Theoretische Informatik
WS03/04

Alexander Böhm
e-mail: timeless@team-cauchy.de

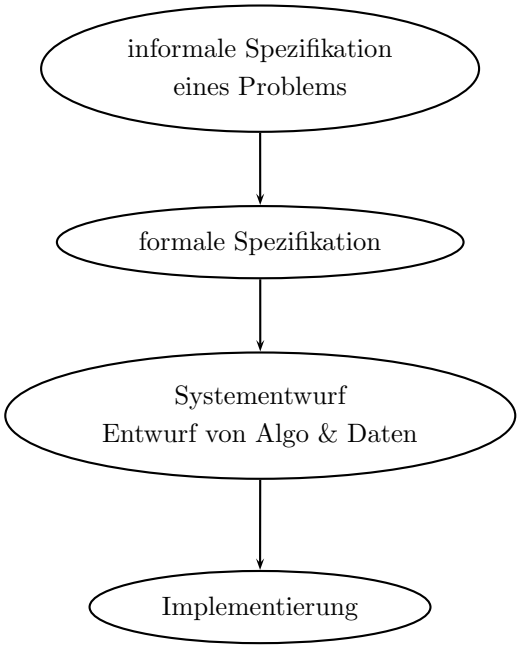
Christian Forler
e-mail: shortie@team-cauchy.de

20. März 2004

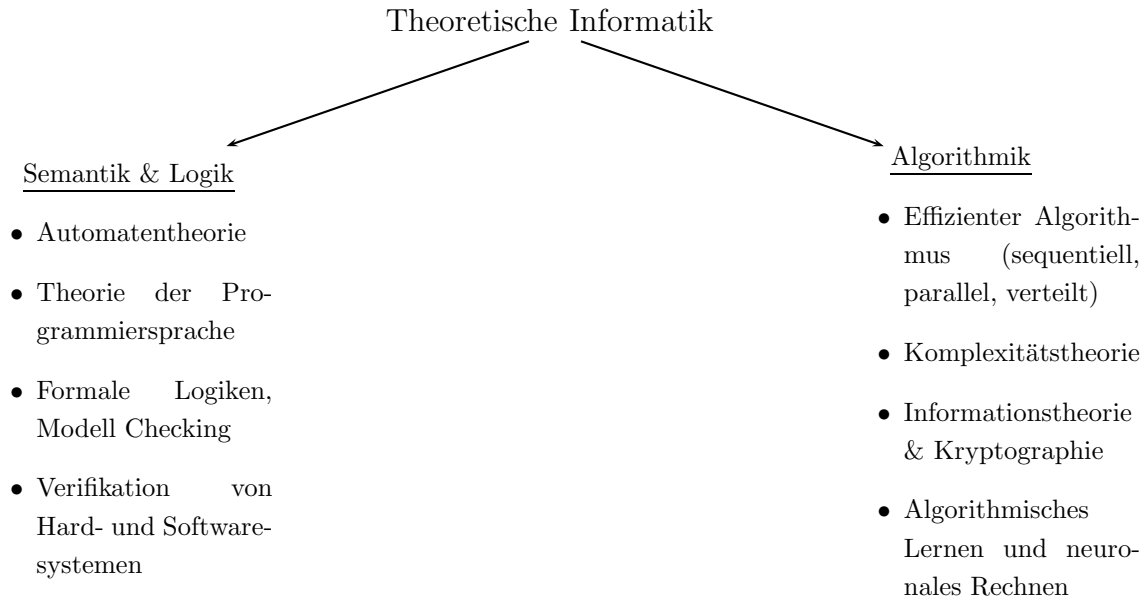
Inhaltsverzeichnis

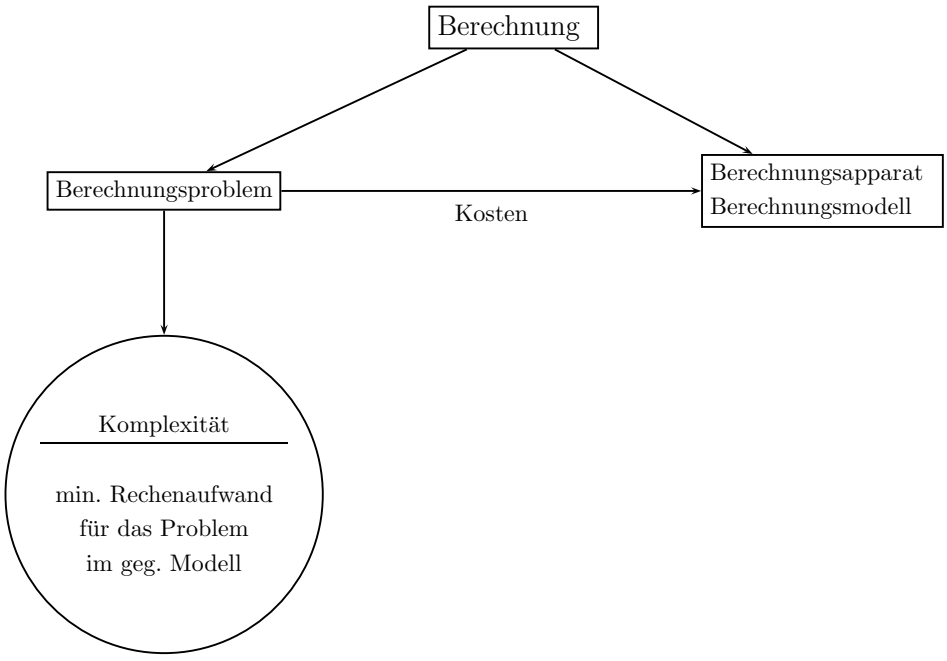
Kapitel 1

Grundlegende Begriffe



Definition 1.1 (Theoretische Informatik)
Sammlung wissenschaftlicher Methoden zur Unterstützung und Ermöglichung obigen Ablaufs. Basiert auf geeigneten Modellbildungen und Formalisierungen, insbesondere des Begriffs $\boxed{\textit{Berechnung}}$.





1.1 Berechnungsproblem

Definition 1.2 (Berechnungsproblem Π)
Das Berechnungsproblem Π *ist eine Relation* $\Pi \subseteq \Sigma^* \times \Sigma^*$ *= Menge von Paaren*

$(x, y) \in \Sigma^* \rightarrow$ *Relation Eingabe - Lösung, wobei x die korrekt kodierte Eingabe und y eine Lösung zu x darstellt*

Formale Spezifikation:

- Arbeitsalphabet Σ (endliche Menge)
- Eingabekodierung, Einbettung der Eingabemenge in Σ^* (wobei Σ^* die Menge aller Worte, die aus dem Alphabet Σ gebildet werden können, ist) und Definition der Eingabelänge
- Berechnungsproblem Π

Beispiel:

Berechnungsproblem Multiplikation natürlicher Zahlen

- Arbeitsalphabet $\Sigma = \{0, 1, *\}$
- korrekt kodierte Eingaben $\underbrace{x_{n-1}, x_{n-2}, ..., x_0}_{x} \quad \# \quad \underbrace{y_{m-1}, y_{m-2}, ..., y_0}_{y}$ mit $n, m \geq 1; x_i, y_i \in \{0, 1\} \forall i \Rightarrow$ binäre Darstellung natürlicher Zahlen
- Mult $= \{(x\#y, z); z = z_{k-1}, ...z_0; z_i \in \{0, 1\}\ \forall i\}$
  $(\sum_{i=0}^{n-1} x_i 2^i) * (\sum_{j=0}^{m-1} y_j 2^j) = (\sum_{l=0}^{k-1} z_l 2^l)$

Spezielle Arten von Berechnungsproblemen

Funktion: zu jeder korrekt kodierten Eingabe existiert genau eine Lösung (Beispiel: Multiplikation)

Entscheidungsprobleme: Funktion mit Ausgabe {0,1} (Beispiel: Primzahltest)
$\Pi : \Sigma^* \rightarrow \{0, 1\}$
Entscheidungsprobleme $\Pi : \Sigma^* \rightarrow \{0, 1\}$
werden oft als Spachen $L_\Pi \subseteq \Sigma^*$ aufgefasst, wobei $\forall x \in L_\Pi$ gilt $\Pi(x) = 1$ (d.h. eine Sprache ist eine Menge von Worten x für die $\Pi(x) = 1$ gilt).
Beispiel: Primzahltest: $\Pi(x_{n-1}, ..., x_0) = 1 \Longleftrightarrow \sum_{i=0}^{n-1} x_i 2^i$ ist Primzahl.

Optimierungsprobleme: zu jeder zulässigen Eingabe x existiert genau eine Menge möglicher Lösungen $\mathcal{L}_\Pi(x)$, die mit einem Kosten- (oder Nutzen-)Parameter versehen sind ($c_\Pi \bigcup_{x \in \Sigma^*} \mathcal{L}_\Pi(x) \rightarrow R$). Das Problem besteht aus Paaren (x,y), wobei x eine zulässige Eingabe und y eine mögliche Lösung $y \in L_\Pi(x)$ mit minimalen Kosten ist.
Beispiel: Travelling Salesman Problem (TSP)
gegeben: Eingabekodierung (n, D=$(d_{ij} \forall i, j \in \{1, .., n\})$ (D = Distanzmatrix, n = Anzahl der Städte). $L_{TSP}(x) = \Theta_n = \{\delta\{1, .., n\} \rightarrow \{1, .., n\} \quad bijektiv\}$
($\Theta_n = n! \Leftrightarrow$ Lösungsanzahl, $L_{TSP}(x) \Leftrightarrow$ Lösungsmenge)
TSP=$\{(x_j\delta); c_{TSP}(x, \delta) \leq c_{TSP}(x, \delta') \forall \delta' \in \Theta_n\} \rightarrow$ minimale Kosten für die optimale Lösung (x, δ).

$c_{TSP}(x, \delta) = \sum_{i=1}^{n-1} d_{\delta(i),\delta(i+1)} + d_{\delta(i),\delta(1)}$ (Kosten für $c_{TSP} = \sum$ aller Teilstrecken + Strecke von Ende zu Anfang)

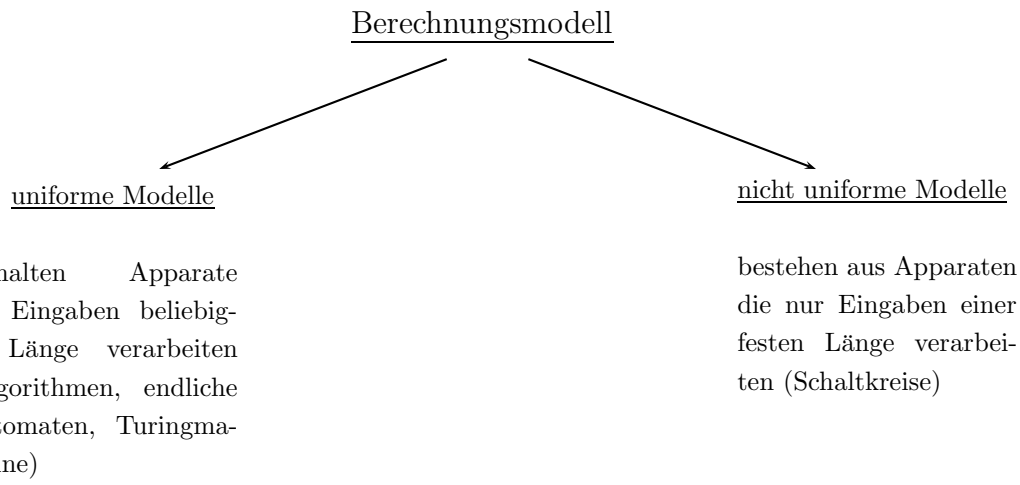
1.2 Berechnungsmodell

Menge gleichartiger Berechnungsapparate (Modell der Schaltkreise, Modell der Turingmaschinen, Modell der endlichen Automaten...)
Berechnungsapparate (engl. computational device) können sehr verschiedenartig sein: Gemeinsamkeiten:

- können Worte aus Σ^* einlesen und ausgeben
- Berechnungsverhalten ist durch die Spezifikation des Apparats und die Eingabe eindeutig bestimmt (gegebenenfalls wird zur Eingabe $x \in \Sigma^*$ unter wohldefiniertem Ressourcenverbrauch eine eindeutig bestimmte Ausgabe $A(x) \in \Sigma^*$ produziert).

Definition 1.3 (Berechnungsapparat)
Der Berechnungsapparat A über Σ berechnet ein gegebenes Problem ($\Pi \subseteq \Sigma^* \times \Sigma^*$) falls für alle zulässigen ($\Leftrightarrow$ korrekt kodierten) Eingaben $x \in \Sigma^*$ gilt:
A produziert Ausgabe $A(x)$ und $(x, A(x)) \in \Pi$.

Definition 1.4 (Berechnungsmodell)
Ein Problem Π heißt im Berechnungsmodell $\mathcal{M}$ berechenbar falls $\exists\ A \in \mathcal{M}$ mit A berechnet Π. Man unterteilt Berechnungsmodelle in zwei Kategorien:



Definition 1.5 (uniforme Berechenbarkeit)
Ein Problem Π ist einem Modell $\mathcal{M}$ berechenbar $\Leftrightarrow \exists$ Berechnungsapparat $A \in \mathcal{M}$ mit A berechnet Π (uniformer Fall).

Problemmatisch: nichtuniformer Fall (Beispiel: Schaltkreis kann 32-Bit Multiplikation aber nicht Multiplikation von Zahlen beliebiger Länge).

Definition 1.6 (nicht uniforme Berechenbarkeit)
$\mathcal{M}$ sei ein nichtuniformes Modell; Π *ist in* $\mathcal{M}$ *berechenbar* $\Leftrightarrow \forall n \in N \quad \exists A_n \in \mathcal{M}$ *mit* A_n *berechnet* $\Pi_n = \{(x,y) \in \Pi; |x| = n\}$*; d.h. Realisierungen von* Π *in* $\mathcal{M}$ *(nichtuniform) sind Folgen von Berechnungsapparaten* $(A_n)_{n\in N}$ *(→ Unterteilung in endliche Teilprobleme)*

Kostenmessung von Berechnungen:

Messen der Kosten als Funktion der Eingabelänge und Vernachlässigung von Konstanten
Annahme: Berechnungsmodell (uniform) mit Kostenmaß C (z.B. Berechnungszeit)

Kosten eines Berechnungsapparates A $\in \mathcal{M}$:

- $C_A(x)$ - Kosten der Berechnung von A auf x
- $C_A(n) = max\{C_A(x); |x| = n\}$

→ Kosten von A sind Funktion $c_A : N \to N$ (sogenannte worst-case Kosten) analog hierzu best-case Kosten: $C_A(n) = min(\{C_A(x); |x| = n\}$ und average-case Kosten: $Exp_{P_n}[C_A(x), |x| = n]$ (wobei $|x|$ die Menge der Eingaben der Länge n darstellt, und Exp_{P_n} entsprechend der Erwartungswert der Wahrscheinlichkeitsverteilung ist).

Definition 1.7 (Komplexität) *Sei* Π *in* $\mathcal{M}$ *berechenbar. Eine Funktion* $f : N \to N$ *heißt die Komplexität von* Π *bezüglich* $\mathcal{M}$ *und C falls:*

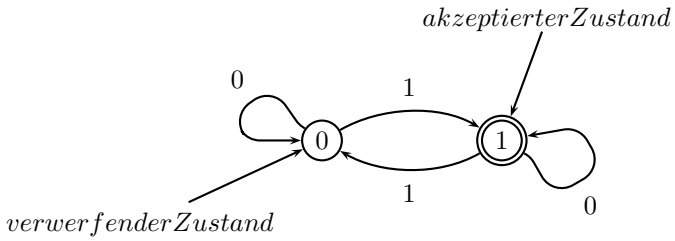
- $\exists\ A^* \in \mathcal{M}$ *der* Π *berechnet mit* $C_A(n) \in O(f(n))$ *(*$A^* \Leftrightarrow$ *optimaler Apparat)*
- $\forall\ A \in \mathcal{M}$*, die* Π *berechnen, gilt* $C_A(n) \in \Omega(f(n))$

Nichtuniformes Modell: Sei Π *in* $\mathcal{M}$ *(nicht uniform) berechenbar. Komplexität* $f : N \to N$ *von* Π *ist definiert durch* $f(n) = min\{C(A_n); A_n$ *berechnet* $\Pi_n\}$ *(C ist Kostenfunktion auf* $\mathcal{M}$ *(z.B. C als Schaltkreisgröße → Minimale Schaltkreiskomplexität für gegebenes Problem))*

Kapitel 2

Endliche Automaten und reguläre Sprachen

2.1 Grundlegende Eigenschaften



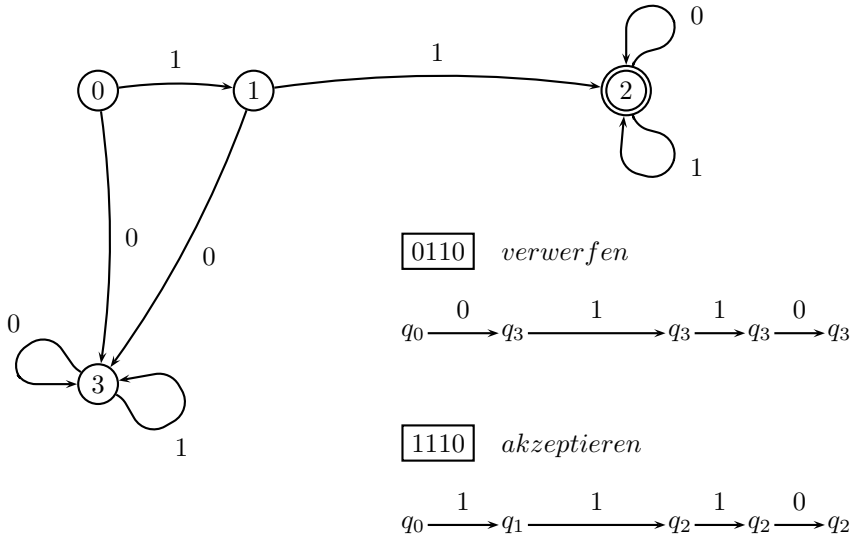
$\boxed{0110}$ *verwerfen*

$$q_0 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{0} q_0$$

$\boxed{1110}$ *akzeptieren*

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1$$

$\text{PARITY} = \{x \in \{0,1\}^*, \sum_{i=1}^{|x|} x_i \; mod \; 2 = 1\}$

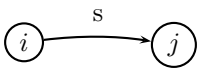


Definition 2.1 (Deterministischer finiter Automate (DFA))
$A = (Q, q_0, F, \delta)$ ist ein endlicher Automat (DFA), wobei

- *Q = endliche Zustandsmenge*
- *$q_0 = q_0 \in Q$ (Anfangszustand)*
- *F = Teilmenge von Q (Menge der akzeptierten Zustände)*
- *$\delta = \delta : Q \times \Sigma \to Q$ (Zustandsüberführungsfunktion $q_{alt}, s \to q_{neu}$ (wobei s das aktuell eingelesene Zeichen darstellt)*

Deterministische finite Automaten sind bezüglich eines endlichen Alphabets Σ definiert.

Veranschaulichung und Notation

1. Zustandsüberführungsgraph (siehe Beispiele)
   - Knoten ≡ Zustände
   - Instanzen $(q_i, s) \xrightarrow{\delta} q_j$ von δ entsprechen den Kanten 

   - $q_i \in F$ werden markiert 
 (akzeptierte Zustände)
2. Zuständsüberführungstabelle

	s_1	s_2	..	s	..	s_t
q_0						
q_1						
..						
q_i				q_j		
..						
q_{r-1}						

gdw. $\delta(q_i, s) = q_j$

$|Q| = r$

Tabelle zu Beispiel 1:

	0	1
q_0	q_0	q_1
$^*q_1$	q_1	q_0

Tabelle zu Beispiel 2:

	0	1
q_0	q_3	q_1
q_1	q_3	q_2
$^*q_2$	q_2	q_2
q_3	q_3	q_3

Definition 2.2 (Berechnungsverhalten von DFAs A über Σ)
Für jede Eingabe $x \in \Sigma^*$ exisitiert ein eindeutiger Zustand $\delta^*(q, x)$ in dem eine Berechnung von A auf x endet, wenn in $q \in Q$ gestartet wird.

- *Fall 1: $x = \epsilon$ (d.h. $|x| = 0$ (leeres Wort)) $\rightarrow \delta^*(q, x) = q$*
- *Fall 2: $x = s$ (d.h. $|x| = 1, s \in \Sigma$) $\rightarrow \delta^*(q, x) = \delta(q, s)$*
- *Fall 3: $|x| > 1$ (d.h. $x = x's; s \in \Sigma; |x'| \geq 1$) $\rightarrow \delta^*(q, x) = \delta(\delta^*(q, x'), s)$*

Äquivalent dazu:
Für alle $q \in Q$ und $x \in \Sigma^*$ gilt:
x definiert einen eindeutig bestimmten Rechenweg $q^{(0)}, q^{(1)}, q^{(2)}, \ldots, q^{(x)}$ in A, wobei für alle $i = 1, \ldots, x$ gilt:
$q^{(i)} = \delta(q^{(i-1)}, x_i)$ *wenn* $x = x_1, x_2, .., x_{|x|}$
$\delta^*(q, x) = q^{(|x|)}$
A(x)=

- *1 falls $\delta^*(q_0, x) \in F$*
- *0 falls $\delta^*(q, x) \notin F$*

$L(A) = \{x \in \Sigma^*; A(x) = 1\}$ *von A berechnete Sprache*

Definition 2.3 (Regulär)
Sprache $L \subseteq \Sigma^*$ heißt regulär, falls $\exists$ DFA mit $L(A) = L$ (d.h. A berechnet L)

Kostenmaße für endliche Automaten

Relevantes Kostenmaß: Anzahl der Zustände (Berechnungszeit wäre trivial (= Länge der Eingabe))

Beobachtung 2.1
DFAs sind uniformes Modell (verarbeiten Eingabe beliebiger Länge)

2.1.1 Nichtreguläre Sprachen

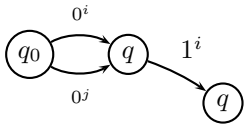
Beispiel:
$L = \{0^n 1^n; n \geq 1 \text{ beliebig }\}$ (beliebig langer Nullblock gefolgt von einem beliebig langen Block von Einsen)

Lemma 2.1 $\exists\, L \notin REG$

Beweis:

Annahme: $\exists$ DFA A für $L \Rightarrow$ Da Q (Zustandsmenge) endlich ist $\exists\ i \neq j$ mit

$\delta^*(q_0, 0^i) = \delta^*(q_0, 0^j)$



$\Rightarrow \delta^*(q_0, 0^i 1^i) = \delta^*(q_0, 0^j 1^i) = \delta^*(q, 1^i)$
$\Rightarrow q' \in F \qquad \Rightarrow q' \notin F \qquad \Rightarrow$ A ist kein DFA für L.

Beobachtung 2.2 *DFAs können nicht zählen*

Wichtige Fragen

1. Wie sieht man $L \in \Sigma^*$ an ob regulär oder nicht?
2. Wenn L regulär, wieviele Zustände hat dann ein minimaler DFA für L?

Lemma 2.2 (Pumping Lemma) *(Aufpumpbarkeitseigenschaft regulärer Sprachen)*
$\forall\, L \in REG\ gilt : \exists\, n_0 \in N\ so\ da\ss \forall\, w \in L\ mit\ |w| > n_0\ gilt :$

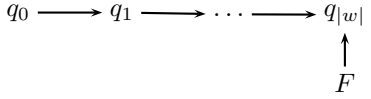
$$\exists\ Zerlegung\ w = xuy\ u \neq \epsilon,\ \forall\, j \geq 0\ gilt :\ xu^j y \in L$$

(w lässt sich an u aufpumpen, ohne dass L verlassen wird)

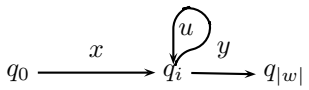
Bemerkung 2.1 $w = xuy \in L \Rightarrow xy, xuuy, xuuuy, \ldots, \in L$

Beweis (Pumping Lemma):

$Fix\ L \in REG,\ DFA\ A = \{Q, q, F, \delta\},\ L, n_0 = |Q|,\ fix\ w\ mit\ |w| > n_0$
Berechnung A auf w:



$Da\ |w| > n_0\ \ \exists\, i \neq j\ \ 0 \leq i;\ j \leq |w|\ und\ q_i = q_j$



Zerlegung von w=xuy
$\Rightarrow \delta^*(q_o, xu^jy) = q_{|w|} \in F \quad \forall\, j \geq 0$
$\Rightarrow xu^jy \in L$ □

Beispiel:

$\{0^n1^n,\ n \geq 0\} \notin REG,\ da\ \ 0\cdots01\cdots1\ \ nicht\ aufpumpbar$

ACHTUNG: Aufpumpbarkeit ist keine hinreichende Bedingung für REG!

