Grundlagen der theoretischen Informatik

Institut für Informatik
Freie Universität Berlin
Dozent: Dr. Klaus Kriegel

Mitschrift: Jan Sebastian Siwy

Sommersemester 2002

Inhaltsverzeichnis

Einleitung

Definition: Eine *Berechnung* ist ein Prozess, der für eine beliebige Eingabe aus einem Eingaberaum in endlich vielen elementaren, deterministischen Schritten eine vorher spezifizierte Ausgabe bestimmt.

Beispiel:

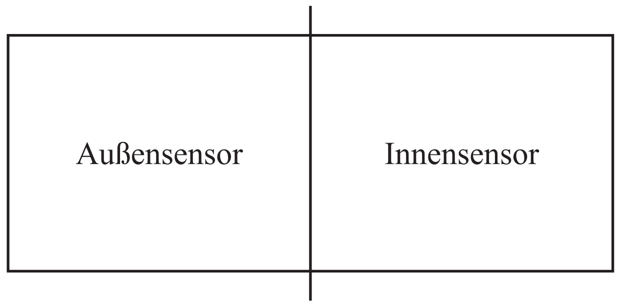
- ganze Zahl $\Rightarrow$ Primzahlzerlegung
- Graph mit Kantengewichten $\Rightarrow$ MST (minimal aufspannender Baum)
- Polynom über $\mathbb{Q}$ $\Rightarrow$ Nullstelle

Themen der Vorlesung:

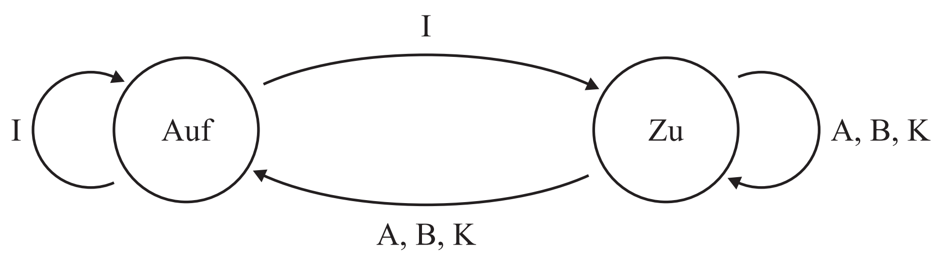
- Modelle für Berechnung
- Gemeinsamkeiten (Simulation) und Unterschiede
- Berechenbarkeit und Endscheidbarkeit
- Church'sche These
- Komplexität einer Berechnung
- P-NP-Problem
- Kontextfreie Sprachen
- Grammatiken und die Chomsky-Hierarchie

Beispiel:

1. Entwurf eines endlichen Automaten zur Steuerung einer Ausgangstür mit Eingangssperre:



Eingaben:	I	–	nur Innensensor
	A	–	nur Außensensor
	B	–	beide
	K	–	keiner



2. Postsches Korrespondenzproblem:
   Alphabet Σ mit den Wörtern $w_1, w_2, \ldots, w_k$ und $v_1, v_2, \ldots, v_k$
   Eingabe: $(w_1, v_2), (w_2, v_2), \ldots (w_k, v_k)$
   Frage: Gibt es eine Indexfolge (mit Wiederholungen) $i_1, i_2, \ldots, i_n$, so dass $w_{i_1} w_{i_2} \ldots w_{i_n} = v_{i_1} v_{i_2} \ldots v_{i_n}$?
   Zwei aller möglichen Eingaben:

   (a) $\overbrace{(1{,}101)}^{a}, \overbrace{(10{,}00)}^{b}, \overbrace{(011{,}11)}^{c}$
   Indexfolge: a, c, b, c
   linke Seite: 1 011 10 011
   rechte Seite: 101 11 00 11

   (b) $(001{,}0), (01{,}011), (01{,}101), (10{,}001)$
   kürzeste Lösung: 66 Paare

   Dieses Problem ist im Allgemeinen nicht entscheidbar (siehe 2.4)!

Kapitel 1

Reguläre Sprachen

1.1 Formale Sprachen

1.1.1 Alphabet

Definition: Ein *endliches Alphabet* ist eine Menge von Symbolen. Alphabete werden häufig mit Σ oder Γ bezeichnet.

Beispiel: $\Sigma = \{0, 1\}$, $\Gamma = \{a, b, c, \ldots z\}$

1.1.2 Wort

Definition: Ein *Wort* (String) über Σ ist eine endliche Folge von Symbolen aus Σ, wobei Wiederholungen erlaubt sind. Das *leere Wort* ε ($\varepsilon \notin \Sigma$) besteht aus null Symbolen. Die Länge des Wortes $|w|$ ist die Anzahl der Symbole in w.

Beispiel: $w_1 = 01101$, $w_2 = abracadabra$

1.1.3 Wortmengen

Um die Menge von Wörtern, die sich aus einem Alphabet bilden lassen, darzustellen, wird folgende Notation vereinbart:

$$\begin{aligned}
\Sigma^0 &= \{\varepsilon\} \\
\Sigma^1 &= \Sigma \\
\Sigma^{k+1} &= \{wa \mid w \in \Sigma^k \wedge a \in \Sigma\} \\
\Sigma^* &= \bigcup_{n=0}^{\infty} \Sigma^n \quad \text{(Menge aller Wörter über } \Sigma\text{)} \\
\Sigma^+ &= \bigcup_{n=1}^{\infty} \Sigma^n \quad \text{(Menge aller Wörter über } \Sigma \text{ ohne das leere Wort)}
\end{aligned}$$

1.1.4 Konkatenation

Definition: Die Operation der *Konkatenation* in Σ^* verbindet zwei Wörter. Das Wort w konkateniert mit dem Wort w' ist ww'. Σ^* mit der Konkatenation ist ein Monoid.

Für die Konkatenation gleicher Wörtern wird folgede Nolation vereinbart:

$$\begin{aligned} w^0 &= \varepsilon \\ w^1 &= w \\ w^{k+1} &= ww^k \\ \\ |w^k| &= k \cdot |w| \end{aligned}$$

Beispiel: 1011 konkateniert mit 011 ist 1011011.

1.1.5 Formale Sprachen

Definition: Eine Untermenge $L \subseteq \Sigma^*$ wird *formale Sprache* über Σ genannt.

Definition: Die folgenden Operationen auf Sprachen nennt man *regulär*:

1. Vereinigung: $A \cup B = \{w \mid w \in A \vee w \in B\}$
2. Konkatenation: $A \circ B = \{ww' \mid w \in A \wedge w' \in B\}$
3. Stern: $A^* = \{w_1 w_2 \ldots w_k \mid k \in \mathbb{N}, w_i \in A\}$

Beispiele:

$$\begin{aligned} A &= \{\text{good}, \text{bad}\} \\ B &= \{\text{girl}, \text{boy}\} \\ A \cup B &= \{\text{good}, \text{bad}, \text{girl}, \text{boy}\} \\ A \circ B &= \{\text{goodgirl}, \text{badgirl}, \text{goodboy}, \text{badboy}\} \\ A^* &= \{\varepsilon, \text{good}, \text{bad}, \text{goodgood}, \text{goodbad}, \text{badgood}, \text{badbad}, \text{goodgoodgood}, \ldots\} \end{aligned}$$

1.2 Endliche Automaten

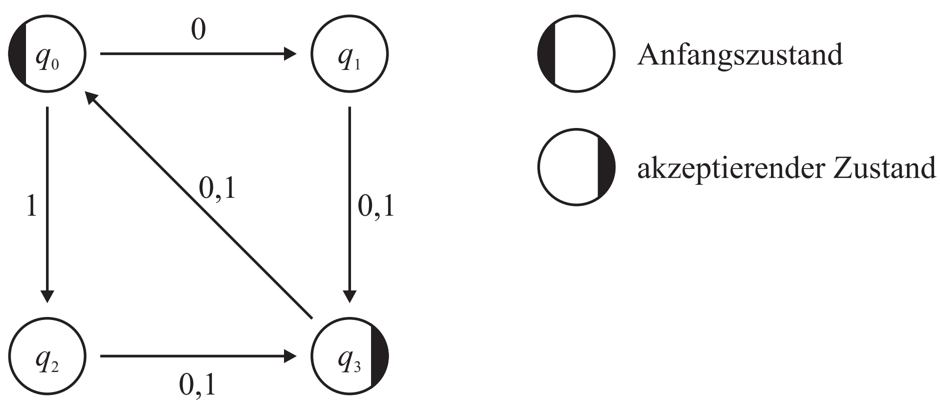
1.2.1 Deterministische Automaten

Definition: Ein *deterministischer endlicher Automat* (deterministic finite automaton, kurz *DFA*) M ist ein 5-Tupel:

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

- Q: endliche Menge von Zuständen
- Σ: Eingabealphabet
- δ: Zustandsüberführungsfunktion ($Q \times \Sigma \to Q$)
- $q_0 \in Q$: Anfangszustand
- F: Menge der akzeptierenden Zustände

Ein DFA kann als gerichteter Graph dargestellt werden:



Für jedes $q \in Q$ und $a \in \Sigma$ gibt es genau eine Kante von q mit dem Wert a.

1.2.2 Akzeptierte Sprachen von DFAs

Um Berechnungen durchführen zu können, wird die Zustandüberführungsfunktion δ induktiv erweitert, so dass sie Wörter verarbeiten kann.

$$\begin{array}{rcl} \hat{\delta} & : & Q \times \Sigma^* \to Q \\ \hat{\delta}(q,\varepsilon) & = & q \\ \hat{\delta}(q,aw) & = & \hat{\delta}(\delta(q,a),w) \end{array}$$

Definition: Ein Wort $w \in \Sigma^*$ wird genau dann akzeptiert, wenn $\hat{\delta}(q_0,w) \in F$.

Definition: Die Menge aller akzeptierten Wörter bildet die *akzeptierte Sprache* des Automaten:

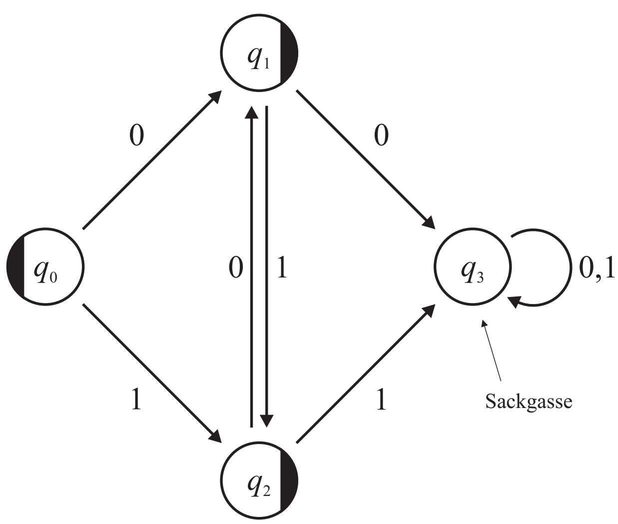
$$L(M) = \{w \in \Sigma^* \mid \hat{\delta}(q,w) \in F\}$$

Beispiel: Die akzeptierte Sprache des Automanten im Abschnitt 1.2.1 lautet:

$$L(M) = \{w \mid w \in \{0,1\}^*, |w| = 3k + 2, k \in \mathbb{N}\}$$

1.2.3 „Sackgassen"-Prinzip

Entwurf eines Automaten, der alle alterierenden 01-Folgen akzeptiert.

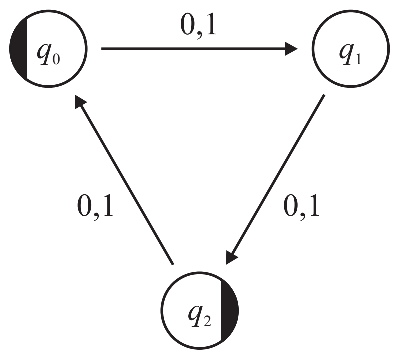


Das „Sackgassen"-Prinzips beruht darauf, dass ein Zustand existiert, bei dem die Zustandsübergangsfunktion für alle Eingaben wieder denselben Zustand liefert. Dies ist nützlich, wenn ab einer bestimmten Stelle im Wort bekannt ist, dass dieses Wort mit Sicherheit akzeptiert bzw. nicht akzeptiert werden soll.

1.2.4 Äquivalenz

Definition: Zwei Automaten M und M' sind *äquivalent* wenn deren Sprachen $L(M)$ und $L(M')$ gleich sind.

Beispiel: Der folgende Automat ist äquivalent zum Automaten im Abschnitt 1.2.1, da er der gleiche Sprache akzeptiert.



1.2.5 Vereinigung von DFA-Sprachen

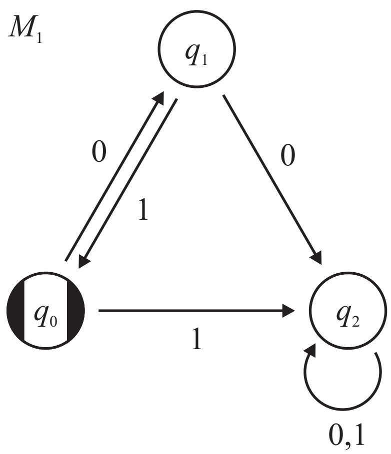
Satz: Sind A und B DFA-Sprachen, dann ist auch $A \cup B$ auch DFA-Sprache.

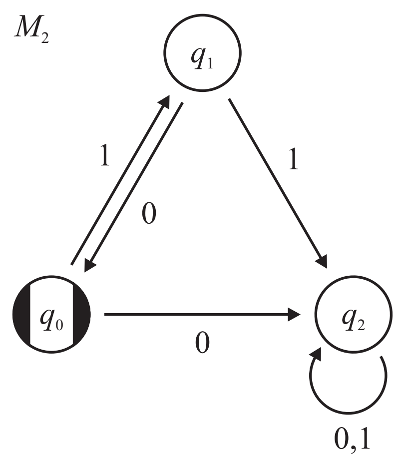
Beweis: Beide Automaten sollen parallel arbeiten. Es sollen alle Wörter akzeptiert werden, welche der erste *oder* der zweite Automat akzeptiert. Dazu bildet man das kartisiches Produkt der Zustandsmengen von M_1 und M_2.

$$\begin{aligned}
\delta &: ((Q_{M_1}, Q_{M_2}), \Sigma) \rightarrow (Q_{M_1}, Q_{M_2}) \\
\delta((q_{M_1}, q_{M_2}), a) &= (\delta_{M_1}(q_{M_1}, a), \delta_{M_2}(q_{M_2}, a)) \\
F &= \{(q_{M_1}, q_{M_2}) \mid q_{M_1} \in F_{M_1} \ \vee \ q_{M_2} \in F_{M_2}\} \\
L(M) &= A \cup B
\end{aligned}$$

Beispiel: Es sind gegeben die beiden folgenden Automaten M_1 und M_2 mit den Sprachen A und B.

