt es maximal $k^{(2|w|-1)}$ Ableitungen der Länge $2|w| - 1$. In Kapitel 8 betrachten wir effizientere Algorithmen.

Satz 7.8

Es gibt einen Algorithmus, der für kfG Γ testet, ob $L(\Gamma) = \emptyset$.

Beweis:
1. Möglichkeit: Prüfe, ob S terminalisierbar ist (Lemma 5.6)
2. Möglichkeit: Teste, ob $\varepsilon \in L(\Gamma)$ nach dem vorigen Satz. Falls $\varepsilon \notin L(\Gamma)$, bilden wir eine ε-produktionsfreie und kettenproduktionsfreie äquivalente kfG Γ' (Lemma 5.8 und Lemma 5.10). Sei n Anzahl der Variablen in Γ'. Dann hat jeder Zweig eines Ableitungsbaumes für ein *kürzestes* $w \in L(\Gamma)$ höchstens n Variablenknoten: Ansonsten käme nämlich eine

Variable A doppelt vor.



Durch pruning (Ersetzung des oberen A-Baumes durch den unteren) erhielte man dann $w' \in L(\Gamma)$ mit $|w'| < |w|$. (Wegen Kettenproduktionsfreiheit und ε-Produktionsfreiheit müßte oberer A-Baum echt breiter sein als unterer A-Baum.)
Man teste alle Ableitungsbäume von Γ' der maximalen Höhe $n+1$ und prüfe, ob sie ein Terminalwort ergeben.

Satz 7.9

Es gibt einen Algorithmus, der testet, ob für eine kfG Γ die Sprache $L(\Gamma)$ endlich ist.

Beweis: Wir zeigen folgendes: Sei Γ in Chomsky-Normalform mit exakt n Variablen. Dann ist $L(\Gamma)$ unendlich g.d.w. es ein $w \in L(\Gamma)$ gibt mit $2^n < |w| \le 2^{n+1}$
„$\Leftarrow$“: Pumping Lemma (Aufblasen)
„$\Rightarrow$“: Es gibt Wörter der Länge $> 2^{n+1}$. Sei w ein kürzestes davon. Nach dem Pumping Lemma ist $w = u_1v_1xv_2u_2$ und $u_1xu_2 \in L(\Gamma)$
$|u_1xu_2| \ge |u_1u_2| = \underbrace{|w| - |v_1xv_2|}_{>2^n}$, wobei $|w| > 2^{n+1}$ und $|v_1xv_2| \le 2^n$ (Pumping Lemma).
D.h. $2^n < \underbrace{|u_1xu_2| \le 2^{n+1}}_{\text{da } w \text{ kürzestes Wort mit } |w| > 2^{n+1}}$
Der Algorithmus muß $w \in L(\Gamma)$ für alle w mit $2^n < |w| \le 2^{n+1}$ prüfen.

Definition 7.10 (Postsches Korrespondenzproblem)

Ein Postsches Korrespondenzsystem ist ein Paar von gleichlangen Listen
$u_1, \ldots, u_n$
$v_1, \ldots, v_n$
von Wörtern über einem Alphabet Σ. Eine nichtleere endliche Folge $i_1, \ldots, i_m$ von natürlichen Zahlen zwischen 1 und n heißt Lösung des Systems, falls
$u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$.
Das Postsche Korrespondenzproblem lautet wie folgt: Gibt einen Algorithmus, der, angewendet auf ein Postsches Korrespondenzsystem, entscheidet, ob es eine Lösung hat?

Bemerkung:

Man stelle sich das Korrespondenzsystem als endliche Menge verschiedener Dominos vor $\boxed{\frac{u_i}{v_i}}$, wobei von jedem Domino beliebig viele Exemplare zur Verfügung stehen. Eine Lösung des Systems ist dann eine Aneinanderreihung von Dominos $\boxed{\frac{u_{i_1}}{v_{i_1}}}\ \boxed{\frac{u_{i_2}}{v_{i_2}}}\ \ldots\ \boxed{\frac{u_{i_m}}{v_{i_m}}}$ derart, daß oben und unten dasselbe Wort steht.

Zur Terminologie: Ein Algorithmus, der entscheidet, ob ein beliebiges Potsches Korrespondenz*system* eine Lösung hat, wäre eine positive Lösung des Postschen Korrespondenz*problems*.

Lemma 7.11

Seien Γ_1, Γ_2 kfG. Wenn es einen Algorithmus zur Entscheidung von $L(\Gamma_1) \cap L(\Gamma_2) = \emptyset$ gibt, dann hat das Postsche Korrespondenzproblem eine positive Lösung.

Beweis:
Sei Postsches Korrespondenzsystem
$u_1, \ldots, u_n;\ v_1, \ldots, v_n$
gegeben mit $u_i, v_i \in \Sigma^*$.

Betrachte das Alphabet $\Sigma' := \Sigma \cup \{c_1, \ldots, c_n\},\ c_i$ neu, und die kfG
$\Gamma_1 := \langle \{S_1\}, \Sigma', \Pi_1, S_1 \rangle$, $\Gamma_2 := \langle \{S_2\}, \Sigma', \Pi_2, S_2 \rangle$.
$\Pi_1 := \{S_1 \to u_i S_1 c_i,\ S_1 \to u_i c_i : 1 \le i \le n\}$
$\Pi_2 := \{S_2 \to v_i S_2 c_i,\ S_2 \to v_i c_i : 1 \le i \le n\}$

Es gilt:
$L(\Gamma_1) = \{u_{i_1} \ldots u_{i_m} c_{i_m} \ldots c_{i_1}\}$
$L(\Gamma_2) = \{v_{i_1} \ldots v_{i_m} c_{i_m} \ldots c_{i_1}\}$

Damit gilt:
$L(\Gamma_1) \cap L(\Gamma_2) \neq \emptyset$ g.d.w. $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$ für gewisse $i_1, \ldots, i_m$
g.d.w. das gegebene Postsche Korrespondenzsystem hat Lösung.

Eine Entscheidung über $L(\Gamma_1) \cap L(\Gamma_2) = \emptyset$ wäre eine Entscheidung über die Lösbarkeit des Korrespondenzsystems.

Bemerkung zum Beweis

Die c_i haben die Funktion, die Zerlegbarkeit der Wörter $u_{i_1} \ldots u_{i_m}$ und $v_{i_1} \ldots v_{i_m}$ in zusammengehörige Paare (Dominos) sicherzustellen.

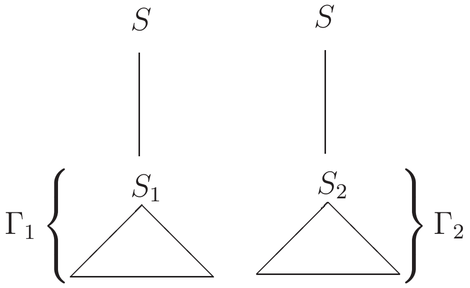
Definition 7.12

Eine kfG Γ heißt eindeutig, falls jedes von Γ erzeugte Wort genau eine Linksableitung hat. Ansonsten heißt sie mehrdeutig.

Lemma 7.13

Wenn es einen Algorithmus gibt, der entscheidet, ob eine kfG eindeutig ist, dann hat das Postsche Korrespondenzproblem eine positive Lösung.

Beweis: Wie im Beweis von Lemma 7.11 konstruieren wir die Grammatiken Γ_1 und Γ_2. Γ_1 und Γ_2 sind eindeutig. Betrachte $\Gamma := \langle\{S, S_1, S_2\}, \Sigma, \Pi_1 \cup \Pi_2 \cup \{S \to S_1, S \to S_2\}, S\rangle$



Da Γ_1 und Γ_2 eindeutig sind, ist Γ eindeutig g.d.w. $L(\Gamma_1) \cap L(\Gamma_2) = \emptyset$.

Bemerkungen:

1. Die Entscheidbarkeit zahlreicher anderer Probleme kontextfreier Sprachen impliziert eine positive Lösung des Postschen Korrespondenzproblems, z.B. (für kfGen Γ_1, Γ_2):
   $L(\Gamma_1) \cap L(\Gamma_2)$ ist unendlich
   $L(\Gamma_1) \cap L(\Gamma_2)$ ist kontextfrei
   $L(\Gamma_1) \subseteq L(\Gamma_2)$
   $L(\Gamma_1) = L(\Gamma_2)$
   $L(\Gamma_1)$ ist regulär
   $L(\Gamma_1)$ ist deterministisch kontextfrei
   (vgl. Schöning 2001, § 2.8)

2. Das Postsche Korrespondenzproblem hat eine negative Lösung (siehe Theorem 16.5).

8 Spezielle Entscheidungsalgorithmen für kontextfreie Sprachen

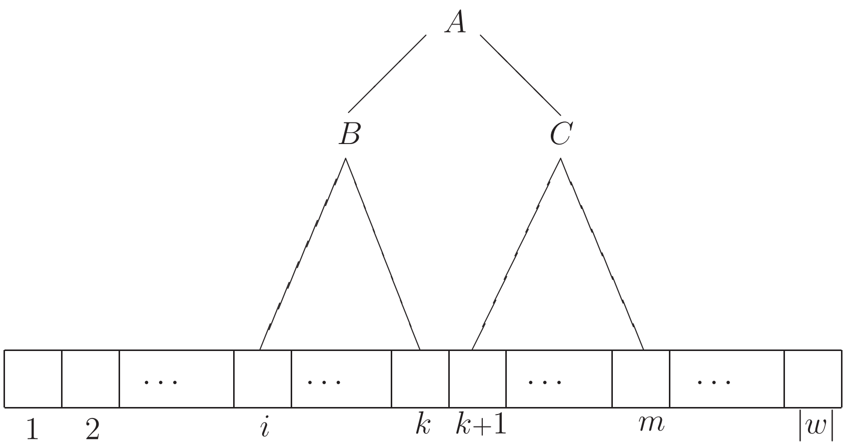
Nach Satz 7.7 gibt es einen (uneffektiven) Entscheidungsalgorithmus für kontextfreie Sprachen. Wir behandeln hier zwei effektivere Algorithmen, den von Cocke, Younger und Kasami für kontextfreie Grammatiken in Chomsky-Normalform und den von Earley, der direkt auf beliebige kontextfreie Grammatiken angewendet werden kann. Der Earley-Algorithmus ist wegen seiner Universalität und Effizienz (im allgemeinen Falle $O(n^3)$, bei vielen wichtigen Grammatiken sogar linear) in der Computerlinguistik von zentraler Bedeutung.

Definition 8.1 (CYK-Algorithmus)

Sei $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$ kfG in CNF. Sei $w \in \mathcal{T}^*$

Der Algorithmus von Cocke, Younger und Kasami (CYK) ist wie folgt definiert:
Wir definieren Mengen von Variablen $\mathcal{V}_{ij}(1 \leq i \leq j \leq |w|)$:
$\mathcal{V}_{ii} := \{A : A \to (w)_i \in \Pi\}$
$\mathcal{V}_{im} := \bigcup_{i \leq k \leq m} \{A : A \to BC \in \Pi, B \in \mathcal{V}_{ik}, C \in \mathcal{V}_{(k+1)m}\}$ für $i < m$.

Der Algorithmus antwortet positiv g.d.w. $S \in \mathcal{V}_{1|w|}$



Theorem 8.2 (Korrektheit und Vollständigkeit des CYK-Algorithmus)

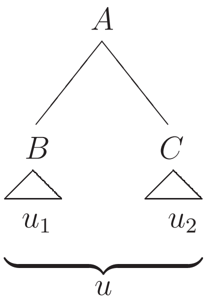
Der CYK-Algorithmus akzeptiert w g.d.w. $w \in L(\Gamma)$.

Beweis:
Wir zeigen:
$A \in \mathcal{V}_{ij}$ g.d.w. $A \overset{*}{\underset{\Gamma}{\Rightarrow}} u$ mit $u = a_1 \dots a_n, a_1 = (w)_i,\ a_n = (w)_j$, d.h. u Teilwort von w vom i-ten bis zum j-ten Zeichen.

Daraus folgt mit $A = S,\ i = 1,\ j = |w|$ die Behauptung.

„$\Rightarrow$“: leichte Induktion über $(j - i)$

„$\Leftarrow$“: Induktion über $|u|$:
$|u| = 0$: tritt nicht auf
$|u| = 1$: Dann ist $A \to u$ Produktion, d.h. $A \in \mathcal{V}_{ii}$
$|u| > 1$: Dann liegt Ableitungsbaum folgender Form vor:



Damit gilt: $B \in \mathcal{V}_{ik},\ C \in \mathcal{V}_{(k+1)j}$ für ein k mit $i \leq k < j$.
Wegen $A \to BC \in \Pi$ gilt damit $A \in \mathcal{V}_{ij}$.

Beispiel 1 (aus Loos, 1989):

Terminalalphabet: $\{a, +, *, (,)\}$
Produktionen:

$S \to T|S + T$
$T \to F|T * F$
$F \to a|(S)$

Grammatik in CNF:

$S \to a|A_1H|TI|SJ$
$T \to a|A_1H|TI$
$F \to a|A_1H$
$A_1 \to ($
$A_3 \to *$
$A_2 \to)$
$A_4 \to +$
$H \to SA_2$
$I \to A_3F$
$J \to A_4T$

Eingabe: $w = a * (a + a)$

	a 1	$*$ 2	$($ 3	a 4	$+$ 5	a 6	$)$ 7
1	$\{S, T, F\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{S, T\}$
2	–	$\{A_3\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{I\}$
3	–	–	$\{A_1\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{S, T, F\}$
4	–	–	–	$\{S, T, F\}$	$\emptyset$	$\{S\}$	$\{H\}$
5	–	–	–	–	$\{A_4\}$	$\{J\}$	$\emptyset$
6	–	–	–	–	–	$\{S, T, F\}$	$\{H\}$
7	–	–	–	–	–	–	$\{A_2\}$

$S \in \mathcal{V}_{17}$ d.h. $w \in L(\Gamma)$.

Beispiel 2:

Grammatik hinter Theorem 5.12 in CNF:

$S \to B_a B_1$
$B_1 \to A B_2$
$B_2 \to B_b B$
$A \to B_a A | a$
$B \to B_b B | b$
$B_a \to a$
$B_b \to b$

Eingabe: *aaabb*

	a 1	a 2	a 3	b 4	b 5
1	$\{A, B_a\}$	$\{A\}$	$\{A\}$	$\emptyset$	$\{S, B_1\}$
2	–	$\{A, B_a\}$	$\{A\}$	$\emptyset$	$\{S, B_1\}$
3	–	–	$\{A, B_a\}$	$\emptyset$	$\{B_1\}$
4	–	–	–	$\{B, B_b\}$	$\{B_2, B\}$
5	–	–	–	–	$\{B, B_b\}$

Definition 8.3 (Zerlegte Produktion)

Sei $\Gamma' =< \mathcal{V}', \mathcal{T}, \Pi', S >$ kfG. Dann betrachte die äquivalente Grammatik $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, \overline{S} >$ mit $\mathcal{V} := \mathcal{V}' \cup \{\overline{S}\},\ \Pi := \Pi' \cup \{\overline{S} \to S\}$.

Eine zerlegte Produktion hat die Form $A \to v_1 \bullet v_2$ für $A \to v_1 v_2 \in \Pi$. Ein Zustand des Earley-Algorithmus für die Eingabe von $w = a_1 \dots a_n$ ist ein Paar, bestehend aus einer zerlegten Produktion und einer natürlichen Zahl ≥ 0 : $\quad (A \to v_1 \bullet v_2; i)$

Definition 8.4 (Earley-Algorithmus)

Der Earley-Algorithmus konstruiert zu einer Eingabe $a_1 \ldots a_n \in \mathcal{T}^*$ eine Folge von Zustandsmengen $Q_0, \ldots, Q_n$ in folgender Weise:

(1) Initialisierung: $Q_0 := \{(\overline{S} \to \bullet S; 0)\}, Q_1 = \ldots = Q_n = \emptyset$

(2) Abschluß: Beginend mit Q_0, bilde für Q_i den Abschluß bezüglich der folgenden drei Operationen und gehe zu Q_{i+1} über, bis Q_n erreicht ist.

   (V) (Voraussage) Falls $(X \to u \bullet Av; j) \in Q_i$, füge für alle $A \to w \in \Pi$ den Zustand $(A \to \bullet w; i)$ zu Q_i hinzu.

   (E) (Eingabeüberprüfung) Falls $(X \to u \bullet av; j) \in Q_{i-1}$ und $a = a_i$, füge $(X \to ua \bullet v; j)$ zu Q_i hinzu (für $i = 0$ nicht definiert).

   (R) (Rekonstruktion) Falls $(A \to w\bullet; j) \in Q_i$, füge alle $(X \to uA \bullet v; g)$ zu Q_i, für die $(X \to u \bullet Av; g) \in Q_j$ gilt (d.h. für die $(A \to \bullet w; j)$ zu Q_j entsprechend (V) hinzugefügt wurde).

(3) Entscheidung: Falls $(\overline{S} \to S\bullet; 0) \in Q_n$, akzeptiere $a_1 \ldots a_n$, ansonsten verwerfe $a_1 \ldots a_n$.

(Für einen Algorithmus im präzisen Sinn ist es noch notwendig, die Reihenfolge der Operationen zu normieren und dementsprechend die Zustandsmengen als *geordnete* Mengen aufzufassen.)

Bemerkungen:

1. Falls $(X \to u \bullet v; j)$ Zustand, dann $X \to uv$ Produktion aus Π.

2. Intuitive Idee: $(X \to u \bullet v; j) \in Q_i$ bedeutet:
   Die Produktion $X \to uv$ wurde bei Lektüre des j-ten Zeichens des Eingabewortes vorausgesagt; jetzt ist man bei der Lektüre des i-ten Zeichens; mit dem Teil u der vorausgesagten Produktion war man bei der Lektüre des $(j+1)$-ten bis zum i-ten Zeichen erfolgreich.

Beispiel 1:

$$\begin{aligned} \Gamma: \quad & \overline{S} \to S \\ & S \to aSb \\ & S \to \varepsilon \end{aligned}$$

$L(\Gamma) = \{a^n b^n : n \geq 0\}$

Überprüfung, ob $aabb \in L(\Gamma)$:

$$
\begin{array}{llll}
Q_0 = \{ & (\overline{S} \to \bullet S;\ 0) & (0,1) & \text{Initialisierung} \\
 & (S \to \bullet aSb;\ 0) & (0,2) & \text{(V) aus (0,1)} \\
 & (S \to \bullet\ ;\ 0) & (0,3) & \text{(V) aus (0,1)} \\
 & (\overline{S} \to S\bullet\ ;\ 0)\ \} & (0,4) & \text{(R) aus (0,3) und (0,1)} \\
 & & & \\
Q_1 = \{ & (S \to a \bullet Sb;\ 0) & (1,1) & \text{(E) aus (0,2)}\ \ (a \text{ gelesen}) \\
 & (S \to \bullet aSb;\ 1) & (1,2) & \text{(V) aus (1,1)} \\
 & (S \to \bullet\ ;\ 1) & (1,3) & \text{(V) aus (1,1)} \\
 & (S \to aS \bullet b;\ 0)\ \} & (1,4) & \text{(R) aus (1,3) und (1,1)} \\
 & & & \\
Q_2 = \{ & (S \to a \bullet Sb;\ 1) & (2,1) & \text{(E) aus (1,2)}\ \ (a \text{ gelesen}) \\
 & (S \to \bullet aSb;\ 2) & (2,2) & \text{(V) aus (2,1)} \\
 & (S \to \bullet\ ;\ 2) & (2,3) & \text{(V) aus (2,1)} \\
 & (S \to aS \bullet b;\ 1)\ \} & (2,4) & \text{(R) aus (2,3) und (2,1)} \\
 & & & \\
Q_3 = \{ & (S \to aSb\bullet;\ 1) & (3,1) & \text{(E) aus (2,4)}\ \ (b \text{ gelesen}) \\
 & (S \to aS \bullet b;\ 0)\ \} & (3,2) & \text{(R) aus (3,1) und (1,1)} \\
 & & & \\
Q_4 = \{ & (S \to aSb\bullet;\ 0) & (4,1) & \text{(E) aus (3,2)}\ \ (b \text{ gelesen}) \\
 & \underbrace{(\overline{S} \to S\bullet;\ 0)}_{\text{Akzeptanz}}\ \} & (4,2) & \text{(R) aus (4,1) und (0,1)}
\end{array}
$$

Beispiel 2 (aus Loos 1989, angepaßt an die hiesige Terminologie)

$$
\begin{array}{ll}
\Gamma: & \overline{S} \to S \\
 & S \to A \\
 & S \to aSb \\
 & A \to a \\
 & A \to Aa
\end{array}
$$

$L(\Gamma) = \{a^n b^m : n > m > 0\}$
Überprüfung, ob $aab \in L(\Gamma)$:

$$
\begin{array}{llll}
Q_0 = \{ & (\overline{S} \to \bullet S;\ 0) & (0,1) & \text{Initialisierung} \\
 & (S \to \bullet A;\ 0) & (0,2) & \text{(V) aus (0,1)} \\
 & (S \to \bullet aSb; 0) & (0,3) & \text{(V) aus (0,1)} \\
 & (A \to \bullet a;\ 0) & (0,4) & \text{(V) aus (0,2)} \\
 & (A \to \bullet Aa;\ 0)\ \} & (0,5) & \text{(V) aus (0,2)}
\end{array}
$$

$Q_1 = \{$	$(S \to a \bullet Sb;\ 0)$	$(1,1)$	(E) aus (0,3) (a gelesen)
	$(A \to a\bullet;\ 0)$	$(1,2)$	(E) aus (0,4)
	$(S \to \bullet A;\ 1)$	$(1,3)$	(V) aus (1,1)
	$(S \to \bullet aSb;\ 1)$	$(1,4)$	(V) aus (1,1)
	$(A \to \bullet a;\ 1)$	$(1,5)$	(V) aus (1,3)
	$(A \to \bullet Aa;\ 1)$	$(1,6)$	(V) aus (1,3)
	$(S \to A\bullet;\ 0)$	$(1,7)$	(R) aus (1,2) und (0,2)
	$(A \to A \bullet a;\ 0)$	$(1,8)$	(R) aus (1,2) und (0,5)
	$(\overline{S} \to S\bullet;\ 0)\ \}$	$(1,9)$	(R) aus (1,7) und (0,1)

$Q_2 = \{$	$(S \to a \bullet Sb;\ 1)$	$(2,1)$	(E) aus (1,4) (a gelesen)
	$(A \to a\bullet;\ 1)$	$(2,2)$	(E) aus (1,5)
	$(A \to Aa\bullet;\ 0)$	$(2,3)$	(E) aus (1,8)
	$(S \to \bullet A;\ 2)$	$(2,4)$	(V) aus (2,1)
	$(S \to \bullet aSb;\ 2)$	$(2,5)$	(V) aus (2,1)
	$(A \to \bullet a;\ 2)$	$(2,6)$	(V) aus (2,4)
	$(A \to \bullet Aa;\ 2)$	$(2,7)$	(V) aus (2,4)
	$(S \to A\bullet;\ 1)$	$(2,8)$	(R) aus (2,2) und (1,3)
	$(A \to A \bullet a;\ 1)$	$(2,9)$	(R) aus (2,2) und (1,6)
	$(S \to A\bullet;\ 0)$	$(2,10)$	(R) aus (2,3) und (0,2)
	$(A \to A \bullet a;\ 0)$	$(2,11)$	(R) aus (2,3) und (0,5)
	$(S \to aS \bullet b;\ 0)$	$(2,12)$	(R) aus (2,8) und (1,1)
	$(\overline{S} \to S\bullet;\ 0)\ \}$	$(2,13)$	(R) aus (2,10) und (0,1)

$Q_3 = \{$	$(S \to aSb\bullet;\ 0)$	$(3,1)$	(E) aus (2,12) (b gelesen)
	$(\overline{S} \to S\bullet;\ 0)\ \}$	$(3,2)$	(R) aus (3,1) und (0,1)

Beispiel 3:

(Grammatik hinter Theorem 5.12)

$$\begin{aligned} \Gamma:\ & \overline{S} \to S \\ & S \to aAbB \\ & A \to aA|a \\ & B \to bB|b \end{aligned}$$

$L(\Gamma) = \{a^n b^m : n, m \geq 2\}$
Überprüfung, ob $aaabb \in L(\Gamma)$:

$Q_0 = \{$	$(\overline{S} \to \bullet S;\ 0)$	$(0,1)$	Initialisierung
	$(S \to \bullet aAbB;\ 0)\ \}$	$(0,2)$	(V) aus (0,1)

$Q_1 = \{$	$(S \to a \bullet AbB;\ 0)$	$(1,1)$	(E) aus (0,2) (a gelesen)
	$(A \to \bullet aA;\ 1)$	$(1,2)$	(V) aus (1,1)
	$(A \to \bullet a;\ 1)\ \}$	$(1,3)$	(V) aus (1,1)

$Q_2 = \{$	$(A \to a \bullet A;\ 1)$	$(2,1)$	(E) aus (1,2) (a gelesen)
	$(A \to a\bullet;\ 1)$	$(2,2)$	(E) aus (1,3)
	$(A \to \bullet aA;\ 2)$	$(2,3)$	(V) aus (2,1)
	$(A \to \bullet a;\ 2)$	$(2,4)$	(V) aus (2,1)
	$(S \to aA \bullet bB;\ 0)\ \}$	$(2,5)$	(R) aus (2,2)

$Q_3 = \{$	$(A \to a \bullet A;\ 2)$	$(3,1)$	(E) aus (2,3) (a gelesen)
	$(A \to a\bullet;\ 2)$	$(3,2)$	(E) aus (2,4)
	$(A \to \bullet aA;\ 3)$	$(3,3)$	(V) aus (3,1)
	$(A \to \bullet a;\ 3)$	$(3,4)$	(V) aus (3,1)
	$(A \to aA\bullet;\ 1)$	$(3,5)$	(R) aus (3,2) und (2,1)
	$(S \to aA \bullet bB;\ 0)\ \}$	$(3,6)$	(R) aus (3,5) und (1,1)

$Q_4 = \{$	$(S \to aAb \bullet B;\ 0)$	$(4,1)$	(E) aus (3,6) (b gelesen)
	$(B \to \bullet bB;\ 4)$	$(4,2)$	(V) aus (4,1)
	$(B \to \bullet b;\ 4)\ \}$	$(4,3)$	(V) aus (4,1)

$Q_5 = \{$	$(B \to b \bullet B;\ 4)$	$(5,1)$	(E) aus (4,2) (b gelesen)
	$(B \to b\bullet;\ 4)$	$(5,2)$	(E) aus (4,2)
	$(B \to \bullet bB;\ 5)$	$(5,3)$	(V) aus (5,1)
	$(B \to \bullet b;\ 5)$	$(5,4)$	(V) aus (5,1)
	$(S \to aAbB\bullet;\ 0)$	$(5,5)$	(R) aus (5,2) und (4,1)
	$(\overline{S} \to S\bullet;\ 0)\ \}$	$(5,6)$	(R) aus (5,5) und (0,1)

Die folgenden zwei Lemmas präzisieren die in der Bemerkung 2 hinter Definition 8.4 gegebene intuitive Idee.

Lemma 8.5

Seien $a_1, \ldots, a_n \in \mathcal{T}$ als Eingabezeichen gegeben.
Seien im folgenden $u, v, w, u', v', w' \in (\mathcal{V} \cup \mathcal{T})^*$
Falls $(X \to u \bullet v; f) \in Q_i$, gibt es w mit

(i) $\overline{S} \overset{*}{\Rightarrow} a_1 \ldots a_f X w$

(ii) $u \overset{*}{\Rightarrow} a_{f+1} \ldots a_i$

Beweis: Induktion über Schrittzahl des Algorithmus

I.A. : $(\overline{S} \to \bullet S;\ 0) \in Q_0$.
Es gilt:

(i) $\overline{S} \overset{*}{\Rightarrow} \overline{S}$ $(w := \varepsilon)$

(ii) $\varepsilon \overset{*}{\Rightarrow} \varepsilon$

I.S. : (V) Sei $(X \to u \bullet Av;\ f) \in Q_i$.
Durch Voraussage werde $(A \to \bullet v';\ i)$ zu Q_i hinzugefügt.
Nach I.V. wissen wir

(i)' $\underbrace{\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_f Xw}_{\text{daraus}}$ (ii)' $u \overset{*}{\Rightarrow} a_{f+1} \dots a_i$

$\overline{S} \overset{*}{\Rightarrow} \underbrace{a_1 \dots a_f uAvw}_{\text{daraus mit (ii)'}}$ (mit X-Produktion)

(i) $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_f a_{f+1} \dots a_i A \underbrace{vw}_{\text{gesuchtes Wort}}$

Ferner gilt:

(ii) $\varepsilon \overset{*}{\Rightarrow} \varepsilon$

(E) Sei $(X \to u \bullet av;\ f) \in Q_{i-1}$
$(X \to ua \bullet v;\ f)$ werde zu Q_i hinzugefügt, wobei $a = a_i$
Nach I.V.:

(i)' $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_f Xw$ für ein w
(ii)' $u \overset{*}{\Rightarrow} a_{f+1} \dots a_{i-1}$
Daraus:
(i) $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_f Xw$
(ii) $ua \overset{*}{\Rightarrow} a_{f+1} \dots a_{i-1} a_i$

(R) Sei $(A \to u' \bullet;\ f) \in Q_i$ und $(X \to u \bullet Av;\ g) \in Q_f$
Durch Rekonstruktion werde $(X \to uA \bullet v;\ g)$ zu Q_i hinzugefügt
Nach I.V gilt:

(i)' $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_f Aw$ für ein w
(ii)' $u' \overset{*}{\Rightarrow} a_{f+1} \dots a_i$
sowie
(i)'' $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_g Xw'$ für ein w'
(ii)'' $u \overset{*}{\Rightarrow} a_{g+1} \dots a_f$
Daraus:

(i) $\overline{S} \overset{*}{\Rightarrow} a_1 \dots a_g Xw'$
(ii) $uA \overset{*}{\Longrightarrow} a_{g+1} \dots a_f A \overset{*}{\Longrightarrow} a_{g+1} \dots a_f u' \overset{*}{\Rightarrow} a_{g+1} \dots a_f a_{f+1} \dots a_i$.