Beispiel:

$L \in \{\{0,1\}^*$; x enthält gleich viele "0" und "1"$\}$
Diese Sprache ist nicht regulär aber aufpumpbar

2.2 Äquivalenz-Relationen

Definition 2.4 (Äquivalenz-Relationen)
Sei M eine Menge und $x, y, z \in M$ dann gilt:
$R \subseteq MxM$ ist eine Äquivalenzrelation $\Longleftrightarrow$

1. *Symmetrie* $\quad (x,y) \in R \Longleftrightarrow (y,x) \in R$
2. *Reflexivität* $\quad (x,x) \in R \quad \forall\, x \in M$
3. *Transitivität* $(x,y),(y,z) \in R \Rightarrow (x,z) \in R$

Beispiele:

- triviale Äquivalenz-Relationen
  - " = " $\quad (x,y) \in R \Longleftrightarrow x = y$
  - $R_{ALL} \quad \{(x,y);\ x,y \in M\}$
- Mod k-Relation auf Z
  $(x,y) \in Mod\ k \Longleftrightarrow x - y \equiv 0$ mod k

Exkurs

Sei $x, y, k, k' \in Z$ mit $0 \leq k' < k$ dann gilt:
$\exists\, r \quad x = r \cdot k + k'$ mit r = x div k und k' = x mod k
x ≡ y mod k $\Longleftrightarrow$ x mod k = y mod k $\Longleftrightarrow$ x-y ≡ 0 mod k
x∘y mod k ≡ (x mod k) ∘ (y mod k) mod k $\quad \circ \in \{\cdot, +, -\}$

Definition 2.5 (Äquivalenzklassen)
Sei M eine Menge, $x \in M$ und $R \subseteq M\mathbf{x}M$ eine Äquivalenzrelation. R definiert Partition von M in Äquivalenzklassen:

$$[x]_R = \{y;\ (x,y) \in R\}$$

Corollar 2.1 (Elementare Eigenschaft)
$\forall\, x, x' \in M$ *gilt:*

$$[x]_R = [x']_R \quad \textit{oder} \quad [x]_R \cap [x']_R = \emptyset$$

$\Rightarrow$ *R unterteilt M in disjunkte Äquivalenzklassen*

Definition 2.6 (Index)

- ${}^{M}/_{R} = \{[x]_R;\ x \in M\}$
- $\left|{}^{M}/_{R}\right| = ind(R)$

Beispiel:

- $R_{=}$: $[x]_{=} = \{x\}$, $\quad {}^{M}/_{=} = \{\{x\},\ x \in M\}$, $\quad ind(=) = |M|$
- R_{ALL}: $[x]_{ALL} = M$, $\quad {}^{M}/_{ALL} = \{M\}$, $\quad ind(ALL) = 1$
- $R_{mod\ k}$: $[x]_{mod\ k} = \{y;\ x \equiv y\ mod\ k\}$, $\quad {}^{M}/_{mod\ k} = \{[0], [1], \ldots, [k-1]\}$
  $ind(mod\ k) = k$

Definition 2.7 (Verfeinerung)
Äquivalenzrelation R auf M heißt Verfeinerung einer Äquivalenzrelation R' auf M falls gilt:

$$R \subseteq R' \qquad (\Longleftrightarrow (x,y) \in R \Longrightarrow (x,y) \in R')$$

Beispiele

- Geburtsstadt ist eine Verfeinerung von Geburtsland
- Geburtstag ist eine Verfeinerung von Geburtsjahr

Beobachtung 2.3
R' *Verfeinerung von* $R \Rightarrow \forall\, x \in M$ *gilt:*

$$[x]_{R'} \subseteq [x]_R \Rightarrow ind(R') \geq ind(R)$$

($\Rightarrow$ *Mehr Äquivalenzklassen* $\Rightarrow$ *größerer Index)*

Eigenschaft 2.1
Ist R' Verfeinerung von R dann gilt:

$$ind(R) = ind(R') \iff R = R'$$

Definition 2.8 (Nerode-Relation)

1. *$L \subseteq \Sigma^*, \Sigma$ beliebiges, endliches Alphabet*
   L definiert folgende Äquivalenzrelation $\sim_L$ auf Σ^*:

$$x \sim_L x' \text{ gdw. } \forall\, y \in \Sigma^* \text{ gilt: } xy \in L \iff x'y \in L$$

2. *$ind(\sim_L) = |^{\Sigma^*}/_{\sim_L}|$ (Nerode-Index von L)*

Beispiel

$L = PARITY \quad x, x' \in \{0,1\}^* x \sim_L x'$
$01101y \qquad 01y \Rightarrow$ Frage: Sind beide Worte äquivalent? $\Rightarrow$ Zwei Fälle:

1. gerade Zahl von Einsen anhängen (als y): beide Worte in L
2. ungerade Zahl von Einsen anhängen (als y): beide Worte nicht in L

$x \sim_{PARITY} x' \iff x$ und x' haben beide ungradzahlig viele Einsen oder x und x' haben beide gradzahlig viele Einsen. $\Rightarrow ind(\sim_{PARITY}) = 2$

Definition 2.9
$x, x',\ y \in \Sigma^*$ haben die Eigenschaft $xy \in L$ und $x'y \notin L$ oder $xy \notin L$ und $x'y \in L$. In diesem Fall heißt y Zeuge, dass $x \not\sim_L x'$.

Beobachtung 2.4
$x \sim_L x' \Rightarrow$ Entweder $x, x' \in L$ oder $x, x' \notin L$, ansonsten ist bereits ϵ Zeuge für $x \not\sim_L x'$.

Beispiel:

L=$\{0^n, 1^n;\ n \geq 1\} \subseteq \{0,1\}^*$
Fall: $0^i \sim_L 0^j$
$i \neq j$ ist 1^i Zeuge dass $0^i \not\sim_L 0^j$, da $0^i 1^i \in L$ und $0^j 1^i \notin L$
$[0^i]_L = \{0^i\} \quad {}^{\{0,1\}^*}/_L \supseteq \{[0^i], i \geq 0\} \Rightarrow ind(\sim_L) = \infty$

Äquivalente Notation der Nerode-Relation

$L \subseteq \Sigma^*, x \in \Sigma^* \qquad L_x = \{x; xy \in L\}$

Beispiel:

PARITY $x \in \{0,1\}^*$

- x ungerade viele Einsen $L_x = \{y;\ y \text{ hat gradzahlig viele Einsen}\}$
- x gerade viele Einsen $L_x = \{y;\ y \text{ hat ungradzahlig viele Einsen}\}$

Beobachtung 2.5
$x \sim_L x' \iff L_x = L_{x'}$
($\Rightarrow$ y an x oder x' angehängt $\Rightarrow$ gleiches Ergebnis)
$\Rightarrow ind(L) = ind(\sim_L) = |^{\Sigma^*}/_{\sim_L}| = |\{L_x;\ x \in \Sigma^*\}|$

Theorem 2.1
Für alle Sprachen $L \subseteq \Sigma^*$ gilt:

1. *Wenn ind(L) endlich $\Rightarrow \exists$ DFA A_L für L mit ind(L) Zuständen*
2. *Ist A DFA für L $\Rightarrow$ A hat mindestens ind(L) Zustände*

Folgerung:

$\forall L \subseteq \Sigma^*$ gilt:

1. L ist regulär $\iff$ ind(L) ist endlich
2. Wenn L regulär $\Rightarrow$ ind(L) gibt die Größe eines minimalen DFA für L an

Beweis zu 1:

Voraussetzung: $L \subseteq \Sigma^*$ fixiert und ind(L) endlich.
Man konstruiert einen Automaten mit ind(L) Zuständen; ind(L) := I.
Es gilt:
$^{\Sigma^*}/_{\sim_L} = \{q_0, q_1, ..., q_{I-1}\}; \quad q_j \subseteq \Sigma^*; q_0 = [\epsilon]_{\sim_L}$ (Äquivalenzklasse des leeren Worts)
Beobachtung:

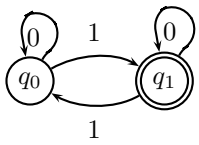
1. $\forall\ i = 0, ..., I-1$ entweder gilt $q_i \subseteq L$ oder $q_i \cap L = \emptyset$
2. $x \sim_L x' \iff \exists i \quad x_i, x' \in q_i$.
   $s \in \Sigma \quad xs \sim_L x's$, denn $L_{xs} = \{y; xsy \in L\}$ (Menge aller Worte nach x, die mit s anfangen) (da $x \sim_L x' = \{y; x'sy \in L = L_{x's}\}$)

Konstruiere $A_L : A_L = (Q_L; q_0; F_L; \delta_L) \qquad Q_L :=^{\Sigma^*}/_{\sim_L} = \{q_0, q_1, ..., q_{I-1}\}$
$q_0 := [\epsilon]_{\sim_L} \quad F_L = \{q_I;\ q_i \subseteq L\}$ (da die Äquivalenzklasse q_i entweder ganz in L oder nicht) (gerechtfertigt durch Beobachtung 1).
$\exists x$ mit $q_i = [x]_{\sim_L} \quad \delta_L(q_i, s) = q_j \Rightarrow q_j = [xs]_{\sim_L}$ (gerechtfertigt durch Beob 2).
Es bleibt zu zeigen: A_L berechnet L

- A_L hat die Eigenschaft dass $\forall\ x \in \Sigma^* \quad \delta_L^*(q_0, x) = q_j \quad mit\ q_j = [x]_{\sim_L}$ (=Äquivalenzklasse von x bezüglich Nerode)
- $[x]_{\sim_L} \subseteq L \iff x \in L \iff q_j \in F_L$

Beispiel

L=PARITY $\quad {}^{\{0,1\}^*}/_{\sim_L} = \{q_0, q_1\}$



q_0={x; x hat geradzahlig viele Einsen}
q_1={x; x hat ungeradzahlig viele Einsen}

Beweis zu 2:

Man definiert eine weitere Äquivalenzrelation $\sim_A$ auf Σ^*. $x \sim_A x' \Leftrightarrow \delta^*(q_0, x) = \delta^*(q_0, x')$ (äquivalent falls x und x' zum gleichen Zustand führen)
Beobachtung:
$|\Sigma^*/\sim_A| = \text{ind}(\sim_A) = |Q| \Rightarrow$ es genügt zu zeigen: $\sim_A$ ist eine Verfeinerung von $\sim_L$, denn dann gilt $\text{ind}(\sim_A) \geq \text{ind}(\sim_L)$.
$\Rightarrow$ es genügt zu zeigen: $x \sim_A x' \Rightarrow x \sim_L x'$
$x \sim_A x' \Leftrightarrow \delta^*(q_0, x) = \delta^*(q_0, x') \rightarrow \forall y \in \Sigma^*$ gilt $\delta^*(q_0, xy) = \delta^*(q_0, x'y)$
$\Rightarrow \forall y \in \Sigma^*$ gilt $xy \in L \Leftrightarrow q' \in F \Leftrightarrow x'y \in L \rightarrow x \sim_L x'$. $|A| \geq |A_L|$

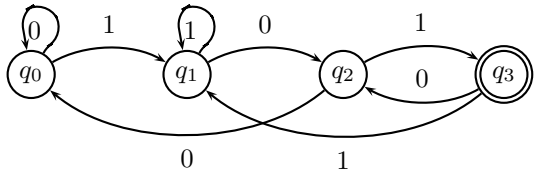
Bemerkung 2.2
Die Aussage ind($\sim_A$)= $|Q|$ gilt nur wenn alle Zustände $q \in Q$ von q_0 aus erreichbar sind, d.h. wenn für alle $q \in Q$ ein $x \in \Sigma^*$ exisitiert mit $\delta^*(q_0, x) = q$. Es gilt jedoch: In jedem DFA A können Zustände die nicht von q_0 aus erreichbar sind ersatzlos gestrichen werden ohne das Berechnungsverhalten zu ändern. Wir setzen im obigen Theorem voraus dass alle Zustände $q \in Q$ von A von q_0 aus erreichbar sein.

Beispiel

$L \subseteq \{0,1\}^* \quad L = \{x \in \{0,1\}^* \quad |x| \geq 3; x \text{ endet mit } 101\}$
$L_\epsilon = L$
$L_{(0)} = \{y; 0y \in L\} = L \Rightarrow 0 \sim_L \epsilon$
$L_{(1)} = \{y; 1y \in L\} = L \cup \{01\} \Rightarrow 1 \not\sim_L 0$
$L_{(10)} = L \cup \{1\}$
$L_{(11)} = L \cup \{01\}$
$L_{(00)} = L$
$L_{(01)} = L \cup \{01\}$
$L_{(000)} = L = L_{(100)}$
$L_{(001)} = L \cup \{01\} = L_{(1)} = L_{(01)} = L_{(11)} = L_{(111)} = L_{(011)}$
$L_{(010)} = L_{(110)} = L_{(10)} = L \cup \{1\}$
$L_{(101)} = L \cup \{01, \epsilon\}$

Beobachtung:
$\forall x = x_1, ...x_n \in \{0,1\}^*, n \geq 4$ gilt $L_x = L(x_{n-2}, xn-1, x)$
$\Rightarrow$ entscheidend sind nur die letzten drei Ziffern
$\{0,1\}^*/L = \{q_0, q_1, q_2, q_3\}$
$q_0 = \{\epsilon, 0, 00, 000, 100, ...\}$
$q_1 = \{1, 01, 11, 001, 011, 111, ...\}$

$q_2 = \{10, 110, 010, ...\}$
$q_3 = \{101, 0101, 1101, ...\}$



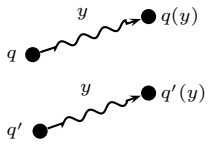
Praktisch interessant:

- Wie konstruiere ich aus einem gegebenen DFA A einen minimalen DFA der Sprache L(A)?
- Gibt es einen effizienten Algorithmus für Eingabe DFA A, Ausgabe minimalen DFA für L(A).

2.3 Minimierung endlicher Automaten

Gegeben ist ein DFA A=(Q, q_0, F, δ) über Σ

Definition 2.10 (Äquivalenzrelation $\sim$ auf Zuständen)
$q \sim q' \iff \forall\, y \in \Sigma^* :$
$\delta^*(q, y) \in F \iff \delta^*(q', y) \in F$



(beide akzeptiert oder beide nicht akzeptiert)
Äquivalente Definition: $\forall\, q \in Q :\ L_q = \{y;\ \delta^*(q, y) \in F\}$

Beobachtung 2.6

- *Sei* $L_{q_0} = L(A)$ *dann gilt:* $q \sim q' \iff L_q = L_{q'}$
- $q \sim q' \iff L(q) = L(q')$

Definition 2.11
q, q' *und* $y \in \Sigma^*$ *mit* $\delta^*(q, y) \in F$ *und* $\delta^*(q', y) \notin F$ *oder* $\delta^*(q, y) \notin F$ *und* $\delta^*(q', y) \in F$ *dann heisst* y *Zeuge für* $q \not\sim q'$.

Lemma 2.3 $\forall\, q \in Q \qquad L^{-1}(q) = \{x;\ \delta^*(q_0, x) = q\}$

Eigenschaften:

1. Falls $q \in F,\ q' \notin F \Rightarrow q \not\sim q'$ und ϵ ist Zeuge dafür $(\forall \in q \in Q;\ \delta^*(q, \epsilon) = q)$

2. $\forall\ q, q', a \in \Sigma$ gilt: $\quad q \sim q' \Rightarrow \delta(q, a) \sim \delta(q', a)$
   <u>denn wenn:</u> $q \sim q' \Rightarrow \forall\ y \in \Sigma^*$ gilt:
   $ay \in L(q) \Longleftrightarrow ay \in L(q') \Longleftrightarrow \forall\ y \in \Sigma^*$ gilt:
   $y \in L(\delta(q, a)) \Longleftrightarrow y \in L((\delta(q', a)) \Longleftrightarrow \delta(q, a) \sim \delta(q', a)$

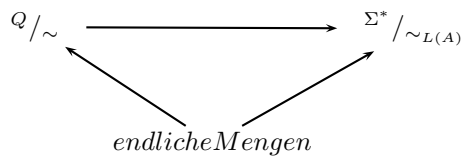
3. äquivalent zu 2: Falls für $q, q', a \in \Sigma$ gilt: $\quad \delta(q, a) \not\sim \delta(q', a) \Rightarrow q \not\sim q'$
   $\Rightarrow$ Rechtfertigen Definiton: ${}^A/_\sim = ({}^Q/_\sim,\ [q_0],\ {}^F/_\sim,\ ; \delta']$
   (Definition ${}^F/_\sim$ gerechtfertigt durch 1
   $(\delta'([q], a) = [\delta(q, a)]$, gerechtfertigt durch 2)
   $\Rightarrow$ DFA ${}^A/_\sim$ für L(A) mit $|A| = \text{ind}(\sim)$)

4. $\forall\ q, q'$ und $x \in L^{-1}(q),\ x' \in L^{-1}(q')$ gilt:
   $q \sim q' \Longleftrightarrow x \sim_{L(A)} x'$
   denn: $q \sim q' \Longleftrightarrow L(q) = L(q') \Longleftrightarrow \forall\ y \in \Sigma^*$ gilt:
   $\delta^*(q, y) \in F \Longleftrightarrow \delta^*(q', y) \in F \Longleftrightarrow \forall\ y \in \Sigma^*$ gilt:
   $\delta^*(q_0, xy) \in F \Leftrightarrow \delta^*(q_0, x'y) \in F \Leftrightarrow \forall y \in \Sigma^*, xy \in L(A) \Leftrightarrow x'y \in L(A) \Longleftrightarrow$
   $L(A)_x = L(A)_{x'} \Longleftrightarrow x \sim_{L(A)} x'$

Theorem 2.2
$\forall$ *DFAs A gilt:*

$$\textit{ind}(\sim) = |{}^Q/_\sim| = \textit{ind}(\sim_{L(A)}),\ \textit{d.h. } {}^A/_\sim \textit{ ist minimaler DFA.}$$

Beweis:



$\phi([q]) := [x]_{\sim_{L(A)}}$ für ein $x \in L^{-1}(q)$ (gerechfertigt durch 4)
Injektivität von ϕ: zu zeigen $\forall\ q, q''$ gilt:
$\phi(q) = \phi(q') \Rightarrow [q] = [q']\phi([q]) = \phi([q']) \Longleftrightarrow$
$[x]_{\sim_{L(A)}} = [x']_{\sim_{L(A)}}$ für$x \in L^{-1}(q), x' \in L^{-1}(q') \Longleftrightarrow q \sim q' \Longleftrightarrow [q] = [q']$
Surjektivität von ϕ: zu zeigen $\forall x \in \Sigma^* \exists\ q \in Q$ mit $\phi([q]) = [x]_{\sim_{L(A)}}$.
Wähle als q den Zustand $\delta^*(q_0, x)$
$\Rightarrow$ ϕ bijektiv. Definiere ϕ und zeige, dass Abbildung ϕ bijektiv (d.h. injektiv und surjektiv) ist.

2.3.1 Effizienter Algorithmus zur Berechnung von $^A/_\sim$ aus A

Datenstrukturen und Subroutinen

Eingabe : $A = (Q, q_0, F, \delta)$ verwalten für alle $i \neq j \in \{0, ..., s-1\}$ eine Liste L(i,j), in die Zustandspaare eingehängt werden können. Der Algorithmus markiert alle Paare nicht-äquivalenter Zustände mittels **markiere(q_i, q_j)** und benutzt dazu eine Funktion **Markiere(q_i, q_j)**.

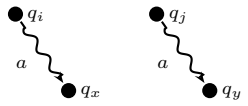
Ausgabe:

	q_0	q_1	..	j	..	q_{s-1}
q_0						
q_1						
..						
i				$*$		
..						
q_{s-1}						

MinDFA

1. Entferne alle nicht erreichbaren Zustände ($\Rightarrow A = (Q, q_0, F, \delta)$ mit $Q = \{q_0, q_1, ..., q_{s-1}\}$)
2. Initialisiere alle L(i,j) mit $\emptyset$
3. for (i=0,...,s-1)
   for (j=0,...,s-1)
   if ($q_i \not\equiv q_j$) // Ein Zustand akzeptiert, der andere nicht
   then markiere (q_i, q_j)
4. for (i=0,...,s-1)
   for (j=0,...,s-1)
   for all $a \in \Sigma$
   if($\delta(q_i, a), \delta(q_j, a)$) unmarkiert
   then füge (q_i, q_j) in L($\delta(q_i, a), \delta(q_j, a)$) ein;
   else Markiere (q_i, q_j);break;

Erläuterung: Untersuche (q_i, q_j) - falls für $a \in \Sigma$ $\delta(q, a) \not\sim \delta(q', a)$ bereits erkannt $\Rightarrow q \not\sim q'$ (falls nicht äquivalent $\Rightarrow$ Abbruch)



falls $\delta(q, a), \delta(q', a)$ nicht markiert $\Rightarrow$ fügen (q, q') in L($\delta(q, a), \delta(q', a)$) ein. Bewirkt: falls $\delta(q, a) \not\sim \delta(q', a)$ zu späterem Zeitpunkt erkannt, kann (q, q') nachträglich markiert werden $\Rightarrow$ markiere $(q_i, q_j) \Rightarrow$ schreibe '*' and die Stelle (i, j).
Markiere $(q_i, q_j) \Rightarrow$ markiert(q_i, q_j) und alle $(q, q') \in$ L(i, j) und alles was in L(q, q') liegt.

Markiere(q_i, q_j)

- markiere(q_i, q_j)
- streiche (q_i, q_j) aus allen Listen L($\delta(q_i, a), \delta(q_j, a)$) in die es vorher geschrieben wurde
- falls L$(q_i, q_j) \neq \emptyset$ für alle $(q, q') \in L(q_i, q_j)$. Markiere (q, q') (Rekursion)

Korrektheit

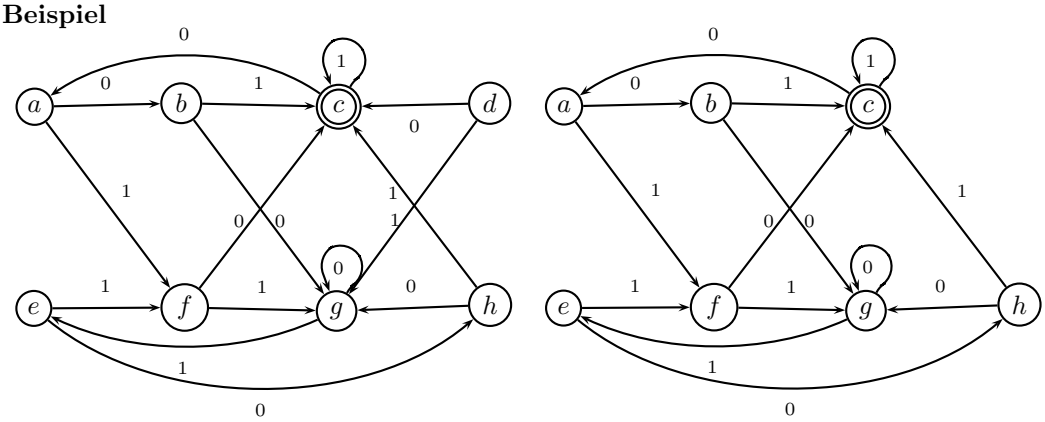
Beobachtung 1: alle markierten (q_i, q_j) sind nicht äquivalent (folgt aus Eigenschaft 1 und 2).
zu zeigen: alle a, q' mit $q \not\sim q'$ werden markiert. Nehme das Gegenteil an:
$\exists (q, q')$ nicht markiert und $q \not\sim q' \Rightarrow \exists$ Zustandspaar (q^*, q'^*) mit kürzestem Zeugen x^* mit dieser Eigenschaft.
Wir wissen $|x^*| > 0$, ansonsten wird (q^*, q'^*) im Schritt 3 markiert $\Rightarrow x^* = ax'$, betrachten (r, r') mit $r = \delta(q^*, a)$ und $r' = \delta(q^*, a)$
$q^* \xrightarrow{a} r \xrightarrow{x^*} \in F$
$q'^* \xrightarrow{a} r' \xrightarrow{x^*} \notin F$
Beobachtung 2: (r, r') hat Zeuge x' kürzer als $x^*, r \not\sim r'$ (aus Eigenschaft 2) $\Rightarrow (r, r')$ wird markiert.
Fall 1: (r, r') wird vor (q^*, q'^*) verarbeitet $\Rightarrow (q^*, q'^*)$ wird in der 'for all $a \in \Sigma$' - Schleife markiert
Fall 2: (r,r') wird nach $(q^*, q'^*$ verarbeitet $\Rightarrow (q^*, q'^*)$ wird in die Liste $L(r, r')$ geschrieben und bei Markiere(r, r') markiert $\Rightarrow$ Widerspruch.

Laufzeit Min-DFA

- Entfernen von Zuständen, die nicht von q_0 aus erreichbar sind
  DFS-Durchlauf (Tiefensuche) startend von q_0: $O(|\Sigma| * |Q|)$
- Schritte 2 und 3: $O(|Q|^2)$
- Schritt 4: es werden folgende Operationen durchgeführt:
  - $T(\cdot, \cdot)$ auf 1 gesetzt: ≤ 1 pro (q, q')
  - Zustandspaare in Listen geschrieben: $\leq |\Sigma|$ pro (q, q')
  - Zustandspaare aus Listen gestrichen: $\leq |\Sigma|$ pro (q, q')
    (durch Streichen wird Markiere(q, q') höchstens einmal aufgerufen (pro (q, q'))

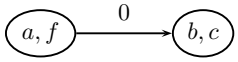
$\Rightarrow O(|\Sigma| * |Q|^2)$ Gesamtlaufzeit

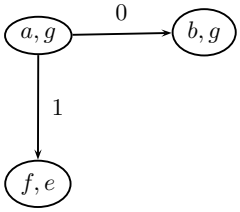
Beispiel

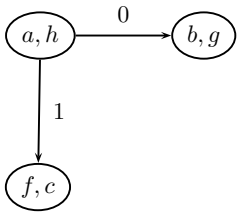


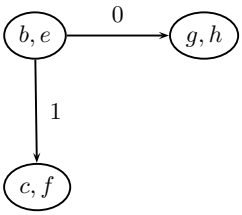
	a	b	c	e	f	g	h
a		1	1		1	1	1
b			1	1	1	1	
c				1	1	1	1
e					1	1	1
f						1	1
g							1
h							

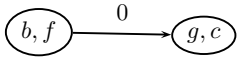
Anmerkung:
Zwei Zustände werden mit einer 1 markiert falls sie nicht äquivalent sind. Wegen der Symetrie der Äquivalenzrelation en reicht es aus das obere Dreieck der Matrix zu betrachten.