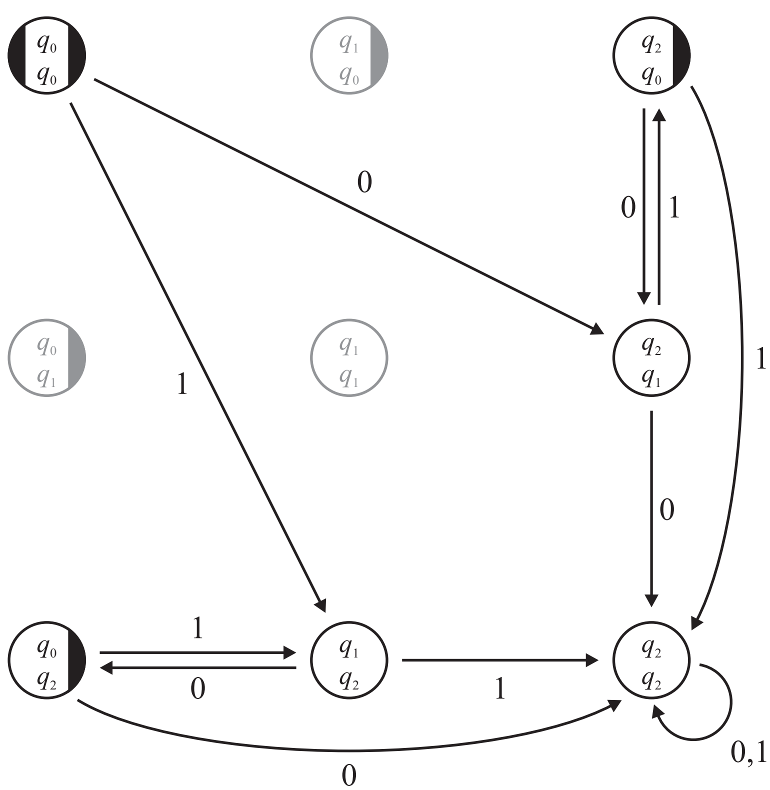
$$\begin{aligned} A = L(M_1) &= \{(01)^k \mid k \in \mathbb{N}\} = (01)^* \\ B = L(M_2) &= \{(10)^k \mid k \in \mathbb{N}\} = (10)^* \end{aligned}$$





Die Vereinigung der beiden Sprachen A und B lautet:

$$A \cup B = \{w \mid w \in (01)^* \lor w \in (10)^*\}$$



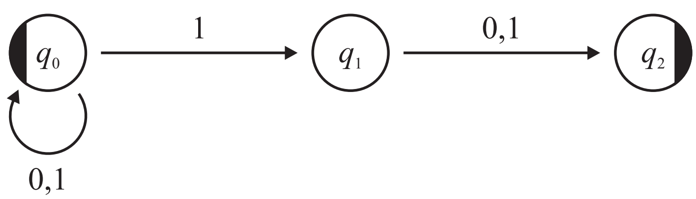
1.2.6 Nichtdeterministische Automaten

Definition: Ein *nichtdeterministischer endlicher Automat* (nondeterministic finite automaton, kurz *NFA*) M ist ein 5-Tupel:

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

- Q: Menge von Zuständen
- Σ: endliches Eingabealphabet
- δ: Zustandsüberführungsfunktion ($Q \times \Sigma \rightarrow \mathcal{P}(Q)$)
- q_0: Anfangszustand
- F: Menge der akzeptierenden Zustände

Beispiel: Dieser NFA akzeptiert alle Wörter, deren vorletztes Symbol 1 ist.



1.2.7 Akzeptierte Sprachen von NFAs

Um eine Berechnungen durchführen zu können, wird die Zustandüberführungsfunktion δ erweitert, so dass sie Wörter verarbeiten kann.

$$\begin{aligned}
\hat{\delta} &: \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q) \\
\hat{\delta}(S, \varepsilon) &= S \\
\hat{\delta}(S, aw) &= \hat{\delta}\left(\bigcup_{q \in S} \delta(q, a), w\right)
\end{aligned}$$

Definition: Ein Wort $w \in \Sigma^*$ wird von M genau dann akzeptiert, wenn es mindestens einen Berechnungszweig für w gibt, der von q_0 zu einem $q \in F$ führt.

$$\hat{\delta}\left(\{q_0\}, w\right) \cap F \neq \emptyset$$

1.2.8 Simulation eines NFA durch einen DFA

Satz: Jeder NFA kann durch einen äquivalenten DFA *simuliert* werden.

Beweis: Die Zustände des DFA M' werden repräsentiert durch alle Teilmengen der Zustandsmenge von des NFA M.

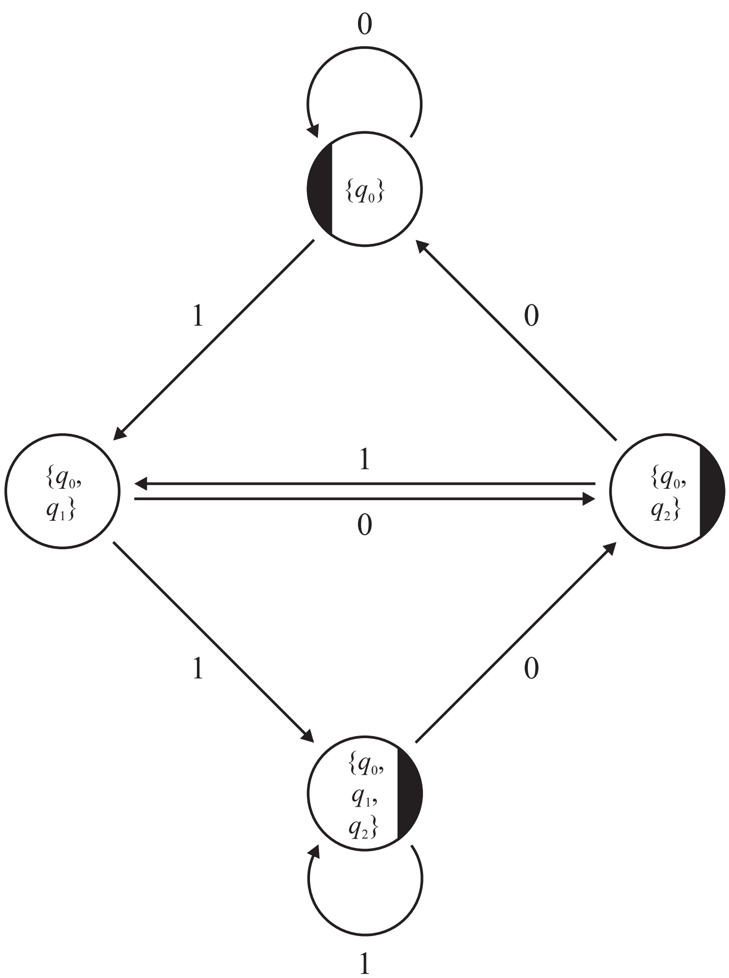
$$M = (Q, \Sigma, \delta, q_0, F) \quad \Rightarrow \quad M' = (\mathcal{P}(Q), \Sigma, \delta', \{q_0\}, F')$$

$$\delta'(S, a) = \bigcup_{q \in S} \delta(q, a)$$

$$F' = \{S \in \mathcal{P}(Q) \mid S \cap F \neq \emptyset\}$$

$$L(M) \quad = \quad L(M')$$

Beispiel: Der folgende DFA ist äquivalent zum NFA im Abschnitt 1.2.6.



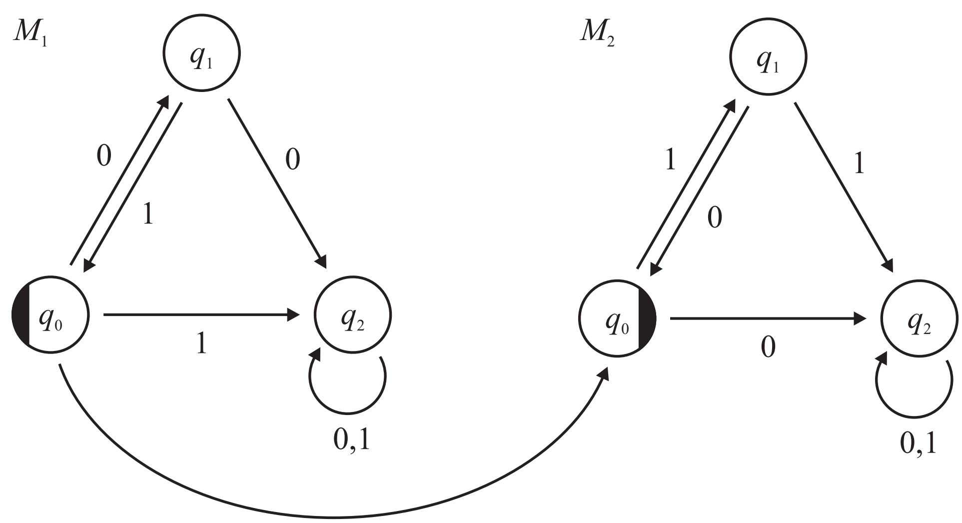
1.2.9 ε-Übergänge

Definition: M ist ein NFA mit *ε-Übergängen*, wenn die Zustandsübergangsfunktion δ auch „spontane“ Zustandübergänge (also Zustandübergänge ohne Verarbeitung eines Symbols a aus Σ) zulässt:

$$\delta \; : \; Q \times (\Sigma \cup \{\varepsilon\}) \;\Rightarrow\; \mathcal{P}(Q)$$

M akzeptiert eine Eingabe $w \in \Sigma^*$, wenn es einen Zweig der Berechnung gibt, der von q_0 zu einem akzeptierenden Zustand führt, dessen konkatenierte Berechnung w ist.

Anwendung: Konkatenation der Sprachen von M_1 und M_2 aus Abschnitt 1.2.5.



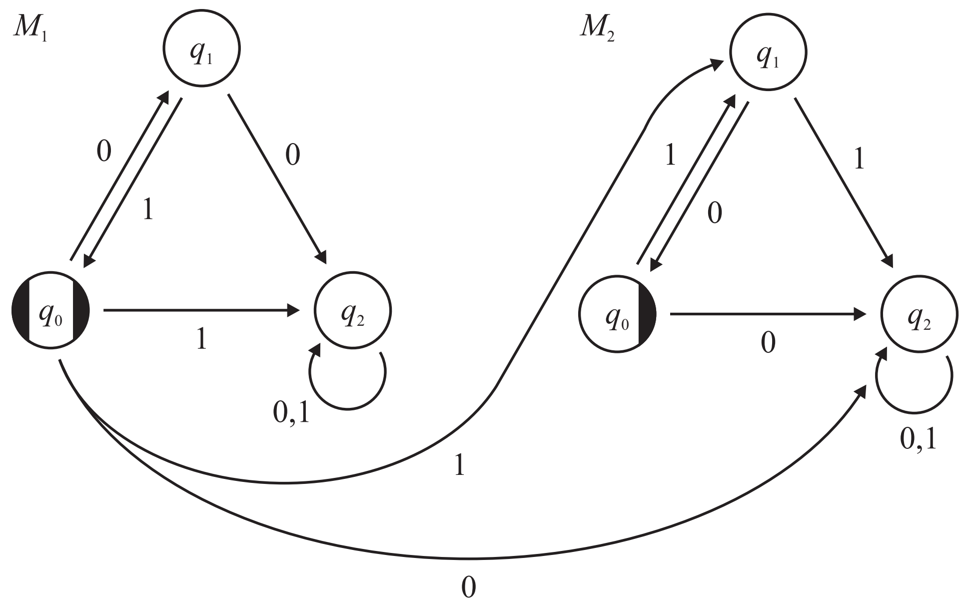
Satz: Jeder NFA M mit ε-Übergängen kann durch einen NFA M' ohne ε-Übergänge simuliert werden. Q, Σ, q_0 und F bleiben dabei gleich, jedoch muss die Zustandsübergangsfunktion angepasst werden:

$$\delta'(q,a) = \bigcup_{i,j \in \mathbb{N}} \hat{\delta}(q, \varepsilon^i a \varepsilon^j) \qquad \text{(für } i,j \in \mathbb{N}\text{)}$$

Ausnahme: Falls der Automat mit ε-Übergängen das leere Wort ε akzeptiert, dann muss die Menge der akzeptierenden Zustände um den Startzustand q_0 erweitert werden.

$$\delta(q_0, \varepsilon) \cap F \neq \emptyset \quad \Rightarrow \quad q_0 \in F'$$

Beispiel: Auflösung der ε-Übergänge

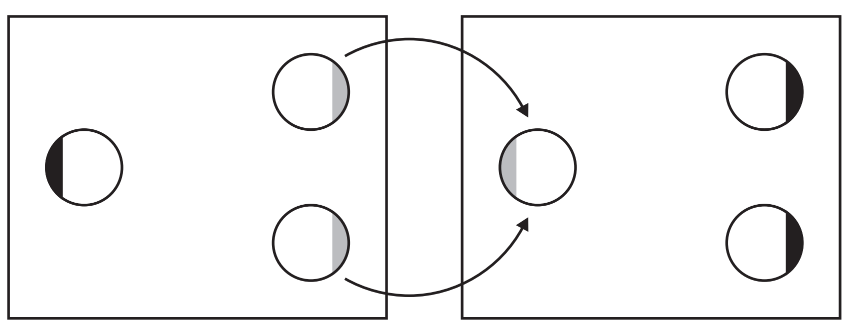


1.2.10 Konkatenation und Stern von DFA-Sprachen

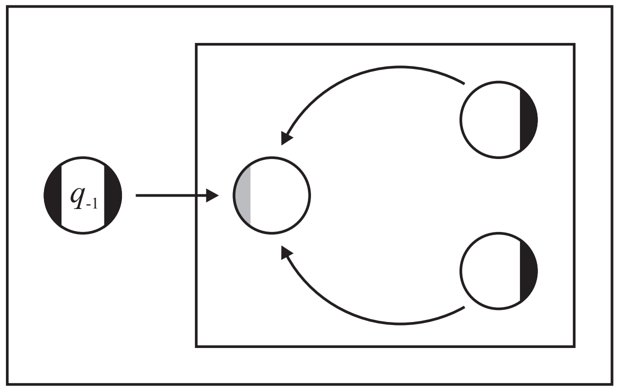
Satz: DFA-Sprachen sind abgeschlossen gegenüber der Konkatenation und dem Stern.

Beweis: M_1 ist DFA für A und M_2 ist DFA für B.

- Schritt 1: Konstruktionen von NFAs mit ε-Übergängen für $A \circ B$ und A^*.
  - M° für $A \circ B$
    - disjukte Vereinigung der Zustandsmengen von M_1 und M_2
    - für jedes $q \in F_{M_1}$ ein ε-Übergang nach q_{0,M_2}
    - $F_{M^\circ} = F_{M_1}$
    - $q_{0,M^\circ} = q_{0,M_1}$



- M^* für A^*
  - Zustandsmenge von M_1 und neuer Startzustand q_{-1}
  - ε-Übergang von q_{-1} nach q_0
  - für jedes $q \in F_{M_1}$ ein ε-Übergang nach q_0



- Schritt 2: Umwandlung der NFAs mit ε-Übergängen in äquivalente NFAs ohne ε-Übergänge
- Schritt 3: Umwandlung der NFAs ohne ε-Übergänge in äquivalente DFAs

1.3 Minimierung endlicher Automanten

1.3.1 Unerreichbare und äquivalenze Zustände

Ziel: L sei DFA-Sprache. Man konstruiert einen DFA M, so dass $L(M) = L$ und die Anzahl der Zustände $|Q_M|$ minimal ist.

Lösung:

1. Eleminierung unerreichbarer Zustände
2. Verschmelzung äquivalenter Zustände

Definition: $q \in Q$ ist *unerreichbar*, wenn es keine Eingabe $w \in \Sigma^*$ gibt, die zu diesem Zustand q führt.

$$\forall w \in \Sigma^* \quad q \neq \hat{\delta}(q_0, w)$$

Satz: Die Menge der unerreichbaren Zustände eines DFAs kann in $O(|Q| \cdot |\Sigma|)$ Zeit bestimmt werden. Für den durch die erreichbaren Zustände induzierten Automaten M' gilt $L(M') = L(M)$.

Beweis: DFS oder BFS im Graphen des Automaten mit Start in q_0 liefert alle erreichbaren Zustände, die restlichen sind nicht erreichbar. Die zweite Behauptung ergibt sich aus der Definition von $\hat{\delta}$.

Definition: M ist ein DFA. Zwei Zustände $q, q' \in Q$ heißen *äquivalent*, wenn für alle $w \in \Sigma^*$ gilt

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

Wir schreiben dann $q \equiv q'$. $\equiv$ ist eine Äquivalenzrelation.

1.3.2 Äquivalenzklassenautomat

Definition: Der *Äquivalenzklassenautomat* M' von M hat die folgende Gestalt:

- $Q' = Q_{/\equiv}$ (Menge der Äquivalenzklassen)
- $\Sigma' = \Sigma$
- $q_0' = [q_0]_\equiv$
- $F' = \{[q]_\equiv \mid q \in F\}$
- $\delta'([q]_\equiv, a) = [\delta(q,a)]_\equiv \in Q'$

Beweise:

1. F' ist wohldefiniert.

   Zu zeigen ist: Aus $q \equiv q'$ folgt, dass $q \in F \Leftrightarrow q' \in F$.