Lemma 8.6

Falls $(X \to u \bullet Aw;\ f) \in Q_i$ und $A \overset{*}{\Rightarrow} a_{i+1} \dots a_l$, dann gilt:
$(X \to uA \bullet w;\ f) \in Q_l$.

Beweis:

Hilfsbehauptung

Falls $(X \to u \bullet a_{i+1} \dots a_l w;\ f) \in Q_i$ dann $(X \to u a_{i+1} \dots a_l \bullet w;\ f) \in Q_l$.

Beweis der Hilfsbehauptung:
$l = i$, d.h. $a_{i+1} \dots a_l = \varepsilon$: trivial. Ansonsten $(l-i)$-facher Eingabevergleich.

Beweis des Lemmas: Durch Induktion über der Länge der Ableitung von $a_{i+1} \dots a_l$ aus A.
$A \overset{0}{\Rightarrow} A$ tritt nicht auf.
Sei $A \overset{1}{\Rightarrow} x \overset{m}{\Rightarrow} a_{i+1} \dots a_l$
Durch Voraussage erhalten wir $(A \to \bullet x;\ i) \in Q_i$. Es gilt:
$x = x_1 X_1 \dots x_s X_s x_{s+1}$ mit $x_i \in \mathcal{T}^*$, wobei x_i oder X_i leer sein kann.
(D.h., Variablen können unmittelbar aufeinander folgen oder brauchen gar nicht vorzukommen.)
Nach I.V. (bzgl. X_i) und mit der Hilfsbehauptung (bzgl x_i) ergibt sich sukzessive:
$(A \to x\bullet;\ i) \in Q_l$.
(D.h., mit der I.V. schaffen wir Variablen nach links und mit der Hilfsbehauptung Terminale.)
Durch Rekonstruktion ergibt sich mit $(X \to u \bullet Av;\ f) \in Q_i :\ (X \to uA \bullet w;\ f) \in Q_l$

Theorem 8.7 (Korrektheit und Vollständigkeit des Earley-Algorithmus)

Ein Wort $w \in \mathcal{T}^*$ wird vom Earley-Algorithmus akzeptiert g.d.w. $w \in L(\Gamma)$

Beweis: „$\Rightarrow$“: Sei $w = a_1 \dots a_n$
„w wird akzeptiert“ bedeutet $(\overline{S} \to S\bullet;\ 0) \in Q_n$ Daraus mit Lemma 8.5:
$S \overset{*}{\Rightarrow} a_1 \dots a_n$, da $f = 0$ und $i = n$.

„$\Leftarrow$“: Sei $w = a_1 \dots a_n \in L(\Gamma)$, d.h. $S \overset{*}{\Rightarrow} a_1 \dots a_n$.
Da gilt $(\overline{S} \to \bullet S;\ 0) \in Q_0$, haben wir mit Lemma 8.6 $(\overline{S} \to S\bullet;\ 0) \in Q_n$, d.h. der Earley-Algorithmus akzeptiert $a_1 \dots a_n$.

9 Die Chomsky-Hierarchie

Die Chomsky-Hierarchie ist eine fundamentale Klassifikation von Sprach- bzw. Grammatiktypen. Es gibt zahlreiche Verfeinerungen, auf die hier nicht eingegangen werden kann.

Definition 9.1 (Sprachtypen)

Sei $\Gamma =< \mathcal{V}, \mathcal{T}, \Pi, S >$ Grammatik. Falls alle Produktionen $u \to v$ aus Π die jeweils im folgenden angegebene Form haben, dann bezeichnet man die Grammatik Γ und die Sprache $L(\Gamma)$ wie angegeben:

Automaten-modell	Form von $u \to v$	Bezeichnung von Γ	Chomsky-Sprachtyp
TM	beliebig	nicht eingeschränkt	0
linear beschränkte TM	$\lvert u\rvert \leq \lvert v\rvert$	nichtverkürzend; monoton	1
	$u = xAy \quad v = xwy \; A \in \mathcal{V}$ $x, y, w \in (\mathcal{V} \cup \mathcal{T})^*; \; w \neq \varepsilon$	kontextsensitiv	
KA	$u \in \mathcal{V}$	kontextfrei	2
DEA	$u \in \mathcal{V}, \; v \in \mathcal{T}\mathcal{V}$ oder $v \in \mathcal{T}$ oder $v = \varepsilon$	regulär	3

Bemerkungen:

1. Die Terminologie ist nicht einheitlich. Oft verbietet man ε-Produktionen bei Typ 2 und Typ 3.

   Dieser Punkt ist nicht sehr fundamental, da, wie wir in Lemma 4.4 und Lemma 5.8 gesehen haben, ε-Produktionen nur für die Erzeugung von ε selbst benötigt werden.

2. Oft bezeichnet man nichtverkürzende Grammatiken als kontextsensitiv. Gelegentlich erwartet man, daß in u mindestens eine Variable vorkommt.

3. Semi-Thue-Systeme sind Paare $< \Sigma, \Pi >$ mit beliebigen Produkionen in Π. Man unterscheidet also nicht Variablen und Terminale und zeichnet auch kein Startsymbol aus.

Lemma 9.2

Falls immer $|u| \geq 1$, sind die durch nichtverkürzende und kontextsensitive Grammatiken erzeugten Sprachklassen identisch.

Beweis:

Kontextsensitiv ⇒ nichtverkürzend: ✓

Nichtverkürzend ⇒ kontextsensitiv:
Definiere zu jedem $a \in \mathcal{T}$ eine neue Variable B_a, ersetze in allen Produktionen a durch B_a und füge die Produktion $B_a \to a$ hinzu. Offenbar ist die neue Grammatik mit der alten geleichwertig.
Dann haben alle Produktionen außer $B_a \to a$ die Form:
$X_1 \dots X_m \to Y_1 \dots Y_n$ mit $n \geq m \geq 1$.
$m = 1$: Ist schon kontextsensitiv.
$m \geq 2$: Ersetze obige Produktion durch folgende Kette von kontextsensitiven Produktionen mit zusätzlichen Variablen $Z_1, \dots, Z_m$:

$$
\begin{array}{l}
X_1 X_2 \dots X_m \to Z_1 X_2 \dots X_m \\
Z_1 X_2 X_3 \dots X_m \to Z_1 Z_2 X_3 \dots X_m \\
\qquad\qquad \vdots \\
Z_1 \dots Z_{m-1} X_m \to Z_1 Z_2 \dots Z_{m-1} Z_m Y_{m+1} \dots Y_n \\
Z_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n \to Y_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n \\
Y_1 Z_2 Z_3 \dots Z_m Y_{m+1} \dots Y_n \to Y_1 Y_2 Z_3 \dots Z_m Y_{m+1} \dots Y_n \\
\qquad\qquad\qquad\qquad\qquad \vdots \\
Y_1 \dots Y_{m-1} Z_m Y_{m+1} \dots Y_n \to Y_1 \dots Y_{m-1} Y_m Y_{m+1} \dots Y_n
\end{array}
$$

Das führe man für jede Produktion durch.

Lemma 9.3

Es gibt eine Typ1-Sprache, die nicht Typ 2 ist.

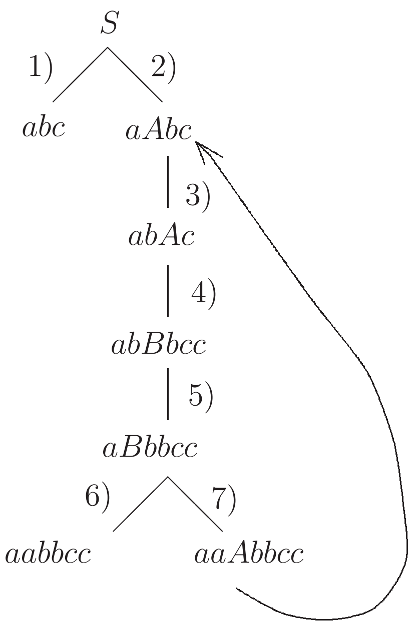
Beweis: Sei $L = \{a^n b^n c^n : n \geq 1\}$.

L ist nicht kontextfrei (Beispiel hinter Theorem 7.1).
L wird erzeugt von der nichtverkürzenden Grammatik[1]:

1)	$S \to abc$	5)	$bB \to Bb$
2)	$S \to aAbc$	6)	$aB \to aa$
3)	$Ab \to bA$	7)	$aB \to aaA$
4)	$Ac \to Bbcc$		

Folgendes Diagramm zeigt die Möglichkeiten, Wörter über $\{a, b, c\}$ in dieser Grammatik abzuleiten.

[1]Vgl. Salooma (1973) Ex.2.1, S.11f., wo sich auch noch andere Grammatiken für L finden.



Theorem 9.4 (Hierarchie-Satz)

Falls $|u| \geq 1$ und die Sprachen kein ε enthalten, dann gilt für die definierten Sprachklassen:
Typ 3-Sprachen$\subsetneq$Typ 2-Sprachen$\subsetneq$Typ 1-Sprachen$\subsetneq$Typ 0-Sprachen

Beweis:

Typ 3$\subsetneq$Typ 2	$\boxed{\subseteq}$	regulär ist Speziallfall von kf	$\boxed{\neq}$	$\{a^n b^n : n \geq 1\}$
Typ 2$\subsetneq$Typ 1	$\boxed{\subseteq}$	kf ist Speziallfall von ks	$\boxed{\neq}$	$\{a^n b^n c^n : n \geq 1\}$
Typ 1$\subsetneq$Typ 0	$\boxed{\subseteq}$	ks ist Speziallfall von nicht-eingeschränkt	$\boxed{\neq}$	Universelles Halteproblem Vgl. Bemerkung zu Satz 15.7

Ohne Beweise werden Tabellen mit Abschluß- und Entscheidbarkeitseigenschaften für verschiedene Sprachklassen angeführt (einschließlich der deterministisch kontextfreien Sprachen) (vgl. Schöning 2001, §1.5).

Für Abgeschlossenheit gilt folgendes:

	$L_1 \cap L_2$	$L_1 \cup L_2$	$\Sigma^* \backslash L$	$L_1 \circ L_2$	L^*
Typ 3	+	+	+	+	+
Det.kf.	-	-	+	-	-
Typ 2	-	+	-	+	+
Typ 1	+	+	+	+	+
Typ 0	+	+	-	+	+

Für Entscheidbarkeit gilt folgendes:

	$w \in L(\Gamma)$	$L(\Gamma) = \emptyset$	$L(\Gamma_1) = L(\Gamma_2)$	$L(\Gamma_1) \cap (\Gamma_2) = \emptyset$
Typ 3	+	+	+	+
Det.kf.	+	+	+	-
Typ 2	+	+	-	-
Typ 1	+	-	-	-
Typ 0	-	-	-	-

Das Problem, algorithmisch zu entscheiden, ob $w \in L(\Gamma)$, nennt man auch das *Wortproblem* für die Sprachen des betrachteten Typs.

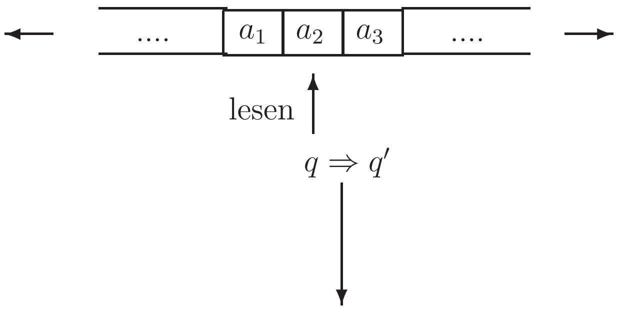
Teil II: Berechenbarkeit

10 Turing-Maschinen

Turing-Maschinen charakterisieren nicht nur eine bestimmte Sprachklasse, sondern modellieren den Berechenbarkeitsbegriff überhaupt. Andere Modelle sind z.B. Registermaschinen, WHILE-Programme, partiell rekursive und λ-definierbare Funktionen. Wir werden außer Turing-Maschinen noch partiell rekursive Funktionen und WHILE-Programme behandeln.

Bei der Definition von Turing-Maschinen folge ich der Darstellung von Boolos & Jeffrey (1989). Diese Definition liegt auch der Macintosh-Software „Turing's World“ von Barwise & Etchemendy (1996) zugrunde, die zum Einüben der Konstruktion von Turing-Maschinen hervorragend geeignet ist. (Auch deren Dokumentation ist sehr empfehlenswert.) Die Definition von Maschinenschemata folgt Lewis & Papadimitriou (1981), die an Hermes (1978) anschließen.

Idee einer Turing-Maschine:



dabei passiert:	1)	Ersetzen des gelesenen Zeichens
oder	2)	Einen Schritt nach rechts gehen
oder	3)	Einen Schritt nach links gehen

Bemerkung: Bei DEAs nur 2. erlaubt!

Definition 10.1 (Turing-Maschine)

Sei Σ ein Alphabet, $\#, L, R \notin \Sigma$. $\#$ heiße „blank“. $\Sigma_\# := \Sigma \cup \{\#\}$.
Eine Turing-Maschine (TM) $\mathcal{M}$ über Σ ist ein Quadrupel $\mathcal{M} = \langle Q, \Sigma, \delta, s\rangle$, wobei
Q nichtleere Zustandmenge, $s \in Q$ ausgezeichneter Anfangszustand,
$\delta : (Q \times \Sigma_\#) \to ((\Sigma_\# \cup \{L, R\}) \times Q)$ partielle Funktion („Übergangsfunktion“).

Bemerkung (Maschinentafel):

Falls $Q = \{q_1 \ldots q_m\}$ und $\Sigma_\# = \{a_1, \ldots a_n\}$, läßt sich $\mathcal{M}$ durch eine Maschinentafel von $m \cdot n$ Quadrupeln darstellen:

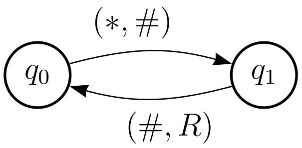
$$\begin{array}{cccc}
q_1 & a_1 & b_{11} & q_{11} \\
\vdots & \vdots & \vdots & \vdots \\
q_1 & a_n & b_{1n} & q_{1n} \\
\vdots & \vdots & \vdots & \vdots \\
q_m & a_1 & b_{m1} & q_{m1} \\
\vdots & \vdots & \vdots & \vdots \\
q_m & a_n & b_{mn} & q_{mn}
\end{array}$$

wobei $\delta(q_i, a_j) = (b_{ij}, q_{ij})$ und wobei die letzten beiden Einträge fehlen, falls $\delta(q_i, a_j)$ nicht definiert ist. (In diesem Fall kann die ganze Zeile weggelassen werden.)

Wenn nichts anderes gesagt ist, soll der in einer Maschinentafel als erster angeführte Zustand der Startzustand sein.
Die andere Möglichkeit der Darstellung ist die als kantenbewerteter, gerichteter Graph. Hier sehen Kanten so aus:

$$q_i \xrightarrow{(a_j, b_{ij})} q_{ij} \quad \text{falls } \delta(q_i, a_j) = (b_{ij}, q_{ij})$$

Beispiel 1: $\Sigma = \{*\};\ \Sigma_{\#} = \{\#, *\};\ Q = \{q_0, q_1\};\ s = q_0$

$\mathcal{M}$: 

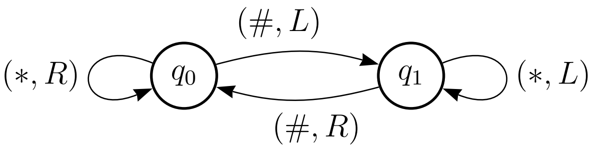
q_0	$*$	$\#$	q_1
q_0	$\#$		
q_1	$*$		
q_1	$\#$	R	q_0

q_0	$*$	$\#$	q_1
q_1	$\#$	R	q_0

(abgekürzt)

Löscht aufeinanderfolgende Sterne und hält dann.

Beispiel 2:



q_0	$*$	R	q_0
q_0	$\#$	L	q_1
q_1	$*$	L	q_1
q_1	$\#$	R	q_0

Pendelt zwischen den Endpunkten einer Folge von Sternen hin und her.

Definition 10.2 (Konfiguration von $\mathcal{M}$)

Eine Konfiguration von $\mathcal{M}$ ist ein Quadrupel (q, u, a, v) mit $q \in Q$,
$a \in \Sigma_{\#},\ u \in \Sigma_{\#}^{*} \backslash (\{\#\} \circ \Sigma_{\#}^{*}),\ v \in \Sigma_{\#}^{*} \backslash (\Sigma_{\#}^{*} \circ \{\#\})$.

Bemerkung (Band):

(q, u, a, v) repräsentiert das Band $\underbrace{\ldots}_{\#} uav \underbrace{\ldots}_{\#}$, von dem a im Zustand q gelesen wird.

Statt (q, u, a, v) schreiben wir auch: $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix}$ oder linear: $(q,\ u\underline{a}v)$.

Wichtig: u hat keine führenden blanks, v hat keine abschließenden blanks. Wir notieren also den minimalen Abschnitt des Bandes, der außer dem gelesenen Zeichen alle Nicht-blanks umfaßt.

Definition 10.3 (Übergänge zwischen Konfigurationen)

Die Relation $\overset{1}{\underset{\mathcal{M}}{\vdash}}$ (kurz $\overset{1}{\vdash}$) zwischen Konfigurationen ist wie folgt definiert:

$\begin{pmatrix} u & a & v \\ & q & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} u & b & v \\ & q' & \end{pmatrix}$ falls $\delta(q, a) = (b, q')$

$\begin{pmatrix} u & a & b & v \\ & q & & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} u & a & b & v \\ & & q' & \end{pmatrix}$ falls $\delta(q, a) = (R, q')$ und $ua \neq \#$ (d.h. nicht: $a = \#$ und $u = \varepsilon$)

$\begin{pmatrix} \# & b & v \\ q & & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} b & v \\ q' & \end{pmatrix}$ falls $\delta(q, \#) = (R, q')$

$\begin{pmatrix} u & a & \\ & q & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} u & a & \# \\ & & q' \end{pmatrix}$ falls $\delta(q, a) = (R, q')$ und $ua \neq \#$

$\begin{pmatrix} u & b & a & v \\ & & q & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} u & b & a & v \\ & q' & & \end{pmatrix}$ falls $\delta(q, a) = (L, q')$ und $av \neq \#$ (d.h. nicht: $a = \#$ und $v = \varepsilon$)

$\begin{pmatrix} u & b & \# \\ & & q \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} u & b \\ & q' \end{pmatrix}$ falls $\delta(q, \#) = (L, q')$

$\begin{pmatrix} & a & v \\ & q & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} \# & a & v \\ q' & & \end{pmatrix}$ falls $\delta(q, a) = (L, q')$ und $av \neq \#$

$\begin{pmatrix} \# \\ q \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} \# \\ q' \end{pmatrix}$ falls $\delta(q, \#) = (R, q')$ oder $\delta(q, \#) = (L, q')$

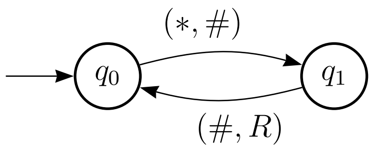
Jedes $\begin{pmatrix} u & a & v \\ & s & \end{pmatrix}$ heißt Anfangskonfiguration. $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix}$ heißt Haltekonfiguration, falls es keinen Nachfolger bzgl. $\overset{1}{\vdash}$ hat, d.h. falls $\delta(q, a)$ nicht definiert ist. Wir schreiben dann auch $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix} \in H_{\mathcal{M}}$ oder $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix} \in H$ wenn klar ist, um welche Maschine es sich handelt.

Für Konfigurationen $k_0, \ldots, k_n$ ist definiert:

$$k_0 \overset{n}{\vdash} k_n \Leftrightarrow \begin{cases} k_0 = k_n & \text{falls } n = 0 \\ k_0 \overset{1}{\vdash} k_1 \overset{1}{\vdash} k_2 \overset{1}{\vdash} \ldots \overset{1}{\vdash} k_n & \text{falls } n > 0 \end{cases}$$

$$k \overset{*}{\vdash} k' \Leftrightarrow (k \overset{n}{\vdash} k' \text{ für irgendein } n)$$

Beispiel: Betrachte die TM



Abfolge von Konfigurationen in dieser Maschine:

$$\begin{pmatrix} * & * & * & * \\ & q_0 & & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} * & \# & * & * \\ & q_1 & & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} * & \# & * & * \\ & & q_0 & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} * & \# & \# & * \\ & & q_1 & \end{pmatrix} \overset{1}{\vdash}$$

$$\begin{pmatrix} * & \# & \# & * \\ & & & q_0 \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} * & \# & \# & \# \\ & & & q_1 \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} * & \# & \# & \# & \# \\ & & & & q_0 \end{pmatrix} \in H$$

Definition 10.4 (Transformation)

Ist $\mathcal{M}$ eine TM, dann soll $(u\underline{a}v) \underset{\mathcal{M}}{\Rightarrow} (u'\underline{a'}v')$ bedeuten:

$$\begin{pmatrix} u & a & v \\ & s & \end{pmatrix} \overset{*}{\underset{\mathcal{M}}{\vdash}} \begin{pmatrix} u' & a' & v' \\ & q & \end{pmatrix} \in H$$

Wir sagen auch: „$\mathcal{M}$ transformiert $(u\underline{a}v)$ in $(u'\underline{a'}v')$“

Definition 10.5 (Akzeptanz, Berechnung, Entscheidung)

$\mathcal{M}$ *akzeptiert* $w \in \Sigma^*$, falls $\begin{pmatrix} \# & w \\ s & \end{pmatrix} \overset{*}{\underset{\mathcal{M}}{\vdash}} k$ gilt für eine Haltekonfiguration k.
$L(\mathcal{M}) := \{w |\ \mathcal{M} \text{ akzeptiert } w\}$.
$\mathcal{M}$ *terminiert* für Eingabe $(w_1, \ldots, w_n) \in (\Sigma^*)^n$, falls es eine Haltekonfiguration k gibt mit $\begin{pmatrix} \# & w_1 & \# & w_2 & \# & \ldots & \# & w_n \\ s & & & & & & & \end{pmatrix} \overset{*}{\underset{\mathcal{M}}{\vdash}} k$.
Sei $f : \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{\text{n-mal}} \rightarrow \Sigma^*$ eine n-stellige, partielle Funktion auf Σ^*. $\mathcal{M}$ *berechnet* f, falls für alle Wörter $w_1, \ldots, w_n \in \Sigma^*$ gilt: Falls $f(w_1, \ldots, w_n) = w$, dann gilt

$$\begin{pmatrix} \# & w_1 & \# & \ldots & \# & w_n \\ s & & & & & \end{pmatrix} \overset{*}{\underset{\mathcal{M}}{\vdash}} \begin{pmatrix} \# & w \\ q & \end{pmatrix} \text{ für Haltekonfiguration } \begin{pmatrix} \# & w \\ q & \end{pmatrix}.$$

Falls $f(w_1, \ldots, w_n)$ nicht definiert ist, dann terminiert $\mathcal{M}$ für das Tupel $(w_1, \ldots w_n)$ nicht. $\mathcal{M}$ darf dabei zusätzliche Hilfszeichen benutzen, ist also eine TM über einem Alphabet $\Sigma' \supseteq \Sigma$.
Sei $L \subseteq \Sigma^*$. Sei $0, 1 \notin \Sigma$. $\mathcal{M}$ *entscheidet* L, falls $\mathcal{M}$ die charakteristische Funktion von L:

$$\chi_L : \Sigma^* \to \{0,1\} : \begin{array}{l} w \mapsto 1 \text{ falls } w \in L \\ w \mapsto 0 \text{ falls } w \notin L \end{array}$$

berechnet. D.h.

$$\begin{pmatrix} \# & w \\ q & \end{pmatrix} \overset{*}{\vdash} \begin{pmatrix} \# & 1 \\ q & \end{pmatrix} \text{ falls } w \in L$$

$$\begin{pmatrix} \# & w \\ q & \end{pmatrix} \overset{*}{\vdash} \begin{pmatrix} \# & 0 \\ q & \end{pmatrix} \text{ falls } w \notin L$$

Bemerkungen:

1. Falls $\mathcal{M}$ die partielle Funktion f berechnet, dann soll $\mathcal{M}$ grundsätzlich nicht halten, falls f nicht definiert ist, auch nicht in einer Konfiguration, die *nicht* die Form $\begin{pmatrix} \# & w \\ q & \end{pmatrix}$ hat.
2. Die zu χ_L gehörende TM $\mathcal{M}$ ist eine TM über dem Alphabet $\Sigma' \cup \{0, 1\}$, wobei $\Sigma' \supseteq \Sigma$. χ_L ist hier eine überall definierte, d.h. totale Funktion.

Beispiel: Arithmetische Funktionen
Zahlen werden im folgenden repräsentiert durch Folgen von Sternen.
Dabei steht $\underbrace{* \ldots *}_{n+1}$ für n $(n \geq 0)$.
$f(n) := n + 1$ wird z.B. berechnet durch folgende TM:

$\mathcal{M} := \langle \{q_1, q_2, q_3\}, \{*\}, \delta, q_1 \rangle$

q_1	$\#$	R	q_2
q_2	$*$	R	q_2
q_2	$\#$	$*$	q_3
q_3	$*$	L	q_3

Es gilt:

$$\begin{pmatrix} \# & \underbrace{* \ldots *} \\ q_1 & n+1 \end{pmatrix} \underset{\mathcal{M}}{\overset{*}{\vdash}} \begin{pmatrix} \# & \underbrace{* \ldots *} \\ q_3 & n+2 \end{pmatrix} \in H$$

Definition 10.6 (Maschinenschemata)

Gegeben sei eine endliche Menge $\mathfrak{M}$ von TM über einem festen Alphabet Σ, wobei eine TM $\mathcal{M}_0 \in \mathfrak{M}$ als Startmaschine ausgezeichnet sei. Ein Maschinenschema über $\mathfrak{M}$ ist ein Tripel $\langle \mathfrak{M}, \eta, \mathcal{M}_0 \rangle$, wobei $\eta : \mathfrak{M} \times \Sigma_\# \to \mathfrak{M}$ partielle Funktion.

Beispiel:

1. $\Sigma = \{a, b\}$ Betrachte die Turing-Maschinen
   R: Gehe ein Feld nach rechts

   $$R := \;\longrightarrow q_0 \xrightarrow{(?,R)} q_1$$

   L: Gehe ein Feld nach links

   $$L := \;\longrightarrow q_0 \xrightarrow{(?,L)} q_1$$

   Das Maschinenschema $\langle\{R, L\}, \eta, R\rangle$ mit $\eta(R, a) = R$; $\eta(R, b) = L$; $\eta(R, \#) = R$, als Diagramm

   $$\begin{tikzcd} > R \arrow[loop above, "{a,\#}"] \arrow[r, "b"] & L \end{tikzcd}$$

   repräsentiert die TM: „Gehe nach rechts bis vor das erste Feld, auf dem b steht“.

2. Das Maschinenschema mit dem Diagramm $> R \xrightarrow{\Sigma_\#} R$, kurz: $> RR$ oder RR repräsentiert die TM: „Gehe zwei Felder nach rechts“.

Bemerkung:

Bei der Darstellung von Maschinenschemata bezeichnen *verschiedene Vorkommen* derselben Bezeichnung *verschiedene* Kopien derselben Maschine mit jeweils *verschiedenen* Zustandsmengen. Im Schema „RR“ bezeichnet das linke und das rechte „R“ jeweils eine Kopie der „Rechtsmaschine“, wobei jede Kopie *separate* Zustände hat. (Ansonsten könnten wir „RR“ nicht vom Maschinenschema $\begin{tikzcd} >R \arrow[loop right, "{(?,R)}"] \end{tikzcd}$ unterscheiden, das eine nichtterminierende TM repräsentiert.)