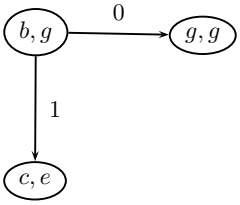








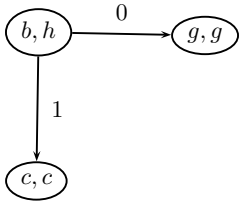


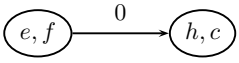


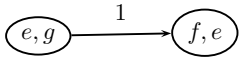
$\boxed{L(b,h)}$
(a,e)

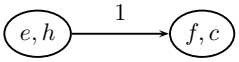
$\boxed{L(b,g)}$
(a,g)

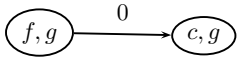
$\boxed{L(e,f)}$
(a,g)

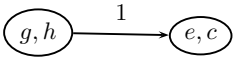




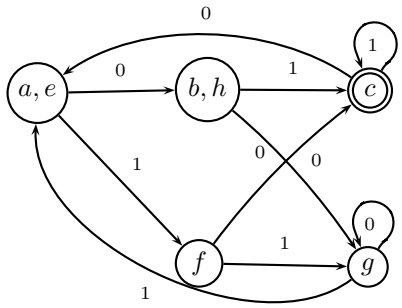








Minimaler DFA



2.4 Operationen auf und Konstruktionshilfen für endliche Automaten

Bezeichnungsweise:

- Sprachen $L \subseteq \Sigma^*$ äquivalente Schreibweise: $L : \Sigma^* \to \{0,1\}$

$$L(x) = \begin{cases} 1 & : \quad x \in L \\ 0 & : \quad x \notin L \end{cases}$$

- $A(Q, q_0, F, \delta), F \subseteq Q$ äquivalente Schreibweise: $F : Q \to \{0,1\}$

$$F(q) = \begin{cases} 1 & : \quad q \in F \\ 0 & : \quad q \notin F \end{cases}$$

erlaubt:

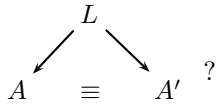
- $L_1 \cap L_2$ ist äquivalent zu $L_1 \wedge L_2$
- $x \in L_1 \cap L_2 \Leftrightarrow x \in L_1$ und $x \in L_2 \Leftrightarrow L_1(x) = 1$ und $L_2(x) = 1 \Leftrightarrow (L_1 \wedge L_2)(x) = 1$
- $L_1 \cup L_2$ ist äquivalent zu $L_1 \vee L_2$
- $L_1 \triangle L_2$ ist äquivalent zu $L_1 \oplus L_2$

Notation

erlaubt boolsche Ausdrücke über Sprachen L_1, L_2, L_3, L_4
$L := (L_1 \wedge L_2) \vee (L_3 \oplus L_4)$ $\quad x \in L$ gdw. $x \in L_1 \cap L_2$ oder $x \in L_3 \triangle L_4$
Beispiel: Komplement einer Sprache L ist $\Sigma^* \setminus L$ äquivalente Notation ist $\neg L$
$(\neg L)(x) = 1 \Leftrightarrow L(x) = 0 \Longleftrightarrow x \in \Sigma^* \setminus L$
Frage: Gegeben sind reguläre Sprachen L_1, L_2 durch DFAs A_1, A_2. Wie konstruiert man DFAs für $L_1 \cap L_2, L_1 \cup L_2, L_1 \triangle L_2$ oder ähnliche? (äquivalent $L_1 \wedge L_2, L_1 \vee L_2, L_1 \oplus L_2$)

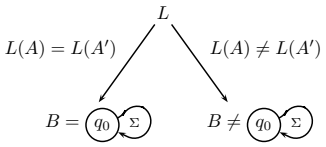
Standard-Anwendungen

1. Verifikation

   Wie testet man  ?

   d.h. $A(x) = A'(x)\ \forall\ x \in \Sigma^*$

   - konstruiere DFA für $L(A) \triangle L(A') := B$
   - Minimiere(B)

2. Synthese
   einfache DFAs $A_1, ...A_K$. Suche DFA A der Art $L = f(L(A_1), ..., L(A_K))$ berechnet mit $f : \{0,1\}^k \to \{0,1\}$ komplizierte boolsche Operationen.
   Gibt es effiziente Synthesealgorithmen?

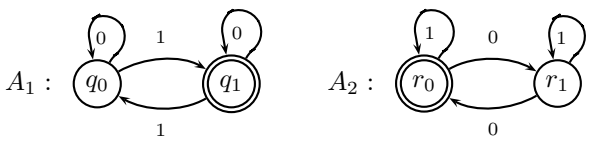
   Beispiele:

   (a) $L = \{x \in \{0,1\}^*$; x hat ungradzahlig viele "1" und gradzahlig viele "0"$\}$
   $\Rightarrow L = L_1 \wedge L_2$:
   $L_1 = PARITY \qquad L_2 = \{x \in \{0,1\}^*$; x hat gradzahlig viele Nullen$\}$

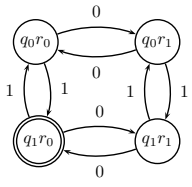
   (b) $A_1 = (Q_1, q_0^1, F_1, \delta_1)$ für L_1
   $A_2 = (Q_2, q_0^2, F_2, \delta_2)$ für L_2
   $F_1 : Q_1 \to \{0,1\}$
   $F_2 : Q_2 \to \{0,1\}$
   A für $L_1 \cap L_2, A = (Q, q_0, F, \delta)$
   $Q = Q_1 \times Q_2 = \{(q_1, q_2), q_1 \in Q_1, q_2 \in Q_2\} \quad q_0 = (q_0^1, q_0^2) \quad F = Q_1 \times Q_2 \to \{0,1\} \quad F(q_1, q_2) = 1 \Leftrightarrow F_1(q_1) = 1 \wedge F_2(q_2) = 1 \Leftrightarrow F(q_1) \wedge F(q_2)$
   $\delta((q_1, q_2); a) = (\delta_1(q,a), \delta_2(q_2, a)) = F(q_1) \wedge F(q_2)$
   (Idee: Parallele Ausführung beider Teilautomaten im Gesamtautomat)

Beispiel

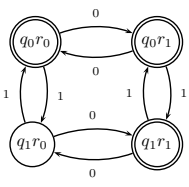
L_1=PARITY, $L_2 = \{x \in \{0,1\}^*$, gradzahlig viele Nullen$\}$
$\Rightarrow L_1 \cap L_2 \equiv$ ungerade Zahl Einsen und gerade Zahl Nullen.



$A_1 \cap A_2$:



$A_1 \cup A_2$:



($A_1 \cup A_2$: berechnet $L_1 \cup L_2 = \{x$; ungerade viele Einsen oder gerade viele Nullen$\}$)

Definition 2.12
$k \geq 1,\ L_1, ..., L_K$ *Sprachen über* Σ *(endliches Alphabet)*
$f : \{0,1\}^k \to \{0,1\}$
definiere $L = f(L_1, ..., L_k)$
$x \in L (bzw.\ L(x) = 1) \iff f(L_1(x), ..., L_k(x)) = 1$
kurz: $L(x) = f(L_1(x), ..., L_k(x))$

Beispiel

$L_1 \cap L_2,\ k = 2,\ f \equiv x_1 \wedge x_2 = f(L_1, L_2) = L_1 \wedge L_2$

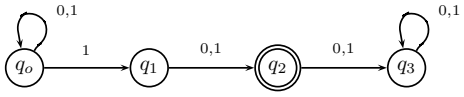
Definition 2.13 (Produktautomat)
$k \geq 1$; *DFAs* $A_1, ..., A_k$ *über* $\Sigma \quad f : \{0,1\}^k \to \{0,1\} \quad A_i = (Q_i, q_0^i, F^i, \delta^i)$
definiere Produktautomaten A:
$A = f(A_1, ..., A_k); \quad A = (Q, q_0, F, \delta)$ *mit* $Q = Q_1 \times ... \times Q_k$
$q_0 = (q_0^1, ..., q_0^k) \quad a \in \Sigma \quad \delta((q_1, ..., q_k), a) = (\delta^1(q_1, a), ..., \delta^k(q_k, a))$
$F(x) = 1 \iff f(F^1(x), ..., F^k(x)) = 1 \quad (x \in F)$

Theorem 2.3 $A : f(A_1, ..., A_k)$ *berechnet* $f(L(A_1), ..., L(A_k))$

Beweis:
$x \in L \Leftrightarrow f(L_1(x), ..., L_k(x) = 1 \Leftrightarrow f(F^1(\delta_1^*(q_0^1, x)), ..., F^k(\delta_k^*(q_0^k, x))) = F(\delta(q_0, x))$

Beobachtung 2.7
$|f(A_1, ..., A_k)| = |A_1| * |A_2| * ... * |A_k|$
(Der Produktautomat ist im Allgemeinen nicht mininal, auch wenn es die Faktor-automaten, aus denen er zusammengesetzt ist, sind.)

2.5 Nichtdeterministische, endliche Automaten (NFA)



q_o ist ein nichtdeterministischer Zustand, denn wenn in q_0 Eins eingegeben wird, darf man sowohl in q_0 bleiben als auch nach q_1 wechseln.
Wirkung: für jede Eingabe kommen verschiedene Zustände als Endzustände in Frage (Bezeichnung: $\delta^*(q_0, x)$ = Menge der Zustände, die als Zielzustände für x fungieren können

Beispiel

$\delta^*(q_0, 100) = \{q_0, q_3\}$ nicht akzeptiert
$\delta^*(q_0, 0000) = \{q_0\}$ nicht akzeptiert
$\delta^*(q_0, 01) = \{q_0, q_1\}$ nicht akzeptiert
$\delta^*(q_0, 011) = \{q_0, q_1, q_2\}$ akzeptiert
$\Rightarrow F = \{q_2\}$

Definition 2.14
NFA akzeptiert eine Eingabe $x \Leftrightarrow$ es existiert ein mit x konsistenter Berechnungsweg, der in einem akzeptierten Zustand endet. Formal: $\delta^*(q_0, x) \cap F \neq \emptyset$.
NFA $N = (Q, q_0, F, \delta)$ Q, q_o, F wie bei DFAs, $\delta : Q \times \Sigma \rightarrow 2^Q := \{Q' \subseteq Q\}$
Beispiel: $N = \{\{q_0, q_1, q_2, q_3\}, q_0, \{q_2\}, \delta\}$
$\delta(q_0, 0) = \{q_0\}$
$\delta(q_0, 1) = \{q_0, q_1\}$
$\delta(q_1, 0) = \{q_2\}$...

Definition 2.15 *Berechnungsverhalten*
definiere: Zustandsfolge $q^{(1)}, q^{(2)}, \ldots, q^{(s)}$ aus Q heißt eine mit $x \in \Sigma^*$ konsistente Berechnung falls $x = x_1, ..., x_{s-1}$ und $q^{(2)} \in \delta(q^{(1)}, x_1), q^{(3)} \in \delta(q^{(2)}, x_2), \ldots, q^{(s)} \in \delta(q^{(s-1)}, x_{s-1})$
$q^{(s)}$= Endzustand der Berechnungsfolge auf x.
$\delta^*(q, x)$ = Menge der Zustände q' für die es eine mit x konsistente, in q' endende und q beginnende Berechnung gibt.
formal: $\forall\, a \in \Sigma \quad \delta^*(q, a) = \delta(q, a) \quad |x| > 1$, *d.h.* $x = x'a$
$\delta^*(q, x) = \bigcup_{q' \in \delta(q, x')} \delta(q', q)$

Definition 2.16
$L(N) = \{x \in \Sigma^*, \delta^*(q_0, x) \cap F \neq \emptyset\}$ *($\exists$ eine mit x konsistente Berechnung von q_0 in einem akzeptierten Zustand)*

Motivation

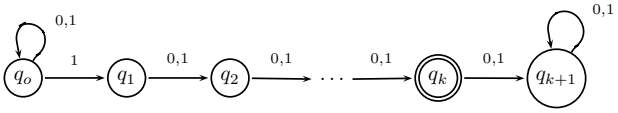
Beobachtung: NFA können potentiell mehr als DFAs (jeder DFA ist ein NFA)

1. Können NFAs nichtreguläre Sprachen berechnen?
   Antwort: Nein. $\exists$ Algorithmus, der jeden NFA in einen äquivalenten DFA überführt (etwas später).

2. $\exists$ Sprachen, für die ein NFA exponentiell kleiner ist als der minimale DFA, d.h. NFAs gestatten eine wesentlich komprimiertere Repräsentierung regulärer Sprachen.

Beispiel

$L_k = \{x \in \{0,1\}^*; |x| \geq k$ und k-te Stelle von hinten ist Eins$\}$

NFA für L_k: 

$\Rightarrow L_k$ hat NFA mit k+2 Zuständen

ind(L_k)?
Voraussetzung: x, x' sind beliebige Worte mit $|x| \geq k, |x'| \geq k$.
Behauptung: $x \sim_L x' \iff$ x und x' stimmen auf den letzten k Stellen überein
denn: Nichtübereinstimmung an i-ter Stelle von hinten $1 \leq i \leq k$:
$|\{xy, x'y\} \cap L_k| = 1$ für y mit $|y| = k-1$
Folgerung: ind(L_k)$\geq 2^k$ da 2^k Möglichkeiten für den Suffix der Länge k bestehen.
denn: $[x]_{L1} \cap [x'] = \emptyset$ für alle x, x' die $|x| \geq k$ und $|x'| \geq k$ erfüllen, die sich auf den letzten k Stellen unterscheiden. $\Rightarrow$ minDFA für L_k hat $\geq 2^k$ Zustände.
Alle zu NFAs äquivalente DFAs haben $\geq 2^k$ Zustände.

Bemerkung 2.3
Gegeben sei ein NFA $N = \{Q, q_0, F, \delta\}$ $\quad \delta : Q \times \Sigma \to 2^Q$ $\quad$ *und* $x \in \Sigma^* : N(x)$ *kann mittels DFS (Tiefensuche) in* $O(|\Sigma| * |Q|)$ *berechnet werden.*

Satz 2.1
Jeder NFA der Größe s (s = Anzahl der Zustände) kann durch einen DFA der Größe $\geq 2^s$ *simuliert werden.*

Beweis durch Potenzmengen-Konstruktion

Algorithmische Idee: Für eine gegebene Eingabe $x = (x_1, ..., x_n)$ verwalten wir für alle $i = 0, ..., n$ die Menge der Zustände, die durch $(x_1, ..., x_i)$ erreicht werden. Wir bezeichnen diese Menge als $Q_0(x), Q_1(x), ..., Q_n(x)$.
Wir wissen:

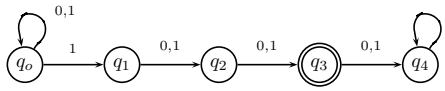
- $Q_0(x) = \{q_0\}$
- x muß akzeptiert werden $\iff Q_n(x) \cap F \neq \emptyset$
- $Q_{i+1}(x) = \bigcup_{q \in Q(x)} \delta(q, x_{i+1})$

Dieser Algorithmus muß nun durch einen DFA nachgebildet werden.
Gegeben: $N = (Q, q_0, F, \delta)$ $\quad$ NFA $\delta : Q \times \Sigma \to 2^Q$
Simulation leistet DFA $A = (\bar{Q}, \bar{q_0}, \bar{F}, \bar{\delta})$
$\bar{Q} = 2^Q$ $\quad \bar{q_0} = \{q_0\}$ $\quad \bar{F} = \{Q' \subseteq \bar{Q}; Q' \cap F \neq \emptyset\}$ $\quad \delta(Q', a) = \bigcup_{q \in Q'} \delta(q, a)$
$a \in \Sigma, Q' \subseteq \bar{Q}$
Behauptung: A berechnet die gleiche Sprache wie L (d.h. L(N))
Beweis: Es genügt zu zeigen: $\forall x \in \Sigma^*$ gilt: $\delta^*(\{q_0\}, x) = \delta^*(q_0, x)$
denn: $x \in L(A) \iff \delta^{-*}(\{q_0\}, x) \in F \iff \delta^{-*}(\{q_0\}, x) \cap F \neq \emptyset \iff$

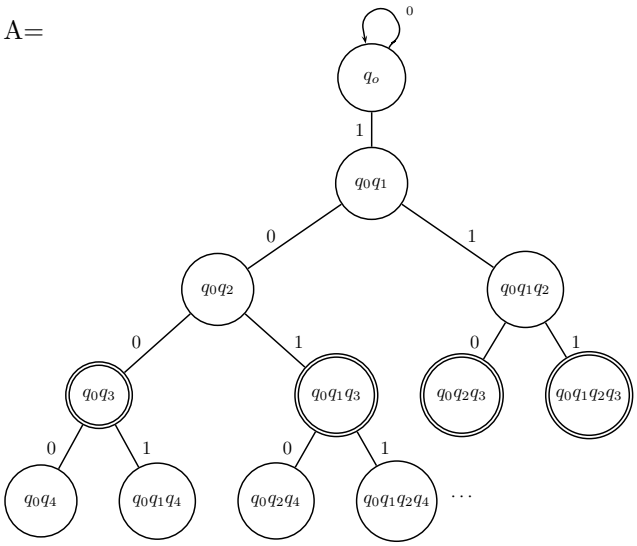
$\delta^*(q_0, x) \cap F \neq \emptyset \Longleftrightarrow x \in L(N)$
Beweis: Per Induktion über Länge von x (folgt direkt aus der Definition)

Folgerungen

1. NFAs können nur reguläre Sprachen berechnen
2. Potenzmengen-Konstruktion ist asymptotisch optimal ((siehe Beispiel k-te Stelle von hinten ist Eins) ⇒ exponentieller Blow-Up in Zustandsgröße NFA - simulierender DFA)
3. Algorithmische Realisierung: In der Regel werden durch Potenzmengen-Konstruktion nicht erreichbare Zustände generiert ⇒ besser: Generiere A aus N, indem auf Basis von DFS oder BFS alle von $\{q_0\}$ aus erreichbaren Zustände generiert werden.

   Beispiel: N= 

(dritte Stelle von hinten ist Eins)

   A= 

4. A muß in der Regel noch minimiert werden
5. Achtung: Die Karthesische-Produkt-Konstruktion funktioniert für NFAs im Allgemeinen nicht, das heißt $N_1, ..., N_k$ NFAs und $f : \{0,1\}^k \to \{0,1\}$ so berechnet $N = f(N_1, ..., N_k)$ im Allgemeinen nicht die Sprache $f(L(N_1), ..., L(N_k))$.

Das Minimierungsproblem für NFAs ist (wahrscheinlich) nicht effizient lösbar (da NP-schwer).

2.6 Reguläre Ausdrücke

Definition 2.17 (Reguläre Ausdrücke über (rekursiv) gegebenem, endlichen Alphabet Σ)

1. *$\epsilon, \emptyset$ und a sind für alle $a \in \Sigma$ reguläre Ausdrücke*
2. *Ist R ein regulärer Ausdruck, so auch $(R)^*$*
3. *sind R_1, R_2 reguläre Ausdrücke, so auch $(R_1) + (R_2)$ und $(R_1) * (R_2)$*

Beispiel: reguläre Ausdrücke über $\{0,1\}$ $\quad 0, 1, \epsilon, \emptyset$
$((0)^*) + ((1)^*) \quad (((0)^*) * ((1)^*)) + (\epsilon) \quad (10^*11) + (01^*00)$

Konvention:

- *wir schreiben R_1R_2 statt $R_1 * R_2$*
- *R^k für $\underbrace{R * R * R * ... * R}_{k-mal}$*
- *Regeln zum Weglassen von Klammern: Hierarchie der Bindekraft von Operatoren: * stärker $*$ stärker $+$*
  Beispiele: $R_1 * R_2 + R3 \equiv ((R_1) * (R_2)) + (R_3), \quad R_1 * R_2^* \equiv (R_1) * ((R_2)^*)$

2.6.1 Semantik von regulären Ausdrücken

Jedem regulärem Ausdruck $R \in (\Sigma \cup \{(,), *, +, \emptyset, \epsilon\})^*$ wird eine Sprache $L(R) \subseteq \Sigma^*$ zugeordnet.

- $L(\emptyset) := \emptyset, \quad L(\epsilon) := \{\epsilon\}, \quad L(a) = \{a\} \quad \forall a \in \Sigma$
- $L(R^*) = L(R)^*$ der kleensche Abschluß von $L(R)$
- $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
- $L(R_1 * R_2) = L(R_1) * L(R_2)$

Definition 2.18
Sei $L, L' \subseteq \Sigma^*$ dann gilt:
$L * L' = \{ww';\ w \in L,\ w' \in L'\}$ *(Konkatenation)*
$\Rightarrow L^k = \{w^{(1)}, ..., w^{(k)};\ w^{(1)}, ..., w^{(k)} \in L\}$
$L^k = \bigcup_{k=0}^{\infty} L^k = \{\epsilon\} \cup \{w;\ \exists k$ *und Unterteilung* $w = w^{(1)}, ..., w^{(k)}$ *mit* $w^{(i)} \in L \ \forall\ i = 1, ..., k\}$

Beispiele

- $(0+1)^* = (\{0\} \cup \{1\})^*$ kleenscher Abschluß $\Longleftrightarrow L^* = \epsilon \cup \{w,\ \exists\ k \geq 1\ \ w = w^{(1)}, ..., w^{(k)}; w^{(i)} \in L;\ i = 1, ..., k\} = \{0,1\}^* \Rightarrow$ Menge aller Worte über 0,1
- k-te Stelle von hinten ist Eins: $(0+1)^* * 1 * (0+1)^{k-1}$
- $PARITY$: - später -

- $PARITY^* : \epsilon \cup \{w; \exists k \geq 1, w = w^{(1)}, ..., w^{(k)}, w^{(i)} \in PARITY, i = 1, ..., k\} = (\{0,1\}^* \setminus \{0\}^*) \cup \{\epsilon\}$, denn: hat w mindestens eine Eins $\rightarrow$ w kann in Teilwerte zerlegt werden, die alle genau eine Eins haben, diese Teilwerte sind in $PARITY$.

Satz 2.2 *Für alle regulären Ausdrücke R gilt: $L(R)$ ist regulär.*

Beweis:
Behauptung klar für $R \in \{\emptyset, \epsilon\} \cup \Sigma$. Es genügt zu zeigen: Wenn R, R_1, R_2 reguläre Ausdrücke sind, für die $L(R), L(R_1), L(R_2)$ regulär $\Rightarrow L(R_1 + R_2), L(R_1 * R_2), L(R^*)$ regulär. Wir fixieren DFAs:

$A_1 = (Q_1, q_0^1, F_1, \delta_1) \quad L(R_1)$
$A_2 = (Q_2, q_0^2, F_2, \delta_2) \quad L(R_2)$
$A = (Q, q_0, F, \delta) \quad L(R)$

Wir konstruieren NFAs für $L(R_1 + R_2), L(R_1 * R_2), L(R^*)$:
Ein DFA für $L(R_1 + R_2) \stackrel{Def.}{=} L(R_1) \cup L(R_2)$ ist $A_1 \vee A_2$
(und hat $|Q_1| * |Q_2|$ Zustände).

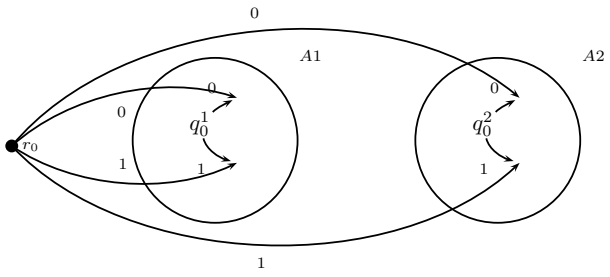
<u>Alternativkonstruktion:</u>

1. kleinerer NFA für $L(R_1 + R_2)$ (mit $|Q_1| + |Q_2| + 1$ Zuständen).
   algorithmische Idee: Raten $i \in \{1, 2\}$, berechnen $A_i(x)$ (d.h. x wird genau dann akzeptiert wenn $A_1(x) = 1$ oder $A_2(x) = 1$, d.h. $L(R_1) \cup L(R_2)$ wird berechnet).
   Definieren: NFA $N^\cup = (Q^\cup, q_0^\cup, F^\cup, \delta^\cup) \qquad Q^\cup = (Q_1 \dot{\bigcup} Q_2 \dot{\bigcup} \{r_0\})$
   ($M \dot{\bigcup} M' \equiv$ disjunkte Vereinigung)
   $q_0^\cup = \{r_0\}$

$$F^\cup = \begin{cases} F_1 \cup F_2 & \text{falls } \epsilon \notin L(R_1) \cup L(R_2) \\ F_1 \cup F_2 \cup \{R_0\} & \text{falls } \epsilon \in L(R_1) \cup L(R_2) \end{cases}$$

   Bemerkung: $\epsilon \in L(R_i) \Longleftrightarrow q_0^i \in F_i \quad i = 1, 2$

$$\delta^\cup(q, a) = \begin{cases} \{\delta^1(q, a)\} & \text{falls } q \in Q_1 \\ \{\delta^2(q, a)\} & \text{falls } q \in Q_2 \\ \{\delta^1(q_0^1, a), \delta^2(q_0^2, a)\} & q = r_0 \end{cases}$$

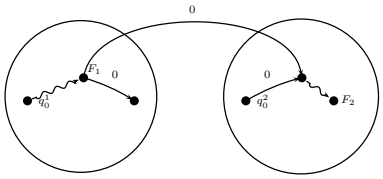


2. Wir konstruieren einen NFA $N^0 = \{Q^0, q_0^0, F^0, \delta^0\}$ für $L(R_1) * L(R_2) = L(A_1) * L(A_2)$
   $Q^0 = Q_1 \cup Q_2 \qquad q_0^0 = q_0^1$

$$\delta^0(q,a) = \begin{cases} \{\delta^1(q,a)\} & q \in Q_1 \setminus F_1 \\ \{\delta^2(q,a)\} & q \in Q_2 \\ \{\delta^1(q,a), \delta^2(q_0^2,a)\} & q \in F_1 \end{cases}$$

$$F^0 = \begin{cases} F_2 & q_0^2 \notin F_2 \\ F_1 \cup F_2 & q_0^2 \in F_2 \end{cases}$$

   $x \in L(R_1 * R_2) \iff \exists\, x = x'x'' \quad x' \in L(R_1), x'' \in L(R_2)$
   Idee: Rate (nicht deterministisch) den Anfang von x''



   Bemerkung: $L_1 \subseteq L_1 * L_2 \leftrightarrow \epsilon \in L_2 \qquad w = w\epsilon$ ist Partition für alle $w \in L_1$

3. Konstruiere NFA $\bar{N} = (\bar{Q}, \bar{q_0}, \bar{F}, \bar{\delta})$ für $L(A)^*$
   $\bar{Q} = Q \cup \{r_0\} \qquad \bar{F} = F \cup \{r_0\} \quad \bar{q_0} = r_0$

$$\bar{\delta}(q,a) = \begin{cases} \{\delta(q,a)\} & \text{für } q \notin \bar{F}, q \neq r_0 \\ \{\delta(q,a), \delta(q_0,a)\} & q \in \bar{F} \\ \delta(q_0,a) & q = r_0 \end{cases}$$

   Idee: In akzeptierenden Zuständen wird nicht deterministisch geraten ob ein neues Wort der Partition anfängt oder nicht.