   Dazu muss man $q \equiv q'$ mit dem Wort $w = \varepsilon$ in die Definition von $\equiv$ einsetzen.

   $$\begin{aligned} \hat{\delta}(q,\varepsilon) \in F \quad &\Leftrightarrow \quad \hat{\delta}(q',\varepsilon) \in F \\ q \in F \quad &\Leftrightarrow \quad q' \in F \end{aligned}$$

2. δ' ist wohldefiniert.

   Zu zeigen ist: Aus $q \equiv q'$ folgt, dass $\forall a \in \Sigma \quad \delta'([q]_\equiv, a) = \delta'([q']_\equiv, a)$

   Aus $q \equiv q'$ folgt, dass $\forall w \in \Sigma^* \quad \hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(q',w) \in F$.

   Insbesondere gilt für jedes $w = aw'$: $\hat{\delta}(q,aw') \in F \Leftrightarrow \hat{\delta}(q',aw') \in F$.

   Laut Definition von $\hat{\delta}$ ist $\hat{\delta}(q,aw') = \hat{\delta}(\delta(q,a),w')$.

   Das bedeutet, dass $\hat{\delta}(\delta(q,a),w) \in F \Leftrightarrow \hat{\delta}(\delta(q',a),w) \in F$.

   Daraus folgt: $\delta(q,a) \equiv \delta(q',a)$, also $\delta'([q]_\equiv, a) = \delta'([q']_\equiv, a)$.

3. $L(M) = L(M')$.

   Aus $w \overset{\in}{\underset{\notin}{}} L(M)$ folgt, dass die Zustandsfolge $q_0, q_1, \ldots q_n$ mit $q_n \overset{\in}{\underset{\notin}{}} F$ endet. Die entsprechende Zustandfolge $[q_0]_\equiv, [q_1]_\equiv, \ldots [q_n]_\equiv$ für den Äquivalenzautomaten endet folglich mit $[q_n]_\equiv \overset{\in}{\underset{\notin}{}} F'$ folgt.

Konstruktion eines Äquivalenzklassenautomaten:

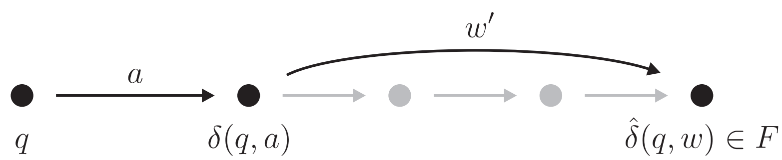
$$q \equiv q' \quad \Leftrightarrow \quad \Big(\forall w \in \Sigma^* \quad \hat{\delta}(q,w) \in F \Leftrightarrow \hat{\delta}(q',w) \in F\Big)$$

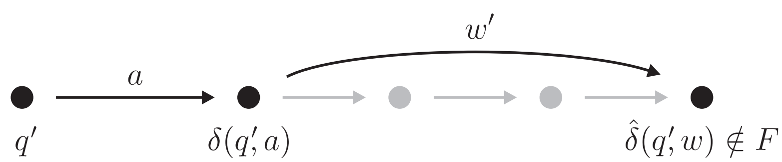
Algorithmische Bestimmung von $\equiv$ über $\not\equiv$:

$$\begin{aligned} q \not\equiv q' \quad \Leftrightarrow \quad & \Big(\exists w \in \Sigma^* \quad \hat{\delta}(q,w) \in F \wedge \hat{\delta}(q',w) \notin F\Big) \text{ oder} \\ & \Big(\exists w \in \Sigma^* \quad \hat{\delta}(q,w) \notin F \wedge \hat{\delta}(q',w) \in F\Big) \end{aligned}$$

Beobachtung:

1. Ist $w = aw'$ Zeuge für $q \not\equiv q'$, dann ist w' Zeuge für $\delta(q,a) \not\equiv \delta(q',a)$





2. Ist w'' der kürzeste Zeuge für $\delta(q,a) \not\equiv \delta(q',a)$, dann ist aw'' ein kürzester Zeuge für $q \not\equiv q'$, der mit a beginnt.

Algorithmus für $\not\equiv$: Man beginnt mit dem Zeugen w der Länge 0 (d.h. $w = \varepsilon$). Dann testet man alle Nichtäquivalenzen durch schrittweise Erhöhung der Zeugenlänge. Wenn nach einer Erhöhung keine *neuen* Nichtäquivalenzen auftreten, stoppt das Verfahren.

1.3.3 Nerode-Relation

Definition: Für $L \subseteq \Sigma^*$ ist R_L in Σ^* die *Nerode-Relation* (für alle $x, y \in \Sigma^*$):

$$x \; R_L \; y \quad \Leftrightarrow \quad (\forall w \in \Sigma^* \quad xw \in L \quad \Leftrightarrow \quad yw \in L)$$

Satz: Die Menge der Äquivalenzklassen $\Sigma^*_{/R_L}$ ist genau dann endlich, wenn L eine DFA-Sprache ist.

Beweis (⇒): Konstruktion eines DFAs für L, mit einem Zustand für jede Äquivalenzklasse $[x]_{R_L}$ von R_L.

- $q_0 = [\varepsilon]_{R_L}$
- $\delta([x]_{R_L}, a) = [xa]_{R_L}$
- $[x]_{R_L} \in F \Leftrightarrow x \in L$
- $L(M) = L$

Das heißt, man kann einen DFA konstruieren, sofern die Anzahl der Äquivalenzklassen der Nerode-Relation R_L endlich ist.

Beweis (⇐): Man wähle $x, y \in \Sigma^*$ beliebig.

- Wenn $\hat{\delta}(q_0, x) = \hat{\delta}(q_0, y)$, dann $x \; R_L \; y$, somit $[x]_{R_L} = [y]_{R_L}$.
- Wenn $\neg(x \; R_L \; y)$, somit $[x]_{R_L} \neq [y]_{R_L}$, dann $\hat{\delta}(q_0, x) \neq \hat{\delta}(q_0, y)$.

Daraus folgt, dass es maximal so viele Äquivalenzklassen in der Nerode-Relation geben kann wie Zustände in einem beliebigen DFA. Damit ist die Anzahl der Äquivalenzklassen endlich.

$$\left|\Sigma^*_{/R_L}\right| \leq |Q|$$

Satz: Die Anzahl der Zustände von $M_\equiv$ entspricht maximal der Anzahl der Äquivalenzklassen von R_L.

Beweis: Man wähle $x, y \in \Sigma^*$ beliebig.

- Wenn $x \; R_L \; y$, dann $\hat{\delta}(q_0, x) \equiv \hat{\delta}(q_0, y)$
- Wenn $\hat{\delta}(q_0, x) \not\equiv \hat{\delta}(q_0, y)$, dann $\neg(x \; R_L \; y)$

Daraus folgt, dass es maximal so viele Zustände im Äquivalenzklassenautomaten gibt wie Äquivalenzklassen in der Nerode-Relation.

$$\left|Q_{/\equiv}\right| \leq \left|\Sigma^*_{/R_L}\right|$$

Satz: Die Äquivalenzklassenautomat $M_\equiv$ von M ist ein Minimalautomat für $L(M)$.

Beweis: Sei M'' ein Minimalautomat für $L(M)$, dann folgt daraus:

$$\left|Q_{/\equiv}\right| \leq \left|\Sigma^*_{/R_L}\right| \leq |Q''|$$

Aber alle $\leq$ sind $=$, sonst wäre Q'' nicht minimal. Daraus folgt

$$\left|Q_{/\equiv}\right| = |Q''|$$

und $M_\equiv$ ist Minimalautomat.

Beispiel: Überprüfung mit Hilfe der Nerode-Relation, ob eine Sprache L DFA-Sprache ist.

- $L = \{w \in \{0{,}1\}^* \mid w \text{ beginnt mit } 01\}$
  Äquivalenzklassen von R_L
  - $[\varepsilon]_{R_L} = \{\varepsilon\}$
  - $[0]_{R_L} = \{0\}$
  - $[1]_{L_R} = \{1, 00\} \circ \Sigma^*$
  - $[01]_{R_L} = \{01\} \circ \Sigma^*$

  Alle Äquivalenzklassen von R_L ergeben wieder L:

  $$L = [\varepsilon]_{R_L} \cup [0]_{R_L} \cup [1]_{R_L} \cup [01]_{R_L}$$

  Da die Anzahl der Äquivalenzklassen von R_L endlich ist, ist L DFA-Sprache.

- $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
  Die Anzahl der Äquivalenzklassen ist unendlich, da $\neg(0^i \; R_L \; 0^j)$ für $i \neq j$.
  Begründung: Für $w = 1^i$ ist $0^i \; 1^i \in L$, aber $0^j \; 1^i \notin L$.
  Damit ist L keine DFA-Sprache.

1.4 Reguläre Sprachen und reguläre Ausdrücke

1.4.1 Einführung

Definition: *Reguläre Sprachen* über Σ sind induktiv definiert:

1. $L = \emptyset$ und $L = \{a\}$ für $a \in \Sigma$ sind regulär.
2. Sind L_1 und L_2 regulär, dann ist auch $L_1 \cup L_2$ regulär.
3. Sind L_1 und L_2 regulär, dann ist auch $L_1 \circ L_2$ regulär.
4. Ist L_1 regulär, dann ist auch L_1^* regulär.

Eine vollständige Beschreibung einer regulären Sprache durch 1. - 4. nennt man einen *regulären Ausdruck*.

Beispiele:

- $L = \{w \in \{0{,}1\}^* \mid w \text{ beginnt mit } 0\} = 0(\{0\} \cup \{1\})^* = 0\{0{,}1\}^* = 0\Sigma^*$
- $L = \{\varepsilon\} = \emptyset^*$
- $L = \{w \in \{0{,}1\}^* \mid w \text{ hat 3 aufeinander folgende } 1\} = \Sigma^*111\Sigma^*$
- $L = \{w \in \{0{,}1\}^* \mid w \text{ enthält } \textit{nicht } 010\}$
  - Fall 1: 01 tritt nicht auf in w: 1^*0^*
  - Fall 2: 01 tritt (endlich oft) auf, muss mit 1 weitergehen, außer beim letzten 01, da dort auch ε möglich ist.

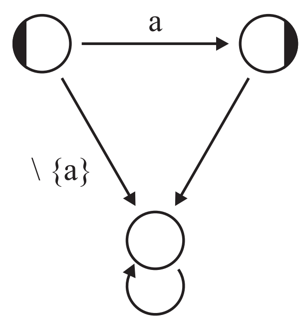
  $L = 1^*0^* \cup 1^*0^*(0111^*0^*)^*01(11^*0^* \cup \emptyset^*)$

1.4.2 Zusammenhang zu DFA-Sprachen

Satz: L ist genau dann eine reguläre Sprache, wenn L DFA-Sprache ist.

Beweis ($\Rightarrow$): Man konstruiert einen DFA.

- $L = \emptyset$ oder $L = \{a\}$



- L ist Vereinigung, Konkatenation oder Stern.

$$\left.\begin{array}{lcl} L = L_1 \cup L_2 & \rightarrow & L(M) = L(M_1) \cup L(M_2) \\ L = L_1 \circ L_2 & \rightarrow & L(M) = L(M_1) \circ L(M_2) \\ L = L_1^* & \rightarrow & L(M) = L(M^*) \end{array}\right\} \text{(siehe 1.2.5 und 1.2.10)}$$

Beweis ($\Leftarrow$): Sei L sei eine DFA-Sprache mit $Q = \{1, \ldots n\}$ und dem Anfangszustand 1. Man konstruiert einen regulären Ausdrucks für L mit dynamischem Programmieren ($1 \leq i, j \leq n$ und $k \geq 0$):

$$R_{i,j}^k = \{w \in \Sigma^* \mid \hat{\delta}(i, w) = j \text{ und alle Zwischenzustände} \leq k\}$$

$R_{i,j}^k$ wird schrittweise aufgebaut $k = 0, 1, 2, \ldots$

- keine Zwischenzustände: $R_{i,j}^0$
  - Für $i \neq j$ $R_{i,j}^0 = \{a \in \Sigma \mid \delta(i, a) = j\}$
  - Für $i = j$ $R_{i,i}^0 = \{a \in \Sigma \mid \delta(i, a) = i\} \cup \{\varepsilon\}$
- induktiver Aufbau: $R_{i,j}^{k-1} \rightarrow R_{i,j}^k$

$$R_{i,j}^k = R_{i,j}^{k-1} \cup R_{i,k}^{k-1} \left(R_{k,k}^{k-1}\right)^* R_{k,j}^{k-1}$$

Alle $R_{i,j}^k$ sind reguläre Ausdrücke. Zur Bestimmung der regulären Sprache von $L(M)$, müssen alle regulären Ausdrücke vom Anfangszustand zu allen akzeptierenden Zuständen vereinigt werden:

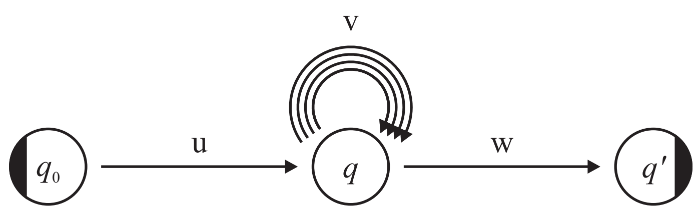
$$L(M) = \bigcup_{j \in F} R_{1,j}^n$$

1.4.3 Pumping-Lemma

Pumping-Lemma: Ist L eine reguläre Sprache, dann gibt es ein $n \in \mathbb{N}$, so dass für jedes $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvw$ existiert mit $|uv| \leq n$ und $|v| \geq 1$ und $uv^iw \in L$ für alle $i \in \mathbb{N}$.

Beweis: Sei M Minimalautomat für die Sprache L mit n Zuständen und z ein Wort aus L mit $|z| \geq n$.
Wir betrachnen die Bearbeitung der ersten n Symbole von z. Dabei treten $n+1$ Zustände auf. Nach dem Schubfachprinzip gilt: Mindestens ein Zustand q tritt doppelt auf.

$$z = u\ v^i\ w \quad (\text{mit } i \geq 1)$$



Anwendungen:

1. $L = \{0^n1^n \mid n \in \mathbb{N}\}$ ist *nicht* regulär.

   Angenommen L ist regulär. Dann gibt es ein n aus dem Pumping-Lemma.
   Man nimmt das Wort $z = 0^n1^n \in L$ mit $|z| \geq n$.
   Pumping-Lemma: $z = uvw$, $|uv| \leq n$ und $|v| \geq 1$
   $\Rightarrow u = 0^i$, $v = 0^j$ mit $i + j \leq n$ und $j \geq 1$
   $\Rightarrow uv^2w = 0^{n+j}1^n \notin L$ Widerspruch!
   $\Rightarrow L$ ist *nicht* regulär.

2. $L = \{w \in \{0{,}1\}^* \mid w^R = w\}$ (Sprache der Palindrome) ist *nicht* regulär.

   Angenommen L ist regulär. Dann gibt es ein n aus dem Pumping-Lemma.
   Man nimmt das Wort $z = 0^n10^n \in L$ mit $|z| \geq n$.
   Pumping-Lemma: $z = uvw$, $|uv| \leq n$, $|v| \geq 1$
   $\Rightarrow u = 0^i$, $v = 0^j$ mit $j \geq 1$
   $\Rightarrow uv^2w = 0^{n+j}10^n \notin L$ Widerspruch!
   $\Rightarrow L$ ist *nicht* regulär.

Achtung: Nicht in jedem Fall kann man die Nicht-Regularität einer Sprache mit dem Pumping-Lemma nachweisen. Das heißt, es gibt nicht reguläre Sprachen, die das Pumping-Lemma erfüllen.