Definition 10.7 (Repräsentierte TM)

Ein Maschinenschemata $\langle \mathfrak{M}, \eta, \mathcal{M}_0\rangle$ repräsentiert folgende TM $\mathcal{M}$:
Sei $\mathfrak{M} := \{\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n\}$. Wir nehmen an, die Zustandsmengen Q_i der $\mathcal{M}_i$ seien disjunkt voneindander (können wir immer erreichen). q^i_j sei der j-te Zustand von $\mathcal{M}_i$.
δ^i sei die Übergangsfunktion von $\mathcal{M}_i$. $\{q_0, \ldots, q_n\}$ sei eine Menge zusätzlicher, d.h. neuer Zustände.

$\mathcal{M} := \langle Q, \Sigma, \delta, s\rangle$ mit $Q := Q_0 \cup Q_1 \cup \ldots \cup Q_n \cup \{q_0, \ldots, q_n\}$
$s = s_{\mathcal{M}_0}$ (Startzustand von $\mathcal{M}_0$: Wir fangen mit erster Maschine an).
δ ist wie folgt definiert:

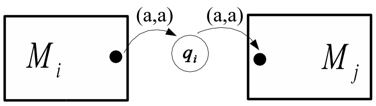
$$\delta(q_j, a) := \begin{cases} (a, s_{\eta(\mathcal{M}_j, a)}) & \text{falls } \eta(\mathcal{M}_j, a) \text{ definiert} \\ \text{undefiniert} & \text{sonst} \end{cases}$$

$$\delta(q_j^i, a) := \begin{cases} \delta^i(q_j^i, a) & \text{falls } \delta(q_j^i, a) \text{ definiert} \\ (a, q_i) & \text{sonst} \end{cases}$$

Bemerkungen:

1. Intuition hinter δ: Innerhalb einer Untermaschine $\mathcal{M}_i$ führt man δ^i aus. Falls man in $\mathcal{M}_i$ eine Haltekonfiguration erreicht, geht man in den Zustand q_i über, führt dann den durch η gegebenen Übergang des Maschinenschemas aus und fährt mit dem Anfangszustand der durch diesen Übergang bestimmten Untermaschine fort.

   $\mathcal{M}_i \xrightarrow{a} \mathcal{M}_j$ bedeutet:

   Offenbar ist es auch möglich, ohne die „Zwischenzustände“ q_i auszukommen.

2. Schreibweise:

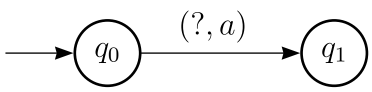
$\mathcal{M}_i \longrightarrow \mathcal{M}_j$	oder $\mathcal{M}_i\mathcal{M}_j$	steht für $\mathcal{M}_i \xrightarrow{\Sigma_\#} \mathcal{M}_j$
$\mathcal{M}_i \xrightarrow{\neq a} \mathcal{M}_j$		steht für $\mathcal{M}_i \xrightarrow{\Sigma_\# \setminus \{a\}} \mathcal{M}_j$
$\mathcal{M}_i \xrightarrow{\overline{\triangle}} \mathcal{M}_j$		steht für $\mathcal{M}_i \xrightarrow{\Sigma_\# \setminus \triangle} \mathcal{M}_j$ für $\triangle \subseteq \Sigma_\#$
$\mathcal{M}^n$		steht für $\underbrace{\mathcal{M} \ldots \mathcal{M}}_{\text{n-mal}}$

   Wenn nicht anders vermerkt, ist die in einem Maschinenschema am weitesten links notierte Maschine die Startmaschine.

Lemma 10.8

Sei $\Sigma = \{a_1, \ldots, a_n\}$.

Jede TM läßt sich als Maschinenschema über $\mathfrak{M} = \{R, L, a_1, \ldots, a_n, \#\}$ schreiben, wobei R und L die in Beispiel 1 hinter Definition 10.6 genannten TM und a folgende TM ist:

$a :=$  für $a \in \Sigma_\#$

Beweis: Ersetze jedes q_i durch die „tut nichts“-Maschine RL und jeden Übergang $\xrightarrow{(a,X)}$ durch $\xrightarrow{a} X \rightarrow$.

Beispiele:
Einige wichtige Maschinen und deren Schemata:

$R_\# : \quad >R \circlearrowright \overline{\#}$ (geht bis auf das erste blank nach rechts)

$L_\# : \quad >L \circlearrowright \overline{\#}$ (geht bis auf das erste blank nach links)

$R_{\overline{\#}} : \quad >R \circlearrowright \#$ (bleibt auf erstem Nicht-blank rechts stehen,)

$L_{\overline{\#}} : \quad >L \circlearrowright \#$ (bleibt auf erstem Nicht-blank links stehen,)

$C : (\underline{\#}u) \underset{C}{\Longrightarrow} (\underline{\#}u\#u)$ Kopierer $\quad$ (u ohne blank)

Schema:

$$\begin{tikzcd}
R_\# \arrow[r] & L \arrow[r, "\alpha \neq \#"] \arrow[d, "\#"] & \#L^2_\# \alpha R^2_\# \alpha \arrow[l, bend right] \\
& L_\# &
\end{tikzcd}$$

(Die Kopie wird links von u angelegt.)
Hier ist α eine Variable für Zeichen aus $\Sigma_\#$ und steht für das von $\#$ verschiedene Zeichen, das bei $\xrightarrow{\alpha\neq\#}$ gelesen wird. Diese Notation erspart Verzweigungen für alle Zeichen des Alphabets.

$S_R : (\underline{\#}u\#v) \underset{S_R}{\Longrightarrow} (\underline{\#}uv)$ Shift rechts

Schema:

$$\begin{tikzcd}
R_\# \arrow[r] & L \arrow[r, "\alpha \neq \#"] \arrow[d, "\#"] & R\alpha L \arrow[l, bend right] \\
& R &
\end{tikzcd}$$

Maschinenschemata zur Berechnung von zahlentheoretischen Funktionen:

$f(n) = 2n$: $CS_R R\#$
$f(m,n) = m + n$: $S_R R\#$
$f(n) = n + 1 : *L$

Definition 10.9 (RTM)

Eine TM mit nur rechtsseitig unendlichem Band (RTM) unterscheidet sich von einer TM wie folgt:

Da das Band linksseitig beschränkt ist, werden bei einer Konfiguration $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix}$ führende blanks notiert, d.h. $u \in \Sigma_{\#}^*$.
Entsprechend gilt:
$\begin{pmatrix} \# & a & v \\ q & & \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} \# & a & v \\ & q' & \end{pmatrix}$ und $\begin{pmatrix} \# \\ q \end{pmatrix} \overset{1}{\vdash} \begin{pmatrix} \# & \# \\ & q' \end{pmatrix}$, falls $\delta(q, \#) = (R, q')$,
d.h. führende blanks werden nicht eliminiert.
Eine Konfiguration $\begin{pmatrix} a & v \\ q & \end{pmatrix}$ hat keinen Nachfolger, falls $\delta(q, a) = (L, q')$ für ein q'.
Eine derartige Konfiguration heißt „Hängekonfiguration“. Eine Haltekonfiguration $\begin{pmatrix} u & a & v \\ & q & \end{pmatrix}$ ist dagegen eine Konfiguration, für die $\delta(q, a)$ nicht definiert ist.

Theorem 10.10 (Simulation von TM durch RTM)

Sei $\mathcal{M}_1 = \langle Q_1, \Sigma_1, \delta_1, s_1 \rangle$ TM. Dann gibt es RTM $\mathcal{M}_2 = \langle Q_2, \Sigma_2, \delta_2, s_2 \rangle$ mit $\Sigma_1 \subseteq \Sigma_2$, so daß für alle $w \in \Sigma_1^*$ gilt:

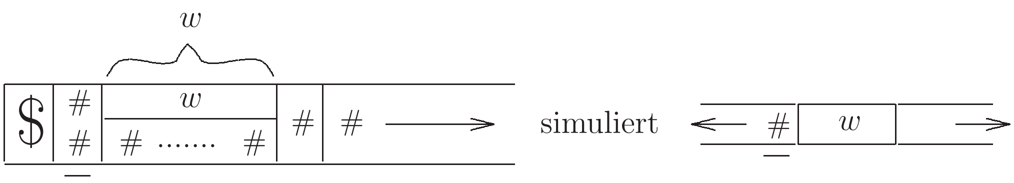
i) $\begin{pmatrix} \# & w \\ s_1 & \end{pmatrix} \underset{\mathcal{M}_1}{\overset{*}{\vdash}} \begin{pmatrix} u & a & v \\ & q & \end{pmatrix} \in H$

$\Rightarrow \begin{pmatrix} \# & w \\ s_2 & \end{pmatrix} \underset{\mathcal{M}_2}{\overset{*}{\vdash}} \begin{pmatrix} u & a & v \\ & q' & \end{pmatrix} \in H$

ii) $\mathcal{M}_1$ terminiert nicht für $w \in \Sigma_1^* \Rightarrow$ $\mathcal{M}_2$ terminiert nicht für $w \in \Sigma_1^*$

Beweis:

Idee: Das beidseitig unendliche Band (Abb. rechts) wird durch das nur rechtsseitig unendliche Band (Abb. links) simuliert.



$\mathcal{M}_2' := \langle Q_2', \{\$\} \cup \Sigma_1 \cup (\Sigma_{1_\#} \times \Sigma_{1_\#}), \delta_2', s_2' \rangle$ sei eine RTM, die $(\underline{\#}w)$ in $\begin{pmatrix} \$ & \begin{matrix} \# & w \\ \underline{\#} & \ldots \# \end{matrix} \end{pmatrix}$ transformiert. (Übung; $\mathcal{M}_2'$ erzeugt also das Band oben links)

$\mathcal{M} := \langle\{\langle q,1\rangle : q \in Q_1\} \cup \{\langle q,2\rangle : q \in Q_1\}, \Sigma_2, \delta, S\rangle$ mit
$\Sigma_2 := \{\$\} \cup \Sigma_{1_\#} \cup (\Sigma_{1_\#} \times \Sigma_{1_\#}) \cup \overline{\Sigma}_{1_\#} \cup (\Sigma_{1_\#} \times \overline{\Sigma}_{1_\#}) \cup (\overline{\Sigma}_{1_\#} \times \Sigma_{1_\#})$ mit
$\overline{\Sigma}_{1_\#} := \{\overline{a} : a \in \Sigma_1\} \cup \{\overline{\#}\}$ und $s = \langle s_1, 1\rangle$

$$\delta(\langle q,1\rangle, \langle a_1,a_2\rangle) = \begin{cases} (\langle a_1', a_2\rangle, \langle q',1\rangle) & \text{falls } \delta(q,a_1) = (a_1', q') \\ (R, \langle q',1\rangle) & \text{falls } \delta(q,a_1) = (R, q') \\ (L, \langle q',1\rangle) & \text{falls } \delta(q,a_1) = (L, q') \end{cases}$$
$$\delta(\langle q,2\rangle, \langle a_1,a_2\rangle) = \begin{cases} (\langle a_1, a_2'\rangle, \langle q',2\rangle) & \text{falls } \delta(q,a_2) = (a_2', q') \\ (R, \langle q',2\rangle) & \text{falls } \delta(q,a_2) = (L, q') \\ (L, \langle q',2\rangle) & \text{falls } \delta(q,a_2) = (R, q') \end{cases}$$

Intuitiv: $\langle q,1\rangle$ entspricht q bei Lektüre des oberen Zeichens, $\langle q,2\rangle$ entspricht q bei Lektüre des unteren Zeichens.

$$\left.\begin{array}{l} \delta(\langle q,1\rangle, \$) = (R, \langle q,2\rangle) \\ \delta(\langle q,2\rangle, \$) = (R, \langle q,1\rangle) \end{array}\right\} \text{Spurwechsel}$$

$$\left.\begin{array}{l} \delta(\langle q,1\rangle, \#) = (\langle\#,\#\rangle, \langle q,1\rangle) \\ \delta(\langle q,2\rangle, \#) = (\langle\#,\#\rangle, \langle q,2\rangle) \end{array}\right\} \text{Erweiterung des 2-Spur-Bandes nach rechts}$$
$$\left.\begin{array}{l} \delta(\langle q,1\rangle, \langle a_1,a_2\rangle) = (\langle\overline{a}_1, a_2\rangle, \langle q,1\rangle) \text{ falls } \delta(q,a_1) \text{ undefiniert} \\ \delta(\langle q,2\rangle, \langle a_1,a_2\rangle) = (\langle a_1, \overline{a}_2\rangle, \langle q,2\rangle) \text{ falls } \delta(q,a_2) \text{ undefiniert} \end{array}\right\} \hookleftarrow$$
$\hookrightarrow$ Markierung der Halteposition

Für Elemente von $\Sigma_{1_\#} \times \overline{\Sigma}_{1_\#}$ und $\overline{\Sigma}_{1_\#} \times \Sigma_{1_\#}$ sind keine Übergänge definiert.

Falls $\begin{pmatrix} \# & w \\ s_1 & \end{pmatrix} \overset{*}{\underset{\mathcal{M}_1}{\vdash}} \begin{pmatrix} u & a & v \\ & q & \end{pmatrix} \in H$, dann gilt

$$\begin{pmatrix} \$ & \begin{array}{cc} \# & w \\ \# & \# \ldots \# \\ s & \end{array} \end{pmatrix} \overset{*}{\underset{\mathcal{M}}{\vdash}} \begin{pmatrix} \$ & \begin{array}{c} c_1 \cdot \vdots \cdot c_n \\ c_1' \cdot \vdots \cdot c_n' \\ \langle q,i\rangle \end{array} \end{pmatrix} \in H, \text{ wobei}$$

$c_n' \cdots c_1'\ c_1 \cdots c_n = \# \ldots \#\ u\overline{a}v\# \ldots \#$, und umgekehrt.

Sei $\mathcal{M}_2'' = \langle Q_2'', \Sigma_2, \delta_2'', s_2''\rangle$ die RTM, die Zeichenketten $\$ \begin{array}{c} c_1 \cdots c_n \\ c_1' \cdots c_n' \end{array}$ in $c_n' \cdots c_1'\ c_1 \cdots c_n$ umformt, blanks am Anfang und am Ende beseitigt und an der markierten Stelle $\overline{a}$ nach Umformung von $\overline{a}$ in a stehenbleibt.
Setze $\mathcal{M}_2 := \mathcal{M}_2'\mathcal{M}\mathcal{M}_2''$.

Theorem 10.11 (Simulation von RTM durch TM)

Sei $\mathcal{M}_1 = \langle Q_1, \Sigma_1, \delta_1, s_1\rangle$ RTM. Dann gibt es TM $\mathcal{M}_2 = \langle Q_2, \Sigma_2, \delta_2, s_2\rangle$ mit $\Sigma_2 \supseteq \Sigma_1$, so daß für alle $w \in \Sigma_1^*$ gilt:

(1) $\begin{pmatrix} \# & w \\ s_1 & \end{pmatrix} \overset{*}{\underset{\mathcal{M}_1}{\vdash}} k \in H \Rightarrow \begin{pmatrix} \# & w \\ s_2 & \end{pmatrix} \overset{*}{\underset{\mathcal{M}_2}{\vdash}} k' \in H$, wobei k' wie k, nur ohne führende blanks.

(2) $\begin{pmatrix} \# & w \\ s_1 & \end{pmatrix} \overset{*}{\underset{\mathcal{M}_1}{\vdash}} k$, wobei k Hängekonfiguration $\Rightarrow \begin{pmatrix} \# & w \\ s_2 & \end{pmatrix} \overset{*}{\underset{\mathcal{M}_2}{\vdash}} k'$, wobei k' ausgezeichnete Haltekonfiguration.

(3) $\mathcal{M}_1$ terminiert nicht für w und Fall (2) ist nicht anwendbar $\Rightarrow \mathcal{M}_2$ terminiert nicht für w.

Beweis: $\Sigma_2 := \Sigma_1 \cup \{\$\},\ Q_2 = Q_1,\ s_2 = s_1,\ \mathcal{M}_2 := L\$R$, d.h. $(\underline{\#}w) \underset{\mathcal{M}_2}{\Longrightarrow} (\$\underline{\#}w)$.
$\delta_2(q, \$)$ sei undefiniert für alle q, sonst $\delta_2 := \delta_1$.

Definition 10.12 (Mehrband-Maschine)

Eine Mehrband-Maschine ist ein Quadrupel $\mathcal{M} = \langle Q, \Sigma, \delta, s\rangle$, wobei $\delta : (Q \times \Sigma_\#^k) \to (\Sigma_\# \cup \{L, R\})^k \times Q$. Konfigurationen haben die Form $\begin{pmatrix} & u_1 & \underline{a_1} & v_1 \\ q, & & \vdots & \\ & u_n & \underline{a_n} & v_n \end{pmatrix}$, d.h. ein Zustand bezieht sich auf Positionen in k Bändern, die unabhängig voneinander in einem einzigen Schritt modifiziert werden können. Akzeptanz von w bedeutet, daß

$$\begin{pmatrix} & \underline{\#} & & w & \\ & \vdots & \cdots & \cdots & \cdots \\ & \vdots & \ddots & \ddots & \ddots \\ s, & \vdots & \ddots & \# & \ddots \\ & \vdots & \ddots & \ddots & \ddots \\ & \underline{\#} & \cdots & \cdots & \cdots \end{pmatrix}$$ in eine Haltekonfiguration überführt wird.

Berechnung von $f(w_1, \ldots, w_n) = w$ bedeutet, daß $$\begin{pmatrix} & \underline{\#} & w_1 & \#\ldots\# & w_n \\ & \vdots & \cdots & \cdots & \cdots \\ & \vdots & \ddots & \ddots & \ddots \\ s, & \vdots & \ddots & \# & \ddots \\ & \vdots & \ddots & \ddots & \ddots \\ & \underline{\#} & \cdots & \cdots & \cdots \end{pmatrix}$$

in eine Haltekonfiguration $\left(q, \dfrac{\#w}{\text{beliebig}}\right)$ überführt wird.

Theorem 10.13 (Simulation von Mehrband-Maschinen durch TM)

Zu jeder Mehrband-Maschine $\mathcal{M}_1$ über Σ gibt es eine TM $\mathcal{M}_2$ über $\Sigma_2 \supseteq \Sigma_1$, so daß gilt:

$$(1)\quad \left(s_1, \begin{array}{cccc} \underline{\#} & & w & \\ \vdots & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots \\ \vdots & \ddots & \# & \ddots \\ \vdots & \ddots & \ddots & \ddots \\ \underline{\#} & \cdots & \cdots & \cdots \end{array}\right) \overset{*}{\underset{\mathcal{M}_1}{\vdash}} \left(q, \begin{array}{ccc} u & \underline{a} & v \\ & w_2 & \\ & \vdots & \\ & w_n & \end{array}\right) \in H \text{ für ein } q \in Q_1,$$

$$\Rightarrow (s_2, \underline{\#}w) \overset{*}{\underset{\mathcal{M}_2}{\vdash}} (q', u\underline{a}v) \in H \text{ für ein } q' \in Q_2$$

(2) $\mathcal{M}_1$ terminiert nicht für $w \Rightarrow \mathcal{M}_2$ terminiert nicht für w.

Beweisskizze:

Simuliere $\left(q, \begin{array}{ccc} u_1 & \underline{a_1} & v_1 \\ & \vdots & \\ u_n & \underline{a_n} & v_n \end{array}\right)$ in der n-Bandmaschine durch $\left(q, \begin{array}{cccc} \# & u_1 & a_1 & v_1 \\ \# & \cdots\# & \S & \cdots\# \\ \vdots & \vdots & \vdots & \vdots \\ \# & u_n & a_n & v_n \\ \underline{\#} & \cdots\# & \S & \cdots\# \end{array}\right)$

in einer geeigneten $2n$-Spur-(jedoch 1-Band-)TM.

Die §-Zeichen markieren jeweils, wo sich der Schreib-Lesekopf der n-Band-Maschine befindet. Jedem einzelnem Schritt der n-Band-Maschine entsprechen n Durchläufe der $2n$-Spur-Maschine. Dabei wird jeweils die Stelle, an der a_i steht, anhand des §-Zeichens darunter aufgesucht, dann je nach Anforderung a_i überschrieben bzw. das darunterstehende §-Zeichen nach rechts oder links bewegt und dann zum Ausgangspunkt zurückgegangen.

Bemerkungen:

1. Eine n-Band-Maschine hat für jedes Band einen eigenen „Lese-Schreib-Kopf". Die n Lese-Schreib-Köpfe können unabhängig voneinander positioniert werden. Eine n-Spur-Maschine hat *einen einzigen* Lese-Schreib-Kopf, der sich auf alle Spuren bezieht; es werden also immer n-tupel an *einer* Position gelesen.

2. Die Simulation von TM durch Mehrband-Maschinen ist trivial, da jede TM eine 1-Band-Maschine ist.

Definition 10.14 (nichtdeterministische TM)

Eine nichtdeterministische TM unterscheidet sich von einer TM dadurch, daß δ als Werte eine Menge hat d.h. $\delta : (Q \times \Sigma_\#) \to \mathcal{P}((\Sigma_\# \cup \{R, L\}) \times Q)$ d.h. $\delta(q, a) \subseteq (\Sigma_\# \cup \{R, L\}) \times Q$.

In der Definition von $\overset{1}{\vdash}$ ist überall $\delta(q, a) = \cdots$ zu ersetzen durch $\delta(q, a) \ni \cdots$

δ ist hier als *totale* Funktion aufgefaßt (die allerdings für manche Agumente den Wert $\emptyset$ haben kann).

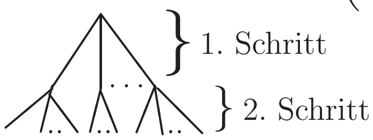
Bemerkung (Theoreme 10.10, 10.11 und 10.13):

Theoreme 10.10, 10.11 und 10.13 gelten auch für nichtdeterministische TM.

Theorem 10.15 (Simulation nichtdeterministischer TM durch TM)

Zu jeder nichtdeterministischen TM $\mathcal{M}_1$ über Σ_1 läßt sich eine TM $\mathcal{M}_2$ über $\Sigma_2 \supseteq \Sigma_1$ konstruieren, so daß gilt: $w \in \Sigma_1$ wird von $\mathcal{M}_1$ akzeptiert g.d.w. w wird von $\mathcal{M}_2$ akzeptiert.

Beweisskizze: Sei $\mathcal{M}_1 = \langle Q_1, \Sigma_1, \delta_1, s_1 \rangle$. Für alle q, a gilt:
$|\delta(q,a)| \leq |Q| \cdot |\Sigma_{1_\#} \cup \{R, L\}| = k$ für eine Konstante k. Die möglichen Konfigurationen von $\mathcal{M}_1$ ausgehend von $\begin{pmatrix} \# & w \\ s_1 & \end{pmatrix}$, lassen sich durch einen Baum beschreiben:



wobei jeder Knoten höchstens k Unterknoten hat. Wenn man den Baum breadth-first durchläuft (d.h. erst alle unmittelbaren Nachfolger eines Knoten betrachtet, bevor man deren Nachfolger betrachtet), erreicht man jede Haltekonfiguration. Man ordne jedem Knoten eine Folge von Zahlen $(p_0, \ldots, p_n)$ mit $p_i \leq k$ zu, wobei p_i die Nummer des im i-ten Schritt betrachteten Unterknotens (von links nach rechts gezählt) ist, und ordne diese lexikographisch. Dann bilde man die 3-Band-Maschine $\mathcal{M}_2$:

1. Band	$\underline{\#}w$	Ausgangswort
2. Band	$\underline{\#}w$	Kopie davon
3. Band	$\underline{\#}p_0\# \ldots \#p_n$	Knotenindex (p_i durch Folge von Sternen repräsentiert)

Das 1. Band bleibt unverändert. Das 2. Band simuliert die Maschine $\mathcal{M}_1$ entsprechend der auf dem 3. Band notierten Schrittfolge. Das 3. Band läuft lexikographisch alle möglichen Schrittfolgen durch, bis auf dem 2. Band kein Nachfolger definiert ist. Dann hält $\mathcal{M}_2$.

Bemerkungen:

1. Die globale Beschränkung der Anzahl der Unterknoten eines Knoten durch k ist wichtig, da dadurch von vornherein klar ist, was auf dem 3. Band zu generieren ist.

2. Das Theorem bezieht sich auf *Akzeptanz*, nicht auf *Berechenbarkeit*.
   $(\#w) \overset{*}{\underset{\mathcal{M}_1}{\vdash}} (\underline{\#}u) \in H$ kann für *mehrere* u gelten.

Exkurs zur Komplexitätstheorie: Das P-NP Problem

Nachdem wir deterministische und nichtdeterministische TM definiert haben, stehen die Hilfsmittel bereit, eines der fundamentalen offenen Probleme der theoretischen Informatik zu formulieren. Für alle detaillierten und weiterführenden Erörterungen sei auf die Komplexitätstheorie verwiesen. In diesem Exkurs stehe „NTM“ für „nichtdeterministische TM“. Elementare Einführung mit weiterführenden Literaturhinweisen: Schöning (2001).

Definition 1 (Laufzeit)

Sei $\mathcal{M}$ TM. Es sei

$$\begin{array}{rlll} time_{\mathcal{M}}: & \Sigma^* & \to & \mathbb{N} \\ & w & \mapsto & \text{Anzahl der Rechenschritte} \\ & & & (\overset{1}{\underset{\mathcal{M}}{\vdash}}\text{-Übergänge) ausgehend von } \begin{pmatrix} \# & w \\ s & \end{pmatrix}\text{, bis } \mathcal{M} \text{ hält} \end{array}$$

Falls $\mathcal{M}$ NTM, wählen wir rechts das Minimum der Anzahl der Rechenschritte. Offenbar ist $time_{\mathcal{M}}$ eine partielle Funktion, da $\mathcal{M}$ nicht notwendigerweise terminiert.

Definition 2 (P, NP)

$$\begin{array}{lll} \mathrm{P} & := & \{L \subseteq \Sigma^*\text{: Es gibt TM } \mathcal{M} \text{ und ein Polynom } p \text{ mit } L = L(\mathcal{M}) \text{ und} \\ & & time_{\mathcal{M}}(w) \leq p(|w|) \text{ für alle } w \in L\} \\ \mathrm{NP} & := & \{L \subseteq \Sigma^*\text{: Es gibt NTM } \mathcal{M} \text{ und ein Polynom } p \text{ mit } L = L(\mathcal{M}) \text{ und} \\ & & time_{\mathcal{M}}(w) \leq p(|w|) \text{ für alle } w \in L\} \end{array}$$

Das P-NP-Problem lautet: $\mathrm{P} \overset{?}{=} \mathrm{NP}$

Bemerkungen:

1. Ein Polynom ist eine Funktion $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$

2. $\mathrm{P} \subseteq \mathrm{NP}$ ist trivial. Das Problem besteht also in der Frage $\mathrm{NP} \overset{?}{\subseteq} \mathrm{P}$.

3. Jedes $L \in \mathrm{NP}$ ist Turing-entscheidbar:
   Sei $\mathcal{M}$ NTM, die L akzeptiert und p ein Polynom, das die Laufzeit für akzeptierende Berechnungen beschränkt. Dann kann man mit Hilfe einer geeigneten deterministischen Mehrband-Maschine für jedes Eingabewort w alle (nichtdeterministischen) Alternativen von Schrittfolgen von $\mathcal{M}$ der maximalen Länge $p(|w|)$ generieren. Falls eine dieser Folgen terminiert, wird 1 ausgegeben, sonst 0. Wegen $L \in \mathrm{NP}$ gilt ja: $w \in L$ g.d.w. eine dieser Folgen terminiert. (Vgl. Beweis von Theorem 10.15)

4. In unserem Kontext spielt es keine Rolle, ob wir (Einband-) TM oder Mehrband-TM als Ausganspunkt wählen, da die Reduktion von Mehrband-TM auf TM in quadratischer Zeit erzielt werden kann.

Definition 3 (Polynomiale Reduzierbarkeit)

Seien $L_1, L_2 \subseteq \Sigma^*$. L_1 heißt auf L_2 polynomial reduzierbar ($L_1 \leq_p L_2$) falls es eine polynomial berechenbare totale Funktion $f : \Sigma^* \to \Sigma^*$ gibt mit $w \in L_1 \Leftrightarrow f(w) \in L_2$ für alle $w \in \Sigma^*$. f heißt dabei polynomial berechenbar, wenn es eine TM $\mathcal{M}$ gibt, die f berechnet und die polynomial beschränkt ist, d.h. $time_{\mathcal{M}}(w) \leq p(|w|)$ für ein Polynom p für alle $w \in \Sigma^*$.

Definition 4 (NP-hart/NP-vollständig)

L heißt NP-hart, falls für alle $L' \in$ NP gilt $L' \leq_p L$.
L heißt NP-vollständig, falls L NP-hart und $L \in$ NP.