Korrektheitsbeweis

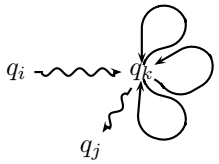
$x \in L(A)^* \iff x \in \epsilon$ oder $\exists\, k \geq 1 \quad x = x^{(1)}, ..., x^{(k)}; x^{(i)} \in L(A)$
$\forall\, i = 1, ..., k \iff \exists$ akzeptierte Berechnung von $\bar{N}$ auf x (springe nach jedem $x^{(i)}$ zum Anfangszustand).

Satz 2.3
Für jede reguläre Sprache $L \subseteq \Sigma^*$ existiert ein regulärer Automat R mit $L(R) = L$. Folgerung: Reguläre Ausdrücke, DFAs und NFAs sind äquivalente Berechnungsmodelle.

Beweis:
Wir zeigen einen Algorithmus der für jeden DFA einen regulären Ausdruck R konstruiert mit $L(A) = L(R)$:
Fixiere $s \geq 1$ und DFA $A = (Q, q_0, F, \delta); \quad Q = \{q_1, ..., q_s\}$
$L_{ij}^k = \{x \in \Sigma^*; \delta^*(q_i, x) = q_j$ und alle Zwischenzustände q_l von q_i durch x nach q_j

erfüllen $l \leq k\}$ $\quad 1 \leq i,j \leq s; \quad 0 \leq k \leq s$
$L(A) = \bigcup_{j,q_j \in F} L_{1j}^s$ $\quad (L_{ij}^s = \{x, \delta^*(q_i,x) = q_j\}$ konstruiere für alle i,j,k einen regulären Ausdruck R_{ij}^k für $K_{ij}^k($ $\Rightarrow R = \sum_{j,q_j \in F} R_{1j}^s)$
Konstruktion vom $R_{ij}^0 : L_{ij}^0 = \{a \in \Sigma; \delta(q_i,a) = q_j\}$ $\quad R_{ij}^0 = \sum_{a \in \Sigma, \delta(q_i,a)=a_j} a$
Es gilt $(\forall k \geq 1; 1 \leq i,j \leq s)$ $\quad R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$, daraus kann ein Algorithmus zur Konstruktion von R direkt abgeleitet werden. L_{ij}^k enthält nur Worte $x \in \Sigma^*$ mit $\delta^*(q_i,x) = q_j$.
Wir betrachten $q_i \to^x q_{l_1} \to^{x_2} ... \to^{x_{n-1}} q_{l_{n-1}} \to^{x_n} q_j$ $\quad x \in Lij^k \Rightarrow l_r \leq k$
$\forall r = 1, ..., n-1$
Fallunterscheidung:

1. $\exists$ kein r mit $l_r = k \Rightarrow x \in L_{ij}^{k-1}$
2. $\exists r$ mit $l_r = k$

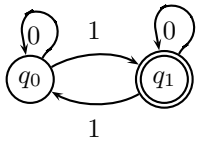


Auf Stücken zwischen q_i und q_k, q_k und q_k, q_k und q_j kommt q_n nicht vor.

$\exists\, t \geq 2$ mit $x = x^{(1)} * x^{(2)} * ... * x^{(t-1)} * x^{(t)}$
$\Rightarrow x \in L_{ik}^{k-1}(L_{kk}^{k-1})^* L_{kj}^{k-1} \Rightarrow L_{ij}^k = L_{ij}^{k-1} \cup L_{ik}^{k-1}(L_{kk}^{k-1})^* L_{kj}^{k-1}$
$\Rightarrow$ falls reguläre Ausdrücke für $L_{i'j'}^k$ für alle i', j' bekannt $\Rightarrow R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}$.

Beispiel: Regulärer Ausdruck für PARITY

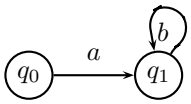
betrachten:



$R = R_{12}^2 = R_{12}^1 + R_{12}^1(R_{22}^1)^* R_{22}^1 = R_{12}^1(R_{22}^1)^* = 0^*1(0 + 10^*1)^*$

allgemein gilt:
$R_{ij}^j = R_{ij}^{j-1} + R_{ij}^{j-1}(R_{jj}^{j-1})^* R_{jj}^{j-1} = R_{ij}^{j-1}\epsilon + R_{ij}^{j-1}[(R_{jj}^{j-1})^* \setminus \epsilon] = R_{ij}^{j-1}(R_{jj}^{j-1})^*$

$R_{i1}^1 = R_{i1}^0(R_{11}^0)^*$



$R_{ij}^i = R_{ij}^{i-1} + \underbrace{R_{ij}^{i-1}(R_{ii}^{i-1})^*}_{(R_{ii}^{i-1})^*} R_{ij}^{i-1} = (R_{ii}^{i-1})^* R_{ij}^{i-1}$

$R_{12}^1 = (R_{11}^0)^* R_{12}^0 = 0^*1$

$R_{22}^1 = R_{22}^0 + R_{21}^0(R_{11}^0)^* R_{12}^0 = 0 + 10^*1$

Kapitel 3

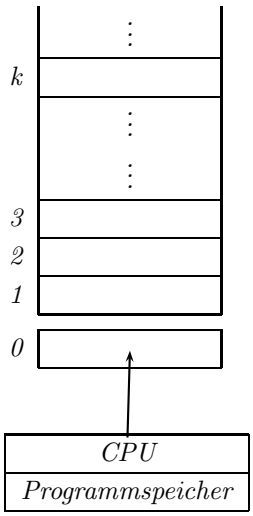
Grundlegende uniforme Berechnungsmodelle

Uniforme Berechnungsmodelle dienen zur formalen Modellierung von Algorithmen.

3.1 Random Access Maschinen (RAM)

Apparat zur Notation von Algorithmen; ist idealisierte, maschinennahe Programmiersprache mit zugehöriger Hardware.

Definition 3.1
RAM orientiert an Verarbeitung natürlicher Zahlen:



RAM arbeitet taktweise, in jedem Takt wird eine Programmzeile abgearbeitet.
Ein RAM-Programm ist eine numerierte, endliche Folge von RAM-Befehlen.
Funktionen: $c : N \to N$ *Content*
$c(k)$ *aktueller Inhalt von Register* k
Parameter z: Programmzeilen-Zähler, gibt aktuell abzuarbeitende Programmzeile an.

	RAM-Befehle	*Wirkung*
Laden	*cload k*	*c(0):=k, z=z+1*
	load k	*c(0):=c(k), z=z+1*
	indload k	*c(0):=c(c(k)), z=z+1*
Speichern	*store k*	*c(k):=c(0), z=z+1*
	indstore k	*c(c(k)):=c(0), z=z+1*
Operationen	*cadd k*	*c(0):=c(0)+k, z=z+1*
	add k	*c(0):=c(0)+c(k), z=z+1*
	csub k	*c(0):=max{c(0)-k,0},z=z+1*
	sub k	*c(0):= max{c(0)-c(k),0},z=z+1*
	cmult k	*c(0):=c(0)*k,z=z+1*
	mult k	*c(0):=c(0)*c(k),z=z+1*
	cdiv k	*c(0):=⌊c(0)/k⌋,z=z+1*
	div k	*c(0):= ⌊c(0)/c(k)⌋,z=z+1*
	cmod k	*c(0):=c(0) mod k,z=z+1*
	mod k	*c(0):=c(0) mod c(k),z=z+1*
	goto k	*z:=k*
	if 0 goto k	*wenn c(0)=0 goto k, sonst z:=z+1*
	stop	

Definition 3.2 *Wir sagen: Eine RAM A berechnet ein Problem* $\Pi \subseteq \Sigma^* \times \Sigma^*$ *falls für alle Eingaben x gilt: Ist x in vorgeschriebener Weise in die Register von A geschrieben, so hält das Programm in einem Zustand, in dem eine Lösung y zu x in vorgeschriebener Weise in die Register geschrieben wird.*
„Vorgeschrieben“ liegt hierbei im Ermessen des Entwerfers.

Beispiele für RAM-Implementierungen bekannter Algorithmen

Problem 1:
gegeben: natürliche Zahlen x und y
gesucht: z = x*y

Ausgangskonfiguration:

x steht in Register 1

y steht in Register 2

Endkonfiguration:

x → Register 1

y → Register 2

z → Register 3

Programm:

1. load 1
2. mult 2
3. store 3
4. stop

3.1.1 Kosten von RAM-Berechnungen

uniforme Kostenmessung (this.uniform ≠ Berechnungsmodellen.uniform)

Definition 3.3 (Zeitmessung)
Jeder RAM-Befehl kostet eine Zeiteinheit, jedes belegte Register kostet eine Speichereinheit.
Sei A RAM, x Eingabe von A, dann gilt

$\mathbf{RAM - time^u_A(x)}$*:= Anzahl der Rechenschritte von A auf x*

$\mathbf{RAM - space^u_A(x)}$*:= Anzahl der benutzten Register*

Im Beispiel: $RAM - time^u_A(x,y) = RAM - space^u_A(x,y) = 4$
$\Rightarrow RAM - time^u_A(n) = RAM - space^u_A(n) = 4$
(Kritisch: Verarbeitung großer Zahlen (≡ langer Eingaben) kostet genausoviel Zeit wie kurze Eingaben.)

logarithmische Kostenmessung

Definition 3.4 (Zeitmessung)
Die Ausführung jedes RAM-Befehls kostet die Summe der Eingabelängen der am Befehl beteiligten Operanden (= Registeradressen und Registerinhalte).

Beispiel: Wende obigen Algorithmus auf zwei n-Bit Zahlen an:
($\Rightarrow |x| = |y| = n \quad x \in N$ ist n-Bit Zahl ≡ Binärdarstellung von x hat n Bits genau dann wenn $x \equiv (x_{n-1}, ..., x_0)$ mit $x = \sum_{i=0}^{n-1} x_i 2^i$ und $x_{n-1} = 1 \Leftrightarrow 2^{n-1} \leq x \leq 2^n - 1$)

Definition 3.5
$\underline{|x| = \lfloor log_2 x \rfloor + 1}$

Beispiel:

	x y
1. load 1	$\|x\| + 1$
2. mult 2	$\|x\| + \|y\| + 2$
3. store 3	$\|x\| + \|y\| - 1 + 2$
4. stop	1
	$O(\|x\| + \|y\|)$

$\Rightarrow RAM - time^{log}_A(n) = O(n)$

Speicherplatzmessung

Definition 3.6 (Speicherplatzbedarf pro Register)
Länge der längsten in dieses Register geschriebenen Zahl

Definition 3.7 (Gesamtspeicherplatzbedarf)
Summe der Kosten aller Register (nicht benutze Register verursachen Kosten 0)

Beispiel:
$RAM - space_A^{log}(x,y) = |x| + |y| + 2|x| + 2|y| - 2 = 3(|x| + |y|) - 2$
Register 1: $|x|$
Register 2: $|y|$
Register 3: $|x| + |y| - 1$
Register 4: $|x| + |y| - 1$
$\Rightarrow RAM - space_A^{log}(n) = \Theta(n)$

Problem 2:

Eingabe: $x \in N$

Ausgabe: $|x|, x_0, x_1, ..., x_{|x|-1} \in \{0,1\}$ mit $x = \sum_{i=0}^{|x|-1} 2^i x_i$

Algorithmus (Pseudocode):

```
output x mod 2;
x := x div 2;
while x > 0
    output x mod 2;
     x := x div 2
```

Beispiel:

x=37	1	
x=18	0	
x=9	1	
x=4	0	$\Rightarrow 37 \equiv 100101$
x=2	0	
x=1	1	
x=0		

RAM-Implementierung

Anfangskonfiguration: $x \rightarrow$ Register 1
Endkonfiguration:
$|x| \rightarrow$ Register 2
$x_0 \rightarrow$ Register 3
$x_1 \rightarrow$ Register 4
...
$x_{|x|-1} \rightarrow$ Register $|x| + 2$

Programm:

1. cload 3
2. store 2
3. load 1
4. cmod 2
5. indstore 2
6. load 1
7. cdiv2
8. if 0 goto 19
9. store 1
10. load 2
11. cadd 1
12. store 2
13. load 1
14. cmod 2
15. indstore 2
16. load 1
17. cdiv 2
18. goto 8
19. load 2
20. csub 2
21. store 2
22. stop

Kostenbedarf

Speicherplatz: $RAM - space_A^{log}(x) = 2 * x + log_2|x| + 1$
$RAM - space(n) = O(n)$
$RAM - time_A^{log}(x) = O(n + (n-1) + ... + 1) = O(n^2)$
In jedem Schleifendurchlauf gilt: Zeit $\in O(|x|)$

Problem 3:

gegeben: $x, y, z \in N$

gesucht: $x^y \ mod \ z$ (Grundoperation von RSA)

Pseudocode:

```
u=1;
while y > 0
    u=u*x mod z;
    y=y-1;
```

Kosten $C(n) \equiv 2^{n-1}$ (Durchläufe der while-Schleife für n-Bit Zahlen x, y, z)
(bei RSA: n=2048)

Ein effizienter Algorithmus: Square and multiply

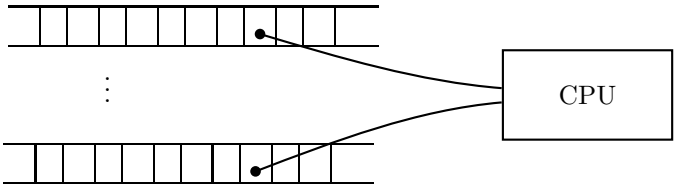
$y = (y_{n-1}, y_{n-2}, ..., y_0) \qquad x^{2^{i+1}} = (x^{2^i})^2$

$x^y \ mod \ z = x^{(\sum_{i=0}^{n-1} y_i 2^i)} \ mod \ z = \prod_{i, y_i=1} x^{2^i} mod \ z$

```
u:=1; v= x mod z
while y > 0
    if (y mod 2) = 1 then u:=u*v mod z
    v:= v^2 mod z
    y:= y div 2
```

3.2 Turing-Maschinen (TM)

k-Band TM über Σ; Σ = Berechnungsalphabet (Voraussetzung: Σ enthält ein Leerzeichen (#), $k \geq 1 \in N$
typischer Fall: $\Sigma = \{0, 1, \#\}$



Eigenschaften und Funktionsweise

- in jedem Feld kann ein Zeichen aus Σ stehen
- auf Bändern agieren Lese-/Schreibköpfe (hier: ein Kopf pro Band)
- Köpfe können in einem Arbeitstakt Zeichen aus Σ schreiben und ein Feld nach links oder nach rechts gehen
- Köpfe werden von einer CPU gesteuert, die ein Turing-Maschinen-Programm enthält (also eine endliche Folge von Befehlen der Art:
  if $(s_1, ...s_k)$ and q then $(s'_1, ..., s'_k)(M_1, ..., M_k)$ and q' $s_i, s'_i \in \Sigma$
  $M_i \in$ MOVE=$\{L, R, N\}$ (Left, Right, Not) $q \in Q \leftarrow$ endliche Zustandsmenge
  Wenn die Maschine TM in Zustand q und die Köpfe ein Zeichen s_1 bzw. $s_2, ..., s_k$ lesen, so schreibe s'_1 bzw. ... bzw. s'_k, gehe für alle $i = 1, ..., k$ entsprechend M_i und die TM geht in Zustand q'.
  Alternativ: q : if $(s_1, ..., s_k)$ then $(s'_1, ..., s'_k)(M_1, ..., M_k)$ goto q'

Formale Definition einer k-Band Turing-Maschine

$M = \{Q, q_o, \delta\}$ (M={Zustandsmenge, wohldefinierter Anfangszustand, Zustandsübergangsfunktion})
$\delta : Q \times \Sigma^k \rightarrow \Sigma^k \times MOVE^k \times Q$
$(q, (s_1, ..., s_k)) \rightarrow (s'_1, ..., s'_k), (M_1, ..., M_k), q')$

Beispiel

TM, die eine gegebene Binärzahl um 1 erhöht
Anfangskonfiguration (1-Band TM):
$\#\#...\#b_{n-1}, ..., b_0 \overset{\downarrow}{\#}\#...\#$ für $b_{n-1}, ..., b_0 \in \{0, 1\}$, $x \in N$
($\overset{\downarrow}{\#}$ kennzeichnet hierbei die initiale Position des Lese-/Schreibkopfs)
Zielkonfiguration:
$\#\#...\#c_n, c_{n-1}, ..., c_0 \overset{\downarrow}{\#}...\#$ $x + 1 \in N$

zugehöriges TM-Programm

	0	1	#
q_0			(#,L,q_1)
q_1	(1,R,q_2)	(0,L,q_1)	(1,R,q_2)
q_2	(0,R,q_2)	(1,R,q_2)	(#,N,q_3)
q_3	(0,N,q_3)	(1,N,q_3)	(#,N,q_3)

Beispiel

3-Band TM (addiert 2 Binärzahlen)
Anfangskonfiguration ($b, c \in \{0,1\}^*$):
$\#b_{n-1}, ..., \overset{\downarrow}{b_0}\#$
$\#c_{n-1}, ..., \overset{\downarrow}{c_0}\#$
$\#\#, ..., \overset{\downarrow}{\#}\#$

Endkonfiguration:
$\#b_{n-1}, ..., b_0\#$
$\#c_{n-1}, ..., c_0\#$
$\#d_{s-n}, ..., d_0\#$

$\sum_{i=0}^{n-1} b_i 2^i + (\sum_{j=0}^{n-1} c_j 2^j) = \sum_{k=0}^{s-1} d_k 2^k$

Annahme: TM schreibt nur auf Band 3

Band 1	#	0	1	#	#	0	0	1	1
Band 2	#	#	#	0	1	0	1	0	1
Band 3	#	#	#	#	#	#	#	#	#
q_0	q_2	(0,LLL,q_0)	(1,LLL,q_0)	(0,LLL,q_0)	(1,LLL,q_0)	(0,LLL,q_0)	(1,LLL,q_0)	(1,LLL,q_0)	(0,LLL,q_1)
q_1	(1,q_2)	(1,LLL,q_0)	(0,LLL,q_1)	(1,LLL,q_0)	(0,LLL,q_1)	(1,LLL,q_0)	(0,LLL,q_1)	(0,LLL,q_1)	(1,LLL,q_1)
q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2	q_2

Beispiel (Akzeptor)

2-Band TM über {0,1,#}
Anfangskonfiguration:
$\#b_{n-1}, ..., \overset{\downarrow}{b_0}\#$
$\#c_{m-1}, ..., \overset{\downarrow}{c_0}\#$

akzeptiere falls $(b_{n-1}, ..., b_0) = (c_{m-1}, ..., c_0)$, verwerfe sonst

Band 1	#	0	1	#	#	0	0	1	1
Band 2	#	#	#	0	1	0	1	0	1
q_0	q_{acc}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	LL	q_{rej}	q_{rej}	LL
q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}	q_{acc}
q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}	q_{rej}

Ab jetzt werden TM-Algorithmen nur noch informal spezifiziert.

Beispiel 4

3-Band TM über {0,1,#}

- Anfangsbedingung Band 1: $b \in \{0,1\}^*$
- TM akzeptiert b $\Longleftrightarrow$ b enthält genau soviele Nullen wie Einsen
- TM zählt auf Band 2 die Einsen
- TM zählt auf Band 3 die Nullen
- TM akzeptiert wenn nach Einlesen der Eingabe Inschriften auf Band 2 und 3 gleich

Beobachtung 3.1 *TM können nichtreguläre Sprachen berechnen.*

3.2.1 Berechnungen durch TM

wichtiger Begriff: Konfiguration einer TM: Gesamtheit aller Informationen, die zu einem gegebenen Zeitpunkt relevant für das weitere Verhalten der TM ist.
konkret: gegeben: k-Band TM M über Σ
Konfiguration:

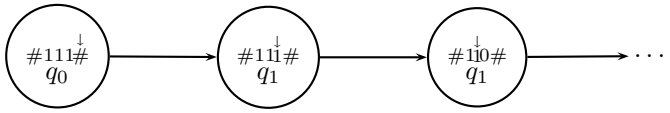
- aktueller Zustand
- aktuelle Beschriftung der Bänder
- aktuelle Kopfpostitionen

Beobachtung 3.2
Zu jeder Konfiguration existiert eine eindeutig bestimmte Nachfolgekonfiguration ($\Rightarrow$ die des nächsten Taktes).
$\Rightarrow$ *Jeder Konfiguration K von M entspricht eine eindeutig bestimmte Folge* $K \rightarrow K_1 \rightarrow ...$ *von Konfigurationen mit* K_1 *Nachfolgekonfiguration von K;* K_i *Nachfolgekonfiguration von* $K_{i-1} \quad \forall i > 1$.
Die Folge heißt Berechnung von M auf K.
Es gibt zwei Möglichkeiten:

1. *Berechnung „hält an“(„terminiert“) falls* $\exists i \quad K_i = K_{i+1} = ...$ *dann heißt* K_i *Stopp-Konfiguration*
2. *Berechnung hält nicht an* $(\forall i \geq 1 \quad K_i \neq K_{i+1})$

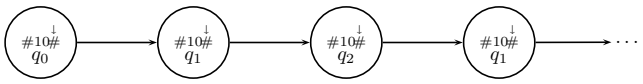
Beispiele

1. (vgl. Beispiel 1:)



2.

	0	1	#
q_0			q_1
q_1	q_2	q_2	q_2
q_2	q_1	q_1	q_1



$\Rightarrow$ hält nicht an, wechselt immer von q_1 nach q_2 und zurück

Startend auf $k_0 = \underbrace{\#11\overset{\downarrow}{\#}}_{q_0}$ hält M nach $time_M(k_0) = 7$ in der Stoppkonfig $\underbrace{110\overset{\downarrow}{\#}}_{q_3}$

Definition 3.8

1. *Berechnung einer TM auf einer gegebenen Konfiguration k ist eine eindeutig bestimmte Folge $k_0 \to k_1 \to k_2 \to ...$ mit*
   (a) $k_0 = k$
   (b) $\forall\, i = 0, 1, ...$ *ist* k_{i+1} *die Nachfolgekonfiguration von* k_i *bezüglich M*
2. *M hält auf k* $\leftrightarrow \exists\ s \geq 0$ *mit* $k_s = k_{s+1} = k_{s+2} = ...$; k_s *heißt Stopp-Konfiguration zu k*
3. *M hält auf* $K \Rightarrow time_M(k) = s =$ *Anzahl der Takte bis zum Erreichen einer Stopp-Konfiguration*
   M hält nicht auf $K \Rightarrow time_M(k) = \infty$

Konvention

Gegeben: Σ (enthält #)
Für jede Eingabe $x \in \Sigma^*$ für eine TM M über Σ bestimmen wir eine eindeutig bestimmte Startkonfiguration $k_0(x)$:
$\underbrace{\overset{\downarrow}{\#}\, x_1, ..., x_n \#}_{q_0}$ Band 1, alle anderen Bänder sind # in allen Bandzellen.
(wenn nicht explizit anders vereinbart)

Konvention

Für alle $y \in \Sigma^*$ definieren wir den Begriff „Stoppkonfiguration“zu y:
Wenn M k-Bank TM, dann steht y auf Band k: $\#y_1, ..., y_n \overset{\downarrow}{\#}$

- Band 1 darf Eingabe enthalten (kein „Muss“), alle anderen Bänder sind leer
- M befindet sich in Stoppkonfiguration (d.h. in einem Stoppzustand q)

Bemerkung: $k_0(\epsilon) \equiv$ Band 1 ist leer, d.h. enthält nur #

Definition 3.9
Gegeben ist ein Berechnungsproblem $\Pi \subseteq \Sigma^* \times \Sigma^*$
Eine TM M berechnet Π*, falls M für alle zulässigen Eingaben* $x \in \Sigma^*$ *auf* $k_0(x)$ *in einer Stoppkonfiguration* $k(y)$ *hält und* $(x, y) \in \Pi$ *(d.h. y Lösung zu x).*

Vereinbarungen

Wir schreiben $time_M(x)$ für $time_M(k_0(x))$ und $time_M(n)$ für $max_{x \in \Sigma^*, |x|=n}(time_M(x))$ worst-case Laufzeit.
$time_M : N \to N$ **wichtigster Kostenparameter**

Definition 3.10
Wir nennen eine Turing-Maschine TM über Σ *t(n)-zeitbeschränkt falls* $time_M(n) \leq t(n)$ *für alle* $n \in N$ *(genauer:* $time_M(x) \leq t(|x|) \quad \forall\, x \in \Sigma^*$ *zulässig).*
Wir nennen eine TM M polynomiell zeitbeschränkt falls $\exists$ *Konstanten* C, k *mit M ist* $C * n^k$ *zeitbeschränkt.*

Frage: Wie aussagekräftig ist das Modell der TM?
Angestrebte Aussage: Wenn Berechnungsmodell Π in irgendeinem vernünftigem Modell einen Polynomialzeitalgorithmus hat, dann auch im TM-Modell.