1.5 Zusammenfassung

$$L \text{ ist regulär} \quad \begin{cases} \Leftrightarrow \quad L \text{ ist DFA-Sprache} \\ \Leftrightarrow \quad \Sigma^*_{/R_L} \text{ ist endlich} \\ \Leftrightarrow \quad \overline{L} \text{ ist regulär } (\overline{L} = \{w \in \Sigma^* \mid w \notin L\}) \\ \Leftrightarrow \quad L^R \text{ ist regulär} \end{cases}$$

$$L \text{ ist regulär} \quad \begin{cases} \Rightarrow \quad L^* \text{ ist regulär} \\ \Rightarrow \quad L \text{ erfüllt das Pumping-Lemma} \end{cases}$$

$$L_1 \text{ und } L_2 \text{ sind regulär} \quad \begin{cases} \Rightarrow \quad L_1 \circ L_2 \text{ sind regulär} \\ \Rightarrow \quad L_1 \cup L_2 \text{ sind regulär} \\ \Rightarrow \quad L_1 \cap L_2 \text{ sind regulär} \end{cases}$$

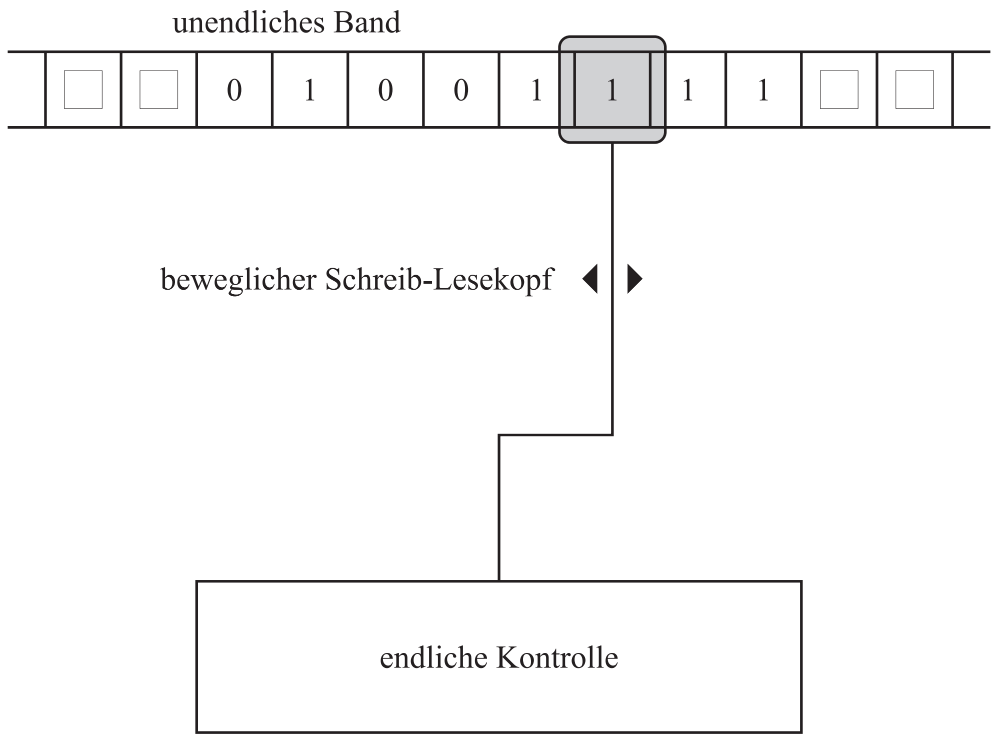
Kapitel 2

Berechenbarkeit

2.1 Turing-Maschinen

2.1.1 Einleitung

Idee: Statte einen DFA mit der Fähigkeit aus, Informationen zu speichern, zu lesen und zu löschen.



Definition: Eine *Turing-Maschine* (TM) ist gegeben durch ein 7-Tupel.

$$M = (q, \Sigma, \Gamma, \delta, q_0, \Box, F)$$

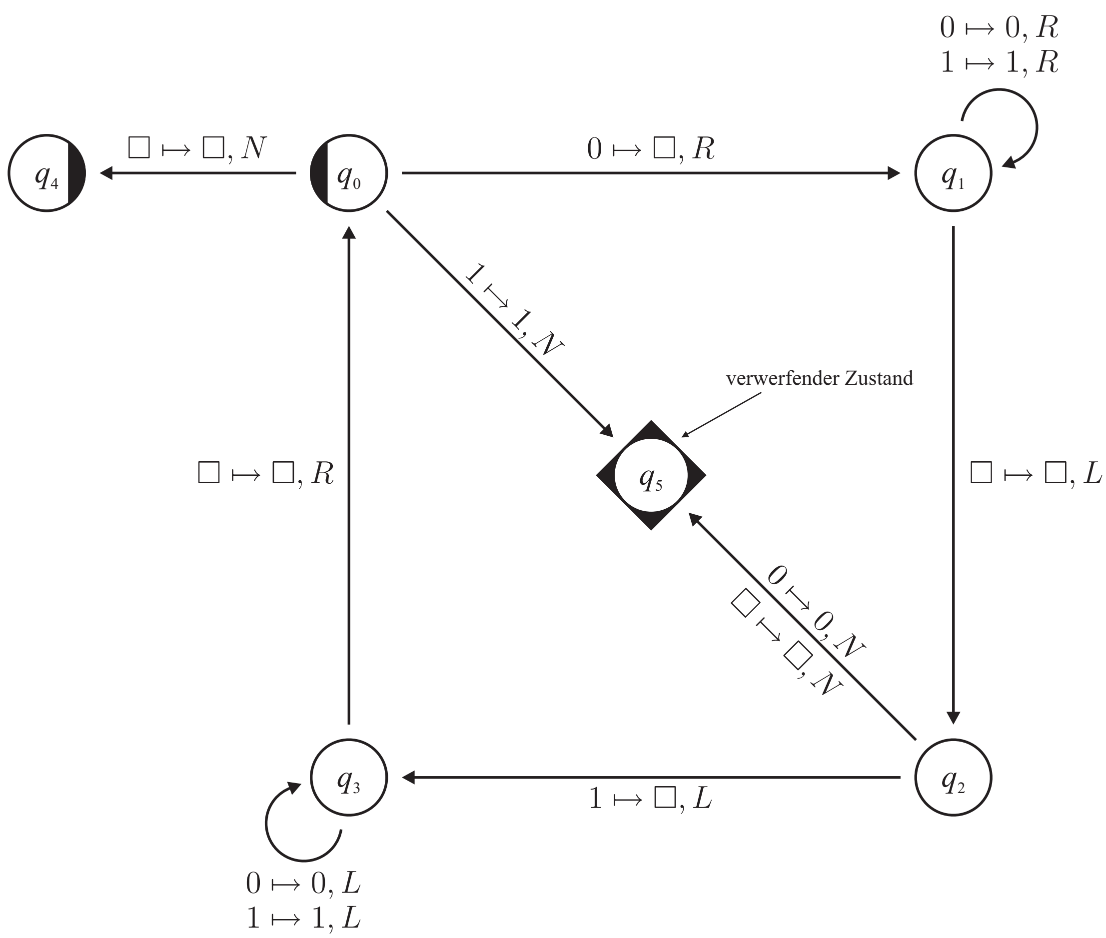
- Q: endlicher Zustandsmenge
- Σ: Eingabealphabet
- Γ: Arbeitsalphabet
- δ: Überführungsfunktion

$$\begin{array}{ccccccccccc}
\delta & : & Q & \times & \Gamma & \hookrightarrow & Q & \times & \Gamma & \times & \{L, R, N\} \\
 & & \uparrow & & \uparrow & & \uparrow & & \uparrow & & \uparrow \\
 & & \substack{\text{alter}\\\text{Zustand}} & & \text{Lesen} & & \substack{\text{neuer}\\\text{Zustand}} & & \text{Schreiben} & & \substack{\text{Kopf-}\\\text{bewegung}}
\end{array}$$

- q_0: Anfangszustand
- $\Box$: Leerzeichen (Blank; $\Box \in \Gamma \setminus \Sigma$)
- F: Endzustände
  - Einteitung in akzeptierende und verwerfende möglich
  - Überführungsfunktion δ muss für $q \in F$ nicht definiert sein

Beispiel: Turing-Maschine für $L = \{0^n1^n \mid n \in \mathbb{N}\}$

```
Solange Band nicht leer
    Lösche eine Null
    Wenn nicht möglich: Verwerfen
    Lösche eine Eins
    Wenn nicht möglich: Verwerfen
Akzeptieren
```



	0	1	$\square$
q_0	$q_1,\ \square,\ R$	$q_5,\ 1,\ N$	$q_4,\ \square,\ N$
q_1	$q_1,\ 0,\ R$	$q_1,\ 1,\ R$	$q_2,\ \square,\ L$
q_2	$q_5,\ 0,\ N$	$q_3,\ \square,\ L$	$(q_5,\ \square,\ N)$
q_3	$q_3,\ 0,\ L$	$q_3,\ 1,\ L$	$q_0,\ \square,\ R$
akzeptieren q_4	–	–	–
verwerfen q_5	–	–	–

2.1.2 Konfiguration

Definition: Eine *Konfiguration* ist eine Momentaufnahme der Arbeit einer Turing-Maschine charakterisiert durch Bandinhalt (ohne $\Box$), Zustand $q \in Q$ und Kopfposition.

Eine Konfuguration wird repräsentiert durch $\alpha\ q\ \beta \in \Gamma^*\ Q\ \Gamma^*$.

- α: Bandinhalt links von Kopfposition
- q: aktueller Zustand
- β: Bandinhalt ab Kopfposition

Die Folgekonfiguration k' einer Konfiguration $k = \alpha\ q\ \beta$ ist bei $\alpha = \alpha'\ a$ und $\beta = b\ \beta'$ folgermaßen charakterisiert:

$$k' = \begin{cases} \alpha'\ q'\ a\ b'\ \beta' & \text{falls} \quad \delta(q,b) = (q', b', L) \\ \alpha\ b'\ q'\ \beta' & \text{falls} \quad \delta(q,b) = (q', b', R) \\ \alpha\ q'\ b'\ \beta' & \text{falls} \quad \delta(q,b) = (q', b', N) \end{cases}$$

Schreibweise:

- $k \vdash k'$ heißt k' ist Folgekonfiguration von k
- $k \overset{*}{\vdash} k'$ heißt $k = k_0 \vdash k_1 \vdash \ldots \vdash k_m = k'$

Definition: $\varepsilon\ q_0\ w$ mit $w \in \Sigma^*$ ist Startkofiguration.

2.1.3 Berechnung und Entscheidung

Definition: Die *Berechnung* einer Turing-Maschine ist eine Konfigurationsfolge

$$k_0 \vdash k_1 \vdash k_2 \vdash k_3 \vdash \ldots$$

die mit der Startkonfiguration beginnt und entweder unendlich ist oder mit einer Endkonfiguration aufhört, für die keine Folgekonfiguration definiert ist.

Bei einer Endkonfiguration ist der Bandinhalt definiert als Ausgabe.

Definition: Während bei einer Berechnung aus einer Eingabe entweder das gesamte Band als Ausgabe folgt (sofern die Turing-Maschine in keine Endlosschleife gerät), folgt bei der *Entscheidung* aus der Eingabe entweder die Ausgabe 0 (verwerfen) oder 1 (akzeptieren).

- Akzeptierender Endzustand: Band löschen und 1 schreiben
- Verwerfender Endzustand: Band löschen und 0 schreiben
- unendliche Berechnungen: Ausgabe nicht definiert

2.1.4 Codierung endliche Informationsmengen in Q

Durch Erweiterung der Zustandsmenge Q mit einem k-stelligen Bitvektor, kann man in jedem Zustand eine fest begrenzte Informationsmenge speichern.

$$Q' = Q \times \underbrace{\{0,1\} \times \{0,1\} \times \ldots \times \{0,1\}}_{k \text{ Bit}}$$

2.1.5 Erweiterung und Reduktion des Bandalphabets

Satz: Das Bandalphabet Γ einer Turing-Maschine kann beliebig zu Γ' erweitert werden.

$$\Gamma \quad \Rightarrow \quad \Gamma' \qquad (\text{mit } \Gamma \subset \Gamma')$$

Satz: Jede Turing-Maschine M mit Bandalphabet Γ kann durch eine Turing-Maschine M' mit $\Gamma' = \{0,1\}$ „simuliert" werden.

Definition: *Simulation* ist die Codierung der Konfigurationen von M in Konfigurationen von M'.

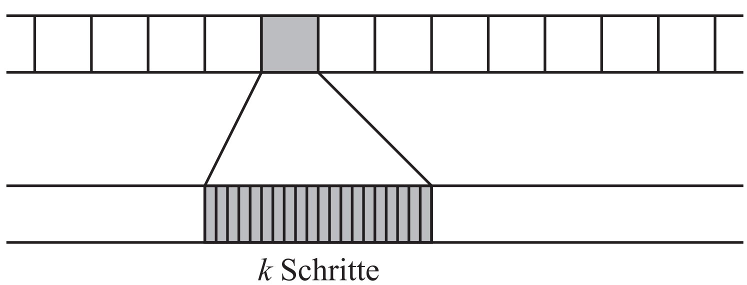
$$k \quad \mapsto \quad c(k)$$

Jeder Schritt von M wird durch endlich viele Schritte von M' simuliert.

$$k \vdash k' \quad \mapsto \quad c(k) \vdash c_1 \vdash c_2 \vdash \ldots \vdash c_m \vdash c(k')$$

Beweis:

- Codierung von Γ in $\{0,1\}^k$ wobei $k = \lceil \log_2 |T| \rceil$. Jede Zelle von M wird durch einen Block von k Zellen auf dem Band von M' dargestellt.



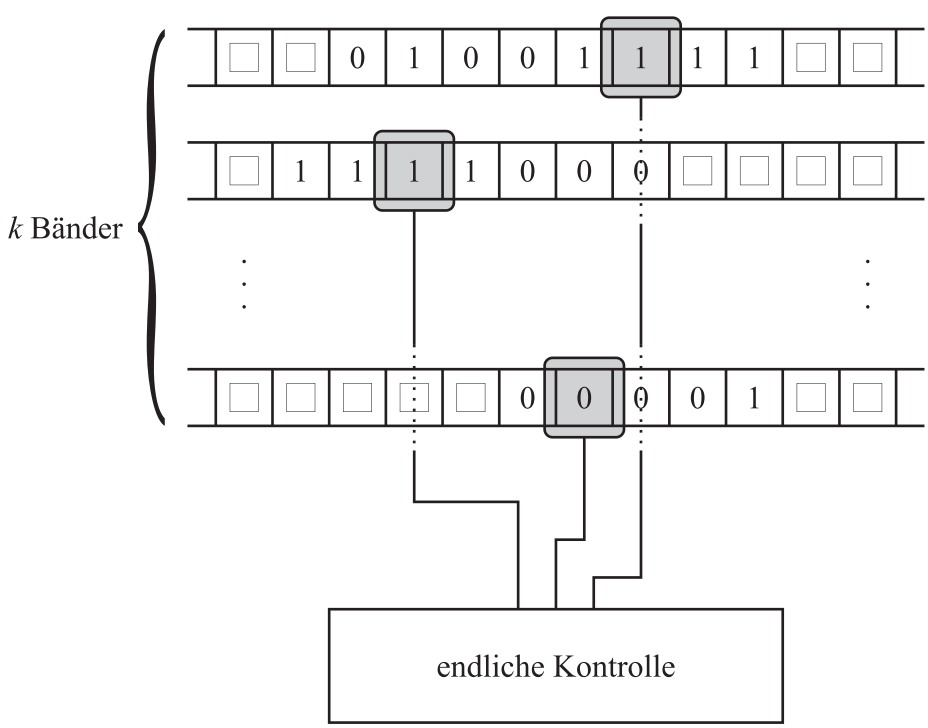
- Simulation eines Schritts von M:
  - M' steht auf erster von k Zellen, welche die aktuelle Zelle von M darstellen.
  - M' geht $k-1$ Schritte nach rechts und „speichert“ Bandinformation (siehe 2.5.2).
  - M' führt intern Übergangsfunktion von M aus. M' kennt neuen Zustand, Schreibanweisung und Kopfbewegung von M.
  - M' geht $k-1$ Schritte nach links und führt Schreibanweisung aus.
  - wenn M den Kopf nach links (rechts) bewegt, macht M' k Bewegungen nach nach links (rechts).