Theorem 5

Sei L NP-vollständig. Dann gilt:
$L \in P \Leftrightarrow$ P=NP

Beweis: leichte Übung
(Für „$\Rightarrow$“ ist zu benutzen: $L' \leq_p L \in P \quad \Rightarrow \quad L' \in P$.)

Beispiele für NP-vollständige Probleme:

1. Erfüllbarkeit aussagenlogischer Formeln (Gibt es eine „1“ in der letzten Spalte der Wahrheitstafel für eine gegebene Formel ?) („SAT “)

2. Travelling-Salesman-Problem: Sei $n \times n$ Matrix (M_{ij}) über $\mathbb{N}$ gegeben.
   Gibt es für gegebenes k eine Permutation π von $(1, \ldots, n)$ mit
   $\sum_{i=1}^{n-1} M_{\pi(i),\pi(i+1)} + M_{\pi(n),\pi(i)} \leq k$?
   (Gibt es eine Rundreise durch n Orte, deren Länge einen vorgegebenen Wert nicht überschreitet ?)

   Daß z.B. $SAT \in NP$, ist offensichtlich: Man bestimme aus einer Formel A deterministisch die vorkommenden Aussagevariablen $p_1, \ldots, p_n$ und wähle dann (nichtdeterministisch) Belegungen $\langle i_1, \ldots, i_n \rangle$ ($i_j \in \{0, 1\}$), für die man jeweils deterministisch bestimmt, ob sie A erfüllen. Die (nichtdeterministische) Rechenzeit ist dann sicherlich polynomial durch die Länge von A beschränkt.

Der schwierige Teil des Nachweises der NP-Vollständigkeit liegt also im Nachweis der NP-Härte (für SAT erstmals von Cook 1971 bewiesen).
SAT stellt ein prominentes Paradigma dar, auf das man häufig die NP-Vollständigkeit anderer Probleme zurückführt.

Beispiel für ein NP-hartes Problem:

Das Wortproblem für Typ-1-Sprachen ist NP-hart, aber möglicherweise nicht NP-vollständig (es ist „PSPACE-vollständig“).

Bemerkung:

Die Bedeutung des P-NP-Problems liegt darin, daß einerseits polynomial beschränkter Rechenzeitaufwand erwartet wird für Probleme, die man effektiv mit dem Rechner behandeln will, es andererseits aber viele zentrale Probleme gibt, die sich nur nichtdeterministisch mit solcherart beschränktem Aufwand lösen lassen. Es wird meistens vermutet, daß P $\neq$ NP gilt.

11 Turing-Aufzählbarkeit, -Akzeptierbarkeit, -Berechenbarkeit, -Entscheidbarkeit

Wir stellen die Beziehung zwischen nicht eingeschränkten Grammatiken und Begriffen der Berechenbarkeitstheorie her. Ferner geben wir Beispiele für ein nicht entscheidbares Problem und eine nicht berechenbare Funktion.

Definition 11.1 (Turing-Eigenschaften)

Eine Sprache L über Σ heißt Turing-akzeptierbar, falls es TM $\mathcal{M}$ über $\Sigma' \supseteq \Sigma$ gibt mit $L = L(\mathcal{M})$.

L heißt Turing-aufzählbar, falls es TM $\mathcal{M}$ über $\Sigma' \supseteq \Sigma$ gibt, so daß gilt: $L = \{w \in {\Sigma'}^*$: Es gibt ein $u \in {\Sigma'}^*$ mit $(\underline{\#}u) \underset{\mathcal{M}}{\Longrightarrow} (\underline{\#}w)\}$

L heißt Turing-entscheidbar, falls es TM $\mathcal{M}$ über $\Sigma' \supseteq \Sigma \cup \{0, 1\}$ gibt, die L entscheidet.

$f : \Sigma^* \times \ldots \times \Sigma^* \to \Sigma^*$ heißt Turing-berechenbar, wenn es eine TM $\mathcal{M}$ über $\Sigma' \supseteq \Sigma$ gibt, die f berechnet.

Bemerkungen:

1. $\Sigma' \supseteq \Sigma$ wegen eventueller Hilfssymbole.
2. L ist Turing-aufzählbar g.d.w. L Wertebereich einer einstelligen Turing-berechenbaren Funktion ist.
3. Beachte, daß nach dieser Definition eine Turing-berechenbare Funktion partiell sein kann, während von einer Turing-entscheidbaren Sprache verlangt wird, daß von jedem Wort festgestellt werden kann, ob es zur Sprache gehört oder nicht. Man kann auch den Begriff der Turing-Entscheidbarkeit partiell definieren (vgl. Definition 16.1). Unser Begriff der Turing-Entscheidbarkeit entspricht dem späteren Begriff der rekursiven, nicht dem der partiell rekursiven Menge.

Definition 11.2 („grammatisch“)

$L \subseteq \Sigma^*$ heißt grammatisch, falls es eine (nicht eingeschränkte) Grammatik Γ gibt mit $L = L(\Gamma)$.

Theorem 11.3

Folgendes ist äquivalent:

(i) L ist grammatisch

(ii) L ist Turing-akzeptierbar

(iii) L ist Turing-aufzählbar

Notation

Sei $\mathcal{M}$ eine 1-Band-Maschine. Im Rahmen eines Maschinenschemas für eine Mehrband-Maschine steht $\mathcal{M}^{(i)}$ für eine Mehrband-Maschine, die sich auf dem i-ten Band wie $\mathcal{M}$ verhält und die anderen Bänder unverändert läßt.

Beweis: (i) $\Rightarrow$ (ii): Sei $L = L(\Gamma)$, wobei Γ Grammatik über Σ mit Anfangssymbol S

A) Wir konstruieren nichtdeterministische TM T_Γ über geeignetem $\Sigma' \supseteq \Sigma$, die folgendes leistet:

1. Alle Haltekonfigurationen sind von der Form $(\underline{\#}w)$ für $w \in \Sigma^*$
2. $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$ g.d.w. $(\underline{\#}u)$ wird von T_Γ in $(\underline{\#}v)$ transformiert. (Übungsaufgabe)

B) Wir konstruieren die Maschine E_Σ, die die Gleichheitsrelation über Σ^* entscheidet:
$(\underline{\#}w\#w) \underset{E_\Sigma}{\Longrightarrow} (\underline{\#}1)$
$(\underline{\#}w\#u) \underset{E_\Sigma}{\Longrightarrow} (\underline{\#}0)$ falls $w \neq u$

C) Wir konstruieren eine 3-Band-Maschine $\mathcal{M}_1$ über $\Sigma \cup \{0\}$, die $\begin{pmatrix} \underline{\#}w \\ \underline{\#} \\ \underline{\#} \end{pmatrix}$ in $\begin{pmatrix} \underline{\#}w \\ \underline{\#}0 \\ \underline{\#}S \end{pmatrix}$ transformiert.

D) Wir konstruieren eine 3-Band-Maschine $\mathcal{M}_2$ über $\Sigma \cup \{0\}$, die $\begin{pmatrix} \underline{\#}w \\ \underline{\#}0 \\ \underline{\#}v \end{pmatrix}$ in $\begin{pmatrix} \underline{\#}w \\ \underline{\#}w\#v \\ \underline{\#}v \end{pmatrix}$ transformiert.

Dann definieren wir eine nichtdeterministische 3-Band-Maschine wie folgt:

$$\begin{tikzcd} \mathcal{M}_1 \arrow[r] & \mathcal{M}_2\, E_\Sigma^{(2)} R^{(2)} \arrow[r, "0"] & T_\Gamma^{(3)} L^{(2)} \arrow[l, bend right] \end{tikzcd}$$

Offenbar wird w von dieser Maschine genau dann akzeptiert, wenn $w \in L(\Gamma)$, d.h. $S \overset{*}{\underset{\Gamma}{\Longrightarrow}} w$.
Intuitiv: Auf Band 3 werden nichtdeterministisch alle in Γ ableitbaren Wörter generiert. Nach jedem Schritt wird das Wort auf Band 2 übertragen und dort mit dem Ausgangswort w verglichen. Sobald Übereinstimmung besteht, hält die Maschine.

(ii) $\Rightarrow$ (iii): Sei $\mathcal{M}_1$ eine Maschine, die L akzeptiert. Sei $w \in L$.
Wir bilden eine 2-Band-Maschine $\mathcal{M}_2$:

1. Band: $\underline{\#}w$ bleibt dort unverändert
2. Band: $\mathcal{M}_1$ wird dort ausgeführt mit der Eingabe $\underline{\#}w$.
Genau dann, wenn $\mathcal{M}_1$ bei Startkonfiguration $(s_1, \underline{\#}w)$ hält, dann hält $\mathcal{M}_2$ bei Startkonfiguration $\left(s_2, \begin{array}{cc} \underline{\#} & w \\ \underline{\#} & \end{array}\right)$ mit Haltekonfiguration $\left(q, \begin{array}{c} \underline{\#}\,w \\ \dots \end{array}\right)$. $\mathcal{M}_2'$ sei 1-Band-Version von $\mathcal{M}_2$ gemäß Theorem 10.13. Es gilt: $(\underline{\#}w) \underset{\mathcal{M}_2'}{\Longrightarrow} (\underline{\#}w)$ falls $w \in L$; falls $w \notin L$, terminiert $\mathcal{M}_2'$ für $(\underline{\#}w)$ nicht. Also wird L durch $\mathcal{M}_2'$ aufgezählt.

Intuitiv: $\mathcal{M}_2'$ repräsentiert die partielle Identitätsfunktion, die w ausgibt, falls w von $\mathcal{M}_1$ akzeptiert wird, und sonst nicht terminiert.

(iii) $\Rightarrow$ (i): Sei $\mathcal{M} =< Q, \Sigma, \delta, s >$ die TM, die L aufzählt.
Es gibt es Grammatik Γ über $\Sigma \cup \{s, h, (,), \#\}$, so daß gilt: $\left(\begin{array}{cc} \# & u \\ s & \end{array}\right)$ wird in Haltekonfiguration $\left(\begin{array}{cc} \# & w \\ q & \end{array}\right)$ transformiert g.d.w. $(s\#u) \overset{*}{\underset{\Gamma}{\Longrightarrow}} (h\#w)$ (Beweis als Übungsaufgabe:
Schreibe $\left(\begin{array}{ccc} u & a & v \\ & q_i & \end{array}\right)$ als $(u\underbrace{s\dots s}_{i-mal}av))$.
Erweitere Γ um folgende Produktionen zu Γ':
$S \to (s\#X)$
$X \to aX$ für alle $a \in \Sigma$
$X \to \varepsilon$
$(h\# \to \varepsilon$
$) \to \varepsilon$.
Dann gilt : $S \overset{*}{\underset{\Gamma'}{\Longrightarrow}} w$ g.d.w. es gibt u so daß $(\underline{\#}u) \underset{\mathcal{M}}{\Longrightarrow} (\underline{\#}w)$.

Theorem 11.4

Sei $L \subseteq \Sigma^*$. L ist Turing-entscheidbar g.d.w. sowohl L als auch $\overline{L}$ sind Turing-aufzählbar (Turing-akzeptierbar).

Beweis: „$\Rightarrow$“: Sei $\mathcal{M}$ eine Maschine, die L entscheidet. Sei $\perp$ eine Maschine, die bei keiner Eingabe terminiert. Dann gilt:
$\mathcal{M}R \overset{0}{\to} \perp$ akzeptiert L; $\mathcal{M}R \overset{1}{\to} \perp$ akzeptiert $\overline{L}$.

„$\Leftarrow$": $\mathcal{M}_1$ akzeptiere L, $\mathcal{M}_2$ akzeptiere $\overline{L}$.
Sei T_1 die Maschine, die eine beliebige Konfiguration in $(\underline{\#}1)$ überführt, T_0 die Maschine, die eine beliebige Konfiguration in $(\underline{\#}0)$ überführt. $\mathcal{M}'$ sei die 3-Band-Maschine, die $\begin{pmatrix} \underline{\#}w \\ \underline{\#} \\ \underline{\#} \end{pmatrix}$ in $\begin{pmatrix} \underline{\#}w \\ \underline{\#}w \\ \underline{\#}w \end{pmatrix}$ überführt.
Die Entscheidungsmaschine für L sei eine 3-Band-Maschine, die erst $\mathcal{M}'$ ausführt und dann „parallel" (abwechselnd je ein Schritt) $\mathcal{M}_1^{(2)}T_1^{(1)}$ und $\mathcal{M}_2^{(3)}T_0^{(1)}$. Wenn eine dieser Maschinen hält, soll die gesamte Entscheidungsmaschine halten.

Korollar 11.5

L Turing-entscheidbar g.d.w. L und $\overline{L}$ grammatisch

Beweis: aus Theorem 11.3

Theorem 11.6

$f : \Sigma^* \times \ldots \times \Sigma^* \to \Sigma^*$ (f kann partiell sein) ist Turing-berechenbar g.d.w. graph(f):= $\{(w_1, \ldots, w_n, w) : f(w_1, \ldots, w_n) = w\}$ ist Turing-aufzählbar (Turing-akzeptierbar).
Wir fassen dabei $(w_1, \ldots, w_n, w)$ als Element von $(\Sigma \cup \{\$\})^*$ auf: $w_1\$ \ldots \$w_n\$w$.

Beweis: "$\Rightarrow$": $\mathcal{M}$ berechne f. Seien $\mathcal{M}_1$ und E folgende 3-Band-Maschinen:

$$\mathcal{M}_1 : \begin{pmatrix} \underline{\#}w_1\$ \ldots \$w_n\$w \\ \underline{\#} \\ \underline{\#} \end{pmatrix} \Longrightarrow \begin{pmatrix} \underline{\#}w_1\$ \ldots \$w_n\$w \\ \underline{\#}w_1\# \ldots \#w_n \\ \underline{\#}w \end{pmatrix}$$

E: $\begin{pmatrix} \underline{\#}w \\ \underline{\#}u \\ \underline{\#}v \end{pmatrix} \Longrightarrow \begin{pmatrix} \underline{\#}w \\ \underline{\#} \\ \underline{\#} \end{pmatrix}$, falls $u = v$, ansonsten terminiere E nicht.

Dann wird graph(f) von der 3-Band-Maschine $\mathcal{M}_1\mathcal{M}^{(2)}E$ akzeptiert.

Für die Umkehrung vergleiche die Bemerkung zu Theorem 15.4.

Nicht Turing-berechenbare Funktionen

Es gibt solche: Die Menge der TM über einem festen Alphabet ist abzählbar, die Menge aller partiellen Funktionen ist nicht abzählbar (Diagonalisierungsargument).

Angabe einer solchen Funktion: Sei $f_1, f_2, \ldots$ Abzählung aller einstelligen Turing-berechenbaren Funktionen über $\mathbb{N}$.

Setze $g(n) := \begin{cases} f_n(n)+1 & \text{falls } f_n(n) \text{ definiert} \\ 1 & \text{sonst} \end{cases}$

Falls g selbst Turing-berechenbar, gilt $g = f_m$ für ein m (da dann g in der Abzählung vorkommen muß). Damit gilt:

$f_m(m) = g(m) = \begin{cases} f_m(m)+1 & \text{falls } f_m(m) \text{ definiert} \\ 1 & \text{sonst} \end{cases}$

Widerspruch: $\begin{array}{lll} f_m(m) \text{ definiert} & \Rightarrow & f_m(m) = f_m(m)+1 \\ f_m(m) \text{ nicht definiert} & \Rightarrow & f_m(m) = 1 \text{ (d.h. } f_m(m) \text{ definiert).} \end{array}$

D.h. g ist nicht Turing-berechenbar.

Andererseits werden wir eine zweistellige Turing-berechenbare Funktion U angeben, so daß

$U(m,n) = \begin{cases} f_m(n) & \text{falls } f_m(n) \text{ definiert} \\ \text{undefiniert} & \text{sonst} \end{cases}$

Damit gilt: Es gibt einstellige Turing-berechenbare Funktion V mit
$V(m) = U(m,m) = f_m(m)$, falls $f_m(m)$ definiert,
d.h. es gibt l, so daß
$f_l(m) = V(m) = f_m(m)$, falls $f_m(m)$ definiert.
Damit gibt es auch ein i, so daß
$f_i(m) = f_m(m)+1$, falls $f_m(m)$ definiert.
Damit
$f_i(i) = f_i(i)+1$, falls $f_i(i)$ definiert.
Also gilt: $f_i(i)$ und damit $U(i,i)$ ist nicht definiert.
Es kann aber nicht Turing-entscheidbar sein, ob $f_m(n)$ definiert ist. Ansonsten ließe sich U total machen (nach dem Muster von g) und $U(i,i)$ wäre definiert.

Definition 11.7 (Halteproblem)

Für eine TM $\mathcal{M}$ heiße $L(\mathcal{M})$ auch das Halteproblem von $\mathcal{M}$. Das universelle Halteproblem für TM ist die Menge $\{(m,n) : f_m(n) \text{ ist definiert}\}$ oder äquivalent
$\{(\mathcal{M}, w) : w \in L(\mathcal{M})\}$.

Theorem 11.8

Das universelle Halteproblem für TM ist nicht Turing-entscheidbar.

Beweis: Ergibt sich aus der Argumentation vor Definition 11.7:
Die universelle Funktion $U(m,n)$ ergibt sich aus dem Kleenschen Normalformensatz (Theorem 13.6) bzw. aus der Existenz universeller TM (Kapitel 15).

Konkretes Beispiel einer nicht-berechenbaren Funktion („busy beaver“- -Funktion)

Sei $\Sigma = \{*\}$. Wir definieren: $p(\mathcal{M}) = n$ g.d.w. $(\underline{\#}) \underset{\mathcal{M}}{\Longrightarrow} (\underline{\#}\underbrace{*\ldots*}_{n+1})$.

$p(\mathcal{M})$ heißt „Produktivität von $\mathcal{M}$“.
$P(k) := \max\{p(\mathcal{M}) : \mathcal{M}$ hat genau k Zustände $\}$. (Produktivität der produktivsten Maschine mit k Zuständen.)
Es gilt: $P(n+1) > P(n)$. („Ersetze # durch $*$ und gehe ein Feld nach links.“)
Daraus erhalten wir:
(1) $m > n \Rightarrow P(m) > P(n)$, und damit $P(m) \leq P(n) \Rightarrow m \leq n$
Ferner gilt:
(2) $P(n+11) > 2n$ falls $n > 0$

Beweis von (2): Es gibt eine Maschine mit 10 Zuständen, die $(\underline{\#}\underbrace{*\ldots*}_{n})$ in $(\underline{\#}\underbrace{*\ldots*}_{2n})$ transformiert ($n > 0$). Mit $n+1$ Zuständen kann man ferner $(\underline{\#}\underbrace{*\ldots*}_{n+1})$ aus $(\underline{\#})$ erzeugen. Daher kann man mit $n+11$ Zuständen $(\underline{\#}\underbrace{*\ldots*}_{2n+2})$ aus $(\underline{\#})$ erzeugen, d.h.

$P(n+11) \geq 2n+1$

Wir zeigen nun: Die „busy beaver“-Funktion P ist nicht Turing-berechenbar.

Annahme: Sei P Turing-berechenbar. Sei $\mathcal{M}_{bb}$ die TM, die P berechnet. $\mathcal{M}_{bb}$ habe k Zustände. Sei T_n die Maschine mit $n+1$ Zuständen, die $(\underline{\#})$ in $(\underline{\#}\underbrace{*\ldots*}_{n+1})$ transformiert.

Wir betrachten $\mathcal{M} = T_n\mathcal{M}_{bb}\mathcal{M}_{bb}$. $\mathcal{M}$ hat $n+1+2k$ Zustände (nach dem Verfahren von Def. 10.7 sogar $n+4+2k$ Zustände weger der eingefügten zusätzlichen Zustände — dieser Unterschied spielt für das folgende Argument keine Rolle).
Es gilt:
$p(\mathcal{M}) = P(P(n))$,
d.h. $P(n+1+2k) \geq p(\mathcal{M}) = P(P(n))$.
Daraus $n+1+2k \geq P(n)$ mit (1).
Das gilt für ein beliebiges n, also auch für $n+11$:
$n+12+2k \geq P(n+11)$. Daraus
$n+12+2k > 2n$ mit (2).
Daraus
$12+2k > n$.
Das gilt für beliebiges n, also auch für $2k+12$. Damit
$12+2k > 12+2k$. Widerspruch.

Intuitiver Grund für diese Situation:

$\mathcal{M}_{bb}\mathcal{M}_{bb}$ stellt $P(P(n))$ dar, d.h. durch Verdopplung der Anzahl der Zustände müßte erreicht werden, was mit Hilfe von $P(n)$ Zuständen erreichbar ist. P ist dabei definiert durch

Quantifikation über *alle* TM: Für geeignetes n ist $P(n)$ größer als die Anzahl der Zustände, mit denen man P durch eine TM realisieren müßte.

These von Alonzo Church/Alan Turing

Turing-Entscheidbarkeit	=	Entscheidbarkeit	Churchsche
Turing-Berechenbarkeit	=	Berechenbarkeit	These

Andere Präzisierungen von Entscheidbarkeit/Berechenbarkeit:

1. Partiell rekursive Funktionen
2. λ-Kalkül-Definierbarkeit
3. Definierbarkeit mit Gleichungen
4. Definierbarkeit durch Registermaschinen

   etc.

Alle diese Präzisierungen sind untereinander äquivalent.
Daraus zieht man den philosophischen Schluß, das Gemeinsame hinter diesen verschiedenartigen formalen Begriffen sei der Begriff der Berechenbarkeit im nicht-formalen Sinne.

12 Primitiv rekursive und partiell rekursive Funktionen

Neben Turingmaschinen behandeln wir primitiv rekursive, partiell rekursive und rekursive Funktionen als Explikation des Begriffs der Berechenbarkeit. Ich folge in diesem und dem nächsten Kapitel im wesentlichen der Darstellung von Hermes (1978), mit dem Unterschied, daß neben primitiv rekursiven durchgängig partiell rekursive und nicht nur rekursive Funktionen betrachtet werden.

Wir behandeln (zunächst totale) zahlentheoretische Funktionen $\mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$. Wir schreiben $\vec{x}$ für eine Folge $x_1, \ldots, x_n$ von natürlichen Zahlen. x' sei die Abkürzung für $x + 1$.

Definition 12.1 (Primitiv rekursive Funktionen)

Die Funktionen

$C_0^0 := 0$ (d.h. 0 aufgefaßt als nullstellige Funktion)

$N(x) := x'$ (Nachfolger)

$U_n^i(x_1, \ldots, x_n) := x_i$ (i-te Projektion)

sind primitiv rekursiv.

Sind $h_1, \ldots, h_r$ n-stellige primitiv rekursive Funktionen, g r-stellige primitiv rekursive Funktion, dann ist f mit
$f(\vec{x}) := g(h_1(\vec{x}), \ldots, h_r(\vec{x}))$ (Einsetzung, Komposition)
eine n-stellige primitiv rekursive Funktion.

Sei g n-stellige primitiv rekursive Funktion, h $(n+2)$-stellige primitiv rekursive Funktion, dann ist f mit

$$\left.\begin{array}{rcl} f(\vec{x}, 0) & = & g(\vec{x}) \\ f(\vec{x}, y') & = & h(\vec{x}, y, f(\vec{x}, y)) \end{array}\right\} \text{primitive Rekursion}$$

eine $(n+1)$-stellige primitiv rekursive Funktion.

Beispiele

(Jeweils in „inoffizieller“ Notation und mit expliziter Rückführung auf die Grundfunktion)

$$V(x): \begin{array}{rcl} V(0) & = & 0 \\ V(y') & = & y \end{array} \qquad \begin{array}{rcl} V(0) & = & C_0^0 \\ V(y') & = & U_2^1(y, V(y)) \end{array}$$

$$x+y: \begin{array}{rcl} x+0 & = & x \\ x+y' & = & (x+y)' \\ & & \end{array} \qquad \begin{array}{rrcl} & +(x,0) & = & U_1^1(x) \\ & +(x,y') & = & h(x,y,+(x,y)) \\ \text{mit} & h(x,y,z) & = & N(U_3^3(x,y,z)) \end{array}$$

$C_n^k(x_1, \ldots, x_n) = k$ (konstant) wird induktiv wie folgt definiert:
C_0^0 : ✓ Ausgangsfunktion
$C_0^{k+1} := N(C_0^k)$
$C_{n+1}^k(\overrightarrow{x}, 0) = C_n^k(\overrightarrow{x}) \quad C_{n+1}^k(\overrightarrow{x}, y') = U_{n+2}^{n+2}(\overrightarrow{x}, y, C_{n+1}^k(\overrightarrow{x}, y))$

$$\mathrm{sg}(x): \begin{array}{lcl} \mathrm{sg}(0) & = & 0 \\ \mathrm{sg}(y') & = & 1 \end{array} \qquad \begin{array}{lcl} \mathrm{sg}(0) & = & C_0^0 \\ \mathrm{sg}(y') & = & C_2^1(y, \mathrm{sg}(y)) \end{array}$$

$$\overline{\mathrm{sg}}(x): \begin{array}{lcl} \overline{\mathrm{sg}}(0) & = & 1 \\ \overline{\mathrm{sg}}(y') & = & 0 \end{array} \qquad \begin{array}{lcl} \overline{\mathrm{sg}}(0) & = & C_0^1 \\ \overline{\mathrm{sg}}(y') & = & C_2^0(y, \overline{\mathrm{sg}}(y)) \end{array}$$

$$x \dot{-} y: \begin{array}{lcl} x \dot{-} 0 & = & x \\ x \dot{-} y' & = & V(x \dot{-} y) \\ & & \end{array} \qquad \begin{array}{rrcl} & \dot{-}(x, 0) & = & U_1^1(x) \\ & \dot{-}(x, y') & = & h(x, y, \dot{-}(x, y)) \\ \text{mit} & h(x, y, z) & = & V(U_3^3(x, y, z)) \end{array}$$

$$|x - y|: \quad |x - y| \;=\; (x \dot{-} y) + (y \dot{-} x) \qquad |-|(x, y) \;=\; +(\dot{-}(x, y), \dot{-}(y, x))$$

$$\begin{array}{llcl} \varepsilon(x, y): & \varepsilon(x, y) & = & \mathrm{sg}(|x - y|) \\ \delta(x, y): & \delta(x, y) & = & \overline{\mathrm{sg}}(|x - y|) \end{array}$$

$$x \cdot y: \quad \begin{array}{lcl} x \cdot 0 & = & 0 \\ x \cdot y' & = & (x \cdot y) + x \end{array}$$

$$\begin{array}{lcl} \cdot(x, 0) & = & C_1^0(x) \\ \cdot(x, y') & = & h(x, y, \cdot(x, y)) \quad \text{mit} \quad h(x, y, z) = +(U_3^3(x, y, z), U_3^1(x, y, z)) \end{array}$$

$$\prod_{i=0}^{y} f(\overrightarrow{x}, i): \qquad \begin{array}{lcl} \prod_{i=0}^{0} f(\overrightarrow{x}, i) & = & f(\overrightarrow{x}, 0) \\ \prod_{i=0}^{y'} f(\overrightarrow{x}, i) & = & (\prod_{i=0}^{y} f(\overrightarrow{x}, i)) \cdot f(\overrightarrow{x}, y') \end{array}$$

(ausführliche Notation als Übung)

$\sum_{i=0}^{y} f(\overrightarrow{x}, i)$ analog.

Lemma 12.2

Sei f jeweils definiert durch

1. $f(x_1, \ldots, x_n) = g(x_1, \ldots, x_i, x_j, x_{i+1}, \ldots, x_n)$,
   wobei g $(n+1)$-stellige primitiv rekursive Funktion $(1 \leq i, j \leq n)$.

2. $f(x_1, \ldots, x_n) = g(x_{\pi(1)}, \ldots, x_{\pi(n)})$,
   wobei g n-stellige primitiv rekursive Funktion und π Permutation von $(1, \ldots n)$

3. $f(x_1, \ldots, x_n, y_1 \ldots, y_m) = g(x_1, \ldots, x_i, h(y_1, \ldots, y_m), x_{i+1}, \ldots, x_n)$,
   wobei g $(n+1)$-stellige und h m-stellige primitiv rekursive Funktion $(1 \leq i \leq n)$

Dann ist f primitiv rekursiv.

Beweis: Übung

Definition 12.3 (Primitiv rekursive Prädikate)

Ein Prädikat P heißt primitiv rekursiv, falls seine charakteristische Funktion

$\chi_P(\vec{x}) = 1$ falls $P(\vec{x})$

$\chi_P(\vec{x}) = 0$ falls nicht $P(\vec{x})$

primitiv rekursiv ist.

Beispiel:

$\chi_<(x,y) := \mathrm{sg}(y \dot{-} x)$ ist die charakteristische Funktion des 2-stelligen primitiv rekursiven Prädikats $x < y$.

$\chi_=(x,y) := \delta(x,y)$ ist die charakteristische Funktion des 2-stelligen primitiv rekursiven Prädikats $x = y$.