3.3 Gegenseitige Simulation von RAM- und TM-Modellen

Allgemeine Vorbemerkungen

Gegeben: Eine zu simulierende TM (bzw. RAM) über der Zustandsmenge (bzw. Menge der Programmzeilennummern) Q.
Wir beschreiben die Simulation durch die simulierende TM (bzw. RAM) M' folgendermaßen:

1. Zu jeder Konfiguration k für M existiert eine entsprechende Konfiguration $\Phi(k)$ für M'
2. Definiere M' durch Angabe von Unterprozeduren M'_0 und $M'(q) \quad q \in Q$ in folgender Weise:
   - M'_0 erzeugt für alle zulässigen Eingaben x für M die Konfiguration $\Phi(k_0(x))$
   - $M'(q)$ hat folgende Eigenschaft: Ist k eine Konfiguration für M im Zustand q und k' ihre Nachfolgekonfiguration, so überführt $M'(q)$ die Konfiguration $\Phi(k)$ in $\Phi(k')$

Veranschaulichung:

		$k_0(x)$	$\rightarrow$	k_1	$\rightarrow$	k_2	$\rightarrow$	k_3	$\rightarrow$	...
		q_0		q_{i_1}		q_{i_2}		q_{i_3}		...
	Simulation	$\downarrow$		$\downarrow$		$\downarrow$		$\downarrow$		
$k_0'(x)$	$\overset{M_0'}{\rightarrow}$	$\Phi(k_0(x))$	$\overset{M(q_0)}{\rightarrow}$	$\Phi(k_1)$	$\overset{M_(q_{i_1})}{\rightarrow}$	...	$\rightarrow$	...	$\rightarrow$	$\Phi(k_2)$

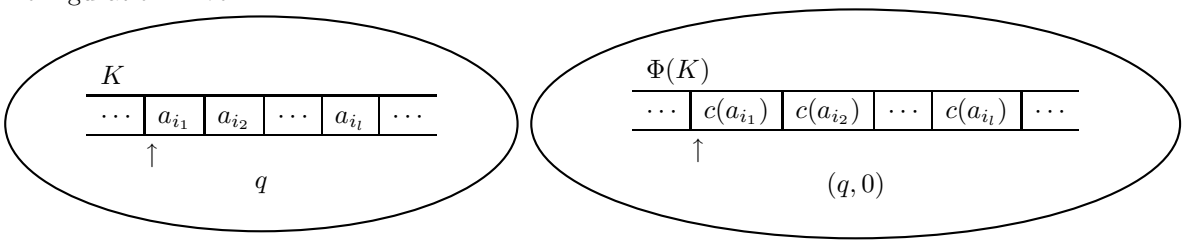
Berechnung von M auf x

Wobei $k_0'(x)$ die Startkonfiguration von M' auf x ist.

Satz 3.1
Jede $t(n)$-zeitbeschränkte TM M über einem beliebig fixem Alphabet Σ lässt sich durch eine $O(t(n))$-zeitbeschränkte TM M' über $\{0,1\}$ mit gleicher Bandanzahl simulieren.

Beweis

Simulation basiert auf einer Kodierungsfunktion c: $\Sigma \rightarrow \{0,1\}^s$ für passend gewähltes $s \quad (s \geq log_2|\Sigma|)$.
Wir betrachten den Fall M 1-Band TM (allgemeiner Fall analog):
Konfiguration K von M:



Erläuterung: Für alle $q \in Q$ sei $M'(q)$ über Zuständen (q,0), (q,1), (q,2) usw. definiert.

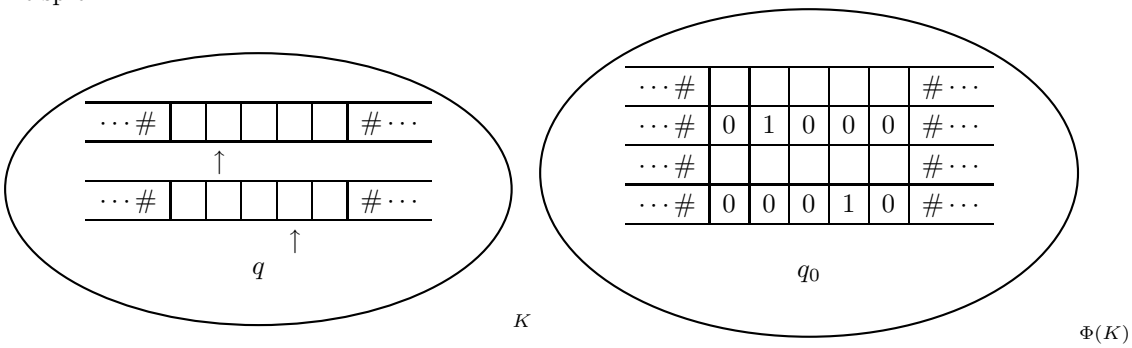
Informale Beschreibung von M(q)

- $M'(q)$ liest den aktuellen Block von links nach rechts und stellt fest, dass dies die Kodierung von $a \in \Sigma$ ist.
  Es sei $\delta(q,a) = (a', M, q') \quad M \in MOVE,\ a' \in \Sigma,\ q' \in Q$
- $M'(q)$ überschreibt aktuellen Block von rechts nach links mit $c(a')$
- $M'(q)$ geht an Startposition des nächsten aktuellen Blocks gemäß M
- $M'(q)$ begibt sich in Zustand $(q',0)$, den Startzustand von $M'(q')$
  $time_{M'(q)}$= O(s) = O(1)

Satz 3.2
Für alle $k > 1$ gilt: Jede $t(n)$-zeitbeschränkte k-Band TM M kann durch $O(t(n)^2)$, zeitbeschränkte 1-Band TM M' simuliert werden.

Beweis

Falls Σ das Bandalphabet von M dann sei $(\Sigma \times \{0,1\})^k$ das Bandalphabet von M'.
Idee: Wir teilen das Band von M' in $2 * k$ Spuren auf.
Beispiel: k=2



Beschreibung von $\Phi(k)$

- Die M' Spuren 1,3,5 enthalten den Inhalt der Bändern 1,2,.. von M; die M'-Spuren 2,4,6,... enthalten die Kopfpositionen wie oben beschrieben.
- Der M'-Kopf steht auf der linken Begrenzung des beschreibenen Teils der Bänder

Satz 3.3 *Jede $t(n)$-zeit- und $s(n)$-platzbeschränkte RAM A kann durch $O(t(n) - s^2(n))$ zeitbeschränkte 4-TM M simuliert werden (logarithmisches Kostenmaß).*

Beweis

Fixiere RAM A, Q Menge aller Programmzeilennummern, $q_0 \in Q$ Nummer des Startbefehls von A
Konfiguration von RAM: $q, (0, c(0))(i_1, c(i_1)), ..., (i_r, c(i_r))$, wobei q den aktuellen Befehl, i die Adressen der nichtleeren Register und $c(i)$ den Inhalt von Register i bezeichnen.

Platzbedarf einer Konfiguration k

$1 + |bin(c(0))| + \sum_{t=1}^{r}(|bin(i_t)| + |bin(c(i_t))|)$
($bin(c(0))$ bezeichnet die Darstellung von $c(0)$ in binärer Form.)
Einer Konfiguration k von A mit q entspricht Konfiguration $\Phi(k)$ von M.
Zustand von $\Phi(k)$: $(q, 0)$ Startzustand von $M(q)$
Band 1: $...\#\#0\#bin(c(0))\#\#bin(i_1)\#bin(c(i_1))\#\#...\#\#bin(i_r)\#bin(c(i_r))\#\#\#...$
Bänder 2-4: sind leer (d.h. nur mit # beschrieben)
Entwurf von $M(q)$: transformiert $\Phi(k)$ in $\Phi(k')$ wobei k' Nachfolgekonfiguration von k ist. Diese entsteht durch die Abarbeitung des Befehls mit Zeilennummer q.
Wir müssen eine Fallunterscheidung bezüglich aller in Frage kommenden RAM-Befehle durchführen:
Exemplarisch: q *ind add i* ist aktuelle Programmzeile
M(q):

- kopiere $bin(c(0))$ auf Band 2
- suche auf Band 1 die Zeichenkette $\#\#bin(i)\#bin(c(i))\#\#$
- kopiere $bin(c(i))$ auf Band 4
- suche $\#bin(c(i))\#bin(c(c(i)))\#\#$ auf Band 1
- kopiere $bin(c(c(i)))$ auf Band 3
- addiere Binärzahlen von Band 2 und Band 3, überschreibe Band 4 mit dem Ergebnis
- kopiere alles rechts von $bin(c(0))$ auf Band 2
- überschreibe $bin(c(0))$ mit Binärzahl auf Band 4
- kopiere hinter diese Binärzahl den Inhalt von Band 2
- lösche Bänder 2,3 und 4
- gehe in Zustand $(q+1, 0)$

Zeitbedarf von $M(q)$: $O(s(k)+$ Kosten der Addition), wobei $s(k)$ der Platzbedarf von k ist. Die Operanden der Addition haben eine Bitlänge $\leq s(k)$, der Zeitbedarf ist damit $O(s(k))$.
$\Rightarrow O(s(k))$
Die Konstruktion $M(q)$ für andere RAM-Befehle läuft ähnlich.
Achtung: mult-, div- und mod-Befehle sind teuerer und verursachen einen Zeitbedarf von $O(s^2(k))$ falls die „Schulmethode"implementiert würde. $\Rightarrow$ Gesamtkosten von M : $\quad time_M(n) \leq t(n) * max_{q \in Q}(time_{M(q)}(s(n))) \in O(t(n)s^2(n))$

Satz 3.4
Jede $t(n)$-zeitbeschränkte TM kann durch $O(t(n)log\ t(n))$ logarithmisch zeitbeschränkte RAM s simuliert werden.

Beobachtung 3.3
$\underbrace{1-TM, k-TM}_{verschiedene Alphabete}$ *, RAM*
In all diesen Modellen ist die Menge der berechenbaren Probleme und die Menge der mit polynomiellem Zeitbedarf berechenbaren Probleme gleich.

Curch'sche These

Die Menge der berechenbaren Probleme ist in allen intuitiv vernünftigen Berechnungsmodellen gleich.

erweiterte Church'sche These

Das Gleiche gilt für die Menge der mit polynomiellem Zeitbedarf berechenbaren Probleme.
$\Rightarrow$ Definition grundlegender Komplexitätsklassen:
$PTIME = \{\Pi; \Pi$ Berechnungsproblem mit Polynomialzeitalgorithmus$\}$
$\mathcal{REK} = \{\Pi; \Pi$ Berechnungsproblem das berechenbar ist$\}$
$PTIME \subseteq \mathcal{REK}$
Wir betrachten gesondert die Berechenbarkeit von Sprachen:
$L \subseteq \Sigma^*$ bzw. $\chi_L : \Sigma^* \to \{0,1\}$
$REK = \{L \subseteq \Sigma^*; L$ berechenbar$\}$
$P = \{L \subseteq \Sigma^*; L$ ist in Polynomialzeit berechenbar $\}$

Beobachtung 3.4

1. $REG \subset P \subseteq REK$
2. $\underbrace{L \in REK}_{\in\{0,1\}} \Longleftrightarrow \exists\, 1-$*Band TM über* $\{0,1,\#\}$*, die* L *berechnet*
3. $L \in P \Longleftrightarrow \exists$ *polynomiell zeitbeschränkte 1-Band TM über* $\{0,1,\#\}$*, die* L *berechnet*

3.4 Universelle TM

Wir entwerfen eine programmierbare TM, die eine beliebige 1-Band TM über $\{0,1,\#\}$ in Echtzeit simulieren kann. Wir kodieren 1-Band TMs M durch Gödel-Nummern.
OBdA $M = \{Q, q_1, \delta\}$
$\delta : Q \times \{0,1,\#\} \to \{0,1,\#\} \times MOVE \times Q; \quad Q = \{q_1, \overset{\downarrow}{q_2}, ..., q_s\}$
$\delta(q_2, a) = (a, N, q_2) \quad \forall a \in \{0,1,\#\}$
Ordnen M eine fixierte Kodierung $< M >$ (genannt Gödel-Nummer) der Funktion δ zu:
Kodierung:

'0'	0	N	0
'1'	$00 = 0^2$	L	00
'#'	$000 = 0^3$	R	000
$\underbrace{q_i}_{1\le i\le s}$	$\underbrace{00...0}_{i} = 0^i$		

Kodierung einer Instanz $\delta(q_i, a) = (a', \underbrace{m}_{MOVE}, q_j)$: $\quad 0^i 10^a 10^b 10^c 10^j = code(I)$

Beispiel

$\delta(q_3, 0) = (\#, L, q_2) \qquad code(I) = (000101000100100)$
$< M >= 111 code(I_1) 11 code(I_2) 11 code(I_3) 11...11 code(I_t) 111$
$I_1, ..., I_t$ sind Instanzen von δ, lexikografisch nach den ersten beiden Komponenten geordnet.

$I_1 = (q_1, 0, ...)(q_1, 1, ...)(q_1, \#, ...)(q_2, 0, ...)...$
$< M >$ entspricht einer eindeutig bestimmten Kodierung als Binärzahl.

Die universelle TM U

- U ist 3-TM über $\{0, 1, \#\}$
- zulässige Eingaben $\underbrace{x}_{\in\{0,1\}^*} \# < M >$, wobei $< M >$ eine syntaktisch korrekte Gödelnummer für eine 1-Band TM M über $\{0, 1, \#\}$ ist.

Arbeitsweise: U simuliert taktweise die Arbeit von M auf x
Phase 0:

- kopiere $< M >$ auf Band 2 und lösche $< M >$ auf Band 1
- schreibe 0^1 auf Band 3

Arbeitsphase (Simulation eines Taktes):

- wenn M in diesem Takt in Zustand q_i, steht auf Band 3 0^i
- die aktuelle Bandinschrift von M steht auf Band 1 von U

U:

1. Falls $i = 2$ stoppe
2. Lese aktuelles Zeichen $a \equiv 0^a$ auf Band 1 und suche Instanz $0^i 10^a ...$ auf Band 2
3. Falls $I = 110^i 10^a 10^b 10^c 10^j$ verändere Band 1 entsprechend 0^b und 0^c und schreibe 0^j auf Band 3

Beobachtung 3.5
$time_M(x, \underbrace{< M >}_{Konstante}) = O(time_M(x))$

Kapitel 4

Berechenbarkeit

Wir betrachten hier ausschließlich 1-Band TM über dem Alphabet $\{0, 1, \#\}$ die Sprachen berechnen, d.h. die eindeutig bestimmte Stopp-Zustände q_2=accept und q_3=reject haben.
Grund: Alle Betrachtungen funktionieren auch für kompliziertere Modelle und Probleme mit mehreren Output-Bits, die Verwendung wäre jedoch umständlicher ohne wesentlichen Erkenntnisgewinn.
genauer: Wir betrachten TMs als geordnete Folge $M_1, M_2, ..., M_i, ...$, wobei M_i die Maschine mit der i-t größten Gödelnummer bezeichnet.
Frage:
Sind alle Sprachen $L \subseteq \{0,1\}^*$ berechenbar?
(d.h. gilt die Aussage: $\forall\, L \subseteq \{0,1\}^* \exists\, i \in N^+$ mit M_i berechnet L?)
Antwort:
Nein. Falls die Aussage richtig wäre könnte man die Menge aller $L \subseteq \{0,1\}^*$ durchnummerieren mit $min(L) := min\{i, M_i \text{ berechnet L }\} \Rightarrow \{L \subseteq \{0,1\}^*\} = \mathcal{P}(\{0,1\}^*)$ abzählbar, jedoch ist $\mathcal{P}(\{0,1\}^*)$ gleichmächtig mit $R(\rightarrow$ überabzählbar)
Da $\mathcal{P}(\{0,1\}^*)$ überabzählbar $\Rightarrow$ „fast alle “Sprachen sind nicht berechenbar, die Menge der berechenbaren Sprachen ist abzählbare Teilmenge in $\mathcal{P}(\{0,1\}^*)$
REK=$\{L \subseteq \{0,1\}^*; \exists$ TM M mit M berechnet $L\}$ (d.h. M hält auf allen $x \in \{0,1\}^*$ und $M(x) = 1 \Longleftrightarrow x \in L; \qquad M(x) = 0 \Longleftrightarrow x \notin L$)
Wir wissen: REG $\subset$ P $\subseteq$ REK $\subset \mathcal{P}(\{0,1\}^*)$
Frage: Ist es möglich nichtberechenbare Sprachen explizit zu definieren?
Antwort: Ja.
Vorbetrachtung:
Definieren Ordnung auf $\{0,1\}^* = \{w_1, w_2, ...\}$ mit $w \prec w'$ genau dann wenn $|w| < |w'|$ oder falls $|w| = |w'|$ ist w lexikografisch kleiner als w'.
Beispiel: $\{0,1\}^*$ geordnet nach $\prec$: $\underbrace{\epsilon}_{w_1}, \underbrace{0}_{w_2}, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, ...$

Definition 4.1 (Diagonalsprache)
Die Diagonalsprache D: $D = \{w_i; M_i(w_i) = 0\} \Rightarrow$ *alle Worte für die 0 auf der Diagonalen steht.*

$$T_{ij} = \begin{cases} 0 & M_j(w_i) = 0 \\ 1 & M_j(w_i) = 1 \\ \# & M_j \textit{ hält nicht auf } w_i \end{cases}$$

T	M_1	M_2	M_3	M_4	M_5
w_1	*0*	.	.	.	.
w_2	.	*0*	.	.	.
w_3	.	.	*0*	.	.
w_4	.	.	.	*0*	.
w_5	.	.	.	.	*0*

Satz 4.1 *$D \notin REK$*

Beweis:
Wir nehmen an $\exists$ TM $M = M_j$ für D. Wir betrachten das Verhalten von M_j auf w_j, genauer:
Ist $w_j \in D$? $w_j \in D \overset{Def.M}{\Longleftrightarrow} M_j(w_j) = 1$ und $w_j \in D \overset{Def.D}{\Longleftrightarrow} M_j(w_j) = 0$
Widerspruch! $\Rightarrow$ es existiert kein $M = M_j$ für D.

Definition 4.2 (Halteproblem $H \subseteq \{0,1,\#\}^*$)
$H = \{< M > \#x; < M >$ *gültige Gödelnummer,* $x \in \{0,1\}^*$, *M hält auf* $x\}$

Satz 4.2 *$H \notin REK$*

Beweis:
Wir nehmen an $H \in$ REK und zeigen dann $D \in$ REK.
$H \in$ REK $\rightarrow \exists$ TM M_H für H; wir bauen mit M_H TM M_D für D.
Definition von M_D: Eingabe: $w_i \in \{0,1\}^*$

- berechnet $< M_i >$
- füttert M_H mit Eingabe $< M_i > \#w_i$

Fall 1: $M_H(< M_i > \#w_i) = 0$ (d.h. M_i hält nicht auf w_i), dann gibt M_0 0 aus
Fall 2: $M_H(< M_i > \#w_i) = 1$ (d.h. M_i hält auf w_i), dann gibt M_0 den Wert $\neg M_i(w_i)$ aus
genauer: Fall 2: Falls $M_H(< M_i > \#w_i) = 1$ M_D startet M_i auf w_i und gibt $\neg M_i(w_i)$ aus.
$\Rightarrow M_D$ berechnet D **Widerspruch**.

Definition 4.3 (Spezielles Halteproblem $H_\epsilon \subseteq \{0,1\}^*$)
$H_\epsilon = \{< M >; M$ *hält auf* $\epsilon\}$ *(d.h. auf leerem Band)*

Satz 4.3 *$H_\epsilon \notin REK$*

Beweis:
Nehmen an $H_\epsilon \in$ REK $\Rightarrow \exists$ TM M_ϵ für H_ϵ, wir zeigen als Folgerung $H \in$ REK.
Wir konstruieren M_H für H mit Unterprozedur M_ϵ
Vorbetrachtung: $\forall\, x \in \{0,1\}^* \exists$ TM M_x die, startend auf dem leeren Band, x auf das Band schreibt.
Arbeitsweise von M_H auf einer Eingabe $< M > \#x$

1. M_H berechnet die Gödelnummer einer M_x^*, die zunächst mittels M_x x auf das Band schreibt und dann M auf x startet.
2. starte M_ϵ auf Eingabe $< M_x^* >$

Beobachtung:

$$M_H(< M > \#x) = M_\epsilon(< M_x^* >) = \begin{cases} 1 & M_x^* \text{ hält auf leerem Band} \Leftrightarrow M \text{ hält auf } x \\ 0 & M_x^* \text{ hält nicht auf leerem Band} \Leftrightarrow M \text{ hält nicht auf } x \end{cases}$$

$\Rightarrow M_H$ berechnet H **Widerspruch**.

Definition 4.4 (Rekursiv aufzählbar)
$L \subseteq \{0,1\}^*$ heißt rekursiv aufzählbar falls eine TM M existiert, die L „aufzählt“
d.h. $M(x) = 1 \Longleftrightarrow x \in L$
(d.h. wenn $x \notin L$ ist $M(x) = 0$ <u>oder</u> M hält nicht auf x gestartet)
ENUM=$\{L \subseteq \{0,1\}^*, \exists$ TM M, die L aufzählt$\}$

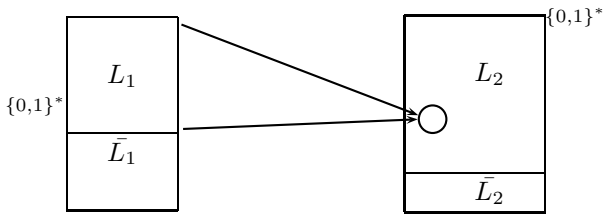
Beobachtung 4.1
$D \in$ ENUM
Definition: $\bar{M}_D$, die D aufzählt
Eingabe: $x \in \{0,1\}^*$

- *berechnet i mit $x = w_i$*
- *berechnet $< M >$ mit $M = M_i$*
- *startet M auf x*
- *gibt falls M auf x hält $\neg M(x)$ aus*

Beobachtung: $\bar{M}_D(x) = 1 \Leftrightarrow M_i(\underbrace{x}_{w_i}) = 0$

4.1 Das Reduktionsprinzip

Definition 4.5 (Reduktion)
Gegeben: $L_1, L_2 \subseteq \{0,1\}^*$ und Funktion $f : \{0,1\}^*$ die berechenbar ist.
Wir sagen f ist Reduktion von L_1 auf $L_2 \leftrightarrow \forall x \in \{0,1\}^*$ gilt:
$x \in L_1 \Longleftrightarrow f(x) \in L_2 (x \notin L_1 \leftrightarrow f(x) \notin L_2)$.



Definition 4.6
$L_1 \leq L_2 \iff \exists$ Reduktion f von L_1 auf L_2. Man kann zeigen: „$\leq$“ist eine reflexive und transitive Relation.
Intuitiv: $L_1 \leq L_2$: *L_1 ist nicht wesentlich schwieriger als L_2, d.h. aus $L_2 \in REK$ folgt $L_1 \in REK$.*

Satz 4.4 *Aus $L_1 \leq L_2$ und $L_2 \in REK$ folgt $L_1 \in REK$*

Beweis:
Wir fixieren eine TM M für f und M_2 für L_2 und konstruieren M_1 für L_1.
Eingabe: $x \in \{0,1\}^*$

- starte M_f auf $x \Rightarrow f(x)$
- starte M_2 auf $f(x); M_1(x) = M_2(f(x))$

Erklärung am Beispiel von Satz 3 ($H_\epsilon \notin$ REK):
Wir definieren eine Reduktion f von H auf H_ϵ (d.h. wir zeigen $H \leq H_\epsilon$)
Definieren f:

- f testet ob die Eingabe die Gestalt $< M > \#x$ mit $< M >$ korrekte Gödelnummer und $x \in \{0,1\}^*$ hat
- f gibt die Gödelnummer einer Maschine M' aus, die folgendes tut:
- M' startet M_x (schreibt die Eingabe x auf das leere Band)
- M' startet M auf x

$f(< M > \#x) =< M' >$
Zu zeigen: f ist eine Reduktion von H auf H_ϵ.
Fall 1: $< M > \#x \in H \Rightarrow M'$ hält auf leerem Band
Fall 2: $< M > \#x \notin H \Rightarrow M'$ hält nicht auf leerem Band

4.2 Der Satz von Rice

Erweiterte Definition der Berechenbarkeit

Wir betrachten die Berechenbarkeit von Funktionen $g : \{0,1\}^* \to \{0,1, \underbrace{*}_{don't care}\}$
partielle Funktion
Sprachen sind Funktionen $L : \{0,1\}^* \to \{0,1\}$

Definition 4.7
Wir sagen eine TM M berechnet die partielle Funktion $g : \{0,1\}^* \to \{0,1,*\}$ falls:
$\forall\, x \in \{0,1\}^*$ und $g(x) \neq *$ gilt: $M(x) = g(x)$.

Eigenschaft 4.1
Im Gegensatz zu volldefinierten Funktionen ist die berechnete partielle Funktion nicht mehr eindeutig definiert. Insbesondere: Eine gegebene TM M berechnet alle partiellen Funktionen g mit der Eigenschaft:
$g^{-1}(1) \subseteq M^{-1}(1) \qquad g^{-1}(0) \subseteq M^{-1}(0)$
Extremfall: $u : \{0,1\}^* \to \{0,1,*\} \qquad u(x) = * \quad \forall x \in \{0,1\}^*$

Beobachtung 4.2
Jede TM M berechnet u.
$R = \{g : \{0,1\}^* \to \{0,1,*\}; \exists$ *TM M, die g berechnet*$\}$

Beobachtung 4.3

- $REK \subseteq R$
- $u \in R$
- $L : \{0,1\}^* \to \{0,1\} \notin R \to L \notin R$

Für $S \subseteq R$ definieren wir $L_S = \{< M >, M$ berechnet $g \in S\}$

Satz 4.5 (Satz von Rice)
Für alle Teilmengen $\emptyset \subset S \subset R$ mit $u \notin S$ gilt: L_S ist nicht berechenbar.

Beispiel

für L_S: $S = \{PRIMES\}; PRIMES(x) = 1 \Leftrightarrow x$ ist eine Primzahl in Binärdarstellung $\Rightarrow L_S$ nicht berechenbar; d.h. es ist nicht berechenbar ob ein gegebener Algorithmus korrekt das Primzahlenproblem oder ein beliebiges anderes Problem berechnet
$\Rightarrow$ Verifikation der Korrektheit von Programmen ist nicht berechenbar.