  Simulationszeit für einen Schritt: $3k-2$ (konstant)

2.1.6 Mehrbandmaschinen

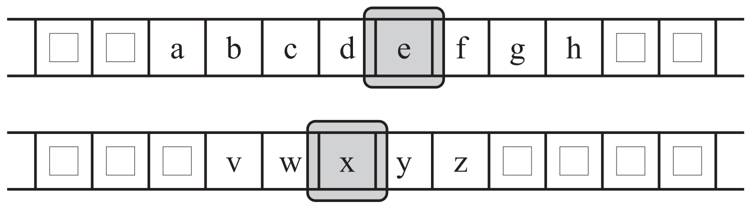
Eine Mehrbandmaschine kann eine endliche Anzahl von Bändern unabhängig voneinander lesen und beschreiben sowie den Schreib-Lesekopf bewegen.

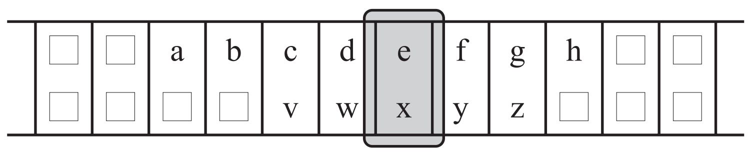
$$\delta : Q \times \Gamma^k \hookrightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$



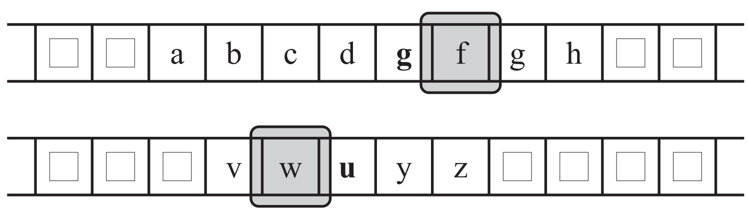
Satz: Jede k-Band-Turing-Maschine M kann durch eine Ein-Band-Turing-Maschine M' simuliert werden.

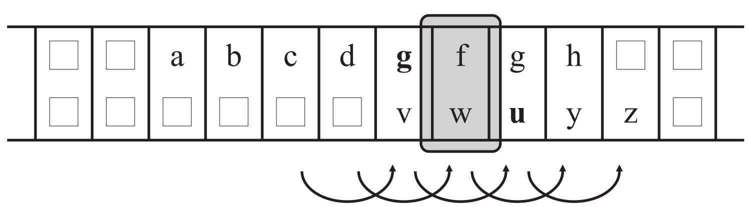
Beweis: M' simuliert k Bänder durch k Spuren auf einem Band $\Gamma' = \Gamma^k$.





Sofern sich jedoch die Köpfe von M auf den einzelnen Bändern in unterschiedliche Richtungen bewegen, werden die Spuren 2 bis k rekonfiguriert, so dass alle Kopfpositionen wieder übereinander liegen.





Simulationszeit: $2 \cdot l$ wobei $l =$ Länge des beschriebenen Bandes

Satz: Jede Berechnung einer k-Band-Turing-Maschine M der Länge t ($t \geq$ Eingabengröße) kann in $c \cdot t^2$ Zeit (c ist konstant) durch eine Ein-Band-Turing-Maschine simuliert werden.

2.1.7 Verwendung von Unterprogrammen

Die Turing-Maschine M soll ein Unterprogramm (Prozedur) nutzen, das von der Turing-Maschine M' ausgeführt wird:

- M bekommt zusätzliches Band und schreibt darauf die Übergabeparameter.
- M' rechnet nur auf dem Zusatzband.
- M kann Ergebnis auf Zusatzband lesen.

Schleifen werden realisiert als Unterprogramme mit Abbruchbedingung.

2.1.8 Verkettete Funktionen

Sei $f : \Sigma^* \to \Gamma^*$ die von M berechnete Funktion.
Sei $g : \Gamma^* \to \Lambda^*$ die von M' berechnete Funktion.
Die Maschine $M; M'$ arbeitet wie folgt:

- Sie führt alle Schritte so wie M aus.
- Falls M stoppt, geht sie auf linkeste beschriebene Stelle des Bandes
- M' startet.

$M; M'$ berechnet die Funktion $gf : \Sigma^* \to \Lambda^*$

2.1.9 Strukturierung von Ein- und Ausgaben

Zur Strukturierung von Ein- und Ausgaben wird das Trennsymbol # verwendet.

Beispiel: z.B. $bin(x)\#bin(y)$ als Eingabe für Addition

2.2 Church'sche These

Definition: Eine Relation $f \subseteq A \times B$ beschreibt eine *partielle Funktion* von A nach B ($f : A \hookrightarrow B$), wenn jedes $a \in A$ zu höchstens einem $b \in B$ in Relation steht ($f(a) = b$ falls solch ein b existiert).

Definition: Jede Turing-Maschine M berechnet eine partielle Funktion $f_M : \Sigma^* \hookrightarrow \Gamma^*$, wobei $f_M(w)$ nicht definiert ist, wenn M bei Eingabe w nicht hält, und der Bandinhalt am Ende der Berechnung $f_M(w)$ ist. Eine Funktion f_M heißt *Turing-berechenbar*.

Beobachtung: ($f : \Sigma^* \hookrightarrow \Gamma^*$)

f ist Turing-berechnenbar	$\Leftrightarrow$	f ist μ-rekursiv
	$\Leftrightarrow$	f ist berechenbar mit λ-Kalkül
	$\Leftrightarrow$	f ist berechenbar durch Registermaschinen (von Neumann-Rechner)
	$\Leftrightarrow$	...

Church'sche These: Die durch die formalen Definitionen der Turing-Berechenbarkeit erfasste Klasse von Funktionen stimmt mit der Klasse der *intuitiv berechnbaren Funktionen* überein.

2.3 Primitiv und μ-rekursive Funktionen

2.3.1 Einführung

Definition: Die Klasse der *primitiv rekursiven Funktionen* (kurz: PRF; $f : \mathbb{N}^k \to \mathbb{N}$) wird folgermaßen gebildet:

- Basisfunktionen:

  1. konstante Funktionen ($c, j \in \mathbb{N}$)

  $$f(x_1, \ldots x_j) = c$$

  2. Projektion ($i, j \in \mathbb{N}$)

  $$p_{i,j}(x_1, \ldots x_j) = x_i$$

  3. Nachfolgerfunktion

  $$s(x) = x + 1$$

- Operationen:

  1. Funktionskomposition ($j, k \in \mathbb{N}$ und $g_1, \ldots g_k, h$ sind PRF)

  $$f(x_1, \ldots x_j) = h(g_1(x_1, \ldots x_j), \ldots g_k(x_1, \ldots x_j))$$

  2. Primitive Rekursion ($j \in \mathbb{N}$ und g, h sind PRF)

  $$\begin{aligned} f(0, x_1, \ldots x_j) &= g(x_1, \ldots x_j) \\ f(y+1, x_1, \ldots x_j) &= h(f(y, x_1, \ldots x_j), y, x_1, \ldots x_j) \end{aligned}$$

Beispiele:

1. Addition

$$\begin{aligned} add : \mathbb{N}^2 &\rightarrow \mathbb{N} \\ add(0,x) &= p_{1,1}(x) \\ add(y+1,x) &= s(p_{1,3}(add(y,x),y,x)) \\ &= s(add(y,x)) \end{aligned}$$

2. Multiplikation

$$\begin{aligned} mul : \mathbb{N}^2 &\rightarrow \mathbb{N} \\ mul(0,x) &= 0 \\ mul(y+1,x) &= add(p_{1,3}(mul(y,x),y,x),p_{3,3}(mul(y,x),y,x)) \\ &= add(mul(y,x),x) \end{aligned}$$

3. Vorgängerfunktion

$$\begin{aligned} pred : \mathbb{N} &\rightarrow \mathbb{N} \\ pred(0) &= 0 \\ pred(y+1) &= p_{2,2}(pred(y),y) \end{aligned}$$

4. Prädikat ungleich 0

$$\begin{aligned} notnull : \mathbb{N} &\rightarrow \{0,1\} \\ notnull(0) &= 1 \\ notnull(y+1) &= 0 \end{aligned}$$

5. Subtraktion (erstes Argument: Minuend, zweites Argument: Subtrahend)

$$\begin{aligned} sub : \mathbb{N}^2 &\rightarrow \mathbb{N} \\ sub(0,x) &= x \\ sub(y+1,x) &= pred(p_{1,3}(sub(y,x),y,x)) \\ &= pred(sub(y,x)) \end{aligned}$$

6. Prädikat kleiner

$$\begin{aligned} less : \mathbb{N}^2 &\rightarrow \{0,1\} \\ less(x,y) &= notnull(sub(x,y)) \end{aligned}$$

2.3.2 μ-Operator

Es gibt zwei grundsätzliche Programmierparadigmen:

- Funktionaler Ansatz
  - Primitiv rekursive Funktionen
  - Eigenschaft: total, d.h. überall definiert
- Imperativer Ansatz
  - Programm mit elementaren Operationen und *statischen* Schleifen
  - Eigenschaft: terminiert immer

Weder mit primitiv rekursiven Funktionen noch mit imperativen Programmen, die nur statische Schleifen haben, lassen sich alle Turing-berechenbaren Funktion realisieren. Jedoch lassen sich beiden Ansätze dahingehend erweitern:

- Funktionaler Ansatz
  - μ-rekursive Funktionen
  - Eigenschaft: nur partielle Funktionen
- Imperativer Ansatz
  - Programm mit elementaren Operationen und `while`-Schleifen oder bedingten Sprunganweisungen
  - Eigenschaft: terminiert *nicht* immer

Definition: Die μ-Operator ist folgendermaßen definiert:

$$\begin{aligned} g &= \mu(f) \\ g(x_1, \ldots x_k) &= \min\{n \mid f(n, x_1, \ldots x_k) = 0 \text{ und} \\ &\qquad \forall m < n \text{ ist } f(m, x_1, \ldots x_k) \text{ definiert}\} \end{aligned}$$

Die Klasse der μ-rekursiven Funktionen ist die kleinste Klasse von partiellen Funktionen, die alle primitiv rekursiven Funktionen enthält und abgeschlossen ist bezüglich der Anwendung des μ-Operator.

Beispiel: Es gibt totale Funktionen, die μ-rekursiv, aber nicht primitiv rekursiv sind, z.B. die Ackermann-Funktion.

$$\begin{aligned} a : \mathbb{N}^2 &\to \mathbb{N} \\ a(0, y) &= y + 1 \\ a(x, 0) &= a(x - 1, 1) \\ a(x, y) &= a(x - y, a(x, y - 1)) \end{aligned}$$

2.4 Entscheidbarkeit

2.4.1 Charakteristische Funktion

Definition: Die *charakteristische Funktion* $\chi_L : \Sigma^* \to \{0,1\}$ und die „halbe“ charakteristische Funktion $\chi_L^* : \Sigma^* \hookrightarrow \{0,1\}$ einer Sprache $L \subseteq \Sigma^*$ sind folgendermaßen definiert:

$$\chi_L(x) = \begin{cases} 1 \text{ falls } x \in L \\ 0 \text{ falls } x \notin L \end{cases}$$

$$\chi_L^*(x) = \begin{cases} 1 & \text{falls } x \in L \\ \text{undefiniert} & \text{falls } x \notin L \end{cases}$$

2.4.2 Entscheidbarkeit und Semi-Entscheidbarkeit

Definitionen:

- Eine Sprache L ist *entscheidbar* (rekursiv), wenn die charakteristische Funktion χ_L Turing-berechenbar ist.
- Eine Sprache L ist *semi-entscheidbar*, wenn die „halbe“ charakteristische Funktion χ_L^* Turing-berechenbar ist.
- Eine Sprache L ist *rekursiv-aufzählbar*, wenn L das Bild einer totalen Turing-berechenbaren Funktion ist.

Satz: Eine Sprache L ist genau dann entscheidbar, wenn L und $\overline{L} = \Sigma^* \backslash L$ semi-entscheibar sind.

Beweis ($\Rightarrow$): Da die Sprache L entscheidbar ist, existiert eine charakteristische Funktion $\chi_L = f_M$ für eine Turing-Maschine M.

- Man konstrutiert zwei Turing-Maschinen M' uns M'' für L und $\overline{L}$.
- M' bzw. M'' arbeiten zuerst wie M.
- Sobald M anhält, verhalten sich M' und M'' folgendermaßen:
  - Falls M akzeptiert, dann akzeptiert auch M' (Symbol 1 auf das Band schreiben) und M'' geht in eine Endlosschleife
  - Falls M verwirft, dann akzeptiert M'' (Symbol 1 auf das Band schreiben) und M' geht in eine Endlosschleife.

Die „halben“ charakteristischen Funktionen χ_L^* und $\chi_{\overline{L}}^*$ sind die von den Turing-Maschinen M' und M'' berechnete Funktionen:

$$\begin{aligned} \chi_L^* &= f_{M'} \\ \chi_{\overline{L}}^* &= f_{M''} \end{aligned}$$

Beweis (⇐): Da die Sprachen L und $\overline{L}$ semi-entscheidbar sind, existieren für beide Sprachen die „halben" charakteristischen Funktionen $\chi_L^* = f_{M'}$ und $\chi_{\overline{L}}^* = f_{M''}$ für die Turing-Maschinen M' und M''.

- Man konstruiert eine 2-Band-Turing-Maschine M.
- Die Turing-Maschine M kopiert zuerst die Eingabe auf das zweite Band.
- Anschliend wird die Maschine M' auf dem ersten Band und M'' auf dem zweiten Band simuliert.
- Die gesamte Maschine M stoppt, wenn die simulierte Maschine M' oder die simulierte Maschine M'' stoppt.
- Falls M' gestoppt hat, dann akzeptiert M (Symbol 1 auf das Band schreiben); falls M'' gestoppt hat, dann verwirft M (Symbol 0 auf das Band schreiben).

Die charakteristische Funktion χ_L ist die von der Turing-Maschine M berechnete Funktion:

$$\chi_L = f_M$$

Satz: Die Sprache L ist genau dann rekursiv-aufzählbar, wenn L semi-entscheidbar ist.

Beweis (⇒): Die Sprache L ist rekursiv-aufzählbar. Damit ist $L = \text{Im}(f_M)$, wobei f_M eine totale Funktion ist, die von M berechenbar ist.

Man konstruiert nun eine Maschine M', mit dem Ziel $f_{M'} = \chi_L^*$, denn damit wäre L semi-entscheidbar.

Die Eingabe für M' sei w und M' soll akzeptieren, wenn $w \in L$.

- Die Maschine M' generiert nacheinander alle Eingaben u für M ($u \in \{\varepsilon, 0, 1, 00, 01, 10, 11, \ldots\}$).
- Für jedes von M' generierte u, wird über eine Simulation von M der Wert $v = f_M(u)$ berechnet und mit w verglichen:
  - Ist $v = w$, dann akzeptiert M' und stoppt.
  - Ist $v \neq w$, dann wiederholt M' die Berechnung für das nächste u.

Die „halbe" charakteristische Funktion χ_L^* ist die von der Turing-Maschine M' berechnete Funktion f_M, denn die Maschine M hält nur, wenn sie eine Übereinstimmung der Eingabe w mit einem Element $v = f_M(u)$ aus dem Bild $\text{Im}(f_M)$ der totalen Funktion f_M findet. Damit ist L semi-entscheidbar.

$$\chi_L^* = f_{M'}$$

Beweis ($\Leftarrow$): Die Sprache L ist semi-entscheidbar. Damit existiert eine „halbe“ charakteristische Funktion $\chi_L^* = f_{M'}$.

Ziel ist es, eine Maschine M mit der totalen Funktion f_M zu finden, so dass $L = \text{Im}(f_M)$. Die Eingabe für M sei w.

- Man nutzt folgende Bijektion: $\varphi : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$.