Lemma 12.4

Mit $P(\vec{x})$ und $Q(\vec{x})$ sind auch $P(\vec{x}) \wedge Q(\vec{x}),\ P(\vec{x}) \vee Q(\vec{x}),\ \neg P(\vec{x})$ primitiv rekursiv.
Mit $P(\vec{x}, y)$ sind auch $(\exists y \leq z)\ P(\vec{x}, y)$ und $(\forall y \leq z)\ P(\vec{x}, y)$ primitiv rekursiv.

Beweis:

$\chi_{P \wedge Q} = \chi_P \cdot \chi_Q;\ \chi_{P \vee Q} = \mathrm{sg}(\chi_P + \chi_Q);\ \chi_{\neg P} = \overline{\mathrm{sg}}(\chi_P)$

$\chi_{(\forall y \leq z)\ P(\vec{x},y)} = \prod_{y=0}^{z} \chi_P(\vec{x}, y)$

$\chi_{(\exists y \leq z)\ P(\vec{x},y)} = \mathrm{sg}(\sum_{y=0}^{z} \chi_P(\vec{x}, y))$

(Die Notation ist hier etwas lax, da wir P nicht von $P(\vec{x})$ unterscheiden und auch logische Operationen für Prädikate nicht definiert haben.)

Beispiele:

$x \leq y :$	$\Leftrightarrow$	$(\exists z \leq y)\ x + z = y$
$x \mid y :$	$\Leftrightarrow$	$(\exists z \leq y)\ x \cdot z = y$
$\mathrm{Prim}(x) :$	$\Leftrightarrow$	$x \neq 0 \wedge x \neq 1 \wedge (\forall z \leq x)\ (z = 0 \vee z = 1 \vee z = x \vee \neg z \mid x)$

Lemma 12.5 (Fallunterscheidung)

Mit $g_1, \ldots, g_n, P_1, \ldots, P_n$ ist auch f mit $f(\vec{x}) := \begin{cases} g_1(\vec{x}) & \text{falls } P_1(\vec{x}) \\ \vdots & \vdots \\ g_n(\vec{x}) & \text{falls } P_n(\vec{x}) \end{cases}$
primitiv rekursiv, vorausgesetzt, auf jedes $\vec{x}$ trifft genau eines der P_i zu.

Beweis:
$f(\vec{x}) = g_1(\vec{x}) \cdot \chi_{P_1}(\vec{x}) + \ldots + g_n(\vec{x}) \cdot \chi_{P_n}(\vec{x})$

Korollar 12.6 (Fallunterscheidung, Endlichkeit)

1. Mit $g_1, \ldots, g_n, g, P_1, \ldots, P_n$ ist auch f mit $f(\vec{x}) = \begin{cases} g_1(\vec{x}) & \text{falls } P_1(\vec{x}) \\ \vdots & \vdots \\ g_n(\vec{x}) & \text{falls } P_n(\vec{x}) \\ g(\vec{x}) & \text{sonst} \end{cases}$
   primitiv rekursiv, vorausgesetzt, die P_i schließen sich aus.
2. P treffe nur auf endlich viele $\vec{x}$ zu. Dann ist P primitiv rekursiv.

Lemma 12.7

1. $\mathrm{Max}(\vec{x})$ und $\mathrm{Min}(\vec{x})$ sind primitiv rekursiv (für feste Länge von $\vec{x}$).
2. Sei f primitiv rekursiv. Dann sind auch $\operatorname*{Max}_{y=0}^{z} f(\vec{x}, y)$ und $\operatorname*{Min}_{y=0}^{z} f(\vec{x}, y)$ primitiv rekursiv.

Beweis: Übung

Definition 12.8 (μ-Operator)

Für ein Prädikat P sei definiert:
$\mu y P(\vec{x}, y) :=$ das kleinste y, so daß $P(\vec{x}, y)$ gilt
(„unbeschränkter μ-Operator“, „Minimalisierung“)
$(\mu y \leq z)\ P(\vec{x}, y) := \begin{cases} \text{das kleinste } y \text{ mit } 0 \leq y \leq z, \text{ so daß } P(\vec{x}, y) \text{ gilt,} \\ \qquad \text{falls es ein solches } y \text{ gibt} \\ 0 \text{ sonst.} \end{cases}$
(„beschränkter μ-Operator“)

Bemerkung:

Der unbeschränkte μ-Operator erzeugt eine partielle Funktion, da es kein y der geforderten Art geben muß. Der beschränkte μ-Operator erzeugt in jedem Fall eine totale Funktion.

Lemma 12.9

Sei P primitiv rekursiv. Dann ist auch f mit $f(\vec{x}, z) = (\mu y \leq z)\ P(\vec{x}, y)$ primitiv rekursiv.

Beweis: Setze

$$h(\vec{x}, z, u) := \begin{cases} u & \text{falls } (\exists y \leq z)\ P(\vec{x}, y) \\ z' & \text{falls } P(\vec{x}, z') \text{ und } \neg(\exists y \leq z)\ P(\vec{x}, y) \\ 0 & \text{sonst} \end{cases}$$

h ist primitiv rekursiv.

Es gilt:
$f(\vec{x}, 0) = 0 \quad f(\vec{x}, z') = h(\vec{x}, z, f(\vec{x}, z))$

Beispiele:

$\dfrac{x}{y} := (\mu z \leq x)(y \cdot z' > x)$

$p_i : \quad p_0 := 2,\ p_{n'} := (\mu y \leq (p_n! + 1))(\mathrm{Prim}(y) \wedge p_n < y)$
$p_i :\ i-$te Primzahl (Zwischen p und $p! + 1$ liegt immer eine Primzahl)
$p_0 = 2, p_1 = 3, p_2 = 5, p_3 = 7, \cdots$

$\exp(n, x) := (\mu z \leq x)\neg(p_n^{z+1} | x)$, auch als $(x)_n$ geschrieben
„Exponent von p_n in der Primzahlzerlegung von x“
$x = 2^{a_0} \cdot 3^{a_1} \cdot 5^{a_2} \cdot 7^{a_3} \cdot \ldots$ a_3 ist $(x)_3$, d.h. Exponent der dritten Primzahl (7)

$$l(x) = \begin{cases} (\mu z \leq x)(\forall w \leq x)\neg(w > z \wedge p_w | x) & \text{falls } x > 0 \\ 0 & \text{sonst} \end{cases}$$

„Index der größten in x enthaltenen Primzahl“

Kodierung von Paaren

Es wird häufig notwendig sein, Paare natürlicher Zahlen *bijektiv* durch einzelne Zahlen zu kodieren, und zwar derart, daß sowohl die Kodierungsfunktion σ_2 als auch die Dekodierungsfunktionen σ_{21} und σ_{22} primitiv rekursiv sind. Dazu benutzen wir folgende Eigenschaft:
Jede Zahl $z > 0$ läßt sich eindeutig schreiben als $z = 2^x \cdot (2y + 1)$. Jedes $z \geq 0$ läßt sich damit eindeutig als $2^x(2y + 1) \dot{-} 1$ (primitiv rekursive Funktion) charakterisieren.

$$\begin{array}{lcl}
\sigma_2(x,y) & = & 2^x(2y+1) \dot{-} 1 \\
\sigma_{21}(z) & = & \exp(0,z+1) \\
\sigma_{22}(z) & = & \dfrac{\dfrac{z+1}{2^{\exp(0,z+1)}} \dot{-} 1}{2}
\end{array}$$

σ_2 ist primitiv rekursive Bijektion zwischen $\mathbb{N} \times \mathbb{N}$ und $\mathbb{N}$ mit primitiv rekursiver Umkehrfunktion $(\sigma_{21}, \sigma_{22})$. Es gilt:

$$\begin{array}{lcl}
\sigma_{21}(\sigma_2(x,y)) & = & x \\
\sigma_{22}(\sigma_2(x,y)) & = & y \\
\sigma_2(\sigma_{21}(z), \sigma_{22}(z)) & = & z
\end{array}$$

Entsprechend kann man für Tripel definieren:

$$\begin{array}{lcl}
\sigma_3(x,y,z) & = & \sigma_2(\sigma_2(x,y),z) \\
\sigma_{31}(u) & = & \sigma_{21}(\sigma_{21}(u)) \\
\sigma_{32}(u) & = & \sigma_{22}(\sigma_{21}(u)) \\
\sigma_{33}(u) & = & \sigma_{22}(u) \\
u.s.w.
\end{array}$$

Lemma 12.10 (Wertverlaufsrekursion)

Seien $g : \mathbb{N}^n \to \mathbb{N}$ und $h : \mathbb{N}^{n+m+1} \to \mathbb{N}$ und $r_1, \ldots, r_m : \mathbb{N}^n \times \mathbb{N} \to \mathbb{N}$ so, daß für alle $\vec{x}, y \in \mathbb{N}$ jeweils $r_i(\vec{x}, y) \leq y$ gilt. Eine Funktion $f : \mathbb{N}^n \times \mathbb{N} \to \mathbb{N}$ werde durch *Wertverlaufsrekursion* definiert, d.h. durch das Schema

$f(\vec{x}, 0) = g(\vec{x})$
$f(\vec{x}, y') = h(\vec{x}, y, f(\vec{x}, r_1(\vec{x}, y)), \ldots, f(\vec{x}, r_m(\vec{x}, y)))$

Es gilt: f ist primitiv-rekursiv, falls $g, h, r_1, \ldots, r_m$ primitiv rekursiv sind.

Beweis: Aufgabe

Lemma 12.11 (Ackermann-Funktion)

Die Ackermann-Funktion $A(x,y)$, die definiert ist durch:

$A(0,y) = y'$
$A(x',0) = A(x,1)$
$A(x',y') = A(x, A(x',y))$

hat folgende Eigenschaften:

1. Für jedes x ist $f_x(y) := A(x,y)$ eine einstellige primitiv rekursive Funktion.
2. A ist nicht primitiv rekursiv.

Beweis:

1. Indukion über x:
   I.A: $f_0(y) := A(0, y)$ ist primitiv rekursiv: $f_0(y) = y'$
   I.S: Es gilt: $f_{n+1}(0) = f_n(1)$; $f_{n+1}(y') = f_n(f_{n+1}(y))$.
   Da f_n primitiv rekursiv, ist damit auch f_{n+1} primitiv rekursiv.

2. Man kann zeigen: Zu jeder einstelligen, primitiv rekursiven Funktion g gibt es ein c, so daß gilt: $g(x) < A(c, x)$ für alle x. D.h., die Ackermann-Funktion wächst schneller als jede primitiv rekursive Funktion. Daraus: $g(c) < A(c, c)$. Wäre A primitiv rekursiv, dann auch $g_A(x) := A(x, x)$. Dann würde gelten: $g_A(c) < A(c, c) = g_A(c)$ für ein gewisses c. Widerspruch. (Ausführlicher Beweis: siehe Hermes (1978))

Bemerkungen:

1. Aus der Schreibweise $f_{n+1}(y') = f_n(f_{n+1}(y))$ ergibt sich:
   $f_{n+1}(y) = \underbrace{f_n(f_n(\ldots f_n}_{(y+1)-fach}(1)\ldots))$.
   Das macht intuitiv das schnelle Wachstum der Ackermann-Funktion klar.

2. Die Ackermann-Funktion ist offensichtlich berechenbar.

3. Die Ackermann-Funktion in der angegebenen Form (wie sie auch in in der Literatur üblich ist) ist nicht die ursprünglich von Ackermann angegebene Funktion, hat jedoch im wesentlichen dieselben Eigenschaften wie die ursprüngliche Funktion.

Definition 12.12 (Partiell rekursive Funktionen und Prädikate)

Die partiell rekursiven Funktionen sind wie folgt definiert:

Die Ausgangsfunktionen C_0^0, N, U_n^i sind partiell rekursiv.

$f(\overrightarrow{x}) := g(h_1(\overrightarrow{x}), \ldots, h_r(\overrightarrow{x}))$ ist partiell rekursiv, falls g und h partiell rekursiv, wobei die Gleichheit wie folgt zu verstehen ist:
$f(\overrightarrow{x})$ ist genau dann definiert, wenn alle $h_i(\overrightarrow{x})$ definiert sind und wenn $g(h_1(\overrightarrow{x}), \ldots, h_r(\overrightarrow{x}))$ definiert ist.

$f(\overrightarrow{x}, 0) = g(\overrightarrow{x})$, $f(\overrightarrow{x}, y') = h(\overrightarrow{x}, y, f(\overrightarrow{x}, y))$ definiert eine partiell rekursive Funktion f, falls g und h partiell rekursiv sind, wobei die Gleichungen so zu verstehen sind, daß $f(\overrightarrow{x}, 0)$ genau dann definiert ist, falls $g(\overrightarrow{x})$ definiert ist und $f(\overrightarrow{x}, y')$ genau dann definiert ist, falls $f(\overrightarrow{x}, y)$ und $h(\overrightarrow{x}, y, f(\overrightarrow{x}, y))$ definiert sind.

$f(\vec{x}) = (\mu y)(g(\vec{x}, y) = 0)$ definiert eine partiell rekursive Funktion f, falls g partiell rekursiv und die Gleichung wie folgt zu verstehen ist:

$$f(\vec{x}) = w \text{ g.d.w.} \begin{cases} g(\vec{x}, y) \text{ ist definiert für alle } y \leq w \\ g(\vec{x}, w) = 0 \\ g(\vec{x}, y) > 0 \text{ für alle } y < w \end{cases}$$

Eine totale partiell rekursive Funktion heißt rekursiv oder μ-rekursiv.
Ein Prädikat P heißt partiell rekursiv, falls seine charakteristische Funktion

$\chi_P(\vec{x}) = 1$ falls $P(\vec{x})$
$\chi_P(\vec{x}) = 0$ falls $P(\vec{x})$ definiert ist, aber nicht gilt
$\chi_P(\vec{x})$ undefiniert falls $P(\vec{x})$ undefiniert

partiell rekursiv ist.

Bemerkung:

Statt $(\mu y)(g(\vec{x}, y) = 0)$ kann man auch $(\mu y)P(\vec{x}, y)$ für ein partiell rekursives Prädikat P verlangen, da dann g als $\overline{\text{sg}}(\chi_P)$ definiert werden kann.

Beispiele:
$\perp_0 := (\mu y)(C_1^1(y) = 0)$
$\perp_1 (x) := (\mu y)(C_2^1(x, y) = 0)$
$\perp_n (x_1, \ldots, x_n) := (\mu y)(C_{n+1}^1(x_1, \ldots, x_n, y) = 0)$
(Die überall undefinierte n-stellige Funktion)

$\frac{x}{y} \cdot := (\mu z)(\varepsilon(y \cdot z, x) = 0)$. $\frac{3}{2} \cdot$ ist undefiniert; $\frac{x}{0} \cdot$ ist undefiniert für $x > 0$; $\frac{0}{0} \cdot = 0$

Lemma 12.13

Die Ackermann-Funktion ist partiell rekursiv (sogar rekursiv, da überall definiert).

Beweisidee:
Man betrachtet die Folge der Terme, die sich ergibt, wenn man $A(x, y)$ sukzessiv in einer bestimmten Ordnung auswertet, z.B.
$A(1, 1),\ A(0, A(1, 0)),\ A(0, A(0, 1)),\ A(0, 2),\ 3.$
Wenn man diese Terme in geeigneter Weise kodiert, erhält man eine primitiv rekursive Funktion
$f(n, x, y) =$ Kodenummer des n-ten Terms in der Auswertungsfolge von $A(x, y)$
(wobei $f(m, x, y) = f(n, x, y)$ für $m > n$, falls die Auswertungsfolge nach n Schritten terminiert).

Beispieltabelle:

Schritt n	Berechnung von $A(1,1)$	abgekürzt	Kodierung	Wert $f(n,1,1)$
1	$A(1,1)$	$1,1$	$2^2 \times 3^2$	36
2	$A(0,A(1,0))$	$0,1,0$	$2^1 \times 3^2 \times 5^1$	90
3	$A(0,A(0,1))$	$0,0,1$	$2^1 \times 3^1 \times 5^2$	150
4	$A(0,2)$	$0,2$	$2^1 \times 3^3$	54
5	3	3	2^4	16
6	3	3	2^4	16
7	3	3	2^4	16
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Ferner läßt sich eine primitiv rekursive Funktion g wie folgt definieren:
$g(x) = \begin{cases} k & \text{falls } x \text{ Kodenummer der (einzelnen) Ziffer } k \\ 0 & \text{sonst} \end{cases}$
In der Primzahlkodierung der Beispieltabelle wäre k der um 1 erniedrigte Exponent von 2, falls k eine Potenz von 2 ist.
Im Beispiel wäre also $g(16) = g(f(5,1,1)) = 3$.
Allgemein gilt:
$A(x,y) = g((\mu z)(f(z,x,y) = f(z+1,x,y))$,
d.h. A ist partiell rekursiv. (Für einen detaillierten Beweis siehe Hermes (1978))
Die Grundidee, die auch bei der Kodierung von TM durch partiell rekursive Funktionen in Kapitel 13 benutzt wird, besteht darin, die Zwischenwerte der Berechnung einer Funktion in Abhängigkeit von der Schrittzahl ihrer Berechnung darzustellen.

13 Turing Maschinen und partiell rekursive Funktionen

Wir zeigen, daß Berechenbarkeit durch Turingmaschinen und Berechenbarkeit durch partiell rekursive Funktionen gleichwertig sind. Es handelt sich also um inhaltlich verschiedene Präzisierungen des (extensional) selben Begriffs.

Theorem 13.1

Alle partiell rekursiven Funktionen sind Turing-berechenbar.

Beweis:

(i) $C_0^0,\ N,\ U_n^i$ sind Turing-berechenbar (Aufgabe)

(ii) Komposition (Aufgabe)

(iii) Primitive Rekursion:

$$\begin{array}{lcl} f(\vec{x},0) & = & g(\vec{x}) \\ f(\vec{x},y') & = & h(\vec{x},y,f(\vec{x},y)) \end{array}$$

Seien TM $\mathcal{M}_g$ und $\mathcal{M}_h$ gegeben, die g und h berechnen. $\overline{m}$ stehe für $\overbrace{*\ldots*}^{m+1}$; $\vec{x}$ stehe für $\overline{x_1}\#\ldots\#\overline{x_n}$. Sei $\overline{\mathcal{M}_h}$ eine TM, welche die Argumente von $\mathcal{M}_h$ in anderer Reihenfolge benutzt, so daß folgendes gilt:
$(\underline{\#}\overline{z}\#\overline{y}\#\vec{x}) \underset{\overline{\mathcal{M}_h}}{\Longrightarrow} (\underline{\#}\overline{v})$ g.d.w. $(\underline{\#}\vec{x}\#\overline{y}\#\overline{z}) \underset{\mathcal{M}_h}{\Longrightarrow} (\underline{\#}\overline{v})$.
Seien $\mathcal{M}_1$-$\mathcal{M}_4$ folgende 2-Band-TM:

$$\begin{pmatrix} \underline{\#}\vec{x}\#\overline{y} \\ \underline{\#} \end{pmatrix} \underset{\mathcal{M}_1}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{x}\#\overline{0}\#\vec{x}\#\overline{1}\#\ldots\#\vec{x}\#(\overline{y-1})\#\vec{x} \\ \underline{\#} \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#}\vec{x}\#\vec{w} \\ \underline{\#} \end{pmatrix} \underset{\mathcal{M}_2}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{w} \\ \underline{\#}\vec{x} \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#}\overline{y}\#\vec{x}\#\vec{w} \\ \underline{\#}\overline{z} \end{pmatrix} \underset{\mathcal{M}_3}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{w} \\ \underline{\#}\overline{z}\#\overline{y}\#\vec{x} \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#} \\ \underline{\#}\overline{z} \end{pmatrix} \underset{\mathcal{M}_4}{\Longrightarrow} \begin{pmatrix} \underline{\#}\overline{z} \\ \underline{\#} \end{pmatrix}$$

Beachte, daß wir die Länge von $\vec{x}$ (als n-Tupel) kennen, d.h. wissen, wo $\vec{x}$ aufhört und wo $\vec{w}$ anfängt.

Im Grenzfall (y=0) produziert die Maschine $\mathcal{M}_1$ die Konfiguration $\begin{pmatrix} \underline{\#}\vec{x} \\ \underline{\#} \end{pmatrix}$, so daß $\vec{w}$ leer sein kann (d.h. $\vec{w}$ ist dann das leere n-Tupel, nicht etwa ein blank $\#$).

$$\mathcal{M}_f := \mathcal{M}_1\mathcal{M}_2\mathcal{M}_g^{(2)}R^{(1)} \begin{array}{l} \xrightarrow{\#^{(1)}} L^{(1)}\mathcal{M}_4 \\ \xrightarrow{\neq \#^{(1)}} L^{(1)}\mathcal{M}_3\overline{\mathcal{M}}_h^{(2)} \end{array}$$

(iv) Minimalisierung: $f(\vec{x}) := (\mu y)(g(\vec{x}, y) = 0)$
Sei $\mathcal{M}_g$ die TM, die $g(\vec{x}, y)$ berechnet.
Seien 2-Band-TM $\mathcal{M}_0$-$\mathcal{M}_3$ wie folgt gegeben:

$$\begin{pmatrix} \underline{\#}\vec{x} \\ \underline{\#} \end{pmatrix} \underset{\mathcal{M}_0}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{x}\#\overline{0} \\ \underline{\#} \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#}\vec{x}\#\overline{y} \\ \underline{\#}w \end{pmatrix} \underset{\mathcal{M}_1}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{x}\#\overline{y} \\ \underline{\#}\vec{x}\#\overline{y} \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#}\vec{x}\#\overline{y} \\ \underline{\#}w \end{pmatrix} \underset{\mathcal{M}_2}{\Longrightarrow} \begin{pmatrix} \underline{\#}\vec{x}\#\overline{y+1} \\ \underline{\#}w \end{pmatrix}$$

$$\begin{pmatrix} \underline{\#}\vec{x}\#\overline{y} \\ \underline{\#}\overline{0} \end{pmatrix} \underset{\mathcal{M}_3}{\Longrightarrow} \begin{pmatrix} \underline{\#}\overline{y} \\ \underline{\#} \end{pmatrix}$$

$$\mathcal{M}_f := \mathcal{M}_0\mathcal{M}_1\mathcal{M}_g^{(2)}R^{(2)}R^{(2)} \begin{array}{l} \xrightarrow{\#^{(2)}} L^{(2)}L^{(2)}\mathcal{M}_3 \\ \xrightarrow{\neq \#^{(2)}} L^{(2)}L^{(2)}\mathcal{M}_2 \end{array}$$

Es wird heraufgezählt, bis $g(\vec{x}, y)$ den Wert 0 hat (falls ein solcher Wert existiert und g für alle früheren Werte von y definiert ist). Ansonsten terminiert die Berechnung nicht.

Der folgende Beweis der Umkehrung von Theorem 13.1 ist etwas aufwendiger.

Definition 13.2 (Kodierung)

Die Kodierung von Zeichen und Zeichenfolgen durch Zahlen und von Prädikaten und Funktionen über Zahlen heißt Gödelisierung. Die den Zeichen zugeordneten Zahlen heißen auch Gödelnummern.

Gödelisierung von TM

Nummerierung der Felder des Bandes

... 11 9 7 5 3 1 0 2 4 6 8 10 12 ...

Primitiv rekursive Funktionen und Prädikate:
$G(x) :\Leftrightarrow 2|x$ (x ist gerade)

$U(x) :\Leftrightarrow \neg 2|x$ (x ist ungerade)

$$R(x) := \begin{cases} x+2 & \text{falls } G(x) \\ 0 & \text{falls } x = 1 \\ x \dot{-} 2 & U(x) \wedge x \neq 1 \end{cases} \quad \text{„Feld unmittelbar rechts von } x\text{“}$$

$L(x)$ analog „Feld unmittelbar links von x“

$$R(x,y) :\Leftrightarrow \begin{cases} G(x) \wedge x > y & \text{falls } G(y) \\ (U(x) \wedge x < y) \vee G(x) & \text{falls } U(y) \end{cases} \quad \text{„}x\text{ liegt irgendwo rechts von y“}$$

$L(x,y)$ analog

$$Z(x,y) := \begin{cases} \frac{|x-y|}{2} & \text{falls } G(x) \wedge G(y) \\ \frac{|x-y|}{2} & \text{falls } U(x) \wedge U(y) \\ \frac{(x+y)+1}{2} & \text{sonst} \end{cases}$$

„Anzahl der Felder zwischen x und y, wobei x, jedoch nicht y mitgerechnet wird“

Kodierung des Bandinhalts

Sei $\Sigma_{\#} = \{a_0, a_1, \ldots, a_m\}$ wobei $a_0 = \#$.
Sei β: Feldnummer $\to$ Zeichennummer
beliebige Füllung des Bandes mit Zeichen, d.h. auf Feld i stehe das Zeichen $a_{\beta(i)}$.
Der Bandinhalt b ist dann wie folgt kodiert:
$b := \prod_{j=0}^{\infty} p_j^{\beta(j)}$ (da $a_0 = \#$, sind nur endlich viele Faktoren $\neq 1$)

Beispiel:

... #	a_1	a_7	#	a_2	#	a_3	# ...	Inhalt: $2^2 \cdot 7^7 \cdot 11^3 \cdot 13^1$
7	5	3	1	0	2	4	6	

Primitiv rekursive Funktionen und Prädikate:
$B(x) :\Leftrightarrow (\forall y \leq l(x))((x)_y \leq m)$ „x ist Gödelnummer eines Bandinhaltes“
(m war maximale Nummer eines Zeichens)
Sei Konfiguration $(\underline{\#}w)$ ($w \in \Sigma^*$) gegeben mit Bandinhalt b, wobei der Lesekopf auf Feld Nr. n steht.
$W_0(n,b) := |w| - 1$.
Ist $(\underline{\#}w)$ eine Haltekonfiguration für eine zahlentheoretische Funktion, dann ist $W_0(n,b)$

deren Wert (um 1 verringerte Länge des Wortes w, das nur aus Sternen $*$ besteht).
Wir zeigen: W_0 ist primitiv rekursiv:
$E_l(n,b) := R(n)$ („linkestes Feld von w“)
$E_r(n,b) := (\mu x \leq b)[(b)_{R(x)} = 0 \wedge (\forall y \leq \mathrm{Max}(R(x),n))(L(y,R(x)) \wedge R(y,n) \rightarrow (b)_y > 0)]$
(„rechtestes Feld von w“; $A \rightarrow B$ steht dabei für $\neg A \vee B$)
$W_0(n,b) = Z(E_l(n,b), E_r(n,b))$

Kodierung der Maschinentafel

Gegeben sei eine Maschinentafel im Sinne der Bemerkung zu Def. 10.1.
Wir haben $m+1$ Zeichen $a_0, a_1, \ldots, a_m$ in $\Sigma_{\#}$(mit $a_0 = \#$)
$n+1$ Zustände $q_0, q_1, \ldots, q_n$
Wir führen ein spezielles Zeichen h ein, das in der 3. Spalte der Maschinentafel steht, falls $\delta(q,a)$ nicht definiert ist. In der 4. Spalte steht dann ein beliebiges q. Die Symbole der Maschinentafel werden wie folgt kodiert:

$q_0 \ldots q_n$	$a_0 \ldots a_m$	R	L	h
$0 \ \ldots \ n$	$0 \ \ldots \ m$	$m+1$	$m+2$	$m+3$

Die Maschinentafel der gegebenen TM ist dann eine $(((m+1)(n+1)) \times 4)$-Matrix A_{ij}, die wie folgt kodiert wird:

$$t = p_0^m \cdot p_1^n \cdot \prod_{i=1}^{(m+1)(n+1)} (p_{\sigma_2(i,3)}^{A_{i3}} \cdot p_{\sigma_2(i,4)}^{A_{i4}})$$

Da die ersten beiden Spalten der Maschinentafel durch m und n festgelegt sind, müssen wir dafür nur die Zahlen m und n kodieren.

Sei h nun die folgende 4-stellige primitiv rekursive Funktion:

$h(p,r,c,t) := (t)_{\sigma_2(((t)_0+1)\cdot c+r+1,p)}$

$(t)_0 + 1$ ist dabei die Anzahl der zur Verfügung stehenden Zeichen, c ist die Zustandsnummer, r die Nummer des gelesenen Zeichens.
(Wenn man 8 Zeichen und 5 Zustände hat und im Zustand q_2 das Zeichen a_3 ausgewertet wird, ist $((t)_0+1)\cdot c+r+1 = 8\cdot 2+4 = 20$, was der 20. Zeile der Maschinentafel entspricht.)