Beweis

Wir zeigen $H_\epsilon \leq L_S$
Wir konstruieren eine Reduktion f von H_ϵ auf L_S.
Wir definieren $f(x)$ für eine Eingabe $x \in \{0,1\}^*$
f testet ob $x =< M >$ für eine TM M.
Die Ausgabe von f ist eine TM $< M' >$ die auf der Eingabe $x' \in \{0,1\}^*$ folgendes tut:

- M' startet M auf dem leeren Band
- falls M anhält so startet M' M_g auf x

Unterprozeduren

Wir wissen:

- $u \notin S$
- $S \neq 0$, d.h. $\exists\, g \in S, g \neq u$

Wir fixieren eine TM M_g, die g berechnet.
Behauptung: $f(< M >) =< M' >$ ist Reduktion von H_ϵ auf L_S
Fall 1: $< M > \notin H_\epsilon \Rightarrow M'$ hält auf keiner Eingabe $x' \in \{0,1\}^* \Rightarrow$ Die einzige partielle Funktion, die von M' berechnet wird ist $u, u \notin S \Rightarrow (< M' > \notin L_S)$.
Fall 2: $< M > \in H_\epsilon \Rightarrow M'$ hat gleiches Ausgabeverhalten wie $M_g \Rightarrow M'$ berechnet $g, g \in S$

4.3 Post'sches Korrespondenzproblem (PKP)

Eingabe: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(s)}, y^{(s)}) \quad s \geq 1$ beliebig; $x^{(i)} \in \{0,1\}^*, y^{(i)} \in \{0,1\}^* \quad \forall i = 1, ..., s$
Es gilt: $I \in \text{PKP} \Leftrightarrow \exists i_1, i_2, ..., i_t \quad (t \geq 1 \text{ beliebig})$ mit $x^{(i_1)}, x^{(i_2)}, ..., x^{(i_t)} = y^{(i_1)}, y^{(i_2)}, ..., y^{(i_t)}$
Beispiel:

1. $I = ((101, 10), (0, 10), (11, 000), (00, 110)) \quad x^{(1)}x^{(2)} = y^{(1)}y^{(2)} \quad I \in \text{PKP}$
2. $I = ((10, 01), (11, 001), (101, 011)) \quad I \notin \text{PKP}$, da alle $x^{(i)}$, jedoch kein $y^{(i)}$ mit 1 anfängt

Satz 4.6 *PKP $\notin$ REK*

Definition 4.8 (Modifiziertes PKP (MPKP))
MPKP, Eingabe I wie bei PKP: $I \in MPKP \Leftrightarrow \exists 1, i_2, i_3, ..., i_t \quad t \geq 1$ beliebig $x^{(i_1)}, x^{(i_2)}, ..., x^{(i_t)} = y^{(i_1)}, y^{(i_2)}, ..., y^{(i_t)}$

Hilfssatz 4.1 *MPKP $\leq$ PKP*

Beweis:
Gegeben: $I = (x_1y_1)...(x_ky_k)$
Wir definieren f mit $f(I) \in \text{PKP} \Longleftrightarrow I \in \text{MPKP}$
f ist Reduktion, $f(I)$ ist Instanz über $\Sigma \cup \{\$, \&\}$
$f(I) = (x'_0y'_0), (x'_1y'_1), ..., (x'_{k+1}y'_{k+1}) \quad i = 1..k$
$x'_i \leftarrow$ schreibe vor jeden Buchstaben von x_i ein &
$x_i = 10110 \rightarrow x'_i = \&1\&0\&1\&1\&0$
$y'_i \leftarrow$ schreibe hinter jeden Buchstaben von y_i ein &
$y_i = 110 \rightarrow y'_i = 1\&1\&0\&$
$x'_0 = x'_1 \qquad x'_{k+1} = \&\$$
$y'_0 = \&y'_1 \qquad y'_{k+1} = \$$
Beobachtung: $\forall x'_i$ beginnen mit „&“, nur y'_0 beginnt mit „&“
$\Rightarrow$ jede Lösung von $f(I)$ beginnt mit x'_0 bzw. y'_0 und enthält danach nur Worte mit x'_i bzw. y'_i mit $i > 0$, da sonst y'_iy_0 zwei „&“hintereinander entstünden, die durch

x'_i nicht generierbar sind.
$i_1, ..., i_s$ ist Lösung zu $f(I) \Longleftrightarrow i_1 = 0, i_2, ..., i_{s-1} \in \{1, ..., k\}, i_s = k+1$
$x'_{i_1}, x'_{i_2}, ..., x'_{i_s} = y'_{i_1}, y'_{i_2}, ..., y'_{i_s}$ und $x_{i_2}, ..., x_{i_k} = y_{i_2}, ..., y_{i_k}$
$\Rightarrow I \in$ MPKP $\Longleftrightarrow f(I) \in$ PKP

Hilfssatz 4.2 *$H \leq MPKP$*

Beweis:
Gegeben $M\#w$, wir ordnen mittels Reduktion $f : I = f(M, w)$ mit $I \in$ MPKP
$\Longleftrightarrow M\#w \in H$ zu.
Eigenschaften von M: M arbeitet über $\{0, 1, \#\}$ und ist eine 1-TM über Q
$Q' \subseteq Q$ bezeichnet die Menge der Stoppzustände. Wir betrachten die Zustandsüberführungsfunktion δ als Menge von Instanzen (q, a, c, M, q') mit:

- $q \in Q \setminus Q'$
- $M \in MOVE$
- $q' \in Q'$
- $a, c \in \{0, 1, \#\}$

Wir kodieren die Konfigurationen von M als $w'_1 w'_2 ... w'_{i-1} q_j w'_i w'_{i+1} ... w'_m$
also:

- akueller Zustand ist q_j
- aktuell bearbeiteter Bereich des Bandes ist $w'_1 ... w'_m$
- aktuelle Kopfposition ist Kopf auf w'_i

„aktuell bearbeitet“heißt: die Felder, die bereits vom Kopf besucht wurden
Beobachtung:

- Startkonfiguration auf $w = w_1 ... w_n$: $\quad q_0 w_1 w_2 ... w_n$
- Stoppkonfiguration: $w'_1 w'_2 ... w'_{i-1} q' w'_i w'_{i+1} ... w'_m \quad q' \in Q'$
- M hält auf $w \rightarrow$ Berechnung von M auf w ist $k_0 \rightarrow k_1 \rightarrow ... \rightarrow k_r$
  $k_0 = k_0(w), \quad k_r$ Stoppkonfiguration
  k_{i+1} ist Nachfolgekonfiguration von $k_i \quad i = 0, ..., r-1$
- M hält nicht auf $w \quad k_0 \overset{M}{\rightarrow} k_1 \overset{M}{\rightarrow} k_2 \overset{M}{\rightarrow} ...$
  Enthält die Berechnung von M auf $k_0(w)$ keine Stoppkonfiguration ist die Folge der Konfigurationen unendlich.

$f(M\#x)$ besteht aus einer Menge von Wortpaaren (Regeln):
Vier Untermengen:

1. Startregel
2. Kopierregeln
3. Überführungsregeln

4. Löschregeln und Schlußregeln

$f(M\#x)$ ist Instanz für MPKP über $\{0,1,\#,\$,\&\} \cup Q$
Sprechweise:
Gilt I ist Instanz zu PKP (oder MPKP) und $i_1, i_2, ..., i_s$ ist Lösung zu dieser Instanz dann heißt $x_{i_1}, x_{i_2}, ..., x_{i_s} = y_{i_1}, y_{i_2}, ..., y_{i_s}$ Lösungswort von I.
Zeige:
$M\#w \in H \Leftrightarrow f(M\#x) \in$ MPKP und das Lösungswort zu $f(M\#w)$ ist
$\$\$k_0(w_1)\$\$ \underbrace{k_1\$\$k_2\$\$...\$\$k_r}_{Phase1} \$\$ \underbrace{k_{r,1}\$\$k_{r,2}\$\$...\$\$k_{r,n}}_{Phase2} \$\$\$$
In Phase 1 werden die Überführungsregeln, in Phase 2 die Löschregeln angewendet.
Beobachtung:

1. Anfangsregel: $x...\$ \qquad y\$\$k_0(w)$
   $\Rightarrow x$ muss $\$k_0w$ kopieren:
   Beispiel: $k_0w = q_0w_1w_2...w_r$ und (q_0, w_1, w_1', R, q)
   $x : \$\$q_0w_1w_2w_3\$...$
   $y : \$\$q_0w_1w_2w_3\$w_1'qw_2w_3\$...$

2. Phase 1: Die x-Folge muss für alle $i = 0, ..., r-1$ Konfigurationen k_i durch Kopier- und Überführungsregeln aufschreiben und erzeugt dabei auf y-Seite Konfigurationen y_{i+1}.
   Dieser Prozess endet genau dann wenn eine Stoppkonfiguration k_r auf die y-Seite geschrieben wird, (ansonsten nie).
   $(\Rightarrow M\#w \notin H \Rightarrow f(M\#w) \notin$ MPKP)
   Annahme: $M\#w \in H$ und H_r Stoppkonfiguration
   $H_r = w_1'w_2'...w_{i-1}'q'w_i'...w_m'$

3. Phase 2: Es sind nur noch Kopier- und Löschregeln anwendbar, x muß H_r auf seine Seite schreiben:
   $x : \$\$H_0(w)\$\$...\$\$w_1'...w_{i-1}'q'w_i'...w_m'$
   $y : \$\$H_0(w)\$\$...\$\$H_r\$\$w_1'...w_{i-2}'q'w_i'...w_m'$
   x muß „um q' herum“Löschregeln anwenden, die die Inschrift auf y-Seite um ein Zeichen verkürzen.
   Schlussregel: $x : \$...\$\$aq'\$\$q'\$\$\$$
   $y : \$...\$\$aq'\$\$q'\$\$\$$

Kapitel 5

NP-Vollständigkeit

5.1 Motivation

- Kürzeste Wege
- Minimaler Spannbaum
- Zyklusfreiheit von Graphen

5.1.1 CNF-Problem

Definition 5.1 (CNF-Formel)
$C = \underbrace{C_1 \wedge C_2 \wedge ... \wedge C_n}_{Klauseln}$ *über* $x_1, ..., x_n$ *(x_i sind Variablen aus {0,1})*

Definition 5.2 (Klauseln)
$C_i = \underbrace{L_1 \vee L_2 \vee ... \vee L_n}_{Literale}$

Definition 5.3 (Triviale Klauseln)
0,1, wobei 0 auch die leere Klausel genannt wird.

Definition 5.4 (Literal)
$x_i, \neg x_i \quad i \in \{1, ..., n\}$ *(z.B.* $1 = x_i \vee \neg x_i$*)*

Wir nehmen OBdA an, daß in jeder Klausel höchstens ein Element aus $\{x_i, \neg x_i\}$ $\forall i$ vorkommt.

Eigenschaft 5.1
$\forall\, f : \{0,1\}^n \to \{0,1\}$ *gilt:* $\exists$ *CNF-Formel, die* f *berechnet*

$$\bigwedge_{w=(w_1,...,w_k)\in\{0,1\}^n} c(w) \qquad c(w) = \bigvee_{i=1}^{n} L_i(w)$$

$$L_i(w) = \begin{cases} x_i & w_i = 0 \\ \neg x_i & w_i = 1 \end{cases}$$

Es gilt

$$c(w)c(w') = \begin{cases} 0 & w = w' \\ 1 & sonst \end{cases}$$

Beispiel

x	y		
0	0	0	$(x \vee y)$
0	1	0	$(x \vee \neg y)$
1	0	0	$(\neg x \vee y)$
1	1	1	$(x \vee y) \wedge (x \vee \neg y) \wedge (\neg x \vee y)$

Definition 5.5
Jede boolsche Formel C über $x_1, ..., x_n$ berechnet eine boolsche Funktion
$f : \{0,1\}^n \to \{0,1\}$ *bzw.* $C : \{0,1\}^n \to \{0,1\}$

Bei CNF-Formel:

$C = \bigwedge_{i=1}^{m} C_i \qquad C(w) = \bigwedge_{i=1}^{m} C_i(w);$

$C_i = L_1 \vee ... \vee L_k \qquad C_i(w) = L_1(w) \vee ... \vee L_k(w) \quad \forall w \in \{0,1\}^n$
$x_j(w) = w_j \qquad \neg x_j(w) = 1 - w_j$

CNF-Formel über $x_1, ..., x_n$

$w \in \{0,1\}^n$ ist erfüllende Bedingung für C falls $C(w) = 1 \Leftrightarrow \forall i = 1, ..., m\ C_i(w) = 1$

Beispiele für praxisrelevante Probleme, für die trotz jahrzehntelanger Bemühungen keine Polynomialzeitalgorithmen gefunden wurden:

5.1.2 SAT-Problem

Eingabe: Boolsche Formel in konjunktiver Normalform (CNF-Formel):
$C = C_1 \wedge ... \wedge C_m$ über $x_1, ..., x_m$ in CNF-Form
Frage: Ist C erfüllbar? ($\exists w \in \{0,1\}^*$ mit $C(w) = 1$)
$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_3)$
$\Rightarrow$ (100) ist z.B. erfüllend
$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2) \Rightarrow$ nicht erfüllbar
$x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_1) \Rightarrow$ nicht erfüllbar

Algorithmen für SAT

Exhaustive Search (Vollständige Suche):
forall $w \in \{0,1\}^n$ do { if $C(w) = 1$ stop 1}
stop 0
Laufzeit: $O(n * m * 2^n)$

Beobachtung 5.1
Gegeben sei $w \in \{0,1\}^n \quad C = C_1 \wedge ... \wedge C_m$, *dann ist* $C(w)$ *in* $O(n*m)$ *berechenbar.*

1. *SAT ist extrem praxisrelevant*
2. *Wir kennen keine Polynomialzeitalgorithmen für SAT*

Situation: Wir wissen nicht ob $SAT \in P$.

Definition 5.6 (K-CNF)
CNF-Formel $C = \bigwedge_{i=1}^{m} C_i$ *heißt k-CNF Formel* $k \in N^+$ *falls* $\forall i = 1, ..., m$
C_i *enthält höchstens k-Literate.*

Definition 5.7 (k-SAT)
Eingabe: k-CNF-Formel C
Ausgabe: $1 \leftrightarrow \exists w$ *mit* $C(w) = 1$

Frage: Für welche $k \geq 1$ gilt k-SAT $\in$ P?
Antwort:

1-SAT $\in$ P 1-Formel: $L_1 \wedge L_2 \wedge ... \wedge L_m$ erfüllbar gdw. $\exists$ für kein j Literal x_j und $\neg x_j$ in C bzw. kein Literal ist 0.

2-SAT $\in$ P es existiert ein schneller Algorithmus für 2-SAT, der auf der Berechnung starker Zusammenhangskomponenten eines passend konstruierten Graphen basiert.

3-SAT: 3-SAT ist genauso schwer wie SAT. 3-SAT $\in$ P $\Rightarrow$ SAT $\in$ P

5.1.3 Travelling Salesman Problem (TSP)

Eingabe:

- n ($\equiv n$ Städte)
- $D = (d_{ij})_{ij=1}^{n} \qquad d_{ij} \in N$ (Distanzmatrix d_{ij} = Distanz von i nach j)

mögliche Lösungen: $\Pi \in \Sigma_n$ (Menge der Permutationen von $1...n$)
Kosten für eine Rundfahrt: $C(\Pi, D) = \sum_{i=1}^{n-1} d_{(\Pi(i)\Pi(i+1))} + d_{(\Pi(n)\Pi(1))}$
TSP $= \{(n, D, \Pi); C(\Pi, D) = min_{\Pi' \in \Sigma_n'} C(\Pi', D)\}$

Entscheidungsvariante von TSP: Dec-TSP

Dec-TSP

Eingabe: $n, D, k \in N$
Frage: $\exists$ Rundfahrt $\Pi : C(\Pi, D) \leq k$

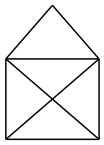
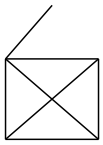
Beobachtung 5.2
Dec-TSP $\in P \Rightarrow$ *TSP* $\in$ *PTIME*

Algorithmus für Dec-TSP

Exhaustive search: Durchsuche die Menge aller $\Pi' \in \Sigma_n$ nach Π mit $C(\Pi, D) \leq k$.
Laufzeit: $\Omega(n * |\Sigma_n|)$ $\quad n! = 2^{\Theta(n\ log\ n)}$
$(\exists c_1, c_2 \in R^+ \quad 2^{c_1 n\ log\ n} \leq n! \leq 2^{c_2 n\ log\ n})$
Bei heutige Rechenleistung:
2^{60} Prozessortakte unter gewaltigen Anstrenungen möglich
2^{80} Prozessortakte „sicher“

5.1.4 Hamilton Circuit (HC)

Eingabe: Ungerichteter Graph $G = (V, E)$
Frage: Hat G einen HC (einfacher Kreis, der alle Knoten enthält)?

Beispiel:  $\in HC$  $\notin HC$

auch hier: HC $\overset{?}{\in}$ P

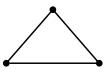
Exhaustive search:
wenn $n = |v| \quad v \in \{v_1, ..., v_n\}$ für alle $\Pi \in G_n$
test ob $\{(v_{\Pi(1)}, v_{\Pi(2)}), (v_{\Pi(2)}, v_{\Pi(3)}), ..., (v_{\Pi(n-1)}, v_{\Pi(n)}), (v_{\Pi(n)}, v_{\Pi(1)})\} \subseteq E$

5.1.5 Clique

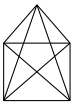
Eingabe: Ungerichteter Graph $G = (V, E), \quad k \in N$
Frage: Enthält G eine Clique der Größe mindestens k (k-Clique)?
Ein Graph heißt vollständig wenn er alle möglichen Kanten enthält.
Ein vollständiger Graph mit k Knoten heißt k-Clique (und hat $\binom{k}{2}$ Knoten).

1-Clique:

2-Clique:

3-Clique: 

4-Clique: 

5-Clique: 

$\left(\text{  }, 4\right) \in Clique \quad \left(\text{}, 5\right) \notin Clique$

Es ist unklar ob Clique $\in$ P.

Exhaustive search:
Teste für alle $V' \subseteq V$ mit $|V'| = k$ ob $\{(v', w'); v' \neq w' \in V\} \subseteq E$
Laufzeit:
$\Omega\binom{|V|}{k}$
$|V| = n$, Algorithmus läuft auf Eingaben $(G = (V, E), \frac{n}{2})$
$\Omega(\binom{n}{n/2}) \approx \frac{2^n}{\sqrt{(n)}}$

5.1.6 Matching

Eingabe: ungerichteter Graph $G = (V, E), \quad k \in N^+$
Frage: Enthält G ein Matching der Größe k, also eine Menge paarweise disjunkter Kanten?
Zwei Kanten e=(v,w) und e'=(v',w') sind paarweise disjunkt $\iff |\{v, v', w, w'\}| = 4$

Beispiele

$\left(\boxtimes, 2\right) \in Matching \quad \left(\boxtimes, 3\right) \notin Matching$

Perfect matching

Eingabe: $G = (V, E)$
Frage: Hat G perfektes Matching, d.h. ein Matching mit $\lfloor \frac{|v|}{2} \rfloor$ Knoten?
Matching $\in$ **P** (bester bekannter Algorithmus: $O(n^3)$)

5.1.7 Rucksack

Eingabe:

- n (Zahl von Objekten)
- $c_1, ..., c_n \in N^+$ Nutzenwerte
- $w_1, ..., w_n \in N^+$ Gewichte
- w Gewichtsschranke
- c Nutzenschranke

Frage: $\exists\ I \subseteq \{1, ..., n\}$ mit $c(I) = \sum_{i \in I} c_i \geq c$ und $w(I) = \sum_{i \in I} w_i \leq w$
Rucksack $\overset{?}{\in}$ P

5.1.8 Partition

Eingabe:

- n
- $w_1, ..., w_n \in N^+$

Frage: $\exists\ I \subseteq \{1, ..., n\}$
$w(I) = \sum_{i \in I} w_i = \sum_{i \notin I} w_i = w(\{1, ..., n\} \setminus I)$

Exhaustive search:
Teste alle $I \subseteq \{1, ..., n\}$ auf die gewünschte Eigenschaft.
Laufzeit: $\Omega(2^n)$
Partition $\overset{?}{\in}$ P

5.2 Nichtdeterministische Touring-Maschinen und die Klasse NP

Definition 5.8 (Nichtdeterministische Touring-Maschinen (NTM))
hier: 1-Band NTMs über Σ $N = (Q, q_0, ACC, REJ, \delta)$*, wobei* Q *die Zustandsmenge,* q_0 *der Anfangszustand und* ACC/REJ *akzeptierende bzw. verwerfende Stoppzustände sind.*
$\delta : Q \times \Sigma \to \Sigma \times MOVE \times 2^Q$
$(q, s) \to (s', m, Q') \leftarrow$ *Menge der möglichen Nachfolgezustände*

Wirkung: Bei DTM (deterministischen TM) existiert für eine gegebene Anfangskonfiguration k_0 ein eindeutig bestimmter Rechenweg $k_0 \to k_1 \to k_2 \to ...$
Bei NTMs gibt es zu jeder gegebenen Konfiguration $k = w_1...w_{i-1}q_jw_i...w_m$ in der Regel mehrere mögliche Nachfolgekonfigurationen (Beispiel: wenn $\delta(q_j, w_i) = (s, L, \{q_k, q_l\})$, dann sind $k' = w_1...w_{i-2}q_kw_{i-1}sw_{i+1}...w_m$ und $k'' = w_1...w_{i-2}q_lw_{i-1}sw_{i+1}...w_m$ die beiden möglichen Nachfolgekonfigurationen).
(Die Terminologie sei hier wie im PKP-Beweis:

- $w_1, ..., w_m$: aktuelle Bandinschrift
- q_j: aktueller Zustand
- i: aktuelle Konfiguration

$\Rightarrow$ für jede Konfiguration k existieren mehrere mögliche Berechnungen von M auf k
$\Rightarrow$ für jede Eingabe x zu M existieren in der Regel verschiedene Berechnungsverläufe von M auf x (bzw. $k_0(x)$), die akzeptierend, verwerfend oder gar nicht enden müssen.

Definition 5.9
gegeben: NTM M*, Eingabe* x *für* M
$time_M(x) =$ *maximale Länge einer Berechnung von* M *auf* x
$time_M(n) = max_{|x|=n}(time_M(x))$
Zeitverhalten $time_M : N \to N$ *von* M

Definition 5.10
Die von einer NTM M *berechnete Sprache bestehe aus genau den Eingaben, für die es eine akzeptierende Berechnung gibt.*

Konvention:

Wir betrachten nur NTMs, die auf jeder möglichen Berechnung anhalten.

Erläuterung

NTMs können nichtdeterministisch einen sogenannten „Ratestring "(einer vorgegebenen Länge $p(n)$) auf das Band schreiben.

- M erzeugt deterministisch die Binärdarstellung $bin(p(n))$
- M schreibt zeichenweise einen String der Länge $p(n)$, wobei bei jedem Bit nichtdeterministisch zwischen „Schreibe 0"und „Schreibe 1"gewählt wird.
  $(q_i, \#, 0, R, \{q_i q_j\})$ $\quad$ $(q_j, \#, 1, R, \{q_i q_j\})$
  $\rightarrow$ dabei wird der Binärzähler entsprechend heruntergezählt.

$$M \quad \begin{array}{c} q_0 x_1 \ldots x_n \\ \downarrow \quad \searrow \quad \searrow \\ x_1 \ldots x_n \# y_1 \ldots y_{p(n)} q \end{array}$$

Definition 5.11 (NP)
NP=$\{L; L$ *hat nichtdeterministische TM mit* $time_M \in n^{O(1)}\}$ $\quad P \subseteq NP$

Bemerkung 5.1
Typische Operation für NTMs ist:
gegeben: $t : N \rightarrow N$, *„Erzeuge nichtdeterministisch einen Ratestring der Länge* $t(|x|)$*"(M erzeugt* $bin(t(|x|))$ *und schreibt einen String der Länge* $t(|x|)$*, wobei bei jedem Bit nichtdeterministisch zwischen „Schreibe 0"und „Schreibe 1"gewählt wird.)*

NP-Algorithmus für das SAT-Problem

Eingabe: n, $C = \bigwedge_{i=1}^{m} C_i$ $\quad C_i$ Klausel über $x_1, ..., x_n, \bar{x_1}, ..., \bar{x_n}$
NP-Algorithmus:

- Rate String $b = (b_1, ..., b_n) \in \{0,1\}^n$ $\qquad (O(n))$
- Verifiziere deterministisch ob $C(b) = 1$ (d.h. ob $C_i(b) = 1 \quad \forall i = 1, ..., n$ $O(n * m)$)

Beobachtung 5.3

1. *Laufzeit ist* $O(n * m)$
2. M^* *berechnet* $L = \{(n, C);\ \exists b = (b_1, ..., b_n)$ *mit* $C(b) = 1\} \stackrel{Def.}{=} SAT.$
   $\Rightarrow SAT \in NP \Rightarrow$ *3-*$SAT \in NP$

NP-Algorithmus für das Dec-TSP-Problem

Eingabe: $D = (d_{ij})_{ij+1}^{n}, k$
NP-Algorithmus:

- Rate eine Permutation $\Pi \in \Sigma_n$
  Bemerkung: Rate String der Länge $n * bin(n)$, der als Permutation integriert wird.
- Verifiziere deterministisch ob $C(\Pi, D) \leq k$
  $C(\Pi, D) = \sum_{i=1}^{n-1} d_{\Pi(i),\Pi(i+1)} + d_{\Pi(n),\Pi(1)}$

Laufzeit: $O(n\ log\ n + n * m)$
$\Rightarrow$ Polynomialzeitalgorithmus, berechnet $L = \{(n, D); \exists \Pi \in \Sigma_n; \quad C(\Pi, D) \leq k\} =$ Dec-TSP

NP-Algorithmus für das Clique-Problem

Eingabe: ungerichteter Graph $G = (V, E), \quad k \in N^+$
NP-Algorithmus:

- Rate Menge $V' \subseteq V$ mit $|V'| = k$
  genauer: $V = \{v_1, ..., v_n\}$. Rate Bitstring der Länge $k(\lfloor log\ n \rfloor + 1)$ und interpretiere diesen, wenn möglich, als Teilmenge $V' \subseteq V$, $|v'| = k$.
  Falls $i_r \neq i_s \quad \forall r, s \Rightarrow V' = \{i_1, i_2, ..., i_k\}$.
- Verifiziere ob für alle $v'.v'' \in V'$ gilt: $v' \neq v'' \Rightarrow (v', v'') \in E$

Laufzeit: $O(k\ log\ n + \binom{k}{2}) = O(n\ log\ n + n^2) = O(n^2)$

NP-Algorithmus für Rucksackproblem

Eingabe: $c_1, ..., c_n, c \in N^+ \qquad w_1, ..., w_n, w \in N^+$
NP-Algorithmus:

- Rate $I \in \{1, ..., n\}$
- Verifiziere ob $w(I) \leq w \quad c(I) \geq c$

Laufzeit: $O(n * m)$

5.2.1 Spezielle NP-Algorithmen

Definition 5.12 (Rate+Verifiziere Algorithmen)
Eingabe: x
Phase 1: Rate nichtdeterministisch einen String $y \in \{0,1\}^{t(|x|)}$, wobei $t : N \to N$ eine vorgegebene Funktion ist.
Phase 2: Arbeite deterministisch auf $x\#y$, bzw. produziere determinitisch 0 oder 1 auf $x\#y$.