	0	1	2	3
0	0	2	5	9
1	1	4	8	13
2	3	7	12	18
3	6	11	17	24

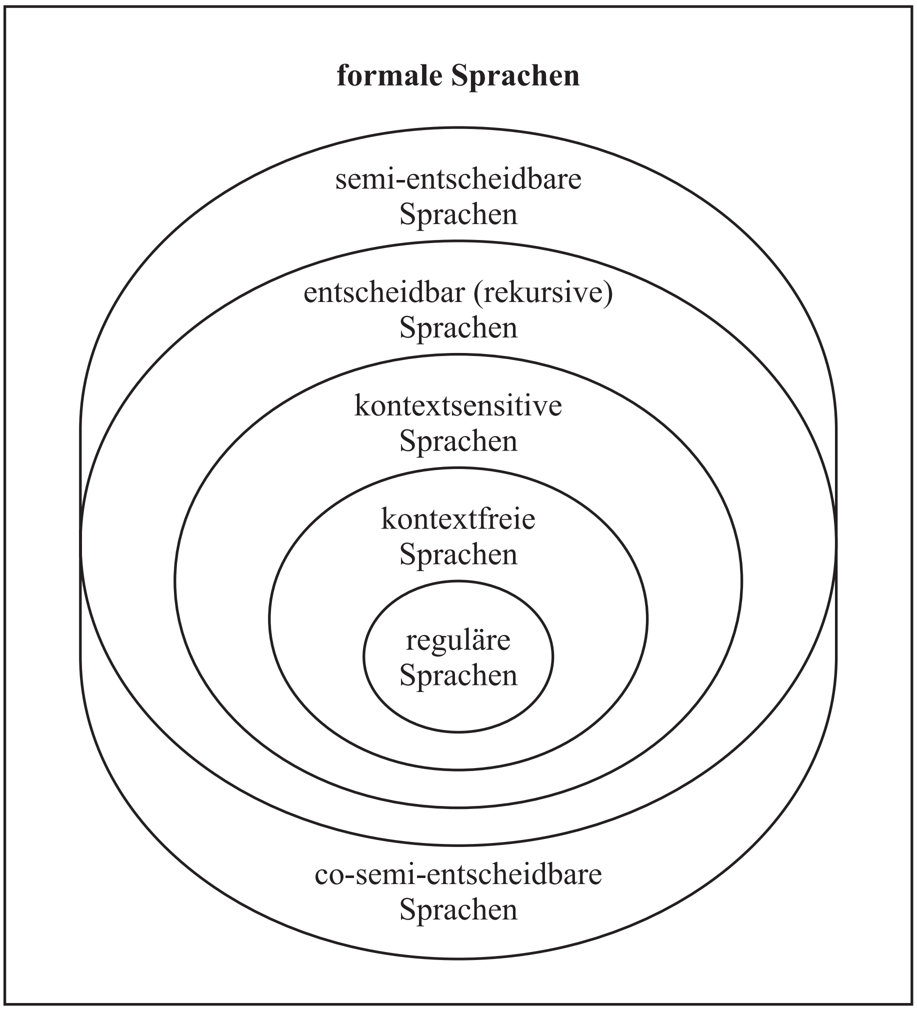
- Man setzt voraus, dass $L \neq \emptyset$ und dass Maschine M ein Wort $v \in L$ kennt.
- M interpretiet die Eingabe $w \in \Sigma^*$ als $n \in \mathbb{N}$ und berechnet $\varphi(n) = (n_1, n_2)$.
- M interpretiert $bin(n_1)$ ohne die erste 1 als Eingabe von M' und simuliert n_2 Schritte von M'. Durch die Begrenzung der Rechenschritte wird verhindert, dass M' in eine Endlosschleife gerät. Dies ist wichtig, denn die von M berechenbare Funktion muss total sein.
- Hat M' das Wort $bin(n_1)$ in n_2 Schritten akzeptiert, dann ist $bin(n_1) \in L$, M gibt $bin(n_1)$ aus und stoppt. Sonst gibt M das bekannte Wort v aus und stoppt.

Bemerkung: Wenn $bin(n_1)$ in n_2 Schritten von M nicht akzeptiert worden ist, heißt es *nicht*, dass $z = bin(n_1)$ nicht in L ist. Für ein größeres n wiederholt sich aufgrund der Definition von φ das Wort z mit einem größeren n_2, so dass die Zugehörigkeit von z zu L mit mehr Schritten überprüft werden kann.

Behauptung: Das Bild der von M berechnete Funktion f_M ist L: Ist ein Wort $w \in L$, dann liegt es im Bild von f_M und kann von M nach n Wiederholungen akzeptiert werden, wobei $\varphi(n) = (1w, m)$. Die Maschine M braucht m Schritte, um w zu akzeptieren.

Folgerungen:

- Ist L semi-entscheidbar, aber nicht entscheidbar, dann ist $\overline{L}$ nicht semi-entscheindbar.
- L ist genau dann co-semi-entscheidbar, wenn $\overline{L}$ semi-entscheidbar ist.



2.5 Unentscheidbarkeit

2.5.1 Einleitung

Das Ziel ist es zu zeigen, dass es unentscheidbare Sprachen gibt. Das soll mit Hilfe von Diagolanisierung gezeigt werden (wie beim Beweis, dass *keine* Menge A existiert, so dass $|A| = |\mathcal{P}(A)|$). Dazu wird eine Codierung einer Turing-Maschine als Eingabe für eine Turing-Maschine benutzt.

2.5.2 Codierung von Turing-Maschinen

Eine Codierung $\langle M \rangle$ für eine Turing-Maschine M soll folgendermaßen aussehen:

- Alphabet: $\Sigma = \{0, 1\}$ (o.B.d.A.)
- Zustände: $Q = \{1, \dots n\}$; Startzustand: 1
- Arbeitsalphabet: $\Gamma = \{a_1, \dots a_m\}$
- Zustandsübergangsfunktion: δ ist eine Folge von $n \cdot m$ Befehlssätzen:

$$\#bin(n)\#bin(m)\#bin(k_1)\#bin(k_2)\# \dots \#bin(k_{n \cdot m})$$

  Jedes k_i repräsentiert eine Zuweisung:

$$\delta(p, a_j) = (q, a_k, K)$$

Satz: Die Sprache C aller gültigen Codierungen für Turing-Maschinen ist entscheidbar.

$$C = \{w \in \{0, 1\}^* \mid w \text{ ist gültige Codierung einer TM}\}$$

Beweis: Zuerst muss überprüft werden, ob die Codierung w mit zwei Blöcken beginnt:

$$\#bin(n)\#bin(m)\#$$

Ist dies nicht erfüllt, dann wird das Wort w verworfen.

Ansonsten wird überprüft, ob $m \cdot n$ weitere Blöcke folgen. Wenn dies nicht der Fall ist, wird w verworfen.

Ansonsten wird jeder einzelne Block auf Korrekkeit geprüft. Wenn alle Blöcke korrekt codiert sind, wird w akzeptiert, sonst wird w verworfen.

2.5.3 Universelle Turing-Maschine

Satz: Es gibt eine universelle Turing-Maschine M_U, die bei einer Eingabe $\langle M\rangle\#w$ die Arbeit von M auf w simuliert.

Beweis: Man konstruiert eine 3-Band-Turing-Maschine:

- Band 1: Band von M
- Band 2: Kopie von $\langle M\rangle$
- Band 3: Hilfsband (aktueller Zustand q von M)

Die Simulation eines Schrittes von M erfolgt folgendermaßen:

1. Auf Band 3 steht der Startzustand von M.
2. M_U liest auf Band 1 das Symbol a aus w.
3. Anschließend sucht M_U den Befehlssatz für $\delta(q, a)$ auf Band 2.
4. M_U führt die Anweisung auf Band 1 aus und schreibt den neuen Zustand q' auf Band 3.

2.5.4 Diagonalsprache

Definition: Eine Turing-Maschine bekommt als Eingabe w ihre eigene Codierung. Dieses Wort w soll sie *nicht* akzeptieren. Die Menge aller Codierungen solcher Turing-Maschinen nennt man *Diagonalsprache*.

$$D = \{w \in \{0,1\}^* \mid w = \langle M\rangle \text{ und } M \text{ akzeptiert } w \textit{ nicht}\}$$

Vorstellung: Es wird in einer Tabelle für alle Turing-Maschinen eingetragen, ob sie die Codierung jeder anderen Turing-Maschine oder sich selber akzeptieren oder verwerfen.

Eingabe \ TMs	M_1	M_2	M_3	M_4	$\cdots$
$\langle M_1\rangle$	akz.	akz.	verw.	akz.	$\cdots$
$\langle M_2\rangle$	akz.	**verw.**	verw.	verw.	$\cdots$
$\langle M_3\rangle$	verw.	akz.	**verw.**	akz.	$\cdots$
$\langle M_4\rangle$	verw.	akz.	akz.	**verw.**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Nun werden nur die Einträge in der Diagonalen betrachtet, in der jeder Turing-Maschine M_i eine Codierung $\langle M_i\rangle$ zugeordnet wird. Alle Codierungen $\langle M_i\rangle$, bei denen M_i die Codierung $\langle M_i\rangle$ verwirft, gehören zur Diagonalsprache D.

Satz: Die Diagonalsprache D ist nicht entscheidbar.

Beweis (indirekt): Angenommen D sei entscheidbar, dann existiert eine Turing-Maschine M_D, so dass die von ihr berechenbare Funktion f_{M_D} die charakteristische Funktion χ_D ist.

Man benutzt nun die Kodierung $\langle M_D \rangle$ als Eingabe für M_D:

- Angenommen M_D akzeptiert ihre Codierung $\langle M_D \rangle$, dann ist nach der Definition von D ihre Codierung $\langle M_D \rangle \notin D$. Da die von M_D berechnete Funktion $f_{M_D} = \chi_D$ die charakteristische Funktion von D ist, ergibt die Berechnung $\chi_D(\langle M_D \rangle) = 0$, was bedeutet, dass M_D die Codierung $\langle M_D \rangle$ *nicht* akzeptiert. Dies ist ein Widerspruch zur Annahme!

- Angenommen M_D akzeptiert ihre Codierung $\langle M_D \rangle$ *nicht*, dann ist nach der Definition von D ihre Codierung $\langle M_D \rangle \in D$. Damit ist $\chi_D(\langle M_D \rangle) = 1$. Also akzeptiert M_D die Codierung $\langle M_D \rangle$. Auch dies ist ein Widerspruch zur Annahme!

$$
\begin{array}{lcl}
M_D \text{ akzeptiert } \langle M_D \rangle & \Leftrightarrow & \langle M_D \rangle \notin D \\
 & \Leftrightarrow & \chi_D(\langle M_D \rangle) = f_{M_D}(\langle M_D \rangle) \neq 1 \\
 & \Leftrightarrow & M_D \text{ akzeptiert } \langle M_D \rangle \text{ nicht} \\
 & & \text{Widerspruch!}
\end{array}
$$

Daraus folgt: D ist nicht entscheidbar.

2.5.5 Spezielles Halteproblem

Definition: Das *spezielle Halteproblem* (Selbstanwendungsproblem) ist gegeben durch die Sprache Hs:

$$Hs = \{w \in \{0,1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält}\}$$

Die Turing-Maschine M_w ist definiert als:

$$M_w = \begin{cases} M & \text{falls } w = \langle M \rangle \text{ eine gültige Codierung ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Dabei ist $\hat{M}$ eine beliebige, aber fest gewählte Turing-Maschine.

Satz: Die Sprache Hs ist nicht entscheidbar.

Beweisidee: Angenommen Hs wäre entscheidbar, dann müsste D auch entscheidbar sein. Dies wäre ein Widerspruch.

Beweis (indirekt): Sei χ_{Hs} die charakteristische Funktion der Sprache Hs, dann konstruiert man eine Turing-Maschine M_D, so dass die von M_D berechnete Funktion f_{M_D} die charakteristische Funktion χ_D der Diagonalsprache D ist. Die Eingabe für M_D sei w.

1. M_D entscheidet, ob $w = \langle M \rangle$ eine gültige Codierung ist (siehe 2.5.2).
   - Wenn ja: M_D fährt fort
   - Wenn nein: M_D verwirft
2. M_D nutzt M_{Hs} und entscheidet, ob $M = M_w$ auf w hält
   - Wenn ja: M_D fährt fort
   - Wenn nein: M_D akzeptiert, da die Maschine M sich damit nicht akzeptieren kann und deswegen zu D gehört.
3. M_D nutzt die Universal-Turing-Maschine T_U, um die Berechnung von $M = M_w$ bei Eingabe w zu simulieren (es ist nun bekannt, dass M_w bei Eingabe w terminiert).
   - Wenn M_w die Eingabe w akzeptiert, verwirft M_D die Eingabe.
   - Wenn M_w die Eingabe w verwirft, akzeptiert M_D die Eingabe.

Damit existiert eine charakteristische Funktion $\chi_D = f_{M_D}$. Daraus folgt, dass die Diagonalsprache D entscheidbar ist. Dies ist ein Widerspruch!

Daraus folgt: Hs ist *nicht* entscheidbar.

2.6 Reduktionen

2.6.1 Definition der Reduktion

Definition: Es seien $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ Sprachen.

L_1 ist reduzierbar auf L_2, wenn eine totale berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ existiert, so dass für alle $w \in \Sigma_1^*$ gilt:

$$w \in L_1 \quad \Leftrightarrow \quad f(w) \in L_2$$

Schreibweise:

$$L_1 \leq L_2$$

Lemma: Ist $L_1 \leq L_2$ und ist L_2 entscheidbar (bzw. semi-entscheidbar), dann ist auch L_1 entscheidbar (bzw. semi-entscheidbar).

Beweis: Sei $L_1 \leq L_2$ mittels f und L_2 sei entscheidbar. Daraus folgt, dass die charakteristische Funktion χ_{L_2} berechenbar ist. Zu zeigen ist Berechenbarkeit der charakterischen Funktion χ_{L_1}.

Um zu entscheiden, ob ein Wort w in der Sprache L_1 liegt, wird $f(w)$ berechnet und entschieden, ob $f(w)$ in L_2 liegt. Wenn dies Fall ist, ist auch $w \in L_1$; wenn nicht, dann ist $w \notin L_2$.

$$\chi_{L_1} \quad = \quad \chi_{L_2} \circ f$$

$$\begin{aligned} \chi_{L_1}(w) = 1 \quad &\Leftrightarrow \quad w \in L_1 \\ &\Leftrightarrow \quad f(w) \in L_2 \\ &\Leftrightarrow \quad \chi_{L_2}(f(w)) = 1 \end{aligned}$$

Damit ist χ_{L_1} berechenbar. Daraus folgt, dass L_1 entscheidbar ist.

Für den Beweis für semi-entscheidbare Sprachen ist χ_{L_1} durch $\chi_{L_1}^*$ und χ_{L_2} durch $\chi_{L_1}^*$ zu ersetzen.

2.6.2 Allgemeines Halteproblem

Definition: Das *allgemeine Halteproblem* ist gegeben durch die Sprache H:

$$H = \{w\#x \mid M_w \text{ angesetzt auf } x \text{ hält}\}$$

Die Turing-Maschine M_w ist definiert als:

$$M_w = \begin{cases} M & \text{falls } w = \langle M \rangle \text{ eine gültige Codierung ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Satz: Die Sprache H ist nicht entscheidbar.

Beweis: Es ist zu zeigen, dass $Hs \leq H$. Wäre H entscheidbar, müsste Hs auch entscheidbar sein. Dies wäre ein Widerspruch.

Gesucht wird eine Funktion f für die Reduktion, so dass $w \in Hs \Leftrightarrow f(w) \in H$. Es wird folgede Zuweisung gewählt:

$$f(w) = w\#w$$

Probe:

$$w \in Hs \quad \Leftrightarrow \quad M_w \text{ hält auf } w \quad \Leftrightarrow \quad w\#w \in H$$

Damit ist Hs auf H reduzierbar. Da aber Hs nicht entscheidbar ist, ist auch H nicht entscheidbar.

2.6.3 Halteproblem auf leerem Band

Definition: Das *Halteproblem auf leerem Band* ist gegeben durch die Sprache H_0:

$$H_0 = \{w \mid M_w \text{ angesetzt auf } \varepsilon \text{ hält}\}$$

Satz: Die Sprache H_0 ist nicht entscheidbar.

Beweis: Es ist zu zeigen, dass $H \leq H_0$. Wäre H_0 entscheidbar, müsste H auch entscheidbar sein. Dies wäre ein Widerspruch.