Jede Zeile der Maschinentafel läßt sich dann schreiben als:

$q_c \quad a_r \quad a_{h(3,r,c,t)} \quad q_{h(4,r,c,t)}$

D.h. h dekodiert die Maschinentafel in der folgenden Weise:
$h(3,r,c,t) :=$ Auszuführende Operation bei Lektüre von a_r im Zustand q_c
$h(4,r,c,t) :=$ Zielzustand bei Lektüre von a_r im Zustand q_c

Damit können wir ein Prädikat definieren, das besagt, daß t Gödelnummer einer TM ist:

$M(t) :\Leftrightarrow (t)_0 \geq 1 \wedge (\forall r \leq (t)_0)(\forall c \leq (t)_1)\ [(h(3,r,c,t) \leq (t)_0 + 3) \wedge (h(4,r,c,t) \leq (t)_1)]$

$(t)_0 \geq 1$ soll gelten, da das Alphabet der TM außer dem blank $\#$ mindestens ein Zeichen enthalten muß.

Folgekonfigurationen

Sei t Kodierung der TM, a Kodierung des Arbeitsfelds, b Kodierung der Bandinschrift, c Kodierung des Zustands

Wir definieren: $A(t,a,b,c)$= „nächstes Arbeitsfeld“
$B(t,a,b,c)$ = „nächste Bandinschrift“
$C(t,a,b,c)$ = „nächster Zustand“
$E_0(t,a,b,c) \Leftrightarrow$ „Es liegt Haltekonfiguration vor“

Formell:

$$E_0(t,a,b,c) :\Leftrightarrow h(3,(b)_a,c,t) = m+3$$

$$A(t,a,b,c) := \begin{cases} R(a) & \text{falls } h(3,(b)_a,c,t) = m+1 \\ L(a) & \text{falls } h(3,(b)_a,c,t) = m+2 \\ a & \text{sonst} \end{cases}$$

$B(t,a,b,c)$ und $C(t,a,b,c)$ als Übung

$K(x) :\Leftrightarrow (\exists a,b,c \leq x)(B(b) \wedge c \leq n \wedge x = \sigma_3(a,b,c))$
„x ist Kodierung einer Konfiguration“ (d.h. eines Tripels, bestehend aus Arbeitsfeld, Bandinschrift und Zustand)

$E(t,k) :\Leftrightarrow E_0(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k))$.
„k ist Kodierung einer Haltekonfiguration einer TM mit Nummer t“

$$\begin{aligned} F(t,k) := \ & \sigma_3(A(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k)), \\ & \quad B(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k)) \\ & \quad C(t,\sigma_{31}(k),\sigma_{32}(k),\sigma_{33}(k)). \end{aligned}$$

„Kodierung der Folgekonfiguration zur Konfiguration mit Nummer k“

$W(k) := W_0(\sigma_{31}(k),\sigma_{32}(k))$
„Zahl n, falls k Gödelnummer einer Konfiguration $(q,\underline{\#}\underbrace{*\ldots*}_{n+1})$“

Kodierung der Auswertungsschritte

$B_0(\vec{x})$= Nummer der Anfangskonfiguration $(q_0,\underline{\#}\vec{x})$ (Übung)

$$K(t,\vec{x},0) := B_0(\vec{x})$$
$$K(t,\vec{x},z') := \begin{cases} K(t,\vec{x},z) & \text{falls } E(t,K(t,\vec{x},z)) \\ F(t,K(t,\vec{x},z)) & \text{sonst} \end{cases}$$

Gödelnummer der z-ten Konfiguration bei Auswertung der TM mit Nummer t für die Anfangskonfigurtaion $(q_0,\underline{\#}\vec{x})$. Falls z-te Konfiguration Haltekonfiguration, soll k für die folgenden Konfigurationen konstant bleiben.

Achtung: Falls $\vec{x} = x_1,\ldots,x_n$, steht „$B_0(\vec{x})$“ für „$B_0(x_1,\ldots,x_n)$“,
jedoch „$(q_0,\underline{\#}\vec{x})$“ für „$(q_0,\underline{\#}\overline{x_1}\#\ldots\#\overline{x_n})$“.

Theorem 13.3

Jede Turing-berechenbare Funktion f ist partiell rekursiv.

Beweis: Sei t die Gödelnummer der TM $\mathcal{M}$, die f berechnet. Dann gilt, falls $\mathcal{M}$ für die Argumente $\overrightarrow{x}$ terminiert:
$f(\overrightarrow{x}) = W[K(t, \overrightarrow{x}, (\mu z)E(t, K(t, \overrightarrow{x}, z)))]$
Andernfalls ist $f(\overrightarrow{x})$ undefiniert.

Definition 13.4 (Kleenesches T-Prädikat)

Sei n die Länge von $\overrightarrow{x}$. Für jedes solches n sei ein primitiv rekursives Prädikat T_n wie folgt definiert:
$T_n(t, \overrightarrow{x}, y) :\Leftrightarrow E(t, K(t, \overrightarrow{x}, \sigma_{21}(y))) \wedge \sigma_{22}(y) = K(t, \overrightarrow{x}, \sigma_{21}(y))$.
„Spätestens nach $\sigma_{21}(y)$ Schritten hält die Maschine mit Nummer t, wobei $\sigma_{22}(y)$ die Nummer der Haltekonfiguration ist“.
T_n heißt auch Kleenesches T-Prädikat.

Lemma 13.5

(i) Falls $(\mu z)E(t, K(t, \overrightarrow{x}, z))$ definiert ist, dann ist $(\mu y)T_n(t, \overrightarrow{x}, y)$ definiert und es ist $(\mu z)E(t, K(t, \overrightarrow{x}, z)) = \sigma_{21}((\mu y)T_n(t, \overrightarrow{x}, y))$.

(ii) Falls $(\mu y)T_n(t, \overrightarrow{x}, y)$ definiert ist, dann ist $(\mu z)E(t, K(t, \overrightarrow{x}, z))$ definiert und es ist $K(t, \overrightarrow{x}, (\mu z)E(t, K(t, \overrightarrow{x}, z))) = \sigma_{22}((\mu y)T_n(t, \overrightarrow{x}, y))$.

Beweis:

(i) Sei $w := (\mu z)E(t, K(t, \overrightarrow{x}, z))$, sei $u := \sigma_2(w, K(t, \overrightarrow{x}, w))$. Dann ist $w = \sigma_{21}(u)$.
Also gilt: $E(t, K(t, \overrightarrow{x}, \sigma_{21}(u)))$. Ferner gilt: $\sigma_{22}(u) = K(t, \overrightarrow{x}, \sigma_{21}(u))$.
Also gilt $T_n(t, \overrightarrow{x}, u)$ (Definition von T_n).
Sei $u' = (\mu y)T_n(t, \overrightarrow{x}, y)$.
Dann gilt: $E(t, K(t, \overrightarrow{x}, \sigma_{21}(u')))$ und $\sigma_{22}(u') = K(t, \overrightarrow{x}, \sigma_{21}(u'))$.
Es gilt: $\sigma_{21}(u') \geq \sigma_{21}(u) = w$ nach Wahl von w (w war minimal gewählt).
Damit wissen wir: $\sigma_{22}(u') = K(t, \overrightarrow{x}, \sigma_{21}(u')) = K(t, \overrightarrow{x}, w) = \sigma_{22}(u)$.
Zusammen: $\sigma_{21}(u') \geq \sigma_{21}(u)$; $\sigma_{22}(u') = \sigma_{22}(u)$.
Daraus folgt: $u' \geq u$. Damit: $u' = u$ (wegen Minimalität).

(ii) Sei $w = (\mu y)T_n(t, \vec{x}, y)$.
Dann gilt $E(t, \vec{x}, K(t, \vec{x}, \sigma_{21}(w)))$ und $\sigma_{22}(w) = K(t, \vec{x}, \sigma_{21}(w))$. Dann gilt nach (i) durch Einsetzen von $\sigma_{21}(w)$: $\sigma_{22}(w) = K(t, \vec{x}, (\mu z)E(t, K(t, \vec{x}, z)))$.

Wir definieren noch eine primitiv rekursive Funktion, die wir für das folgende Theorem benötigen: $U(x) := W(\sigma_{22}(x))$

Ferner führen wir eine Notation ein:
$g(\vec{x}) \simeq f(\vec{x})$ („vollständig gleich") heißt:
Für jedes $\vec{x}$: $g(\vec{x})$ ist definiert g.d.w. $f(\vec{x})$ ist definiert,
wobei im definierten Fall gilt: $f(\vec{x}) = g(\vec{x})$.
Analog: $P(\vec{x}) \overset{\sim}{\leftrightarrow} Q(\vec{x})$ („vollständig äquivalent")

Theorem 13.6 (Kleenesche Normalform)

Zu jeder partiell rekursiven Funktion f läßt sich eine Zahl t angeben, so daß gilt:
$f(\vec{x}) \simeq U((\mu y)T_n(t, \vec{x}, y))$.

Beweis: Sei t die Gödelnummer der TM, die f berechnet. Falls $\mathcal{M}$ für die Argumente $\vec{x}$ terminiert, gilt:
$f(\vec{x}) = W[K(t, \vec{x}, (\mu z)E(t, K(t, \vec{x}, z))]$. Mit Lemma 13.5 (ii) ergibt sich dann:
$f(\vec{x}) = W[\sigma_{22}((\mu y)T_n(t, \vec{x}, y))] = U((\mu y)T_n(t, \vec{x}, y))$

Bemerkung:

Dieses Theorem besagt insbesondere, daß sich jede partiell rekursive Funktion unter Verwendung von *höchstens einem* unbeschränkten μ-Operator definieren läßt.

14 LOOP- und WHILE-Programme

Als weitere Charakterisierung des Berechenbarkeitsbegriffs führen wir LOOP- und WHILE-Programme ein. Primitiv bzw. partiell rekursive Funktionen beschreiben die denotationale Semantik solcher Programme.
Literatur: Engeler & Läuchli (1992), Schöning (2001)

Definition 14.1 (LOOP-Programme)

LOOP_n $(n > 0)$ ist die Menge der Programme, die durch folgende Syntax erklärt ist:

$$
\begin{array}{rcl}
\langle\text{Variable}\rangle & ::= & x_1|\dots|x_n \\
\langle\text{Wertzuweisung}\rangle & ::= & \langle\text{Variable}\rangle := 0 \;\; | \\
 & & \langle\text{Variable}\rangle := \langle\text{Variable}\rangle \;\; | \\
 & & \langle\text{Variable}\rangle := N(\langle\text{Variable}\rangle) \;\; | \\
 & & \langle\text{Variable}\rangle := V(\langle\text{Variable}\rangle) \\
\langle\text{Anweisung}\rangle & ::= & \langle\text{Wertzuweisung}\rangle \;\; | \;\; \textbf{loop} \; \langle\text{Variable}\rangle \; \textbf{do} \; \langle\text{Programm}\rangle \; \textbf{end} \\
\langle\text{Programm}\rangle & ::= & \langle\text{Anweisung}\rangle \;\; | \\
 & & \langle\text{Programm}\rangle ; \langle\text{Programm}\rangle
\end{array}
$$

LOOP_n ist also eine kontextfreie Sprache über dem Terminalalphabet
$\{x_1, \dots, x_n, 0, N, V, :=, ;, (,), \textbf{loop}, \textbf{do}, \textbf{end}\}$
$\text{LOOP} := \bigcup\limits_{n \geq 1}^{\infty} \text{LOOP}_n$

Bemerkung:

Programme lassen sich *eindeutig* als Folgen von Anweisungen schreiben.

Definition 14.2 (Informelle Semantik von LOOP-Programmen)

$x_i := 0$	ordnet der Variablen x_i den Wert 0 zu
$x_i := x_j$	ordnet der Variablen x_i den Wert zu, den die Variable x_j hat
$x_i := N(x_j)$	ordnet der Variablen x_i den Wert $n + 1$ zu, wenn x_j den Wert n hat
$x_i := V(x_j)$	ordnet der Variablen x_i den Wert $n - 1$ zu, falls x_j den Wert $n > 0$ hat, und den Wert 0, falls x_j den Wert 0 hat

$P_1; P_2$	führt zuerst das Programm P_1 aus und dann das Programm P_2
loop x_i **do** P **end**	führt n mal hintereinander das Programm P aus, falls x_i den Wert n hat (der Wert von x_i *vor* Eintritt in die Schleife ist maßgeblich — eine eventuelle Veränderung dieses Wertes im Rahmen der Ausführung von P ist für die Anzahl der Durchläufe von P unerheblich).

Beispiele:

1. **loop** x_2 **do** $x_1 := N(x_1)$ **end**
   Wenn x_1 den Wert n und x_2 den Wert m hat, erhält x_1 den Wert $n + m$.

2. **loop** x_2 **do** $x_1 := V(x_1)$ **end**
   Wenn x_1 den Wert n und x_2 den Wert m hat, erhält x_1 den Wert $n \dot{-} m$.

3. $x_3 := 0$; **loop** x_2 **do loop** x_1 **do** $x_3 := N(x_3)$ **end end**; $x_1 := x_3$
   Wenn x_1 den Wert n und x_2 den Wert m hat, erhält x_1 den Wert $n \cdot m$.

4. $x_2 := 0$; **loop** x_1 **do** $x_2 := N(0)$ **end**; $x_1 := x_2$
   (genauer: statt $x_2 := N(0)$: $\quad x_2 := 0; x_2 := N(x_2)$)
   Wenn x_1 den Wert 0 hat, erhält x_1 den Wert 0, sonst erhält x_1 den Wert 1.

Definition 14.3 (Formale [denotationale] Semantik von LOOP - Programmen)

Jedem Programm $P \in \text{LOOP}_n$ wird eine Funktion $f^P : \mathbb{N}^n \to \mathbb{N}^n$ zugeordnet, die beschreibt, wie sich die Werte von Variablen aufgrund der Anwendung von P verändern. Wir beschreiben f^P durch seine Komponenten $f_l^P : \mathbb{N}^n \to \mathbb{N}$ $(1 \leq l \leq n)$. $\vec{k}$ stehe für $k_1, \ldots, k_n$.

$$f_l^{x_i:=0}(\vec{k}) = \begin{cases} 0 & \text{falls } l = i \\ k_l & \text{sonst} \end{cases}$$

$$f_l^{x_i:=x_j}(\vec{k}) = \begin{cases} k_j & \text{falls } l = i \\ k_l & \text{sonst} \end{cases}$$

$$f_l^{x_i:=N(x_j)}(\vec{k}) = \begin{cases} k_j + 1 & \text{falls } l = i \\ k_l & \text{sonst} \end{cases}$$

$$f_l^{x_i:=V(x_j)}(\vec{k}) = \begin{cases} k_j - 1 & \text{falls } l = i \text{ und } k_j > 0 \\ 0 & \text{falls } l = i \text{ und } k_j = 0 \\ k_l & \text{sonst} \end{cases}$$

$f_l^{C;P}(\overrightarrow{k}) = f_l^P(f_1^C(\overrightarrow{k}), \ldots, f_n^C(\overrightarrow{k}))$
(C steht für eine Anweisung)

$f_l^{\mathbf{loop}\ x_i\ \mathbf{do}\ P\ \mathbf{end}}(\overrightarrow{k}) = \overline{f}_l^P(\overrightarrow{k}, k_i)$

wobei

$\overline{f}_l^P(\overrightarrow{k}, 0) = k_l$

$\overline{f}_l^P(\overrightarrow{k}, m+1) = f_l^P(\overline{f}_1^P(\overrightarrow{k}, m), \ldots, \overline{f}_n^P(\overrightarrow{k}, m))$

Bemerkung:

Es wird benutzt, daß sich ein Programm eindeutig als Folge von Anweisungen schreiben läßt. Es wird nämlich $f_l^{C;P}$ und nicht etwa $f_l^{P_1;P_2}$ definiert. So spart man sich den Nachweis der Eindeutigkeit der definierten Funktion f_l^P.

Lemma 14.4

Die Funktionen f_l^P sind primitiv rekursiv.

Beweis:

Dies ergibt sich aus den Definitionsgleichungen in Def. 14.3. Bei der Definition von $\overline{f}_1^P, \ldots, \overline{f}_n^P$ wird dabei ein Schema der *simultanen* Rekursion benutzt: $\overline{f}_l^P(\overrightarrow{k}, m+1)$ wird durch Rückgriff auf $\overline{f}_i^P(\overrightarrow{k}, m)$ für alle i $(1 \leq i \leq n)$ definiert. Dies läßt sich durch geeignete Kodierung als primitiv-rekursive Definition nachweisen (Übung).

Definition 14.5 (LOOP-berechenbare Funktionen)

Eine n-stellige Funktion $f : \mathbb{N}^n \to \mathbb{N}$ heißt LOOP-berechenbar, wenn es ein Programm $P \in \text{LOOP}_m$ für $m \geq n$ gibt, so daß gilt:

$$f(k_1, \ldots, k_n) = f_1^P(k_1, \ldots, k_n, \underbrace{0, \ldots, 0}_{m-n})$$

D.h. der Wert von $f(k_1, \ldots, k_n)$ muss nach Ablauf des Programms P der Wert der Variablen x_1 sein, falls „vor“ Beginn des Programmablaufs die Werte von $x_1, \ldots, x_n$ auf $k_1, \ldots, k_n$ und die Werte von eventuellen Hilfsvariablen $x_{n+1}, \ldots, x_m$ auf 0 gesetzt werden.

Bemerkung:

Die Wahl von x_1 als Speicherplatz für den Wert der Funktion und die Wahl von 0 als Ausgangsbelegung der Hilfsvariablen ist willkürlich.

Theorem 14.6 (Äquivalenz LOOP-berechenbar/primitiv rekursiv)

Die LOOP-berechenbaren Funktionen sind genau die primitiv-rekursiven Funktionen.

Beweis:

1. Jede LOOP-berechenbare Funktion f ist primitiv-rekursiv:
   Es ist zu zeigen, daß f_1^P primitiv rekursiv ist. Dies ergibt sich aus Lemma 14.4.

2. Jede primitiv-rekursive Funktion ist LOOP-berechenbar:
   Für Ausgangsfunktionen und Komposition ist das Verfahren offensichtlich. Bei primitiver Rekursion geht man wie folgt vor:
   Sei
   $$\begin{aligned} f(\vec{x},0) &= g(\vec{x}) \\ f(\vec{x},y') &= h(\vec{x},y,f(\vec{x},y)) \end{aligned}$$

   G und H seien die LOOP-Programme, die g und h berechnen. Sei $\vec{x} = x_1, \ldots, x_n$. Seien $G, H \in \text{LOOP}_k$ (und damit $k \geq n+2$). Dann wird f durch folgendes Programm P aus LOOP_{k+n+2} berechnet, d.h. P benutzt $n+2$ Hilfsvariablen:

$x_{k+1} :=$	$x_1; \ldots; x_{k+n+1} := x_{n+1};$	% Argumente von f werden in Hilfsvariablen % gespeichert
G;		
$x_{n+1} :=$	$0;$	% Rekursionsvariable wird auf 0 gesetzt
$x_{n+2} :=$	$x_1;$	% Ergebnis von G wird an erste Ausführung % von H übergeben
loop	x_{k+n+1}	
do	$x_1 := x_{k+1}; \ldots; x_n := x_{k+n};$	% Setze bei jeder Iteration Parameter auf % Ausgangswert
	$x_{n+3} := 0; \ldots; x_k := 0;$	% Setze Hilfsvariablen von H auf 0
	$H;$	
	$x_{n+1} := N(x_{n+1});$	% Erhöhe Rekursionsvariable
	$x_{n+2} := x_1$	% Übergebe Wert von H für nächste % Iteration
end		

Operationale Semantik von LOOP-Programmen

Wir hatten eine *denotationale* Semantik von LOOP-Programmen angegeben. Diese beschrieb die Leistung eines LOOP-Programms durch eine Funktion, die einer Eingabe-Belegung der Variablen eine Ausgabe-Belegung zuordnete. Da diese Funktion primitiv rekursiv war, ergab sich sofort der erste Teil des Theorems. Man kann die Semantik eines LOOP-Programms auch *operational* angeben. Dann ist der Beweis des ersten Teils des

Theorems etwas aufwendiger, da man die primitiv rekursive Funktion erst noch konstruieren muss. Eine operationale Semantik gibt an, wie sich Konfigurationen bei jedem einzelnen Schritt der Programmausführung verändern. Eine operationale Semantik für LOOP wäre z.B. folgende.

Definition 14.7 (Konfiguration und Übergänge in LOOP-Programmen)

Eine *Konfiguration* ist ein Paar, bestehend aus einer *Belegung* β der Variablen durch natürliche Zahlen und einem Programm P, geschrieben als $\begin{pmatrix} \beta \\ P \end{pmatrix}$. $\beta[x_i := n]$ sei diejenige Belegung, die x_i durch n belegt und ansonsten mit β identisch ist. $\begin{pmatrix} \beta \\ \square \end{pmatrix}$ bezeichne eine Endkonfiguration.
Dann definieren wir folgende Übergangsregeln:

$$\begin{pmatrix} \beta \\ x_i := 0; P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta[x_i := 0] \\ P \end{pmatrix}$$

$$\begin{pmatrix} \beta \\ x_i := x_j; P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta[x_i := \beta(x_j)] \\ P \end{pmatrix}$$

$$\begin{pmatrix} \beta \\ x_i := N(x_j); P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta[x_i := \beta(x_j + 1)] \\ P \end{pmatrix}$$

$$\begin{pmatrix} \beta \\ x_i := V(x_j); P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta[x_i := \beta(x_j \mathbin{\dot{-}} 1)] \\ P \end{pmatrix}$$

$$\begin{pmatrix} \beta \\ \textbf{loop } x_i \textbf{ do } Q \textbf{ end}; P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta \\ \underbrace{Q; \ldots; Q}_{\beta(x_i)\text{-mal}}; P \end{pmatrix}$$

Ein Programm P transformiert β in β', formell $\beta \underset{P}{\Longrightarrow} \beta'$, falls man aus $\begin{pmatrix} \beta \\ P \end{pmatrix}$ die Endkonfiguration $\begin{pmatrix} \beta' \\ \square \end{pmatrix}$ mit den angegebenen Regeln gewinnen kann.

Bemerkungen:

1. Die Regeln sind so zu verstehen, daß sie entsprechend auch für einzelne Anweisungen gelten, bei denen die Fortsetzung P fehlt. Formal kann man das bewerkstelligen, indem man als zusätzliche Regel $\begin{pmatrix} \beta \\ C \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta \\ C; \square \end{pmatrix}$ für Anweisungen C einführt.

2. Da sich jedes Programm eindeutig als Folge von Anweisungen lesen läßt, ist die operationale Semantik deterministisch, d.h. für jedes $\begin{pmatrix} \beta \\ P \end{pmatrix}$ ist eindeutig ein Nachfolger festgelegt. Auch läßt sich leicht beweisen, daß die Anwendung der Regeln nach endlich vielen Schritten terminiert, d.h. in einer Endkonfiguration endet.

Man kann jetzt folgendes nachweisen:

Satz 14.8

Falls P zu LOOP_n gehört und β,β' entsprechend nur die Variablen $x_1, \ldots, x_n$ belegen, dann gilt $\beta \underset{P}{\Longrightarrow} \beta'$ g.d.w. $f^P(\beta(x_1), \ldots, \beta(x_n)) = (\beta'(x_1), \ldots, \beta'(x_n))$

Beweis: Übung

Definition 14.9 (WHILE-Programme)

WHILE_n-Programme unterscheiden sich von LOOP_n-Programmen dadurch, daß die Syntax von Anweisungen wie folgt gegeben ist:

$\langle$Anweisung$\rangle$ ::= $\langle$Wertzuweisung$\rangle$ | **while** $\langle$Variable$\rangle \neq 0$ **do** $\langle$Programm$\rangle$ **end**

Das Terminalalphabet enthält also „**while**“ statt „**loop**“ sowie „$\neq$“ als zusätzliches Zeichen.

Definition 14.10 (Informelle Semantik von WHILE-Programmen)

while $x_i \neq 0$ **do** P **end**

iteriert das Programm P, bis nach einer Durchführung von P die Variable x_i den Wert 0 hat. (Wenn x_i zu Beginn den Wert 0 hat, wird P überhaupt nicht ausgeführt. Wenn x_i nie den Wert 0 erhält, terminiert die **while**-Anweisung nicht. Es wird nicht verlangt, daß P terminiert.)

Beispiele:

1. **while** $x_1 \neq 0$ **do** $x_1 := x_1$ **end**

   berechnet die Funktion

   $$\begin{cases} f(0) = 0 \\ f(x+1) \ \ \text{undefiniert} \end{cases}$$

2. $x_2 := x_1; x_3 := 1;$ [genauer: $x_3 := 0; x_3 := N(x_3);$]
   while $x_2 \neq 0$ **do** $x_3 := 0; x_2 := V(x_2)$ **end** ;
   while $x_3 \neq 0$ **do** $x_3 := x_3$ **end** ;

   $x_1 := V(x_1)$
   berechnet die partielle Vorgängerfunktion

   $$\begin{cases} V_p(0) & \text{undefiniert} \\ V_p(x+1) = x \end{cases}$$

   (Zunächst wird kodiert: $x_3 = 1$ falls $x_1 = 0$, und $x_3 = 0$ falls $x_1 \neq 0$)

Lemma 14.11

Alle LOOP-berechenbaren Funktionen sind WHILE-berechenbar.

Beweis: Übung

Definition 14.12 (Denotationale Semantik von WHILE-Programmen)

Wir ersetzen in Def. 14.3 die Definition von $f_l^{\textbf{loop } x_i \textbf{ do } P \textbf{ end}}(\vec{k})$ durch
$f_l^{\textbf{while } x_i \neq 0 \textbf{ do } P \textbf{ end}}(\vec{k}) = \overline{f}_l^P(\vec{k}, (\mu y)(\overline{f}_i^P(\vec{k}, y) = 0))$.

Lemma 14.13

Die Funktionen f_l^P sind partiell rekursiv.

Theorem 14.14 (Äquivalenz WHILE-berechenbar/partiell rekursiv)

Die WHILE-berechenbaren Funktionen sind genau die partiell rekursiven Funktionen.

Beweis:

1. Jede WHILE-berechenbare Funktion ist partiell rekursiv:
   Dies folgt aus Lemma 14.13.

2. Jede partiell rekursive Funktion ist WHILE-berechenbar:
   Sei $f(\overrightarrow{x}) = (\mu y)(h(\overrightarrow{x}, y) = 0)$.
   Sei H ein WHILE-Programm, das h berechnet.
   Sei $\overrightarrow{x} = x_1, \ldots, x_n$.
   Sei $H \in \text{WHILE}_k$, d.h. $k \geq n+1$ und $x_{n+2}, \ldots, x_k$ sind die Hilfsvariablen von H.
   Dann wird f durch folgendes Programm $P \in \text{WHILE}_{k+n+1}$ berechnet, d.h. P hat $n+1$ Hilfsvariablen.

$x_{k+1} :=$	$x_1; \ldots; x_{k+n} := x_n;$	% Argumente von f werden in Hilfsvariablen
		% gespeichert
$H;$		
while	$x_1 \neq 0$	
do	$x_{k+n+1} := N(x_{k+n+1});$	% Erhöhe Schleifenzähler
	$x_1 := x_{k+1}; \ldots; x_n := x_{k+n};$	% Setze Parameter auf Ausgangswert
	$x_{n+1} := x_{k+n+1};$	% Übergebe Schleifenzähler an H
	$x_{n+2} := 0; \ldots; x_k := 0;$	% Setze Hilfsvariablen von H auf 0
	H	
end;		
$x_1 :=$	x_{k+n+1}	% Übergebe Schleifenzähler als Resultat

Definition 14.15 (Operationale Semantik von WHILE-Programmen)

An die Stelle des **loop**-Übergangs in Def. 14.7 werden folgende Übergänge gesetzt:

$$\begin{pmatrix} \beta \\ \textbf{while } x_i \neq 0 \textbf{ do } Q \textbf{ end}; P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta \\ P \end{pmatrix} \quad \text{falls } \beta(x_i) = 0$$

$$\begin{pmatrix} \beta \\ \textbf{while } x_i \neq 0 \textbf{ do } Q \textbf{ end}; P \end{pmatrix} \Longrightarrow \begin{pmatrix} \beta \\ Q; \textbf{while } x_i \neq 0 \textbf{ do } Q \textbf{ end}; P \end{pmatrix} \quad \text{falls } \beta(x_i) \neq 0$$

Bemerkung:

Die Abfolge von Übergängen ist zwar wie bei LOOP-Programmen deterministisch, d.h. eindeutig bestimmt, aber nicht mehr notwendigerweise terminierend.

15 Aufzählbarkeit

Neben dem Begriff der Berechenbarkeit selbst ist der Begriff der Aufzählbarkeit der zentrale Grundbegriff der Berechenbarkeitstheorie. Wir betrachten hier die Aufzählbarkeit einer Menge natürlicher Zahlen durch eine berechenbare Funktion.

Definition 15.1 (Rekursiv aufzählbar)

$M \subseteq \mathbb{N}$ heißt [partiell, primitiv] rekursiv aufzählbar, falls M leer ist oder es eine einstellige [partiell, primitiv] rekursive Funktion f gibt, so daß gilt: $x \in M \Leftrightarrow (\exists y) f(y) = x$.
Ein einstelliges Prädikat P hat die entsprechende Eigenschaft, falls $\{x \in \mathbb{N} : P(x)\}$ diese Eigenschaft hat.