Satz 5.1
$\forall L \in$ NP gilt: Es existiert ein polynomieller Rate+Verifiziere Algorithmus für L

Beweis:
$L \in$ NP $\Rightarrow$ wir können polynomiell zeitbeschränkte NTM M für L finden.

Wir fixieren ein Polynom $p : N \to N \quad time_M(n) \leq p(n)$
M habe die Eigenschaft, dass es in jedem Zustand höchstens zwei Nachfolgezustände gibt. Wir simulieren M durch eine Rate+Verifiziere TM M':
Eingabe x:

- M' rät String $y \in \{0,1\}^{O(|x|)}$
- Lesekopf von y steht anfänglich auf y_1
- M' startet M auf $x^{(y)}$: Falls nichtdeterministisch ein Zustand mit Nachfolgezuständen q_0, q_1 erreicht wird, so geht M' nach q_0 falls das aktuelles y-Bit 0 ist und nach q_1 falls 1. Der Lesekopf auf y geht ein Bit nach rechts.

Bemerkung 5.2
Jede NTM M mit Zeitbedarf $t(n)$ kann durch $O(t(n))$ zeitbeschränkte NTM $\bar{M}$ simuliert werden, für die es in jedem Zustand höchstens 2 Nachfolgezustände gibt.
$I = (q, s, s', m, \{q_1, ..., q_s\}) \quad I \in M$
$M' : I_1 = (q, s, s', m, \{q_1, q_2'\})$
$I_2 = (q_2', s', s', N, \{q_2, q_3'\})$
$I_3 = (q_3', s', s', N, \{q_3, q_4'\})$
...
$I_{s-1} = (q_{s-1}', s', s', N, \{q_s\})$
$\Rightarrow$ *Logische Charakterisierung von NP*

Satz 5.2
NP=$\{L \subseteq \Sigma^*; \exists$ *Polynom* $p = p(n)$ *und* $\exists$ *Sprache* $L' \in P$ *mit* $x \in L$ *gdw.* $\exists\, y \in \{0,1\}^{p(|x|)}$ *mit* $x\#y \in L'\}$ $\quad L'$ *heißt die zu L gehörige Verifikationssprache.*

Beweis:
$L \in$ NP $\Rightarrow \exists$ polynomieller Rate+Verifiziere Algorithmus für L, d.h. $\exists\, p = p(n) \in n^{O(1)}$, so dass:
Eingabe: x
Rate $y \in \{0,1\}^{p(|x|)}$; Verifiziere (deterministisch in Polynomialzeit) $x\#y$
$L' = \{x\#y$; Rate+Verifiziere Algorithmus akzeptiert $x\#y\}$; $\quad L' \in$ P
Beispiel:
Die Verifikationssprache zu SAT ist $L' = \{(n, C, b), n \in N^+, C$ CNF-Formel über $\{x_1, ..., x_n, \bar{x_1}, ..., \bar{x_n}\}, b \in \{0,1\}^*$ mit $C(b) = 1\}$
denn: $(n, C) \in$ SAT $\Leftrightarrow \exists b \in \{0,1\}^n$ mit $(n, C, b) \in L'$
Die Verifikationssprache zu Clique ist $L' = \{(n, G, K, V'); V' \subseteq V$ Clique zu $G; |V'| = k\}$
P $\overset{?}{=}$ NP

Definition 5.13 (EXPTIME)
EXPTIME $= \{L; \exists 2^{n^{O(1)}}$*-zeitbeschränkte TM für* $L\} \subseteq$ *REK*

Satz 5.3 *NP* $\subseteq$ *EXPTIME*

Beweis:
$L \in$ NP

$\Rightarrow \exists$ Polynom $p : N \to N$ polynomiell zeitbeschränkte TM M' mit $x \in L \iff$
$\exists\, y \in \{0,1\}^{p(|x|)} \qquad M'(x\#y) = 1 \Rightarrow$ EXPTIME-Algorithmus für L.

1. forall $y \in \{0,1\}^{p(|x|)}$ do
   if $M'(x\#y) = 1$ accept x;
2. reject x

5.2.2 Polynomielle Reduktion

Definition 5.14 (Polynomielle Reduktion)
$f : \{0,1\}^* \to \{0,1\}^*$ heißt polynomielle Reduktion einer Sprache $L \subseteq \{0,1\}^*$ auf $L' \subseteq \{0,1\}^*$ falls

1. $\forall x \in \{0,1\}^* : \quad x \in L \iff f(x) \in L'$
2. $f \in PTIME$

$L \leq_{pol} L'$ (L ist polynomiell reduzierbar auf L') $\iff \exists$ polynomielle Reduktion von L auf L'

Bemerkung 5.3 *$L \leq_{pol} L'$ intuitiv: L ist nicht wesentlich schwieriger als L'*

Lemma 5.1 *$L \leq_{pol} L'$ und $L' \in P \to L \in P$*

Beweis:
$L \leq_{pol} L'$; $\quad L' \in$ P
Wir konstruieren polynomiell zeitbeschränkte TMs M_f für f und M' für L'; f ist eine polynomielle Reduktion von L auf L'
Definition von M: Eingabe: $x \in \{0,1\}^*$: Starte M_f auf x $\Rightarrow$ liefert $x' = f(x)$. Starte M' auf x'; akzeptiere $x \iff M'(x') = 1$. Genauso gilt:

Lemma 5.2 *$L \leq_{pol} L', \quad L' \in NP \Rightarrow L \in NP$*

Beispiele

1. HC $\leq_{pol}$ DEC-TSP
   HC: Eingabe: $G = (V, E)$ ungerichter Graph; $G \in$ HC $\iff G$ hat hamiltonian circuit
   DEC-TSP: $\{(D, K), D = (d_j)_{i,j=1}^m; k \in N; \exists$ Rundfahrt bzgl. D mit Länge $\leq k\}$ Konstruktion von $f \in$ PTIME $\qquad f(G) = (D_G, k_G) \quad G \leq$ HC $\iff$ $(D_G, k_G) \in$ DEC-TSP
   $G = (V, E) \quad V = \{v_1, ..., v_n\} \quad D_G = (D_G^{ij})_{i,j=1}^n$

$$D_G^{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 2 & (v_i, v_j) \notin E \end{cases}$$

   $k_G = n$
   Behauptung: $G \to_f (D_G, k_G)$ ist eine polynomielle Reduktion von HC auf DEC-TSP.

- $f \in$ PTIME
- $G \in$ HC $\Rightarrow (D_G, k_G) \in$ DEC-TSP, denn nehme an $v_{\Pi(1)} - v_{\Pi(2)} - v_{\Pi(3)} - ... - v_{\Pi(1)}$ sein HC in $G \Rightarrow C_D(\Pi) = n$
- $G \notin$ HC $\Rightarrow \forall\, \Pi \in \Sigma_n \exists i, 1 \leq i \leq n, (v_{\Pi(i)}, v_{\Pi(i+1 mod n)}) \notin E$
  $\Rightarrow C_G(\Pi) \geq n - 1 + 2 = n + 1 > n \Rightarrow (D_G, k_G) \notin$ DEC-TSP

2. 3-SAT $\leq_{pol}$ SAT
   gegeben: 3-SAT Formel C; C ist auch Eingabe für SAT; $C \in$ 3-SAT $\Rightarrow C \in SAT \Rightarrow C \rightarrow_f C$ ist polynomielle Reduktion von 3-SAT auf SAT, da 3-SAT ein Unterproblem von SAT ist.

3. SAT $\leq_{pol}$ 3-SAT (wir wissen dass 2-SAT $\in$ P ist)
   gegeben: CNF-Formel $C = \bigwedge_{i=1}^{n} C_i$; $\quad C_i$ ist Klausel über $x_1, ..., x_n, \neg x_1, ..., \neg x_n$
   $C \rightarrow C' = \bigwedge_{i=1}^{m} \bigwedge_{l_i=1}^{m_i} C'_{il_i} \quad C'_{il_i}$ sind Klauseln mit höchstens 3 Literalen mit

   (a) C erfüllbar $\Longleftrightarrow C'$ erfüllbar

   (b) $f \in$ PTIME

   <u>Definition der Transformation</u>
   Wir definieren für alle $i = 1, ..., m$ die 3-CNF Formel C'_i
   Fall 1: C_i enthält ≤ 3 Literale $\Rightarrow C'_i = C_i$
   Fall 2: C_i enthält ≥ 4 Variablen, d.h. $C_i = L_1 \vee L_2 \vee .. \vee L_s; s \geq 4 \quad L_r \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\} \quad \forall r = 1, \ldots, s$
   Wir definieren eine 3-CNF Formel C'_i über $L_1, \ldots, L_s, y_1^{(i)}, \ldots, y_{s-3}^{(i)}, \neg y_1^{(i)}, \ldots, \neg y_{s-3}^{(i)}$ mit folgender Eigenschaft: $\forall b \in \{0,1\}^n$ gilt $C_i(b) = 1 \Longleftrightarrow \exists d \in \{0,1\}^{s-3}\, C'_i(b,d) = 1$. Diese Eigenschaft ist hinreichend: $C \in$ SAT $\Longleftrightarrow \exists b \in \{0,1\}^n \quad \forall i = 1, ..., m \quad C_i(b) = 1 \Longleftrightarrow \exists b \in \{0,1\}^n\ \forall i = 1, ..., m\ \exists\ d^{(i)} \in \{0,1\}^{s(i)-3}$ mit $C'_i(b, d^{(i)}) = 1 \Longleftrightarrow C' \in$ 3-SAT
   <u>Idee der Konstruktion</u>
   $C_i = x_1 \vee x_2 \vee x_3 \vee x_4$
   $\Rightarrow C'_i = (x_1 \vee x_2 \vee y)(\neg y \vee x_3 \vee x_4)$
   Behauptung: $\forall\, b \in \{0,1\}^n$ gilt $C_i(b) = 1 \leftrightarrow \exists$ Belegung d für y mit $C'_i(b,d) = 1$
   Wir betrachten:

   (a) $b = 0010 \quad (C_i(b) = 1) \quad$ müssen y=1 setzen $\Rightarrow C'_{(b,d)} = (0 \vee 0 \vee 1)(0 \vee 1 \vee 0)$

   (b) $b = 0100 \quad$ müssen $y = 0$ setzen ($\Rightarrow$ 2. Klausel), 1. Klausel erfüllt, da $x_2 = 1$

   (c) $b = 0000 \Rightarrow C'_i(b, \cdot)$ nicht erfüllbar

   <u>Allgemeine Konstruktion:</u>
   $C_i = L_1 \vee L_2 \vee ... \vee L_s \quad s \geq 4 \Rightarrow C'_i = (L_1 \vee L_2 \vee y_1^{(i)}) \wedge (\neg y_1^{(i)} \vee L_3 \vee y_2^{(i)}) \wedge (\neg y_2^{(i)} \vee L_4 \vee y_3^{(i)}) \wedge ... \wedge (\neg y_{s-4}^{(i)} \vee L_{s-2} \vee y_{s-3}^{(i)}) \wedge (\neg y_{s-3}^{(i)} \vee L_{s-1} \vee L_s)$
   Wir müssen zeigen:

   (a) $f \in$ PTIME (klar!)

(b) $C_i(b) = 1 \Rightarrow \exists\, d^{(i)}$ von $y_1^{(i)}, ..., y_{s-3}^{(i)}$ mit $C_i'(b, d^{(i)}) = 1$

(c) $C_i(b) = 0 \Rightarrow \forall\, d^{(i)} \in \{0,1\}^{s-3} \quad C_i'(b, d^{(i)}) = 0$

Zu Punkt (b):
$b \in \{0,1\}^n, C_i(b) = 1 \Rightarrow \exists r \in \{1, s\}$ mit $L_r(b) = 1$, wir konstruieren $d^{(i)}$ mit $C'(b, d^{(i)}) = 1$.
Wir beobachten: $\neg y_{r-2}^{(i)} \vee L_r \vee y_{r-1}^{(i)}$ ist erfüllt, da $L_r(b) = 1$.
Wir setzen $d_{r-1}^{(i)} = 0 \quad d_{r-2}^{(i)} = 1$
Konkret setzen wir $d_1^{(i)} = ... = d_{r-2}^{(i)} = 1; \quad d_{r-1}^{(i)} = ... = d_{s-3}^{(i)} = 0$
$\Rightarrow d^{(i)} = (1...10...0) \Rightarrow$ alle Klauseln sind erfüllt

Definition 5.15 (NP-Vollständig)
$L \subseteq \Sigma^*$ heißt NP-vollständig $\leftrightarrow$

1. *$L \in NP$*
2. *$\forall L' \in NP \quad L' \leq_{pol} L$*

Intuitiv: L ist „schwerstes“Problem in NP

Definition 5.16 *$NPC = \{L; L$ ist NP-vollständig$\}$*

Entscheidende Eigenschaft:

Satz 5.4
$NPC \cap P \neq \emptyset \Longleftrightarrow P = NP$
Folgerung: $P \neq NP \rightarrow NPC \cap P = \emptyset$
d.h. $P \neq NP \Rightarrow$ kein NPC Problem hat eine Polynomialzeitalgorithmus
$\Rightarrow$ falls $L \in NPC$ gezeigt werden kann ist $L \notin P$

Beweis:
P=NP $\Rightarrow$ P $\cap$ NPC $\neq \emptyset$ (falls NPC $\neq \emptyset$)
P $\cap$ NPC $\neq \emptyset \Rightarrow$ P = NP; wir setzen P $\cap$ NPC $\neq \emptyset$ voraus. Sei $L \in$ NP beliebig, zu zeigen: $L \in$ P. Es gilt $\exists\, L^* \in$ P $\cap$ NPC $\Rightarrow L \leq_{pol} L^* \overset{L^* \in P}{\Longrightarrow} L \in$ P

Frage: Gibt es NP-vollständige Probleme?

5.2.3 Satz von Cook (1973)

Satz 5.5 (Cook (1973)) *SAT ist NP-vollständig*

Beweis:

1. SAT $\in$ NP (schon überprüft)
2. fixieren beliebiges $L \in$ NP, zu zeigen: $L \leq_{pol}$ SAT
   Wir konstruieren eine polynomielle Reduktion L auf SAT, d.h.: konstruieren einen Polynomialzeitalgorithmus für $x \in \{0,1\}^* \Rightarrow$ CNF-Formel C_x mit C_x erfüllbar $\Longleftrightarrow x \in L$

Konstruktion von C_x

Wir wissen: $\exists$ Polynome $p = p(n), q = q(n) \in n^{O(1)}$
$\exists\ q'(n)-$zeitbeschränkte, deterministische TM M' mit $x \in L \iff \exists y \in \{0,1\}^{p(|x|)}$, so dass $M(x\#y) = 1$
speziell:

- M ist eine 1-Band TM $M = \{Q, q_0, F, \delta\}$ über $\{0, 1, \#\}$
- $F \subseteq Q$ die Menge der akzeptierenden Stoppzustände
- M stoppt für alle $x \in \{0,1\}^*$ und alle $y \in \{0,1\}^{p(|x|)}$ auf $x\#y$ nach genau $t(|x|)$ Takten

Jede polynomiell zeitbeschränkte TM lässt sich in eine „spezielle“polynomiell zeitbeschränkte TM umwandeln $\Rightarrow \forall\ x \in \{0,1\}^*$ gilt: $x \in L \iff \exists y \in \{0,1\}^{p(|x|)}, \exists$ Konfigurationsfolge $k_0 \to k_1 \to k_2 \to ... \to k_{t(|x|)}$ mit

1. $k_0 = k_0(x\#y)$
2. $k_i \overset{M}{\to} k_{i+1} \quad \forall i = 0, ..., t(|x|) - 1$
3. $k_{t(|x|)}$ ist akzeptierende Stoppkonfiguration

Wir kodieren $\forall i = 0, ..., t(|x|)$ die Konfiguration k_i durch geeignet gewählte boolsche Variablen und konstruieren eine CNF-Formel C_x mit diesen Variablen mit $C_x(y, k_0, ..., k_{t(|x|)}) = 1 \iff k_0 \to ... \to k_{t(|x|)}$ ist akzeptierte Berechnung von M auf $x\#y$
$\Rightarrow C_x$ erfüllbar $\iff x \in L$ (fertig)
Wir fixieren $x \in \{0,1\}^*$ beliebig, $|x| = n \Rightarrow C_x$ ist definiert über den Variablen $Y \cup H \cup P \cup Q$

- $Y = \{y_1, ..., y_{p(n)}\}$
  y_i definiert den Inhalt von Bandzelle $n + 1 + i$ in Konfiguration k_0:
  Bandinschrift zum Zeitpunkt 0 ist $x_1, ..., x_n\#y_1, ..., y_n$
- $H = \bigcup_{-t(n) \le p \le t(n); 0 \le t \le t(n)} H_{t,l}$
  $H_{t,l} = \{H^0_{t,l}, H^1_{t,l}, H^\#_{t,l}\}$
  Interpretation: $\forall$ Zeitpunkte $t \in \{0, ..., t(n)\}$ und Bandpositionen $-t(n) \le l \le t(n)$ und $b \in \{0, 1, \#\}$ gilt $H^b_{t,l} = 1 \iff$ zum Zeitpunkt t steht an Position l das Zeichen b, sonst ist $H^b_{t,l} = 0$
- $P-$Variablen
  $P = \{P_{t,l}; t = 0, ..., t(n); l = -t(n), ..., t(n)\}$
  $P_{t,l} = 1 \iff$ zum Zeitpunkt t steht der Kopf auf Position l
- falls $\{q_1, ..., q_s\}$ Zustandsmenge von M ist, ist $Q = \{Q_{t,r}; r = 1, ..., s; t = 0, ..., t(|x|)\}$
  Interpretation: $Q_{t,r} = 1 \iff M$ ist zum Zeitpunkt t in Zustand q_r

Beobachtung:
Jede Berechnung von M auf $x\#y$ kann in offensichtlicher und eindeutiger Weise als Belegung von $Y \cup H \cup P \cup Q$ kodiert werden. (Bemerkung: In $t(n)$ Schritten kann lediglich der Bereich $-t(n), ..., t(n)$ beschrieben werden)
C_x muss leisten:
Es dürfen nur die Belegungen von $Y \cup H \cup P \cup Q$ durch C_x akzeptiert werden, die akzeptierten Berechnungen von M auf $x\#y$ entsprechen. $\rightarrow \forall$ anderen Belegungen von $Y \cup H \cup P \cup Q$ müssen Klauseln in C_x existieren, die diese Belegungen verwerfen.
$C_x = C_x^1 \wedge C_x^2 \wedge C_x^3 \wedge C_x^4$
C_x^i sind Mengen von Klauseln, die bestimmte Arten von fehlerhaften Belegungen aussondern

- C_x^1 stellt sicher, dass zu jedem Zeitpunkt der Kopf auf genau einer Position steht und zu jedem Zeitpunkt in jeder Bandzelle genau ein Zeichen steht. C_x^1 benutzt eine spezielle CNF-Formel $E_n(z_1, ..., z_n)$
  $E_n(z_1, ..., z_n) = (z_1 \vee z_2 \vee ... \vee z_n) \wedge \bigwedge_{1 \leq i < j \leq n} (\neg z_i \vee \neg z_j)$ es gilt: $E_n(b_1, ..., b_n) = 1 \iff \sum_{i=1}^{n} b_i = 1 \Rightarrow C_x^1 = \bigwedge_{t=0}^{t(n)} [E_{2t(n)+1}(P_{t,-t(n)}, ..., P_{t,t(n)}) \wedge \bigwedge_{l=-t(n)}^{t(n)} E_3(H_{t,l}) \wedge E_S(Q_{t,1}, ..., Q_{t,s})]$

- C_x^2 stellt sicher, dass $H_0 = H_0(x\#y)$ gilt
  $\Rightarrow C_x^2 = \bigwedge_{l=-t(n)}^{0} H_{0,l}^{\#} \wedge \bigwedge_{l=1}^{n} H_{0,l}^{x_l} \wedge H_{0,n+1}^{\#} \wedge \bigwedge_{l=n+2}^{n+p(n)+1} (H_{0,l}^0 \vee Y_{l-p(n)+1}) \wedge (H_{0,l}^1 \vee Y_{l-(p(n)+1)}) \wedge \bigwedge_{l=n+p(n)+2}^{t(n)} H_{0,l}^{\#} \wedge Q_{0,1} \wedge P_{0,1}$

- Klauselgruppe C_x^3 stellt sicher, dass für alle $t = 1, ..., t(n)$ k_t die Nachfolgekonfiguration von k_{t-1} bezüglich M ist.
  Zur Erinnerung: $(\neg x \vee y) \equiv x \rightarrow y \qquad \neg(x_1 \wedge ... \wedge x_r) \vee y \equiv x_1 \wedge ... \wedge x_r \rightarrow y \equiv (\neg x_1 \vee ... \vee \neg x_r \vee y)$ Beispiel in $C_x^2 : Y_{l-(n+1)} \vee H_{0,l}^0 \equiv \neg Y_{l-(n+1)} \rightarrow H_{l,0}^0$
  $\neg Y_{l-(n+1)} \vee H_{0,l}^1 \equiv Y_{l-(n+1)} \rightarrow H_{l,0}^1$
  $C_x^3 = \bigwedge_{t=1}^{t(n)} \bigwedge_{l=-t(n)}^{t(n)} ((\neg P_{t-1,l} \wedge H_{t-1,l}^0) \rightarrow H_{t,l}^0) \wedge ((\neg P_{t-1,l} \wedge H_{t-1,l}^1) \rightarrow H_{t,l}^1) \wedge ((\neg P_{t-1,l} \wedge H_{t-1,l}^{\#}) \rightarrow H_{t,l}^{\#})$
  $\Rightarrow$ Wenn der Kopf zum Zeitpunkt $t-1$ nicht auf l ist, dann steht zu t das Gleiche im Feld l wie zu $t-1$.
  $\delta : Q \times \{0, 1, \#\} \rightarrow \{0, 1, \#\} \times MOVE \times Q$
  $\bigwedge_{r=1}^{s} \bigwedge_{b \in \{0,1,\#\}} ((P_{t-1,l} \wedge Q_{t-1,r} \wedge H_{t-1,l}^b) \rightarrow H_{t,l}^{b'}) \wedge ((P_{t-1,l} \wedge Q_{t-1,r} \wedge H_{t-1,l}^b) \rightarrow P_{t,l+\triangle}) \wedge ((P_{l,t-1} \wedge Q_{t-1,r} \wedge H_{t-1,l}^b) \rightarrow Q_{t,r'})$ für $\delta(q_r, b) = (b', m, q_{r'})$ und

$$\triangle = \begin{cases} 1 & m = R \\ 0 & m = N \\ -1 & m = L \end{cases}$$

- C_x^4 stellt sicher, dass $k_{t(n)}$ akzeptierte Stoppkonfiguration ist
  $\Rightarrow C_x^4 = \bigvee_{r,q_r \in F} Q_{t(n),r}$

Bemerkung: Es gilt:

- $x \in L \iff C_x$ erfüllbar
- $x \to C_x$ ist in Polynomialzeit berechenbar
  1. Für jede Klausel k aus C_x gilt: k ist in Linearzeit aus x, M und $t(|x|)$ berechenbar
  2. Die Anzahl der Klauseln in C_x ist polynomiell in $|x|$ beschränkt
     $|C_x^1| = O(t(n) * (t^2(n) + t(n) + 1)) = O(t^3(n))$
     $|C_x^2| = O(t(n))$
     $|C_x^3| = O(t(n) * t(n) * s * 3) = O(t^2(n))$
     $\Rightarrow |C_x| \in O(t^3(n))$

Wir wollen die NP-Vollständigkeit auch für andere Probleme zeigen.