Es wird folgende Funktion für die Reduktion gewählt: Man ordnet jedem Wort $w\#x$ eine Turing-Maschine zu, die bei Start auf leerem Band x auf das Band schreibt und sich dann wie M_w verhält. Man bezeichnet mit $f(w\#x)$ die Codierung dieser Maschine.

$$\begin{aligned} w\#x \in H \quad &\Leftrightarrow \quad M_w \text{ hält auf } x \\ &\Leftrightarrow \quad \text{die durch } f(w\#x) \text{ beschriebene Maschine hält auf leerem Band} \\ &\Leftrightarrow \quad f(w\#x) \in H_0 \end{aligned}$$

Damit ist H auf H_0 reduzierbar. Da aber H nicht entscheidbar ist, ist auch H_0 nicht entscheidbar.

2.6.4 Universalsprache

Definition: Die Universalsprache U ist folgendermaßen definiert:

$$U = \{w\#x \mid M_w \text{ angesetzt auf } x \text{ akzeptiert}\}$$

Die Turing-Maschine M_w ist folgendermaßen definiert, wobei hier $\hat{M}$ eine Turing-Maschine ist, die immer akzeptiert:

$$M_w = \begin{cases} M & \text{falls } w = \langle M \rangle \text{ eine gültige Codierung ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Satz: Die Sprache U ist nicht entscheidbar.

Beweis: Es ist zu zeigen, dass $\overline{D} \leq U$ und $\overline{D}$ nicht entscheidbar ist.

Die Sprache $\overline{D}$ ist nicht entscheidbar, da D nicht entscheidbar ist.

$$\overline{D} = \{w \in \{0,1\}^* \mid w \neq \langle M \rangle \text{ oder } (w = \langle M \rangle \text{ und } M \text{ akzeptiert } w)\}$$

Es wird folgende Funktion für die Reduktion gewählt:

$$f(w) = w\#w$$

Probe:

$$\begin{aligned} w \in \overline{D} &\Leftrightarrow w \neq \langle M \rangle \text{ oder } (w = \langle M \rangle \text{ und } M \text{ akzeptiert } w) \\ &\Leftrightarrow M_w = \hat{M} \text{ oder } M_w = M \text{ und } M_w \text{ akzeptiert } w \\ &\Leftrightarrow M_w \text{ akzeptiert } w \text{ (da } \hat{M} \text{ alles akzeptiert)} \\ &\Leftrightarrow w\#w \in U \end{aligned}$$

Damit ist $\overline{D}$ auf U reduzierbar. Da aber $\overline{D}$ nicht entscheidbar ist, ist auch U nicht entscheidbar.

Satz: Die Universalsprache U ist semi-entscheidbar.

Beweis: Es existiert eine Turingmaschine M, so dass die von M berechnete Funktion f_M die „halbe“ charakteristische Funktion χ_U^* der Universalsprache U ist:

1. Die Maschine M überprüft zuerst, ob die Eingabe die Form $w\#x$ hat
   - Wenn ja: M fährt fort
   - Wenn nein: M geht in eine Endlosschleife
2. M simuliert M_w mit der Eingabe x
3. Falls M_w hält, wird geprüft, ob M_w das Wort x akzeptiert hat
   - Wenn ja: M akzeptiert die Eingabe $w\#x$
   - Wenn nein: M geht in eine Endlosschleife
4. Hinweis: Sollte M_w nicht halten, kann M auch nicht halten.

Satz: Die Sprache $\overline{U}$ ist *nicht* semi-entscheidbar.

Beweis (indirekt): Angenommen $\overline{U}$ ist semi-entscheidbar. Da U semi-entscheidbar ist, würde aus der Semi-Entscheidbarkeit von U und $\overline{U}$ folgen, dass U entscheidbar ist. Dies wäre ein Widerspruch.

2.6.5 Postsches Korrespondenzproblem

Definition: Das *Postsche Korrespodenzproblem* ist gegeben durch die Sprache PCP:

$$PCP = \{(u_1, v_1), \ldots (u_k, v_k) \mid \exists i_1 \ldots i_n \quad u_{i_1} \circ \ldots \circ u_{i_n} = v_{i_1} \circ \ldots \circ v_{i_n}\}$$

$$\vdots$$
$$\vdots$$
$$\vdots$$

Vorlesung vom 12.6.2002 (fehlt)

$$\vdots$$
$$\vdots$$
$$\vdots$$

Kapitel 3

Komplexität

3.1 Die Komplexitätsklasse P

3.1.1 Komplexitätsklassen

Definition: Ist M eine Turing-Maschine und $w \in \Sigma^*$, dann bezeichnet $time_M(w)$ die Anzahl der Schritte die M bei der Eingabe w macht:

$$time_M : \Sigma^* \to \mathbb{N} \cup \{\infty\}$$

Definition: Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die *Komplexitätsklasse* $\mathsf{TIME}(f(n))$ besteht aus allen Sprachen L, für die eine Mehrband-Turing-Maschine M existiert, mit der Eigenschaft $L = L(M)$ und die Anzahl der Schritte $time_M(w)$, die M bei der Eingabe w macht, ist kleiner oder gleich $f(|w|)$ für alle $w \in \Sigma^*$. Insbesondere gilt, dass M immer hält.

$$\begin{aligned} \mathsf{TIME}(f(n)) \;\; = \;\; & \{L \subseteq \Sigma^* \mid \exists \text{ TM } M \text{ mit } L(M) = L \text{ und} \\ & \forall w \in \Sigma^* \; time_M(w) \leq f(|w|)\} \end{aligned}$$

Beispiel: Die Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N}$ ist folgendermaßen definiert:

$$s(n) = n + 1$$

Alle regulären Sprachen $L_{\text{regulär}}$ liegen in der Komplexitätsklasse $\mathsf{TIME}(s(n))$.

3.1.2 Polynomialzeitprobleme

Definition: Die Komplexitätsklasse P der „Polynomialzeitprobleme“ ist folgendermaßen definiert:

$$\begin{aligned} \mathsf{P} &= \{L \mid \exists \text{ TM } M \quad \exists \text{ Polynom } p(n) \\ &\quad\; \text{mit } L = L(M) \text{ und } \forall w \in \Sigma^* \; time_M(w) \leq p(|w|)\} \end{aligned}$$

$$\mathsf{P} = \bigcup_{p(n) \text{ Poylnom}} \mathsf{TIME}(p(n))$$

$$L \in \mathsf{P} \;\Leftrightarrow\; \exists \text{ TM } M \quad \exists k \in \mathbb{N} \quad \forall w \in \Sigma^* \quad L = L(M) \quad time_M(w) = \mathcal{O}(|w|^k)$$

Die Klasse P stimmt mit der Klasse der in Polynomialzeit entscheidbaren Probleme auf Registermaschinen überein, wenn alle Operationen mit logarithmischem Kostenmaß angerechnet werden (d.h. Kosten für eine Zuweisung, Addition etc. sind gleich der Anzahl der beteiligten Bits).

Erweiterte Church'sche These: Die Klasse P ist die Klasse der effizient berechenbaren Probleme.

Konsequenz: Zum Nachweis, dass $L \in \mathsf{P}$, reicht es aus, die Algorithmen (Pseudocode, Flussdiagramme etc.) zu analysieren.

Beispiele: Folgende Sprachen liegen in P:

- $\{p(x)\#n \mid p(x)$ ist rationales Polynom, $n \in \mathbb{N}$ und $p(n) = 0\}$

  Begründung: Horner-Schema

- $\{\langle$lineares Gleichungssystem$\rangle \mid$ das System hat eine Lösung$\}$

  Begründung: Gauss-Elimination

- $\{bin(a)\#bin(b)\#bin(c) \mid a, b, c, \in \mathbb{N}$ und $ggT(a, b) = c\}$

  Begründung: Euklidischer Algorithmus

- $\{\langle G\rangle \mid$ Graph G ist zusammenhängend$\}$

  Begründung: Tiefensuche (DFS) bzw. Breitensuche (BFS)

Beispiele: Von folgenden Problemen ist nicht bekannt, ob sie in P liegen (wahrscheinlich nicht):

- $HAM = \{\langle G\rangle \mid G$ hat einen Hamilton-Kreis$\}$

  Bemerkung: Ein Hamilton-Kreis ist ein Kreis, der jeden Knoten genau einmal besucht.

- $PRIME = \{bin(n) \mid n$ ist eine Primzahl$\}$

3.2 Die Komplexitätsklasse NP

3.2.1 Nichtdeterministische Turing-Maschinen

Hinweis: Die bisherige Definition von Turing-Maschinen war deterministisch. Das heißt, jede Konfiguration hat eine eindeutige (oder keine) Nachfolgekonfiguration.

Definition: Eine *nichtdeterministische Turing-Maschine* (kurz: *NTM*) ist ein 7-Tupel:

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

Alle Komponenten bis auf δ sind wie bei der deterministischer Turing-Maschine.

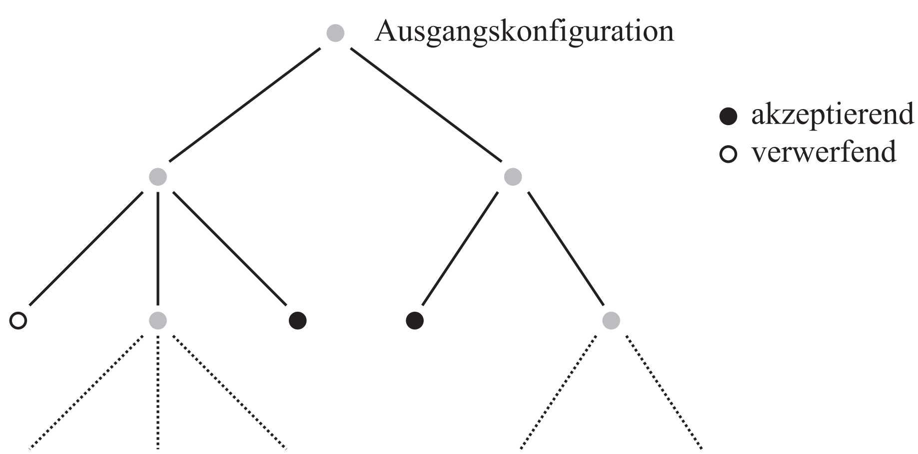
- δ: Zustandsüberführungsfunktion

  $$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$$

  Falls $\delta(q, a) = \emptyset$, stoppt die Turing-Maschine.

  Endkonfigurationen sind immer akzeptierend oder verwerfend.

Definition: Eine nichtdeterministische Turing-Maschine M akzeptiert eine Eingabe w genau dann, wenn im Baum der Folgekonfigurationen mindestens ein Weg von der Startkonfiguration $\varepsilon q_0 w$ zu einer akzeptierenden Endkonfiguration führt. Die akzeptierenden Wörter bilden die Sprache $L(M)$.



3.2.2 Komplexitätsklassen

Definition: Für jede nichtdeterministische Turing-Maschine M existiert die Funktion $ntime_M$:

$$ntime_M(w) = \begin{cases} \text{Minimale Weglänge von } \varepsilon q_0 w \\ \text{zu einer akzeptierenden Endkonfiguration} & \text{falls} \quad w \in L(M) \\ \\ 0 & \text{falls} \quad w \notin L(M) \end{cases}$$

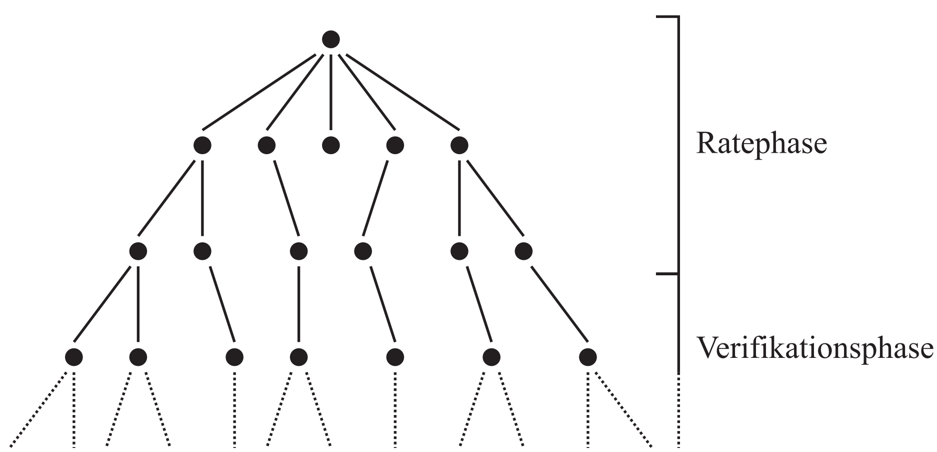
Definition: Sei $f : \mathbb{N} \to \mathbb{N}$ eine Funktion. Die Komplexitätsklasse $\mathsf{NTIME}(f(n))$ besteht aus allen Sprachen L, für die eine nichtdeterministische Mehrband-Turing-Maschine M existiert, mit der Eigenschaft $L = L(M)$ und die minimale Anzahl der Schritte $ntime_M(w)$ von der Startkonfiguration $\varepsilon q_0 w$ zu einer akzeptierenden Endkonfiguration, ist kleiner oder gleich $f(|w|)$ für alle $w \in \Sigma^*$.

$$\begin{aligned} \mathsf{NTIME}(f(n)) &= \{L \subseteq \Sigma^* \mid \exists \text{ NTM } M \text{ mit } L(M) = L \\ &\qquad \text{und } \forall w \in \Sigma^* \; ntime_M(w) \leq f(|w|)\} \end{aligned}$$

3.2.3 Effizient verifizierbare Probleme

Definition: Die Komplexitätsklasse NP der effizient (in Polynomialzeit) verfizierbaren Probleme ist folgendermaßen definiert:

$$\mathsf{NP} = \bigcup_{p(n) \text{ Poylnome}} \mathsf{NTIME}(p(n))$$



Beispiele: Folgende Sprachen liegen in NP:

- Alle Sprachen $L \in \mathsf{P}$

  Begründung: $\mathsf{P} \subseteq \mathsf{NP}$

- $COMPOSITE = \{bin(n) \mid n > 1$ und n ist keine Primzahl$\}$

  Bemerkung: Test auf zusammengesetzte Zahlen

  - M rät $bin(k)$ und $bin(l)$ wobei $1 < |bin(k)|, |bin(l)| \leq |bin(n)|$.
  - M prüft, ob $k \cdot l = n$ ($bin(n)$ ist Eingabe) in $c \cdot |bin(n)|^2$ Zeit.
    - Wenn ja: M akzeptiert
    - Wenn nein: M geht in eine Endlosschleife

- $HAM = \{\langle G \rangle \mid G$ hat einen Hamilton-Kreis$\}$

  - M rät eine Permutation der Knoten von G.
  - M prüft, ob die Permutation einen Kreis in G beschreibt.
    - Wenn ja: M akzeptiert
    - Wenn nein: M geht in eine Endlosschleife

Bemerkung: Das Raten erfolgt bitweise, d.h. die Maschine schreibt eine zufällige Anzahl zufälliger Bits auf ein Band und überprüft dann beispielsweise, ob die geratene Bitfolge eine Permutation repräsentiert.

3.3 NP-Vollständigkeit

3.3.1 Polynomiale Reduktion

Definition: Eine Sprache $L_1 \subseteq \Sigma^*$ ist auf $L_2 \subseteq \Gamma^*$ polynomial reduzierbar, falls eine totale von einer Polynomialzeit-beschränkten deterministischen Turing-Maschine berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ existiert, so dass

$$w \in L_1 \quad \Leftrightarrow \quad f(w) \in L_2$$

Schreibweise:

$$L_1 \leq_{\mathsf{P}} L_2$$

Satz: Ist $L_1 \leq_{\mathsf{P}} L_2$ und $L_2 \in \mathsf{P}$, dann ist $L_1 \in \mathsf{P}$.

Beweis: Es sei $w \in \Sigma^*$ und $|w| = n$.

- Bestimme $f(w)$ in $p(n)$ Schritten

$$\Rightarrow |f(w)| \leq p(n)$$

- Prüfe, ob $f(w) \in L_2$ in $q(p(n))$ Zeit.

$$\Rightarrow \text{Polynomialzeit}$$

$f(w) \in L_2 \quad \Leftrightarrow \quad w \in L_1$

Folgerung: Ist $L_1 \leq L_2$ und $L_1 \notin \mathsf{P}$, dann ist $L_2 \notin \mathsf{P}$.