Bemerkungen:

1. Im partiellen Fall kann die Bedingung, daß M leer ist, weggelassen werden.
2. Statt $x \in M \Leftrightarrow (\exists y) f(y) = x$ kann man auch schreiben: $f(\mathbb{N}) = M$.
3. Für mehrstellige Prädikate verfährt man analog, indem man sie als einstellige Prädikate von n-Tupeln auffaßt.
4. Der Begriff der Turing-Aufzählbarkeit (Def. 11.1) von Mengen natürlicher Zahlen (repräsentiert durch Wörter über dem Alphabet $\{*\}$) ist offensichtlich gleichwertig zu dem der rekursiven Aufzählbarkeit.

Satz 15.2

P ist [partiell, primitiv] rekursiv aufzählbar g.d.w. es ein 2-stelliges [partiell, primitiv] rekursives Prädikat Q gibt, so daß $P(x) \Leftrightarrow (\exists y) Q(x, y)$.

Beweis:
„$\Rightarrow$“: Sei P [partiell, primitiv] rekursiv aufzählbar. Dann ist

1. P leer oder
2. $P(x) \Leftrightarrow (\exists y) f(y) = x$ für ein [partiell, primitiv] rekursives f.

Im 1. Fall setze $Q(x, y) :\Leftrightarrow y \neq y$.
Im 2. Fall setze $Q(x, y) :\Leftrightarrow f(y) = x$.

„$\Leftarrow$“: Falls P leer (d.h. $\{x : P(x)\} = \emptyset$), ist die Behauptung trivial.
P sei nicht leer, d.h. es gelte $P(k)$ für ein k, das für den Rest des Beweises fest gewählt sei.
Es sei $P(x) \Leftrightarrow (\exists y)Q(x, y)$
Sei $f(y) := \begin{cases} \sigma_{21}(y) & \text{falls } Q(\sigma_{21}(y), \sigma_{22}(y)) \\ k & \text{sonst} \end{cases}$
Es gilt: $P(x) \Leftrightarrow (\exists z)Q(x, z)$
$\Leftrightarrow (\exists y)(Q(\sigma_{21}(y), \sigma_{22}(y))$ und $x = \sigma_{21}(y)$
(mit $y := \sigma_2(x, z)$ bzw $z : \sigma_{22}(y))$
$\Rightarrow (\exists y)f(y) = x$ (nach Definition von f)
Umgekehrt gilt: $(\exists y)f(y) = x$
$\Rightarrow$ entweder $x = \sigma_{21}(y)$ und $Q(\sigma_{21}(y), \sigma_{22}(y))$ oder $x = k$.
In beiden Fällen gilt $P(x)$.

Bemerkung:

Für die Richtung „$\Leftarrow$“ ist die Fallunterscheidung, ob P leer oder nichtleer ist, wesentlich und geht in die Definition von f ein. Konstruktiv gilt der Satz unter der Annahme, daß P nichtleer ist.

Theorem 15.3 (Kleenesches Aufzählungstheorem)

Zu jedem 2-stelligen partiell rekursiven Prädikat $R(x, y)$ gibt es ein $t \in \mathbb{N}$, so daß
$(\exists y)R(x, y) \Leftrightarrow (\exists y)T_1(t, x, y)$.

Beweis:
Sei $g(x, y) = \begin{cases} 0 & \text{falls } R(x, y) \\ 1 & \text{falls } R(x, y) \text{ definiert und } \neg R(x, y) \\ \text{undefiniert} & \text{sonst} \end{cases}$
d.h., $g = \overline{sg}(\chi_R)$ (vgl. Bemerkung zu Def. 12.12).

Sei t Gödelnummer der TM $\mathcal{M}$, die $(\mu y)(g(x, y) = 0)$ berechnet. Sei $(\exists y)R(x, y)$. Dann hält $\mathcal{M}$ nach endlich vielen Schritten. Also gilt: $(\exists z)E(t, K(t, x, z))$.
Damit nach Lemma 13.5 (i) $(\exists y)T_1(t, x, y)$.
Sei umgekehrt $(\exists y)T_1(t, x, y)$. Damit nach demselben Lemma $(\exists z)E(t, K(t, x, z))$.
Also erreicht $\mathcal{M}$ eine Haltekonfiguration nach endlich vielen Schritten,
also ist $(\mu y)(g(x, y) = 0)$ definiert, also gilt $(\exists y)R(x, y)$.

Bemerkung:

Hier steht "$\Leftrightarrow$" und nicht etwa $\overset{\sim}{\leftrightarrow}$, da R *partiell* rekursiv und T_1 *primitiv* rekursiv (und damit *total*) ist.

[Um diese Bemerkung zu verstehen, müßte man 3-wertige Logiken einführen und dabei insbesondere solche behandeln, für die sich bei der Äquivalenz von Formeln der Wahrheitswert „unbestimmt“ oder „undefiniert“ ergibt, falls nur eine der Formeln undefiniert ist (in unserem Falle, falls für ein x die Formel $R(x, y)$ für alle y undefiniert ist). Vgl. dazu Kleene (1952), § 64.]

In jedem Fall gilt:
$\{x : (\exists y)R(x, y)\} = \{x : (\exists y)T_1(t, x, y)\}$

Theorem 15.4

Für $M \subseteq \mathbb{N}$ ist folgendes äquivalent:

(i) M ist partiell rekursiv aufzählbar

(ii) M ist rekursiv aufzählbar

(iii) M ist primitiv-rekursiv aufzählbar

Beweis: (iii) $\Rightarrow$ (ii) $\Rightarrow$ (i) trivial

(i) $\Rightarrow$ (iii): M partiell rekursiv aufzählbar
$\Leftrightarrow M = \{x : (\exists y)Q(x, y)\}$ nach Satz 15.2
$\Leftrightarrow M = \{x : (\exists y)T_1(t, x, y)\}$ nach Theorem 15.3 für gewisses t
$\Leftrightarrow M$ primitiv rekursiv aufzählbar nach Satz 15.2, da T_1 primitiv rekursiv

Bemerkungen:

1. Der intuitive Grund für „(i) $\Rightarrow$ (iii)“ liegt darin, daß wir den Wertebereich einer partiell rekursiven Funktion auf primitiv rekursive Weise in Abhängigkeit von der Zahl der Berechnungsschritte aufzählen können.

2. Das Theorem liefert für den Fall der natürlichen Zahlen die noch offene Richtung im Beweis von Theorem 11.6: Wir können davon ausgehen, daß die TM, die graph(f) aufzählt, immer terminiert. Also können wir für gegebene $w_1, \ldots, w_n$ sukzessiv den Graphen durchlaufen. Falls $f(w_1, \ldots, w_n)$ definiert, erreichen wir mit Sicherheit $w_1\$ \ldots \$w_n\$w$ für ein w.

Definition 15.5

Eine Menge M heißt [partiell, primitiv] rekursiv, falls das sie charakterisierende Prädikat $P(x) \Leftrightarrow x \in M$ [partiell, primitiv] rekursiv ist[2].

[2] Vgl. Definition 16.1 und die erste Bemerkung dazu.

Theorem 15.6

Eine Menge $M \subseteq \mathbb{N}$ ist genau dann rekursiv, wenn M und $\overline{M} = \mathbb{N} \backslash M$ rekursiv aufzählbar sind .

Beweis:

„$\Rightarrow$“: $x \in M \Leftrightarrow (\exists y)(x \in M \wedge y = y)$
$x \in \overline{M} \Leftrightarrow (\exists y)(x \in \overline{M} \wedge y = y)$

„$\Leftarrow$“: $M = f(\mathbb{N}), \overline{M} = g(\mathbb{N})$.
Dann $x \in M \Leftrightarrow x = f((\mu y)(f(y) = x \vee g(y) = x))$

Bemerkung:

Das Theorem gilt nicht für „primitiv rekursiv“ oder „partiell rekursiv“ an Stelle von „rekursiv“. Ansonsten wären wegen Theorem 15.4 „partiell rekursiv“, „rekursiv“ und „primitiv rekursiv“ äquivalent.

Satz 15.7

Es gibt jeweils Mengen natürlicher Zahlen mit den folgenden Eigenschaften:

(i) M ist nicht rekursiv aufzählbar

(ii) M ist rekursiv aufzählbar, M ist nicht primitiv-rekursiv

(iii) M ist rekursiv aufzählbar, M ist nicht rekursiv

(iv) M ist rekursiv aufzählbar, M ist nicht partiell-rekursiv

(v) M ist rekursiv aufzählbar, $\overline{M}$ ist nicht rekursiv aufzählbar

(vi) M und $\overline{M}$ sind nicht rekursiv aufzählbar

Beweis:

(i) Graph der busy-beaver-Funktion

(ii) - (v) Universelles Halteproblem

Für $H(t,m)$ als „das Paar (t,m) gehört zum universellen Halteproblem“ gilt $H(t,m) \Leftrightarrow (\exists y) T_1(t,m,y)$.

(vi) Die Menge aller rekursiv aufzählbaren Mengen natürlicher Zahlen ist rekursiv aufzählbar, also insbesondere abzählbar. Also muß es aus dem (nicht abzählbaren) Bereich der Mengen natürlicher Zahlen Mengen mit der geforderten Eigenschaft geben.

Bemerkung:

Es läßt sich leicht zeigen, daß kontextsensitive Sprachen rekursiv sind, da sie sich durch nichtverkürzende Grammatiken darstellen lassen. Da nach Theorem 11.3 die Typ-0-Sprachen genau die rekursiv aufzählbaren Sprachen sind, zeigt Satz 15.7 (iii), daß der Bereich der Typ-0-Sprachen echt über den der Typ-1-Sprachen hinausgeht. (Diese Bemerkung setzt voraus, daß man die in diesem Kapitel behandelten Aufzählbarkeitsbegriffe von Mengen natürlicher Zahlen auf Sprachen überträgt.)

Universelle Turing-Maschinen

Ziel: Sei t Gödelnummer einer TM $\mathcal{M}$. Wir wollen eine universelle TM $\mathcal{M}_u^f$ definieren, die folgendes leistet:
$(\underline{\#}\overline{t}\#\overrightarrow{x}) \underset{\mathcal{M}_u^f}{\Longrightarrow} (\underline{\#}\overline{y})$ g.d.w. $(\underline{\#}\overrightarrow{x}) \underset{\mathcal{M}}{\Longrightarrow} (\underline{\#}\overline{y})$.

Eine solche TM $\mathcal{M}_u^f$ heißt universell, da sie in einer einzigen Maschine die Berechnung beliebiger Funktionen in beliebigen Maschinen simuliert nach Eingabe der Nummer der jeweiligen Maschine als zusätzlichem Argument. Wir definieren zunächst eine universelle Maschine $\mathcal{M}_u^1$, die unmittelbare Übergänge zwischen Konfigurationen simuliert, und dann eine universelle Maschine $\mathcal{M}_u$, die Akzeptanz simuliert.

Wir modifizieren zunächst $K(t,\overrightarrow{x},z)$ zu einer partiell rekursiven Funktion $K(t,k,z)$ wie folgt:

$K(t,k,0) := k$

$$K(t,k,z') := \begin{cases} F(t,K(t,k,z)) & \text{falls } \neg E(t,K(t,k,z)) \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Dabei sei $K(t,k,n)$ überhaupt nur dann definiert, wenn k Nummer einer Konfiguration von $\mathcal{M}$ ist.
$\mathcal{M}_u^1$ sei die TM, die folgendes leistet:
$(\underline{\#}\overline{t}\#\overline{k}) \underset{\mathcal{M}_u^1}{\Longrightarrow} (\underline{\#}\overline{t}\#\overline{K(t,k,1)})$, falls $K(t,k,1)$ definiert.

$(\underline{\#}\bar{t}\#\bar{k}) \underset{\mathcal{M}_u^1}{\Longrightarrow} (\underline{\#}\#\bar{t}\#\bar{k})$, falls $E(t,k)$. Ansonsten soll $\mathcal{M}_u^1$ nicht terminieren.
(Konstruktion als Übung)

Dann gilt für
$\ulcorner\mathcal{M}\urcorner$ Gödelnummer einer TM $\mathcal{M}$
$\ulcorner k\urcorner$ Gödelnummer einer Konfiguration k
$\ulcorner w\urcorner$ Gödelnummer eines Wortes w

$$\begin{cases} k_1 \overset{1}{\underset{\mathcal{M}}{\vdash}} k_2 & \text{g.d.w.} \quad (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner k_1\urcorner}) \underset{\mathcal{M}_u^1}{\Longrightarrow} (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner k_2\urcorner}) \\ k \in H_{\mathcal{M}} & \text{g.d.w.} \quad (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner k\urcorner}) \underset{\mathcal{M}_u^1}{\Longrightarrow} (\underline{\#}\#\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner k\urcorner}) \end{cases}$$

$\mathcal{M}_u^1$ simuliert also einzelne Schritte von $\mathcal{M}$.

Sei $\mathcal{M}_i$: $(\underline{\#}\bar{t}\#\overline{\ulcorner w\urcorner}) \underset{\mathcal{M}_i}{\Longrightarrow} (\underline{\#}\bar{t}\#\overline{k_0})$, wobei k_0 Gödelnummer von Anfangskonfiguration $(s, \underline{\#}w)$ für TM mit Nummer t.

$$\text{Sei } \mathcal{M}_u : \begin{tikzcd} \mathcal{M}_i\mathcal{M}_u^1 R \arrow[r, "\neq \#"] & L \arrow[l, bend right] \end{tikzcd}$$

Terminierende Schrittfolgen lauten dann:
$(\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner w\urcorner}) \underset{\mathcal{M}_i}{\Longrightarrow} (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{k_0}) \underset{\mathcal{M}_u^1}{\Longrightarrow} (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{k_1}) \Longrightarrow \ldots \Longrightarrow (\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{k_n})$
(k_n Nummer von Endkonfiguration von $\mathcal{M}$)

Es gilt: $(\underline{\#}w) \underset{\mathcal{M}}{\Longrightarrow} k$ für ein k g.d.w. $(\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner w\urcorner}) \underset{\mathcal{M}_u}{\Longrightarrow} k'$ für ein k'
$\mathcal{M}_u$ simuliert also Akzeptanz für jede beliebige TM $\mathcal{M}$.
Sei $W'(t,k) = \ulcorner w\urcorner$, falls k Gödelnummer einer Endkonfiguration $(\underline{\#}w)$.
Sei $\mathcal{M}_e : (\underline{\#}\bar{t}\#\bar{k}) \underset{\mathcal{M}_e}{\Longrightarrow} (\underline{\#}\overline{W'(t,k)})$

$$\text{Sei } \mathcal{M}_u^f : \begin{tikzcd} \mathcal{M}_i\mathcal{M}_u^1 R \arrow[r, "\neq \#"] \arrow[d, "= \#"] & L \arrow[l, bend right] \\ \mathcal{M}_e & \end{tikzcd}$$

Es gilt dann:
$(\underline{\#}w_1) \underset{\mathcal{M}}{\Longrightarrow} (\underline{\#}w_2)$ g.d.w. $(\underline{\#}\overline{\ulcorner\mathcal{M}\urcorner}\#\overline{\ulcorner w_1\urcorner}) \underset{\mathcal{M}_u^f}{\Longrightarrow} (\underline{\#}\overline{\ulcorner w_2\urcorner})$
$\mathcal{M}_u^f$ simuliert also Funktionsberechnung.
Entsprechend gibt es eine universelle partiell rekursive Funktion f_u mit folgender Eigenschaft:
$f_u(t_g, \vec{x}) = n$, falls t_g Nummer der TM, die g berechnet und $g(\vec{x}) = n$, und undefiniert sonst.

f_u kann man als die universelle Funktion U im Beweis der Unentscheidbarkeit des universellen Halteproblems verwenden (vergleiche Theorem 11.8).

Bemerkungen:

1. Zur Wiederholung: Wäre das universelle Halteproblem entscheidbar, könnte man f_u total machen:
   $\overline{f}_u(m,n) := \begin{cases} f_u(m,n) & \text{wenn } f_u(m,n) \text{ definiert} \\ 1 & \text{sonst} \end{cases}$
   Setze
   $g(n) := \overline{f}_u(n,n) + 1$
   Da g (total) rekursiv, gilt
   $g(x) = f_u(t,x)$ für ein t und für alle x.
   Damit
   $g(t) = f_u(t,t) = \overline{f}_u(t,t)$,
   jedoch
   $g(t) = \overline{f}_u(t,t) + 1$ nach Definition.

2. Eine universelle (total) rekursive oder primitiv rekursive Funktion für Nummern von (total) rekursiven oder primitiv rekursiven Funktionen bzw. eine entsprechende universelle Turing-Maschine kann es nicht geben, da dieses Diagonalargument wegen der Totalität der Funktion direkt zum Widerspruch führt.

Einige Ausblicke in die Rekursionstheorie

Die Rekursionstheorie geht von der Tatsache aus, daß man jede partiell rekursive Funktion durch eine natürliche Zahl kennzeichnen kann, und baut darauf eine abstrakte Theorie der Berechenbarkeit und Unentscheidbarkeit auf. Hier können nur einige ganz elementare (aber fundamentale) Tatsachen beschrieben werden. Ich orientiere mich dabei an Engeler & Läuchli (1992). Für die weiterführende Theorie vgl. Kleene (1952), Rogers (1987) und Shoenfield (1967). Die Rekursionstheorie wurde maßgeblich von Kleene begründet. Vgl. dazu den Überblick über Kleenes mathematisches Werk in Shoenfield (1995).

Theorem 15.8 (Aufzählbarkeitstheorem)

Für jedes n gibt es ein partiell rekursives $\Phi^n : \mathbb{N}^{n+1} \to \mathbb{N}$, so daß es zu jedem partiell-rekursiven $f : \mathbb{N}^n \to \mathbb{N}$ ein $e \in \mathbb{N}$ gibt mit $\Phi^n(e, x_1, \ldots, x_n) \simeq f(x_1, \ldots, x_n)$.
Dieses e heißt auch Index von f.

Beweis: $\Phi^n(e, \vec{x}) := U((\mu y) T_n(e, \vec{x}, y))$ (vgl. Theorem 13.6)

Bemerkungen:

1. Wir schreiben auch $\Phi^n_e(\vec{x})$ statt $\Phi^n(e, \vec{x})$ und lassen n häufig weg.

2. Man kann sogar annehmen, daß umgekehrt auch jedes $e \in \mathbb{N}$ Index einer partiell rekursiven Funktion ist. Dann ist e etwas anders zu definieren.

3. Das Aufzählbarkeitstheorem kann man als Ausdruck der Tatsache verstehen, daß Programme (= Funktionen f) zugleich als Daten (= Zahlen, Indizes e) verstanden werden können.

Theorem 15.9 (*S*-*m*-*n*-Theorem)

Für $\vec{x} = x_1, \dots, x_m$ und $\vec{y} = y_1, \dots, y_n$ gibt es eine rekursive Funktion S_n^m, so daß gilt:
$\Phi_e(\vec{x}, \vec{y}) \simeq \Phi_{S_n^m(e,\vec{x})}(\vec{y})$

Beweis: $S_n^m(e, \vec{x})$ ist wie folgt definiert:
Sei e Gödelnummer einer TM $\mathcal{M}$. $\mathcal{M}_1$ leiste folgendes: $(\underline{\#}\vec{y}) \underset{\mathcal{M}_1}{\Longrightarrow} (\underline{\#}\vec{x}\#\vec{y})$.
Es sei $\mathcal{M}' := \mathcal{M}_1\mathcal{M}$.
$S_n^m(e, \vec{x})$ sei die Gödelnummer von $\mathcal{M}'$. Es läßt sich zeigen, daß S_n^m rekursiv ist.

Theorem 15.10 (Rekursionstheorem)

Zu jeder partiell rekursiven Funktion f gibt es ein e, so daß $\Phi_e(\vec{x}) \simeq f(e, \vec{x})$ ist.

Beweis: Sei a Index von $f(S_n^1(y,y), \vec{x})$, d.h. $\Phi_a(y, \vec{x}) \simeq f(S_n^1(y,y), \vec{x})$.
Mit S-m-n-Theorem: $f(S_n^1(y,y), \vec{x}) \simeq \Phi_{S_n^1(a,y)}(\vec{x})$.
Mit $y := a$ und $e := S_n^1(a,a)$ ergibt sich $f(e, \vec{x}) \simeq \Phi_e(\vec{x})$.

Bemerkungen:

1. Intuitiver Inhalt: Zu jeder partiell rekursiven Modifikationsvorschrift für TM gibt es eine TM, die sich selbst entsprechend dieser Vorschrift modifiziert. Z.B. gibt es eine TM, die (ohne weitere Argumente) ihren eigenen Index generiert, d.h. $\Phi_e = e$ (Setze $f := \mathrm{id}$) („Selbstreproduktion").

2. Das Rekursionstheorem besagt, daß sich eine rekursive Funktionsgleichung immer lösen läßt, in der eine Funktion unter Rückgriff auf ihren Index (und damit indirekt unter Rückgriff auf sich selbst) definiert wird. Bezeichnet man mit $\{f\}$ den Index einer partiell rekursiven Funktion f, und ist g eine $(n+1)$-stellige partiell rekursive Funktion, dann ist
$F(\vec{x}) = g(\{F\}, \vec{x})$
eine rekursive Funktionsgleichung in der Funktionsvariablen F. Diese Gleichung hat nach dem Rekursionstheorem die Lösung $F = \Phi_e$ für ein geeignetes e.

3. Das nach dem Theorem existierende e ist eindeutig, wenn man es minimal wählt.

Beispiel für die Anwendung des Rekursionstheorems:

Wir wollen nachweisen, daß die Gleichungen $\begin{cases} f(x,0) &= g(x) \\ f(x,y') &= h(x,y,f(2x,y)) \end{cases}$ eine rekursive Funktion f definieren, vorausgesetzt, g und h sind rekursiv.
Wir definieren eine Hilfsfunktion $\overline{f}$

$\begin{cases} \overline{f}(z,x,0) &= g(x) \\ \overline{f}(z,x,y') &= h(x,y,\Phi_z(2x,y)) \end{cases}$ $\overline{f}$ ist offensichtlich rekursiv (keine „echte“ Rekursion).

Nach dem Rekursionstheorem gibt es ein e, für das gilt:

$\Phi_e(x,y) = \overline{f}(e,x,y)$

Es gilt also:

$\begin{cases} \Phi_e(x,0) &= g(x) \\ \Phi_e(x,y') &= h(x,y,\Phi_e(2x,y)) \end{cases}$

Setze $f := \Phi_e$.
(vgl. Shoenfield 1995)

Theorem 15.11 (Fixpunktsatz)

Sei REK^n der Bereich aller n-stelligen partiell rekursiven Funktionen.
Sei f einstellig partiell rekursiv. Dann hat
$G_f : \mathrm{REK}^n \to \mathrm{REK}^n$
$\Phi^n_y \mapsto \Phi^n_{f(y)}$
einen Fixpunkt, d.h. es gibt ein e, so daß $\Phi^n_e(\overrightarrow{x}) \simeq \Phi^n_{f(e)}(\overrightarrow{x})$.

Beweis: Setze $g(z,\overrightarrow{x}) := \Phi_{f(z)}(\overrightarrow{x})$.
Nach dem Rekursionstheorem gibt es ein e, so daß gilt $\Phi_e(\overrightarrow{x}) \simeq g(e,\overrightarrow{x})$. Daraus folgt die Behauptung.

Bemerkung:

Intuitiver Inhalt: Zu jeder Modifikationsvorschrift f für TM gibt es eine Maschine, deren Leistung nach Modifikation unverändert bleibt. D.h., es gibt eine „*vollständige*“ Modifikation, die sich nicht weiter modifizieren läßt.

Korollar 15.12

Sei F ein n-stelliges Funktionszeichen; sei $G(F)$ wie eine partiell rekursive Funktion definiert, wobei zusätzlich zu C_0^0, N und U_n^i auch F als Ausgangssymbol zugelassen ist („$G(F)$ ist partiell rekursiv in F“). Dann gilt: Es gibt eine n-stellige partiell rekursive Funktion f, so daß $f(\vec{x}) = G(f)(\vec{x})$, d.h. die Funktionsgleichung für die Variable F:
$F(\vec{x}) = G(F)(\vec{x})$
hat eine partiell rekursive Funktion f als Lösung.

Beweis: Aufgrund der partiell rekursiven Zuordnung
$F \mapsto G(F)$
ist $G(\Phi_e)$ partiell rekursiv in e, d.h.

$$\begin{aligned} G(\Phi_e)(\vec{x}) &= g(e, \vec{x}) \text{ für ein partiell rekursives } g \\ &= \Phi_{e'}(e, \vec{x}) \text{ für dessen Index } e' \\ &= \Phi_{S_n^1(e',e)}(\vec{x}) \end{aligned}$$

Nach dem Fixpunktsatz gibt es ein e'', so daß gilt $\Phi_{e''}(\vec{x}) = \Phi_{S_n^1(e',e'')}(\vec{x})$. Wähle f so, daß e'' Index von f ist. Dann gilt: $G(f)(\vec{x}) = G(\Phi_{e''})(\vec{x}) = \Phi_{S_n^1(e',e'')}(\vec{x}) = \Phi_{e''}(\vec{x}) = f(\vec{x})$

Beispiel:

1. Die Gleichung

   $$F(x) = F(x) + 1$$

   hat ein partiell rekursives f als Lösung, für das gilt:

   $$f(x) = f(x) + 1$$

   Offenbar ist f die partiell rekursive Funktion, die überall undefiniert ist.

2. Die Gleichung

   $$F(x) = \begin{cases} 1 & \text{falls } x = 0 \\ x \cdot F(x \dot{-} 1) & \text{sonst} \end{cases}$$

   hat ein partiell rekursives f als Lösung, nämlich die Fakultätsfunktion.
   Diese Definition lautet in ML: fun $f(x) =$ if $x = 0$ then 1 else $x * f(x - 1)$.
   (vgl. z.B. Paulson 1996)

Bemerkungen:

1. Hier geht es um die Lösung von Funktionsgleichungen, bei denen eine Funktion *direkt* unter Rückgriff auf sich selbst (ohne Umweg über ihren Index) definiert ist.

2. Die Tatsache, daß sich Funktionsgleichungen der Form

$$F(\overrightarrow{x}) = G(F)(\overrightarrow{x})$$

   immer durch eine partiell rekursive, d.h. berechenbare Funktion lösen lassen, wenn nur die linke Seite partiell rekursiv von der rechten Seite abhängt, ist fundamental für die Semantik von (insbesondere funktionalen) Programmiersprachen. Ihre Erweiterung zu berechenbaren Funktionen, die über den Bereich der natürlichen Zahlen hinausgehen, erfährt sie in der denotationellen Semantik.

16 Unentscheidbarkeit

Wir geben einige unentscheidbare Probleme an, und zwar neben allgemeinen Problemen wie dem universellen Halteproblem und dem Wortproblem für Semi-Thue-Systeme zwei spezielle Probleme: Das Postsche Korrespondenzproblem (Behandlung nach Davis & Weyuker 1983) und die Unentscheidbarkeit der Hornklausellogik.

Definition 16.1 (Entscheidbarkeit von Prädikaten)

Ein Prädikat P heißt entscheidbar, falls $P(x)$ partiell rekursiv ist (vgl. Def. 12.12). Eine Menge M heißt entscheidbar, falls M partiell rekursiv ist. M heißt partiell rekursiv, falls es ein partiell rekursives Prädikat P gibt, durch das sich M darstellen läßt, d.h. für das $M = \{x : \ P(x)\}$ gilt. Ein Problem heißt entscheidbar, wenn die Menge der Lösungen des Problems entscheidbar ist.

Bemerkungen:

1. Es werden also Mengen M betrachtet, für die $a \in M$ nicht unbedingt definiert ist.

   Sei D der Bereich, für den $a \in M$ definiert ist. Dann gilt:
   M partiell rekursiv $\Leftrightarrow$ M und $D \backslash M$ rekursiv aufzählbar. $D \backslash M$ ist in der Regel verschieden von $\overline{M}$.

2. Unser Begriff der Turing-Entscheidbarkeit (Definition 11.1) entspricht der Entscheidbarkeit für totale Mengen, d.h. der Rekursivität. Man hätte auch Turing-Entscheidbarkeit für partielle Mengen definieren können, d.h. für Sprachen L, für die $w \in L$ nicht für alle w definiert sein muß.

Theorem 16.2

Folgende Probleme sind unentscheidbar:

(i) Hält $\mathcal{M}$ bei Eingabe w ? ($w \in L(\mathcal{M})$?)

(ii) Hält $\mathcal{M}$ bei leerer Eingabe ? ($\varepsilon \in L(\mathcal{M})$?)

(iii) Gibt es ein w, so daß $\mathcal{M}$ bei Eingabe w hält ? ($L(\mathcal{M}) = \emptyset$?)