3.3.2 NP-vollständige Probleme

Definition: Eine Sprache L ist NP-schwer (auch: NP-hart), falls für alle $L' \in \mathsf{NP}$ gilt, dass $L' \leq_{\mathsf{P}} L$.

Definition: L ist NP-vollständig, falls L NP-schwer und $L \in \mathsf{NP}$.

3.3.3 *SAT*-Erfüllbarkeitsproblem

Definition: Das *Erfüllbarkeitsproblem für Formeln der Aussagenlogik* ist gegeben durch die Sprache SAT (F ist eine Codierung einer Formel):

$$SAT = \{\langle F \rangle \in \Gamma^* \mid F \text{ ist erfüllbar}\}$$

Satz: Die Sprache SAT liegt in NP.

Beispiel:

$$F = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$$

Hinweis: F ist erfüllbar durch $x_1 = 0$ und $x_2 = 1$.

Vorgehen:

- M rät eine Belegung der Variablen in F.
- M berechnet den Wert von F unter dieser Belegung.
  - Wenn der Wert 1 ist: M akzeptiert
  - Wenn der Wert 0 ist: M geht in eine Endlosschleife

Satz (Cook / Levin): Das Erfüllbarkeitsproblem für Formeln der Aussagenlogik SAT ist NP-vollständig.

Beweis:

1. Es ist zu zeigen, dass SAT NP-schwer ist, d.h. für beliebiges $L \in \mathsf{NP}$ gilt $L \leq_{\mathsf{P}} SAT$, d.h. es gilt eine totale in Polynomialzeit berechenbare Funktion $f : \Sigma^* \to \Gamma^*$, so dass $w \in L \quad \Leftrightarrow \quad f(w) \in SAT$.

2. Idee: $L \in \mathbb{N}\mathsf{P}$, d.h. $\exists$ NTM M

   - $L = L(M)$
   - M ist Polynomialzeit-beschränkt

   $$w = a_1, a_2, \ldots a_n \;\; \text{Eingabe für } M$$

   $w \in L \quad \Leftrightarrow \quad M$ erreicht einen akzeptiereden Endzustand bei Eingabe w

   Codiere Berechnung von M auf w durch Formel, die genau dann erfüllbar ist, wenn M akzeptierende Endkofiguration erreichen kann.

   Vereinfachung: $Q = \{q_0, q_1 \ldots q_k\}$

   M akzeptiert durch einen akzeptierenden Endzustand $q_{\text{accept}} \in Q$ (nicht mit 1 auf Band).

$\vdots$

$\vdots$

$\vdots$

Vorlesung vom 21.6.2002 (fehlt)

$\vdots$

$\vdots$

$\vdots$

Kapitel 4

Kontextfreie Sprachen

4.1 Grammatiken und die Chomsky-Hierarchie

Hinweis: Eine *Grammatik* ist ein System, um Ausdrücke (Wörter) nach bestimmten Regeln zu bilden, und beschreibt eine Sprache.

Bemerkung: Dieses Prinzip ist bereits aus der Bildung von Formeln, Termersetzungsverfahren, Aufbau von Programmiersprachen bekannt.

Definition: Eine Grammatik ist eine 4-Tupel $G = (V, \Sigma, P, S)$:

- V: endliche Menge von Variablen (Großbuchstaben)
- Σ: endliches Terminalalphabet ($\Sigma \cap V = \emptyset$)
- $S \in V$: Startvariable
- P: endliche Menge von Regeln (Produktionen)

$$P \subseteq (\Sigma \cup V)^+ \times (\Sigma \cup V)^*$$

Ist $u = xyz, v = xy'z \in (\Sigma \cup V)^*$ und ist $(y, y') \in P$, dann sagen wir, dass v aus u ableitbar (unter G) ist (oder u geht in v über). Wir schreiben dafür:

$$u \Longrightarrow_G v \quad \text{(kurz: } u \Longrightarrow v \quad)$$

$\stackrel{*}{\Longrightarrow}_G$ ist reflexiver und transitiver Abschluss von $\Longrightarrow_G$, d.h. $u \stackrel{*}{\Longrightarrow}_G v$, wenn $u = u_0 \Longrightarrow_G u_1 \Longrightarrow_G \ldots \Longrightarrow_G u_k = v$

$$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Longrightarrow}_G w\}$$

Beispiel:

1. Korrekte Klammerausdrücke:

$$G = (\{S\}, \{(,)\}, P, S) \quad \text{mit} \quad P = \{(S, ()), (S, (S)), (S, SS)\}$$

Ableitung und Syntaxbaum:

$$\begin{aligned} S &\Longrightarrow_G S\underline{S} \\ &\Longrightarrow_G S(\underline{S}) \\ &\Longrightarrow_G \underline{S}(SS) \\ &\Longrightarrow_G ()(S\underline{S}) \\ &\Longrightarrow_G ()(\underline{S}()) \\ &\Longrightarrow_G ()(()()) \end{aligned}$$

2. Palindrome

$$G = (\{S\}, \{0, 1\}, P, S) \quad \text{mit} \quad P = \{(S, 0), (S, 1), (S, 00), (S, 11), (S, 0S0), (S, 1S1)\}$$

Beispiel:

$$10001 : S \Longrightarrow 1S1 \Longrightarrow 10S01 \Longrightarrow 10001$$

Kürzere Darstellung der Regeln mit einer Variable auf der linken Seite:

$$S \to 0 \mid 1 \mid 00 \mid 11 \mid 0S0 \mid 1S1$$

Nennt man *Backus-Naur-Form* (*BNF*)

Definition:

- Jede Grammatik ist (zunächst) vom Typ 0 (keine Einschränkung).
- Eine Grammatik ist vom Typ 1 (kontextsensitiv), falls für alle Regeln $w_1 \to w_2$ gilt: $|w_1| \leq |w_2|$.
- Eine Grammatik ist vom Typ 2 (kontextfrei), falls für alle Regeln $w_1 \to w_2$ so sind, dass $w_1 \in V$ und $|w_2| \geq 1$
- Eine Grammatik ist vom Typ 3 (regulär), falls für alle Regeln $w_1 \to w_2$ so sind, dass $w_1 \in V$ und $w_2 \in \Sigma \cup \Sigma V$

4.2 Kontextfreie Sprachen und Normalform

Definition: Sprache L ist kontextfrei, wenn eine Typ 2-Grammatik existiert, so dass $L = L(G)$.

Beispiel: $L = \{a^n b^n \mid n \in \mathbb{N}^+\}$ ist kontextfrei (aber nicht regulär).

$$S \to ab \mid aSb$$

Definition: Eine kontextfreie Grammatik G mit $\varepsilon \notin L(G)$ ist in *Chomsky-Normalform* (kurz: *CNF*), falls alle Regeln die folgende Form haben:

$$\begin{aligned} A &\to a \qquad && A \in V, a \in \Sigma \\ A &\to BC \qquad && A, B, C \in V \end{aligned}$$

CNF impliziert binäre Syntaxbäume der Ableitungen, wenn man den letzten Schritt der Umwandlung in Terminalsymbole vernachlässigt.

Beispiel: CNF von $L = \{a^n b^n \mid n \in \mathbb{N}^+\}$:

$$\begin{aligned} S &\to AT \\ T &\to SB \mid b \\ A &\to a \\ B &\to b \end{aligned}$$

GRAFIK: Binärbaum für *aabb*

Schematisch:

GRAFIK: Dreick (binär) mit Rechteck drunter (letzter Schritt)

Satz: Für jede kontextfreie Grammatik G existiert eine CNF G', so dass $L(G) = L(G')$ (d.h. G und G' sind äquivalent).

Beweis:

1. Elemination von Regeln der Form $A \to B$

   (a) Wenn ein Kreis $A_1 \to A_2 \to \ldots \to A_k \to A_1$ ersetze $A_1, A_2, \ldots A_k$ durch neue Variable A (für alle Regeln).

   (b) wenn weiterhin eine Regel $B_1 \to B_2$ existiert, streichen diese Regel und ergänzen die aus B_1 abgeleiteten Regeln mit denen, die aus B_2 abgeleitet werden.

2. Alle Regeln haben die Form

$$A \to a \quad \text{oder} \quad A \to x \in (V \cup \Sigma)^*$$

(a) Für jedes $a \in \Sigma$ fügt man ein neues B_a zu V hinzu sowie die Regeln $B_a \to a$.

(b) Regel $A \to x = A_1 A_2 b_1 A_3 b_2$ wird ersetzt durch (am Beispiel)

$$A \to A_1 A_2 B_{b_1} A_3 B_{b_2}$$

(c) Für jeden Suffix der Länge ≥ 2 Hilfsvariable einführen (am Beispiel)

$$\begin{aligned} A &\to A_1 C_1 \\ C_1 &\to A_2 C_2 \\ C_2 &\to B_{b_1} C_3 \\ C_3 &\to A_3 B_{b_2} \end{aligned}$$

Anwendung: Umformung für $S \to ab \mid aSb$:

$$\begin{array}{lcl} A & \to & a \\ B & \to & b \\ S & \to & AB \mid AC \\ C & \to & SB \end{array}$$

Definition: Eine kontextfreie Grammatik ist in *Greibach Normalform* (kurz: *GNF*), falls alle Regeln die Form $A \to aB_1 \ldots B_k$ ($k \in \mathbb{N}$) haben.

Satz: Für jede kontextfreie Grammatik gibt es eine äquivalente GNF (ohne Beweis).

CYK-Algorithmus:

- Gegeben: $G = (V, \Sigma, P, S)$ in CNF und $w = a_1, a_2, \ldots a_n \in \Sigma^*$
- Frage: $w \in L(G)$?
- Idee: $w \in L(G)$ betrachte oberste Stufe des Syntaxbaums einer Ableitung: $S \Longrightarrow_G AB$, dann wird aus A ein Präfix von w abgeleitet, aus B ein Suffix.
- Cocke, Younger, Kasami: Anwendung von dynamischer Programmierung

$$w_{i,j} = \underbrace{a_i a_{i+1} \ldots a_{i+j-1}}_{\text{fängt mit } a_i \text{ an und hat Länge } j}$$

$$T_{i,j} = \{A \in V \mid A \stackrel{*}{\Rightarrow}_G w_{i,j}\}$$

$$S \in T_{1,n} \quad \Leftrightarrow \quad S \stackrel{*}{\Rightarrow}_G w_{1,n} \quad \Leftrightarrow \quad w \in L(G)$$

- Initialisierung:

$$T_{i,1} = \{A \mid (A, a_i) \in P\}$$

```
for i = 2 to n
    for i = 1 to n - j + 1
```

$$T_{i,j} = \bigcup_{k=1}^{j-1} \{A \mid \exists B \in T_{i,k}, C \in T_{i+k,j-k} \text{ und } (A, BC) \in P\}$$

- Laufzeit: $\mathcal{O}(n^3)$

4.3 Pumping-Lemma

Satz: Für jede kontextfreie Sprache L existiert ein $n \in \mathbb{N}$, so dass für jedes $z \in L$ mit $|z| \geq n$ eine Zerlegung $z = uvwxy$ existiert mit:

- $|vwx| \leq n$
- $|vx| \geq 1$
- $\forall i \in \mathbb{N}$ ist $z' = uv^iwx^iy \in L$

Beispiele:

1. $L = \{a^kb^k \mid k \in \mathbb{N}\}$ ist kontextfrei, also erfüllt es das Pumping-Lemma.

   Man setzt $n = 2$ imd betrachtet ein $z \in L$ mit $|z| \geq n = 2$. Dann ist

   $$z = a^kb^k \text{ mit } k \geq 1$$

   und man findet eine Zerlegung

   $$z = \underbrace{a\ldots}_{u}\underbrace{a}_{v}\underbrace{\varepsilon}_{w}\underbrace{b}_{x}\underbrace{\ldots b}_{y}$$

   Wie man leicht sieht:

   $$z' = uv^iwx^iy = a^{k+(i-1)}b^{k+(i-1)} \in L$$

2. $L = \{a^kb^kc^k \mid k \in \mathbb{N}\}$ ist nicht kontextfrei.
   Indirekter Beweis: Angenommen L wäre kontextfrei, dann gibt es ein n aus dem Pumping-Lemma, ...

   $$\begin{aligned} a^nb^nc^n &= uvwxy \\ \text{Aus 1) } vwx &\in \underbrace{a^*b^*}_{\text{Fall 1}} \cup \underbrace{b^*c^*}_{\text{Fall 2}} \\ z' = uv^2wx^2y & \end{aligned}$$

- Fall 1: Anzahl der a's oder Anzahl der b's in z' ist $> n$, Anzahl der c's gleich n
  $\Rightarrow z' \notin L \Rightarrow$ Widerspruch
- Fall 2: Anzahl der b's oder Anzahl der c's in z' ist $> n$, Anzahl der a's gleich n
  $\Rightarrow z' \notin L \Rightarrow$ Widesrspruch

Widerspruch zur Annahme, d.h. L ist nicht kontextfrei.

Beweis der Pumping-Lemmas: Sei G CNF für L.

$$G = (V, \Sigma, P, S)$$

Man definiert: $k = |V|$ und $n := 2^k$
Sei $z \in L$ mit $|z| \geq n$
Aufgabe: Zerlegung finden
Syntaxbaum T für eine Ableitung von z ohne Terminalsymbole ist binär und hat $|z|$ Blätter. Daraus folgt, dass die Tiefe von $T \geq k$.
Man betrachtet einen Weg maximaler Länge (also $\geq k$), dann muss auf dem Weg eine Variable doppelt austreten.
Man betrachtet erste Variablendopplung von unten und die entsprechenden Unterbäume

GRAFIK: Unterbäume

Beide haben höchstens Tiefe k höchstens $2^k = n$ Blätter.
Zerlegung $uvwxy$:

- $|vwx| \leq n$, da größerer Baum $\leq n$ Blätter hat
- $|vx| \geq 1$, da die Bäume verschieden sind (kleiner Baum ist Unterbaum)
- $z' = uv^j wx^j y = uwy$ wird durch folgenden baum abgebildet:

GRAFIK: kleinen Baum nach oben

$z'' = uv^2wx^2y$

GRAFIK: großer Baum doppelt

Jede weitere $uv^{i+1}wx^{i+1}y$ durch Ersetzung des kleinen Baum durch den großen

4.4 Kellerautomaten

Idee: Automat darf Eingabe nur einmal von links nach rechts lesen und kann Informationen in einem Kellerspeicher aufbewahren.

GRAFIK: Schema

Formel: Ein nichtdeterministischer Kellerautomat (kurz: *NPDA*) wird durch ein 6-Tupel beschrieben:

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

- Q: endliche Zustandsmenge
- Σ: Eingabealphabet (kleine Buchstaben)
- Γ: Kelleralphabet (große Buchstaben)
- δ: Zustandsüberführungsfunktion

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

  $\underset{e}{\mathcal{P}}$ heißt: Menge aller endlichen Teilmengen

- $q_0 \in Q$: Startzustand
- $\# \in \Gamma$: unterstes Kellerzeichen

Arbeitsweise: $(q', B_1 \ldots B_k) \in \delta(q, a, A)$
Wenn M im Zustand q und a auf dem Band und A in oberster Kellerzelle liest, so kann er in q' übergehen, A löschen und $B_1 \ldots B_k$ in den Keller speichern (B_1 oben).
$(q', B_1 \ldots B_k) \in \delta(q, \varepsilon, A)$
Wenn M im Zustand q auf oberster Kellerzelle A liest, kann er ohne Eingabesymbol zu lesen in q' übergehen, A löschen und $B_1 \ldots B_k$ speichern.

Konfiguration: $k \in \underbrace{Q}_{\text{aktueller Zustand}} \times \underbrace{\Sigma^*}_{\text{ungelesen}} \times \underbrace{\Gamma^*}_{\text{Kellerinhalt}}$
$k \vdash k'$: k' ist direkte Folgekonfiguration von k
$k \overset{*}{\vdash} k'$: $k = k_0 \vdash k_1 \vdash \ldots \vdash k_n = k'$

Akzeptierende Endkonfiguration: $(q, \varepsilon, \varepsilon)$ wobei q beliebig

Definition: $w \in L(M) \quad \Leftrightarrow \quad (q_0, w, \#) \overset{*}{\vdash} (q, \varepsilon, \varepsilon)$