(iv) Hält $\mathcal{M}$ bei jeder Eingabe ? ($L(\mathcal{M}) = \Sigma^*$?)

(v) Hält $\mathcal{M}$ bei unendlich vielen Wörtern ? (Gibt es ein k, so daß $|L(\mathcal{M})| < k$?)

(vi) Halten $\mathcal{M}_1$ und $\mathcal{M}_2$ bei denselben Eingaben ? ($L(\mathcal{M}_1) = L(\mathcal{M}_2)$?)

(vii) Hält $\mathcal{M}_2$ bei einer Obermenge der Eingaben, bei denen $\mathcal{M}_1$ hält ?
($L(\mathcal{M}_2) \supseteq L(\mathcal{M}_1)$?)

(viii) Ist die von $\mathcal{M}$ akzeptierte Sprache regulär ?
kontextfrei ?
kontextsensitiv ?
entscheidbar ?

Hierbei stehen $\mathcal{M}$, $\mathcal{M}_1$ und $\mathcal{M}_2$ für beliebige TM, nicht für feste Maschinen.

Beweis:

(i) Dies ist das universelle Halteproblem. Vgl. Theorem 11.8.

(ii) Sei die Maschine w so definiert, daß $(\underline{\#}) \underset{w}{\Longrightarrow} (\underline{\#}w)$ gilt. Dann gilt: $\mathcal{M}$ hält bei Eingabe w g.d.w. $w\mathcal{M}$ bei leerer Eingabe hält. Damit haben wir eine Zurückführung auf das universelle Halteproblem.

(iii) Definiere δ: $(\underline{\#}w) \underset{\delta}{\Longrightarrow} (\underline{\#})$.
$\delta\mathcal{M}$ hält bei irgendeiner Eingabe g.d.w. $\mathcal{M}$ bei leerer Eingabe hält. Damit haben wir Zurückführung auf Punkt (ii).

(iv) $\delta\mathcal{M}$ akzeptiert jede Eingabe g.d.w. $\delta\mathcal{M}$ irgendeine Eingabe akzeptiert.
Auf (iii) zurückgeführt.

(v) $\delta\mathcal{M}$ akzeptiert unendlich viele Wörter g.d.w. $\delta\mathcal{M}$ irgendein Wort akzeptiert.
Auf (iii) zurückgeführt.

(vi) Wähle $\mathcal{M}_1$ als die triviale Maschine, die sofort, d.h. bei jeder Eingabe hält. Die Entscheidung darüber, ob $\mathcal{M}_1$ und $\mathcal{M}_2$ bei derselben Eingabe halten, ist dann gleichwertig mit der Entscheidung darüber, ob $\mathcal{M}_2$ bei jeder Eingabe hält. Zurückführung auf (iv).

(vii) wie (vi).

(viii) Sei $\mathcal{M}'$ die 2-Band-Maschine $\mathcal{M}' := \mathcal{M}^{(2)}\mathcal{M}_u^{(1)}$, wobei $\mathcal{M}_u$ die universelle TM für Akzeptanz ist.
$\mathcal{M}'$ akzeptiert das universelle Halteproblem (vgl. Def. 11.7) falls $\mathcal{M}$ für leere Eingabe terminiert. $\mathcal{M}'$ akzeptiert $\emptyset$, falls $\mathcal{M}$ für leere Eingabe nicht terminiert. Wegen (ii) ist daher nicht entscheidbar, ob $L(\mathcal{M}')$= universelles Halteproblem oder ob $L(\mathcal{M}') = \emptyset$. Da $\emptyset$ regulär, kontextfrei, kontextsensitiv und entscheidbar ist, das universelle Halteproblem jedoch nichts dergleichen ist (da unentscheidbar), würde die Entscheidung über eine dieser Eigenschaften die Entscheidung darüber bedeuten, ob $L(\mathcal{M}') = \emptyset$ oder $L(\mathcal{M}') =$ universelles Halteproblem.

Bemerkungen:

1. Wir wissen sogar von einer festen Maschine $\mathcal{M}$, daß das spezielle Halteproblem unentscheidbar ist. Sei $\mathcal{M}_u$ die universelle TM für die Akzeptanz von Sprachen. Setze $\mathcal{M} := \mathcal{M}_d\mathcal{M}_u$, wobei $(\underline{\#}\bar{t}) \underset{\mathcal{M}_d}{\Longrightarrow} (\underline{\#}\bar{t}\#\bar{t})$. Die Lösbarkeit des speziellen Halteproblems für $\mathcal{M}$ würde die für das universelle Halteproblem nach sich ziehen.

2. Analoge (unentscheidbare) Probleme kann man für (nicht-eingeschränkte) Grammatiken formulieren: $w \in L(\Gamma)$; $\varepsilon \in L(\Gamma)$; $L(\Gamma) = \emptyset$; $L(\Gamma) = \Sigma^*$;
$|L(\Gamma)| \leq k$; $L(\Gamma_1) = L(\Gamma_2)$; $L(\Gamma_2) \supseteq L(\Gamma_1)$; $L(\Gamma)$ regulär, kontextfrei, kontextsensitiv, entscheidbar ?

3. Das allgemeine Problem, welche Sprachmengen (z.B. innerhalb der Menge aller rekursiv aufzählbaren Sprachen) entscheidbar oder rekursiv aufzählbar sind, wird in der weiterführenden Theorie behandelt, z.B. in den Sätzen von Rice und von Greibach (vgl. Hopcroft & Ullman 1990, Kap. 8).

Definition 16.3 (Semi-Thue-System)

Ein Semi-Thue-System ist ein Paar $\Gamma =< \Sigma, \Pi >$, wobei Σ Alphabet, Π Menge von Produktionen über Σ. Die Relation $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$ ist wie bei Grammatiken erklärt.
Das Wort-Problem für Semi-Thue-Systeme besagt:
$u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$?
(D.h. Wortproblem für Semi-Thue-Probleme $= \{(u, \Gamma, v) : \ u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v\}$)

Bemerkungen:

1. Ein Thue-System ist ein Semi-Thue-System, bei dem mit $u \to v$ auch $v \to u$ zu Π gehört.

2. Axel Thue (1863–1922) war ein norwegischer Mathematiker.

Theorem 16.4

Das Wortproblem für Semi-Thue-Systeme ist unentscheidbar.

Beweis:
Ansonsten wäre das universelle Halteproblem entscheidbar, da man für beliebige TM $\mathcal{M}$ die Relation $\overset{*}{\underset{\mathcal{M}}{\vdash}}$ durch $\overset{*}{\underset{\Gamma}{\Longrightarrow}}$ für eine geeignete Grammatik Γ ausdrücken kann (Übung). Γ kann durch Zusammenfassung von Variablen und Terminalen in einem einzigen Alphabet als Semi-Thue-System aufgefaßt werden.

Bemerkungen:

1. Es gibt ein spezielles Semi-Thue-System, dessen Wortproblem unentscheidbar ist. Wähle eine Grammatik Γ, die der in Bemerkung 1 nach Theorem 16.2 genannten TM entspricht.

2. Es gibt sogar ein solches spezielles Semi-Thue-System über einem zweielementigen Alphabet $\{a, b\}$. Man repräsentiere jedes Wort $a_{j_1} \dots a_{j_n}$ über dem Alphabet $\{a_1, \dots, a_n\}$ durch $ba^{j_1}ba^{j_2} \dots ba^{j_n}b$ sowie ε durch ε.

3. Das Wortproblem für Thue-Systeme ist ebenfalls unentscheidbar. Das ist nicht ganz trivial, da mit einem Thue-System ein spezieller Typ von Grammatik vorliegt.

Theorem 16.5

Das Postsche Korrespondenzproblem hat eine negative Lösung, d.h. es gibt keinen Algorithmus, der entscheidet, ob ein gegebenes Postsches Korrespondenzsystem eine Lösung hat.

Beweis:
Sei Γ' ein Semi-Thue-System über $\{a, b\}$, dessen Wortproblem unentscheidbar ist. Γ entstehe aus Γ' durch Hinzufügung der Produktionen $a \to a$ und $b \to b$. Seien u und v Wörter über $\{a, b\}$. Wir konstruieren ein Postsches Korrespondenzsystem $P_{u,v}$, so daß gilt:
$P_{u,v}$ hat Lösung g.d.w. $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$ (g.d.w. $u \overset{*}{\underset{\Gamma'}{\Longrightarrow}} v$).
Damit Rückführung auf Wortproblem für Γ bzw. Γ'.
Alphabet von $P_{u,v}$: $\{a, b, \widetilde{a}, \widetilde{b}, *, \widetilde{*}, [,]\}$.
Für $w \in \{a, b\}^*$ sei $\widetilde{w} \in \{\widetilde{a}, \widetilde{b}\}^*$ das korrespondierende Wort, in dem $\sim$ auf jedem Zeichen steht.
$P_{u,v}$ habe folgende Dominos:

$$\boxed{\frac{[u*}{[}}\ \boxed{\frac{]}{\widetilde{*}v]}}\ \boxed{\frac{*}{\widetilde{*}}}\ \boxed{\frac{\widetilde{*}}{*}}\ \underbrace{\boxed{\frac{h}{\widetilde{g}}}\ \boxed{\frac{\widetilde{h}}{g}}}_{\text{für jedes } g \to h \text{ in } \Gamma} \quad .$$

Wegen $a \to a$ und $b \to b$ können wir auch Dominos $\boxed{\frac{\widetilde{w}}{w}}$ und $\boxed{\frac{w}{\widetilde{w}}}$ für jedes $w \in \{a, b\}^*$ annehmen.

Wir zeigen:

1. $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$ impliziert: $P_{u,v}$ hat Lösung.

   Sei $u = u_1 \overset{1}{\underset{\Gamma}{\Longrightarrow}} \ldots \overset{1}{\underset{\Gamma}{\Longrightarrow}} u_n = v$ mit n ungerade (ist möglich wegen $a \to a$ und $b \to b$).
   Dann ergibt sich eine Lösung von $P_{u,v}$ mit Lösungswort $[u_1 * \widetilde{u}_2 \widetilde{*} u_3 * \ldots * \widetilde{u}_{n-1} \widetilde{*} u_n]$ wie folgt:
   Sei $u_i = x_i g_i y_i$, $u_{i+1} = x_i h_i y_i$ mit Produktion $g_i \to h_i$:

   $$\boxed{\frac{[u*}{[}}\ \underbrace{\overbrace{\boxed{\frac{\widetilde{x}_1}{x_1}}\boxed{\frac{\widetilde{h}_1}{g_1}}\boxed{\frac{\widetilde{y}_1}{y_1}}}^{\widetilde{u}_2}}_{u_1}\ \boxed{\frac{\widetilde{*}}{*}}\ \underbrace{\overbrace{\boxed{\frac{x_2}{\widetilde{x}_2}}\boxed{\frac{h_2}{\widetilde{g}_2}}\boxed{\frac{y_2}{\widetilde{y}_1}}}^{u_3}}_{\widetilde{u}_2}\cdots\boxed{\frac{]}{\widetilde{*}v]}}$$

   (Die ungerade Anzahl von Ableitungsschritten ist notwendig, damit vor dem letzten Wort u_n das Zeichen $\widetilde{*}$ und nicht $*$ steht.)

2. $P_{u,v}$ hat Lösung impliziert $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$.

   Die Folge von Dominos, die zu einer Lösung gehört, muß wie folgt aussehen. Dabei wählen wir die kürzeste Folge, bei der eckige Klammern nur außen auftreten.

   $$\boxed{\frac{[u*}{[}}\ \underbrace{\overbrace{\boxed{\frac{\widetilde{h}_{i_1}}{g_{i_1}}}\cdots\boxed{\frac{\widetilde{h}_{i_k}}{g_{i_k}}}}^{\widetilde{u}_2}}_{u=u_1}\ \boxed{\frac{\widetilde{*}}{*}}\ \underbrace{\overbrace{\boxed{\frac{g_{j_1}}{\widetilde{h}_{j_1}}}\cdots\boxed{\frac{g_{j_r}}{\widetilde{h}_{j_r}}}}^{u_3}}_{\widetilde{u}_2}\boxed{\frac{*}{\widetilde{*}}}\cdots\boxed{\frac{]}{\widetilde{*}v]}}$$

   Man muß unten immer gegenüber oben „nachhalten“, damit sich unten dasselbe Wort wie oben ergibt.
   Das Lösungswort ist also $[u * \widetilde{u}_2 \widetilde{*} u_3 \ldots \widetilde{*} v]$.
   Dabei gilt aufgrund der Definition der vorhandenen Dominos:
   $u = u_1 \overset{*}{\Longrightarrow} u_2 \overset{*}{\Longrightarrow} u_3 \overset{*}{\Longrightarrow} \ldots \overset{*}{\Longrightarrow} u_n = v$

Bemerkung:

Für Teil 2 des Beweises ist entscheidend, daß wir bei der Definition von $P_{u,v}$ Zeichen mit Tilde $\widetilde{\ }$ verwendet haben. Ansonsten hätte die Lösung nicht die geforderte Form (das Lösungswort müsste z.B. nicht mit einer eckigen Klammer anfangen). Teil 1 ließe sich dagegen allein mit den Dominos

$$\boxed{\frac{[u*}{[}}\ \boxed{\frac{]}{*v]}}\ \boxed{\frac{*}{*}}\ \boxed{\frac{h}{g}}$$

bestreiten.

Korollar 16.6

Folgende Probleme sind nicht entscheidbar:

(i) Ist der Durchschnitt zweier kontextfreier Sprachen leer ?

(ii) Ist eine kontextfreie Grammatik eindeutig ?

Beweis: Lemma 7.11 und Lemma 7.13.

Unentscheidbarkeit eines deduktiven Systems

Wir definieren ein deduktives System, das man als einen Meta-Kalkül $\overline{\Gamma}$ zu einem Semi-Thue-System Γ auffassen kann. Wir interpretieren dabei Wörter $a_1 \ldots a_n$ als Listen der Form $(a_1. \ \ldots (a_{n-1}.(a_n. \text{ nil})) \ldots)$. ε wird durch nil interpretiert.
Sei $\Gamma =< \Sigma, \Pi >$ ein Semi-Thue-System, sei $\Sigma = \{a_1, \ldots, a_n\}$.
Das deduktive System $\overline{\Gamma}$ ist wie folgt definiert:

Definition 16.7 (Terme)

Terme von $\overline{\Gamma}$ seien die Konstanten $a_1, \ldots, a_n$ sowie nil, die Variablen $x_1, x_2, x_3, \ldots, x, y, z$ sowie alle Ausdrücke der Form $(t_1.t_2)$ für Terme t_1, t_2.

Bemerkung:

Außenklammern können wegfallen. Z.B ist $(a_1.a_2).a_3$ ein Term.

Wörter über Σ werden wie folgt in Terme übersetzt:
$\overline{\varepsilon} := \text{nil}$ $\overline{au} := (a.\overline{u})$

Lemma 16.8

Für Wörter $u, v \in \Sigma^*$ gilt:

$u = v$ g.d.w. $\overline{u} = \overline{v}$

Definition 16.9 (Formeln)

Formeln von $\overline{\Gamma}$ seien alle Ausdrücke der Form $S(t), W(t), V(t_1, t_2, t_3), VV(t_1, t_2, t_3, t_4)$,
$P(t_1, t_2), A_0(t_1, t_2), A_1(t_1, t_2), A_*(t_1, t_2)$ für Terme t, t_1, t_2, t_3, t_4.

Definition 16.10 (Axiome und Regeln)

Axiome von $\overline{\Gamma}$ sind die im folgenden angegebenen Formeln.
Regeln von $\overline{\Gamma}$ sind die im folgenden angegebenen Ausdrücke der Form $F \leftarrow F_1, \ldots, F_n$ für Formeln $F, F_1, \ldots, F_n$:
(In Anführungszeichen steht immer die intendierte Bedeutung der Prädikate, für die Axiome bzw. Regeln angegeben werden. Diese Bedeutung ist jedoch nicht Bestandteil der Definition des formalen Systems.)

$$\left.\begin{array}{l} S(a_1) \\ \vdots \\ S(a_n) \end{array}\right\} \text{„}t\text{ ist ein Symbol“}$$

$$\left.\begin{array}{l} W(\mathrm{nil}) \\ W(x.y) \leftarrow S(x), W(y) \end{array}\right\} \text{„t ist ein Wort“}$$

$V(\mathrm{nil}, x, x) \leftarrow W(x)$
$V(x_1.x_2, y, x_1.x_3) \leftarrow S(x_1), W(x_2), W(y), W(x_3), V(x_2, y, x_3)$
„Die Verkettung von t_1 und t_2 ergibt t_3“

$VV(x_1, x_2, x_3, x_4) \leftarrow V(x_1, x_2, x), V(x, x_3, x_4)$
„Die Verkettung von t_1, t_2 und t_3 ergibt t_4“

$P(\overline{u}, \overline{v})$ (für jede Produktion $u \to v$ in Γ)

$A_0(x, x) \leftarrow W(x)$ „t ist aus t in 0 Schritten ableitbar“

$A_1(x_1, x_2) \leftarrow VV(y_1, x, y_2, x_1), VV(y_1, y, y_2, x_2), P(x, y)$.
„Aus t_1 ist t_2 unmittelbar (in 1 Schritt) ableitbar“

$$\left.\begin{array}{l} A_*(x_1, x_2) \leftarrow A_0(x_1, x_2) \\ A_*(x_1, x_2) \leftarrow A_1(x_1, x), A_*(x, x_2) \end{array}\right\} \begin{array}{l} \text{„Aus } t_1 \text{ ist } t_2 \text{ in irgendeiner} \\ \text{Anzahl von Schritten ableitbar“} \end{array}$$

Definition 16.11 (Substitution)

F' ist Substitutionsinstanz einer Formel F, wenn F' aus F durch Ersetzung von Variablen durch Terme hervorgeht. $F' \leftarrow F'_1, \ldots, F'_n$ ist Substitutionsinstanz einer Regel $F \leftarrow F_1, \ldots, F_n$, falls $F' \leftarrow F'_1, \ldots, F'_n$ aus $F \leftarrow F_1, \ldots, F_n$ durch Ersetzung von Variablen durch Terme hervorgeht. Dabei müssen immer gleiche Variablen durch gleiche Terme ersetzt werden.[3]

[3] Wir haben es hier mit dem logischen Variablenbegriff zu tun, bei dem immer alle Vorkommen einer Variablen gleichzeitig durch denselben Term ersetzt werden. Der Variablen- (=Nichtterminal-) begriff der Grammatiktheorie bezieht sich stattdessen auf Variablen*vorkommen*, die, insbesondere bei kfG, entsprechend den Produktionen verschieden ersetzt werden können. Vgl. die Fußnote zu Beginn von Kapitel 4.

Ableitung

Jede Substitutionsinstanz F eines Axioms ist eine Ableitung von F in $\overline{\Gamma}$. Sind $D_1, \ldots, D_n$ jeweils Ableitungen von $F_1, \ldots, F_n$ und ist $F \leftarrow F_1 \ldots, F_n$ Substitutionsinstanz einer Regel, dann ist

$$\frac{D_1 \ldots D_n}{F}$$

eine Ableitung von F in $\overline{\Gamma}$. Wir schreiben
$\overline{\Gamma} \vdash F$ („F ist in $\overline{\Gamma}$ ableitbar"),
falls es eine Ableitung in $\overline{\Gamma}$ gibt, die mit F endet.

Beispiel:
Wir zeigen: $\overline{\Gamma} \vdash \underbrace{W(\overline{a_1(a_2a_3})}_{W(a_1.a_2.(a_3.\mathrm{nil})))}$

$$\cfrac{S(a_1) \qquad \cfrac{S(a_2) \qquad \cfrac{S(a_3) \quad W(\mathrm{nil})}{W(a_3.\mathrm{nil})}}{W(a_2.(a_3.\mathrm{nil}))}}{W(a_1.(a_2.(a_3.\mathrm{nil})))}$$

Lemma 16.12

Falls u Wort, dann $\overline{\Gamma} \vdash W(\overline{u})$
Falls $\overline{\Gamma} \vdash W(t)$, dann $t = \overline{u}$ für ein $u \in \Sigma^*$

Beweis: Übung

Lemma 16.13

Für alle Wörter u_1, u_2, u_3 gilt:
$w = u_1u_2u_3$ g.d.w. $\overline{\Gamma} \vdash VV(\overline{u_1}, \overline{u_2}, \overline{u_3}, \overline{w})$

Beweis: Übung

Theorem 16.14

$u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$ g.d.w. $\overline{\Gamma} \vdash A_*(\overline{u}, \overline{v})$.

Beweis: Induktion über Länge der Ableitung in Γ

„$\Rightarrow$“: Sei $u = v$. Dann ist $\overline{u} = \overline{v}$ nach Lemma 16.8. Ferner $\overline{\Gamma} \vdash W(\overline{u})$ nach Lemma 16.12. Also $\overline{\Gamma} \vdash A_0(\overline{u}, \overline{v})$ mit der Regel $A_0(x, x) \leftarrow W(x)$, somit $\overline{\Gamma} \vdash A_*(\overline{u}, \overline{v})$ mit der Regel $A_*(x_1, x_2) \leftarrow A_0(x_1, x_2)$.
Sei $u \overset{1}{\underset{\Gamma}{\Longrightarrow}} w \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$:
Nach I.V. ist $A_*(\overline{w}, \overline{v})$ in $\overline{\Gamma}$ ableitbar.
Ferner ist $u = u_1u_2u_3$ und $w = u_1w_2u_3$ mit $u_2 \to w_2 \in \Pi$.
Nach Lemma 16.13 gilt dann: $VV(\overline{u_1}, \overline{u_2}, \overline{u_3}, \overline{u})$ und $VV(\overline{u_1}, \overline{w_2}, \overline{u_3}, \overline{w})$ sind in $\overline{\Gamma}$ ableitbar.
Ferner ist $P(\overline{u_2}, \overline{w_2})$ Axiom. Damit haben wir folgende Ableitung in $\overline{\Gamma}$:

$$
\dfrac{\dfrac{\begin{matrix}\vdots\\ VV(\overline{u_1}, \overline{u_2}, \overline{u_3}, \overline{u})\end{matrix} \quad \begin{matrix}\vdots\\ VV(\overline{u_1}, \overline{w_2}, \overline{u_3}, \overline{w})\end{matrix} \quad P(\overline{u_2}, \overline{w_2})}{A_1(\overline{u}, \overline{w})} \quad \begin{matrix}\vdots\\ A_*(\overline{w}, \overline{v})\end{matrix}}{A_*(\overline{u}, \overline{v})}
$$

„$\Leftarrow$“: Übung

Korollar 16.15

Es ist nicht entscheidbar, ob $\overline{\Gamma} \vdash F$ für beliebige $\overline{\Gamma}$ und Formeln F ohne Variablen.

Beweis:

Ansonsten wäre nach dem Theorem $u \overset{*}{\underset{\Gamma}{\Longrightarrow}} v$, d.h. das Wortproblem für Semi-Thue-Systeme entscheidbar.

Definition 16.16 (Hornklauseltheorie)

Eine Hornklauseltheorie $\mathcal{H} =< \mathcal{K}, \mathcal{F}, \mathcal{P}, \mathcal{C} >$ ist wie folgt gegeben:

$\mathcal{K}$ ist eine Menge von Konstanten

$\mathcal{F}$ ist eine Menge von Funktionszeichen

$\mathcal{P}$ ist eine Menge von Prädikatzeichen

$\mathcal{C}$ ist eine Menge von Axiomen und Regeln

Terme, Formeln und Regeln werden analog zum vorhergehenden definiert, nur daß jetzt beliebige Zeichen aus $\mathcal{K}, \mathcal{F}, \mathcal{P}$ und $\mathcal{C}$ vorkommen. (Variablen $x_1, x_2, \ldots$ sind natürlich vorhanden. Explizit angegeben werden nur diejenigen Konstanten, die für eine spezifische Theorie charakteristisch sind.)

Bemerkungen:

1. Das oben zu einem beliebigen Semi-Thue-System Γ definierte deduktive System $\overline{\Gamma}$ ist also ein Spezialfall einer Hornklauseltheorie mit
$$\begin{aligned} \mathcal{K} &= \{a_1, \ldots, a_n, \mathrm{nil}\} \\ \mathcal{F} &= \{.\} \\ \mathcal{P} &= \{S, W, V, VV, P, A_0, A_1, A_*\} \\ \mathcal{C} &= \text{ (siehe Def. 16.10)} \end{aligned}$$

2. Hornklauseln sind nach dem Mathematiker A.Horn benannt. Sie bezeichnen Axiome und Regeln der hier betrachteten speziellen Form.

Theorem 16.17

Es gibt eine Hornklauseltheorie $\mathcal{H}$, für die $\mathcal{H} \vdash F$ für beliebige Formeln F ohne Variablen unentscheidbar ist. Damit ist auch unentscheidbar, ob $\mathcal{H} \vdash F$ für beliebige Hornklauseltheorien $\mathcal{H}$ und Formeln F ohne Variablen gilt.

Beweis: Wähle $\overline{\Gamma}$ als $\mathcal{H}$, wobei Γ ein Semi-Thue-System mit unentscheidbarem Wortproblem ist.

Bemerkung:

Es läßt sich zeigen, daß Hornklauseltheorien ohne Funktionszeichen entscheidbar sind.

Korollar 16.18

Es gibt keinen Prolog-Interpreter,

1. dessen Antworten alle korrekt sind.
2. der auf jede Anfrage hin antwortet.

Beweis:

Ein Prolog-Interpreter soll für eine Hornklauseltheorie $\mathcal{H}$ (= Prolog-Programm) und eine Formel F entscheiden, ob es eine Substitutionsinstanz F' von F gibt, für die $\mathcal{H} \vdash F'$ gilt.

Theorem 16.19 (Unentscheidbarkeit der Prädikatenlogik)

Die Prädikatenlogik erster Stufe ist unentscheidbar.

Beweisskizze:

Die Hornklausellogik ist ein Fragment der Prädikatenlogik erster Stufe. Aus der Entscheidbarkeit der Prädikatenlogik erster Stufe würde sich also die Entscheidbarkeit der Hornklausellogik ergeben, was Theorem 16.17 widerspricht.

Literatur

Asteroth, A. & Baier, C. (2002). *Theoretische Informatik: Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispielen*. München: Pearson.

Barwise, J. & Etchemendy, J. (1996). *Turing's World 3.0 for the Macintosh: An Introduction to Computability Theory*. Stanford: CSLI Publications.

Boolos, G. S. & Jeffrey, R. C. (1989). *Computability and Logic*. Cambridge University Press (3. Auflage).

Cook, S. A. (1971), The Complexity of Theorem Proving Procedures, in: *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, 151–158.

Davis, M. D. & Weyuker, E. J. (1983). *Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science*. New York: Academic Press (2. Auflage mit R. Sigal als drittem Autor 1994).

Engeler, E. & Läuchli, P. (1992).*Berechnungstheoorie für Informatiker*. Stuttgart: Teubner (2. Auflage).

Erk, K. & Priese, L. (2002). *Theoretische Informatik: Eine umfassende Einführung*. Heidelberg: Springer (2. Auflage).

Harrison, M. A. (1978). *Introduction to Formal Language Theory*. Reading Mass.: Addison-Wesley.

Hermes, H. (1978). *Aufzählbarkeit, Entscheidbarkeit, Berechenbarkeit: Einführung in die Theorie der rekursiven Funktionen*. Heidelberg: Springer (3. Auflage).

Hopcroft, J. E. & Ullman, J. D. (1990). *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Bonn: Addison-Wesley (engl. Ausgabe 1979; revidierte [simplifizierte] Ausgabe mit R. Motwani als drittem Autor 2001).

Kleene, S. C. (1952). *Introduction to Metamathematics*. Amsterdam: North-Holland (zahlreiche reprints).

Kozen, D. C. (1997). *Automata and Computability*. New York: Springer.

Lewis, H. R. & Papadimitriou, C. H. (1981). *Elements of the Theory of Computation*. Englewood Cliffs NJ: Prentice-Hall (2. Auflage 1997).

Loos, R. (1989). *Formale Sprachen: Theorie und Anwendungen*. Skriptum, Wilhelm-Schickard-Institut.

Milner, R. (1999). *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press.

Paulson, L.C. (1996). ML *for the Working Programmer*. Cambridge University Press (2. Auflage).

Rogers, H. (1987). *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill.

Salomaa, A. (1973). *Formal Languages*. New York: Academic Press.

Schöning, U. (2001). *Theoretische Informatik — kurzgefasst*. Heidelberg: Spektrum (4. Auflage).

Schulz, K. (1991). *Formale Sprachen*. Skriptum, Wilhelm-Schickard-Institut.

Shoenfield, J. R. (1967). *Mathematical Logic*. Reading Mass.: Addison-Wesley.

Shoenfield, J. R. (1995). The Mathematical Work of S.C. Kleene. *Bulletin of Symbolic Logic* 1, 9–43.

Winter, R. (2002). *Theoretische Informatik: Grundlagen mit Übungsaufgaben und Lösungen*. München: Oldenbourg.