Satz 5.6 *$L' \in NP$, $L \in NPC$, $L \leq_{pol} L' \Rightarrow L' \in NPC$*

Beweis:
Wir fixieren $\bar{L} \in$ NP $\Rightarrow \bar{L} \leq_{pol} L$ z.z. $\bar{L} \leq_{pol} L'$.$\exists$ polynomielle Reduktion $\bar{f}$ von $\bar{L}$ auf L und polynomielle Reduktion f von L auf L', wir definieren eine polynomielle Reduktion f' von $\bar{L}$ auf L' durch $f'(x) = f(\bar{f}(x))$ z.z. f' ist polynomielle Reduktion von $\bar{L}$ auf L' $\forall x \in \{0,1\}^* : x \in \bar{L} \iff \bar{f}(x) \in L \leftrightarrow f(\bar{f}(x)) \in L' = f'(x)$.
Wir haben gezeigt: $\leq_{pol}$ ist transitiv.
Folgerung: 3-SAT $\in$ NPC; wir haben gezeigt SAT $\leq_{pol}$ 3-SAT

Satz 5.7 *DEC-Clique $\in$ NPC*

Beweis:
Wir zeigen: 3-SAT $\leq_{pol}$ DEC-Clique. Konstruieren polynomielle Reduktion f von 3-SAT auf DEC-Clique: $C = \bigwedge_{i=1}^{m} C_i \to G_C = (V_C, E_C, m_C)$ mit: G_C hat Clique der Größe $m_C \iff C$ ist erfüllbar.
Wir setzen voraus: $\forall i = 1, ..., m$ bestehe C_i aus genau 3 Variablen; $L = L \vee L \vee L$; $L \vee L' = L \vee L' \vee L'$; C ist über $x_1, ..., x_n, \neg x_1, ..., \neg x_n$ definiert.
$G_C : V_C = \underbrace{\{v_{1,1}, v_{1,2}, v_{1,3}, ..., v_{m,1}, v_{m,2}, v_{m,3}\}}_{3*m \; Knoten}$
$E_C = \{(v_{i,k}, v_{j,l});\ i \neq j$ und $L_{i,k} \neq \neg L_{j,l}\}$, d.h. $\exists$ Belegung $b \in \{0,1\}^n$ mit $L_{i,k}(b) = L_{j,l}(b) = 1$, wir schreiben $C = \bigwedge_{i=1}^{m} (L_{i,1} \vee L_{i,2} \vee L_{i,3})$
z.z. G_C hat $m-$Clique $\iff C$ erfüllbar $\iff \exists\, b \in \{0,1\}^n$ mit $\forall i = 1, ..., m$ existiert L_{i,k_i} mit $L_{i,k_i}(b) = 1$ $\exists b \in \{0,1\}^n$ mit $\forall i \neq j \in 1, ..., m$ gilt: $\exists k_i, l_j \in \{1,2,3\}$ mit $L_{i,k_i}(b) = L_{j,l_j}(b) = 1 \iff (v_{i,k_i}, v_{j,l_j}) \in E \iff \{v_{i,k_i}; i = 1, ..., m\}$ ist Clique in G_C.

Generelle Methode zum Zeigen, dass $L \in$ NPC für gegebenes L

Wähle $\bar{L}$, für das $\bar{L} \in$ NPC bekannt ist und konstruiere eine polynomielle Reduktion $\bar{L} \leq_{pol} L$. Wir haben gezeigt:
3-SAT $\in$ NPC (SAT $\leq_{pol}$ 3-SAT); DEC-CLIQUE $\in$ NPC (3-SAT $\leq_{pol}$ DEC-CLIQUE)

Satz 5.8 *RUCKSACK* $\in$ *NPC*

Beweis:
Eingabe: Nutzen: $c_1, \ldots, c_n$ Gewichte: $w_1, \ldots, w_n$ Gewichtsschranke: w
Nutzenschranke: c
Frage: $\exists\ I \leq \{1, \ldots, n\}\quad c(I) \geq c$ und $w(I) \leq w$; $\quad c(I) = \sum\limits_{i \in I} c_i \quad w(I) = \sum\limits_{i \in I} w_i$
Wir konstruieren eine polynomielle Reduktion von 3-SAT auf RUCKSACK.
Fixieren eine beliebige 3-SAT-Formel $C = \bigwedge\limits_{i=1}^{m} C_i; C_i = L_{i,1} \vee L_{i,2} \vee L_{i,3}$;
$L_{i,j} \in \{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\} \forall i = 1, \ldots, m \quad j = 1, 2, 3$
Wir ordnen eine spezielle Instanz zu mit $\forall i = 1, \ldots, n \quad c_i = w_i$ und $c = w$
Frage: $\exists\ I \subseteq \{1, \ldots, n\} \quad w(I) = c(I) = w = c$?
Wir ordnen C Zahlen zu: $a_1 \ldots a_n b_1 \ldots b_n c_1 \ldots c_n d_1 \ldots d_n A$
Frage: $\exists\ I, I' \subseteq \{1, \ldots, n\} \quad J, J' \subseteq \{1, \ldots, m\}$
$\sum\limits_{i \in I} a_i + \sum\limits_{i' \in I'} b_{i'} + \sum\limits_{j \in J} c_j + \sum\limits_{j' \in J'} d_j = A$
Definition: Wir geben $a_i b_i c_j d_j, A$ als Dezimalzahlen an, $A = (\underbrace{44 \ldots 4}_{m} \underbrace{11 \ldots 1}_{n})$;
alle Zahlen sind $(n+m)$-stellig.
$\forall i = 1, \ldots, m \quad a_i = (a_1^i \ldots a_m^i 0 \ldots 0 \underbrace{1}_{i} 0 \ldots 0)$; a_j^i =Anzahl des Vorkommens von x_i in Klausel c_j
$b_i = (b_1^i \ldots b_m^i 0 \ldots 0 \underbrace{1}_{i} 0 \ldots 0) \qquad d_j = 2c_j$
Wir müssen zeigen: C erfüllbar $\Leftrightarrow [a_i b_i c_j d_j A] \in$ RUCKSACK; wir versuchen I, I', J, J' so zu wählen, dass $\sum\limits_{i \in I} a_i + \sum\limits_{i' \in I'} b_{i'} + \sum\limits_{j \in J} c_j + \sum\limits_{j' \in J'} d_j = A$
Beobachtung 1:
Wir müssen $\forall i = 1, \ldots, n$ genau ein Element aus $\{a_i, b_i\}$ hinzunehmen (sonst ist der Suffix $(\ldots 11 \ldots 11)$ von A nicht erreichbar) $\Rightarrow$ wir entscheiden uns für eine Belegung $\beta = (\beta_1, \ldots, \beta_n) \in \{0,1\}^n$ von C.

$$\beta_i = \begin{cases} 1 & \text{falls } a_i \text{ genommen} \\ 0 & \text{falls } b_i \text{ genommen} \end{cases}$$

$I(\beta) = \{i, \beta_i = 1\} \qquad I'(\beta) = \{i, \beta_i = 0\}$
$\sum\limits_{i \in I(\beta)} a_i + \sum\limits_{i' \in I'} b_i = (s_1(\beta) \ldots s_m(\beta) \underbrace{11 \ldots 1}_{n}) = s(\beta) \qquad s_j(\beta) = \#\{i, \beta_i = 1 \text{ und } x_i \in C_j\} + \#\{i', \beta_{i'} = 0 \text{ und } \neg x_{i'} \in C_j\} \Rightarrow s_j(\beta) = \#\{k, L_{jk}(\beta) = 1\} \in \{0,1,2,3\}$
$\Rightarrow$ Beobachtung: $C(\beta) = 1 \iff s_j(\beta) > 0 \quad \forall j = 1, \ldots, m \qquad C(\beta) = 0 \iff \exists j\ s_j(\beta) = 0$
Schritt 2: Wir wählen J, J' so dass $s(\beta) + \sum\limits_{j \in J} c_j + \sum\limits_{j' \in J'} d_{j'} = A$. Wir fixieren j beliebig. Unterschiedliche Fälle:

1. $s_j(\beta) = 3 \Rightarrow j \in J, j \in J' (3 + 1 = 4)$

2. $s_j(\beta) = 2 \Rightarrow j \notin J, j \in J'(2+2=4)$

3. $s_j(\beta) = 1 \Rightarrow j \in J, j \in J'(1+1+2=4)$

4. $s_j(\beta) = 0 \Rightarrow$ es besteht keine Möglichkeit auf 4 zu kommen

$\Rightarrow \exists J, J' \subseteq \{1, \ldots, n\}$ mit $s(\beta) + \sum_{j \in J} c_j + \sum_{j' \in J'} d_{j'} = A$ gdw. $c(\beta) = 1$
$\Rightarrow [a_i b_i c_j d_j A] \in$ RUCKSACK $\Leftrightarrow C \in$ 3-SAT
Die Reduktion ist in Polynomialzeit durchführbar.
Beobachtung 2:
Wir haben nicht nur 3-SAT $\leq_{pol}$ RP ($\equiv$ RUCKSACK) gezeigt, sondern sogar 3-SAT $\leq_{pol}$ RP*.
RP* ist „spezielles RP“:
Eingabe: $a_1, \ldots, a_n, A \quad (a_1, \ldots, a_n, A) \in$ RP* $\Leftrightarrow \exists I \subseteq \{1, \ldots, n\} \quad \sum_{i \in I} a_i = A$

Satz 5.9 *PARTITION $\in$ NPC*

Beweis:
Eingabe: $a_1, \ldots, a_i \in N$
Frage: $\exists I \leq \{1, \ldots, n\} \quad \sum_{i \in I} a_i = \sum_{i \notin I} a_i$

Bemerkung: PARTITION ist Unterproblem von RP*. $(a_1, \ldots, a_n, \frac{\sum_{i=1}^{n} a_i}{2})$
(Bedingung: $\sum_{i=1}^{n} a_i$ ist gerade)
Beweis: Wir zeigen RP* $\leq_{pol}$ PARTITION. Wir konstruieren eine polynomielle Reduktion, $(a_1, \ldots, a_n, A)$ ist Instanz für RP*, $(a_1, \ldots, a_n, A) \rightarrow (a_1, \ldots, a_n, \underbrace{S - A + 1}_{a_{n+1}}, \underbrace{A + 1}_{a_{n+2}}) \in$ PARTITION. Wir versuchen nun $I \subseteq \{1, \ldots, n+2\}$ zu konstruieren mit $\sum_{i \in I} a_i = S + 1$.
Beobachtung:
Es kann nicht $\{n+1, n+2\} \subseteq I$ gelten, da $(n+1)+(n+2) = S+2 \Rightarrow \{n+1, n+2\} \cap I \neq \emptyset$, da sonst $\{n+1, n+2\} \subseteq \bar{I} = \{1, \ldots, n+2\} \setminus I \Rightarrow |\{n+1, n+2\} \cap I| = 1$
OBdA: $\{n+1\} \in I$ (ansonsten betrachte $\bar{I}$ statt I); $I = \{n+1\} \cup I', I' \subseteq \{1, \ldots, n\}$.
Es gilt: $\sum_{i \in I} a_i = S+1 \Leftrightarrow \sum_{i \in I'} a_i + S - A + 1 = S + 1 \Leftrightarrow \sum_{i' \in I'} a_{i'} = S + 1 - (S - A + 1) = A \Rightarrow (a_1, \ldots, a_{n+2}) \in$ PARTITION $\Leftrightarrow \exists I' \subseteq \{1, \ldots, n\}$
$\sum_{i \in I'} a_i = A \Longleftrightarrow (a_1, \ldots, a_n, A) \in$ RP*
Die Transformation läuft in polynomieller Zeit.
Wir wollen zeigen: TSP $\in$ NPC; dazu zeigen wir DHC $\in$ NPC, dann DHC $\leq_{pol}$ HC.
Wir wissen: HC $\leq_{pol}$ TSP $\Rightarrow$ DHC, HC, TSP $\subseteq$ NPC.

Directed Hamiltionian Circuit (DHC)

Eingabe: gerichteter Graph $G = (V, E); \quad V = \{v_1, \ldots, v_n\}$
Frage: $\exists \Pi \in \Sigma_n \{(v_{\Pi(1)}, v_{\Pi(2)}), (v_{\Pi(2)}, v_{\Pi(3)}), \ldots, (v_{\Pi(n-1)}, v_{\Pi(n)}), (v_{\Pi(n)}, v_{\Pi(1)})\} \subseteq E$ gerichteter HC (DHC)?

Satz 5.10 *DHC $\in$ NPC*

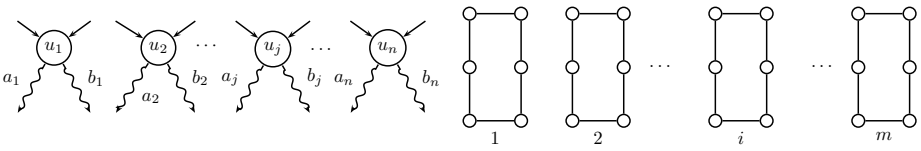
Beweis:
Wir zeigen 3-SAT $\leq_{pol}$ DHC
Wir konstruieren eine polynomielle Reduktion von 3-SAT auf DHC:
$C = \bigwedge_{j=1}^{m} C_j \Rightarrow G(C) = (V(C), E(C))$; C erfüllbar $\Longleftrightarrow G(C) \in$ DHC

Satz 5.11 *3-SAT $\leq_{pol}$ DHC*

Beweis:
Wir ordnen jeder 3-CNF Formel $C = \bigwedge_{i=1}^{m} C_j$ über $\{x_1, \ldots, x_n, \neg x_1, \ldots, \neg x_n\}$ einen gerichteten Graphen $G_C = (V_C, E_C)$ zu mit

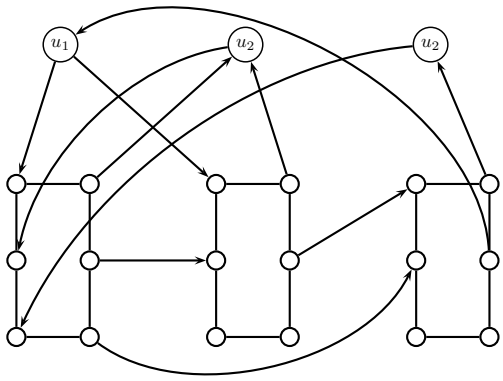
1. C erfüllbar $\Leftrightarrow G_C \in DHC$
2. $C \Rightarrow G_C \in$ PTIME (offensichtlich)

$j = 1, \ldots, m \quad C_j = L_{1,j} \vee L_{2,j} \vee L_{3,j}$, wobei die Literale in einer geordneten Reihenfolge bezüglich $(x_1, \neg x_1, x_2, \neg x_2, \ldots, x_n, \neg x_n)$ sind.
Beispiel:
$(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2)(x_2 \vee \neg x_3)$, wir schreiben $(x_1 \vee x_2 \vee \neg x_3)(\neg x_1 \vee x_2 \vee x_2)(x_2 \vee \neg x_3 \vee \neg x_3)$ $\quad G_C : V_C = \{v_1, \ldots, v_n\} \quad u = \bigcup_{j=1}^{m} \{v_{1,j}, v_{2,j}, v_{3,j}, w_{1,j}, w_{2,j}, w_{3,j}\}$
E_C : Für alle $i = 1, \ldots, n \quad outdeg(u_i) = indeg(u_i) = 2$

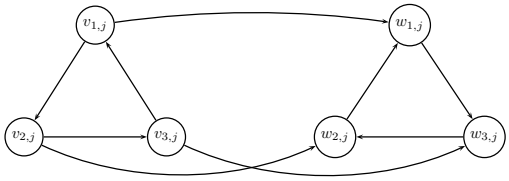


- Die Komponenten $1, \ldots, m$ haben 3 Eingangsknoten $v_{1,j}, v_{2,j}, v_{3,j}$ und 3 Ausgangsknoten $w_{1,j}, w_{2,j}, w_{3,j}$. $\forall k = 1, 2, 3$ und $i = 1, \ldots, n$ und $j = 1, \ldots, m$ heißt: $v_{k,j}$ x_i-Eingang bzw. $\neg x_i-$Eingang falls $L_{k,j} = x_i$ bzw. $L_{k,j} = \neg x_i$; entsprechend $w_{k,j}$ x_i-Ausgang bzw. $\neg x_i-$Ausgang.
- Für alle $i = 1, \ldots, n$ gelten: Die Kante a_i zeigt auf den x_i-Eingang von Komponente $min(x_i)$, wobei $min(x_i) = min\{j; C_j$ enthält $x_i\}$. Die Kante b_i zeigt auf den $\neg x_i-$Eingang von Komponente $min(\neg x_i)$.

Für alle $i = 1, \ldots, n$ gilt: Wenn $j_1 < j_2 < \ldots < j_s$ die Klauseln bezeichnen, die x_i enthalten, so geht für alle $t = 1, \ldots, s-1$ eine Kante vom x_i-Ausgang von Komponente j_t zum x_i-Eingang von Komponente j_{t+1}. Außerdem geht eine Kante vom x_i-Ausgang von Komponente j_s zu $u_{i+1 \bmod n}$. Gleiches gilt für $\neg x_i$. $\Rightarrow$ Für $i = 1, \ldots, n$ existiert ein Weg zu x_i (beziehungsweise ein Weg zu $\neg x_i$), der in u_i startet, über a_i (beziehungsweise b_i) alle Komponenten durchläuft die x_i beziehungsweise $\neg x_i$ enthalten und in $u_{i+1 \bmod n}$ endet.
Konvention:
Enthält C_j x_i (bzw. $\neg x_i$) mehr als einmal, so ist $v_{k,j}$ mit $k = min\{k'; L_{k',j} = x_i\}$ der x_i-Eingang bzw. $w_{k,j}$ der x_i-Ausgang von Komponente j.
Zum Beispiel:

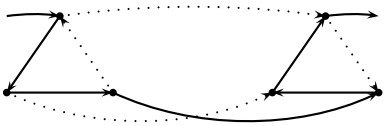


Die Komponente j:

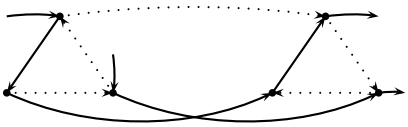


Es gibt drei Möglichkeiten, wie ein DHC die Komponente j durchläuft:

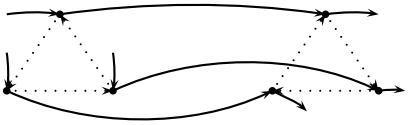
1.



2.



3.



Wichtig: Andere Möglichkeiten gibt es nicht:
Wenn der DHC über $v_{k,j}$ „reingeht“dann geht er stets über $w_{k,j}$ wieder raus.
Zu zeigen: C erfüllbar $\Leftrightarrow$ G_C hat DHC $\Leftrightarrow \exists\, c = (c_1, \ldots, c_n) \in \{0,1\}^n$
$C_j(c) = 1\ \forall j = 1, \ldots, m \Longleftrightarrow \forall j = 1, \ldots, m\ \exists\, i \in \{1, \ldots, n\}$
$c_i = 1 \Rightarrow C_j$ enthält x_i $\quad$ $c_i = 0 \Rightarrow C_j$ enthält $\neg x_i$.
Wir versuchen einen DHC in G_C zu konstruieren:
Beobachtung:
Jeder Kreis, der $u_1, \ldots, u_n$ enthält, enthält für alle $i = 1, \ldots, n$ entweder a_i oder $b_i \Rightarrow$ Kreis definiert eine Belegung $c = (c_1, \ldots, c_n)$ mit

$$c_i = \begin{cases} 0 & \text{falls } b_i \in \text{Kreis} \\ 1 & \text{falls } a_i \in \text{Kreis} \end{cases}$$

$\Rightarrow$ Kreis besteht aus n Teilwegen entsprechend den Literalen x_1 bzw. $\neg x_1, \ldots, x_n$ bzw. $\neg x_n$ je nachdem ob $c_i = 0$ (dann $\neg x_i$) oder $c_i = 1$ (dann x_i).
$\Rightarrow$ Falls $C_j(c) = 0$ für ein $j \in \{1, \ldots, m\} \Rightarrow$ Komponente j wird vom Kreis nicht berührt $\Rightarrow$ Kreis ist kein DHC
$\Rightarrow$ Falls $\forall j = 1, \ldots, m \quad C_j(c) = 1 : \forall j = 1, \ldots, m$; Kreis berührt Komponente j:

- einmal, falls 1 Literal in C_j durch c erfüllt
- zweimal, falls 2 Literale in C_j durch c erfüllt
- dreimal, falls 3 Literale in C_j durch c erfüllt

DHC $\leq_{pol}$ HC

5.3 Schlussbemerkung

5.3.1 Wie zeigt man $L \in$ NPC für gegebenes L?

In der Praxis: Suche in Literatur ein Problem L', für das $L' \in$ NPC bereits gezeigt wurde und entweder $L = L'$ oder man zeigt $L' \leq_{pol} L$. Wir benötigen eine Sammlung möglichst vieler NPC-Probleme $\Rightarrow$ Gerey/Johnson: erste große Sammlung von NPC-Problemen. Heute sind tausende von NPC-Problemen bekannt.
Wir benötigen ein ähnliches Konzept für Probleme, die keine Sprachen sind:
Beispiele hierfür sind TSP, MAXSAT, MAXCLIQUE.

Definition 5.17

- *Ein Problem* Π *heißt NP-schwer falls aus* $P \neq NP$ *folgt:* $\Pi \notin PTIME$ *(bzw. aus* $\Pi \in PTIME$ *folgt P=NP)*
- *Ein Problem* Π *heißt NP-leicht falls aus P=NP folgt* $\Pi \in PTIME$
- *Ein Problem* Π *heißt NP-äquivalent falls es NP-leicht und NP-schwer ist*

Kapitel 6

Erzeugung von Sprachen durch Grammatiken (Formale Sprachen)

6.1 Grammatiken und Chomsky Hierarchie

Definition 6.1 *G = (V,T,S,R)*

V: *Endliches Alphabet von Hilfsvariablen*

T: *Endliches Alphabet von Terminalzeichen*

S: $S \in V$ *Startsymbol*

R: *Endliche Menge von Regeln über* $V \cup T$ *(*$V \cap T = \emptyset$*)*

Regeln

Regeln haben die Gestalt (u,v) (auch geschrieben als u→v) $u \in (V \cup T)^+, v \in (V \cup T)^*$ $\boxed{\Sigma^+ = \Sigma^* \backslash \{\epsilon\}}$

Beispiel

$G = \{\{S, A\}\}, \{0, 1\}, S, \{S \to 0S, S \to 1A, A \to 0A, A \to 1S, A \to \epsilon\}$

Lemma 6.1 *Grammatiken erzeugen Sprachen L(G)*$\subseteq T^*$

Allgemeine Regel

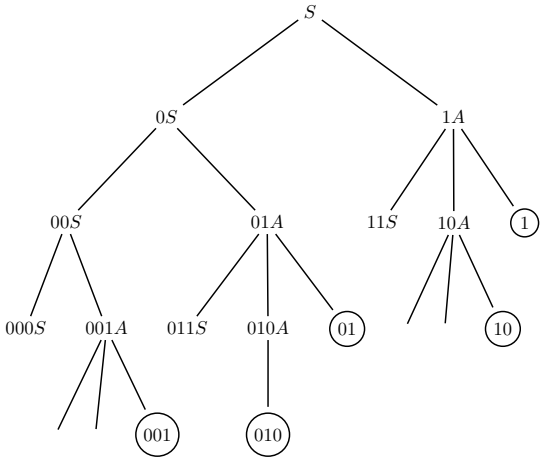
- Ein Wort $x \in (V \cup T)^*$ aus einem Wort $w \in (V \cup T)^+$ in einem Schritt mittels G abzuleiten (Schreibweise $w \underset{G}{\rightarrow} x$) heißt:
  Suche Teilwort u in Wort w (d.h. w= w'uw") und Regel $(u, v) \in R_G$. Ersetze u durch v (d.h. x=w'vw")

- Wir schreiben $w \overset{k}{\underset{G}{\rightarrow}} x$ falls x aus w in k ($k \geq 1$) Schritten abgeleitet werden kann, d.h. $\exists\, w^{(0)}, \ldots, w^{(k)} \in (V \cup T)^*$ mit $w^{(0)} = w,\ w^{(k)} \underset{G}{\rightarrow} w^{(i+1)}$
  $i = 0 \ldots k-1,\ w^{(k)} = x$
- Wir schreiben $w \overset{*}{\underset{G}{\rightarrow}} x$ falls $\exists\, k \geq 1$ mit $w \overset{*}{\underset{G}{\rightarrow}} x$

$\Longrightarrow$ $\boxed{L(G) = \{x \in T^*;\ S \overset{*}{\underset{G}{\rightarrow}} x\}}$ ist die von G erzeugt Sprache

Beispiele

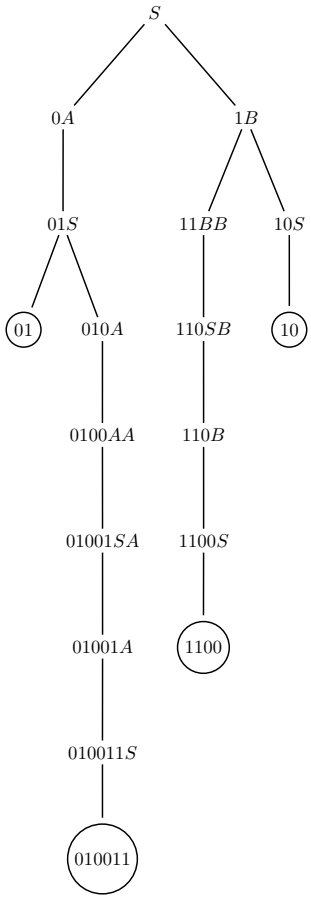
1. $G = \{\{S, A\}\}, \{0,1\}, S, \{S \to 0S,\ S \to 1A,\ A \to 0A,\ A \to 1S,\ A \to \epsilon\}\}$



Beobachtung 6.1
Man kann zwei Sorten von Worten mit Hilfsvariablen ableiten

- $bS \quad b \in \{0,1\}^* \quad b \notin PARITY$
- $b'A \quad b \in \{0,1\}^* \quad b \in PARITY$

2. G mit $R = \{S \to \epsilon,\ S \to 0S1\}$
   $L(G) = \{0^n 1^n,\ n \geq 0\}$
3. G mit $R = \{S \to 0A,\ S \to 1B,\ S \to \epsilon,\ A \to 1S,\ A \to 0AA,\ B \to 0S,$
   $B \to 1BB\}$



Beh: $L(G) = \{w \in \{0,1\}^*$, w hat genausoviel '1' wie '0'$\}$

Ziele:

1. Welche Sprachen sind durch Grammatiken erzeugbar?
2. Welche Sprachen sind durch Grammatiken mit beschränkten Einschränkungen erzeugbar?
3. Komplexität des „Compilierungsproblems“: wie schwer ist es für geg. G und $x \in T^*$ zu berechen ob $x \in L(G)$

Definition 6.2 (Spezielle Grammatiken für G=(V,T,S,R))

1. *G heißt kontextsensitiv (oder monoton) falls*
   (a) $\forall\ (u,v) \in R$ *gilt* $|u| \leq |v| \quad (Ausnahme:\ S \to \epsilon)$
   (b) *S kommt nie auf der rechten Regelseite vor*
2. *G heißt kontextfrei falls* $\forall\ (u,v) \in R$ *gilt* $u \in V,\ v \in (V \cup T)^*$ *(Beispiele 1,2,3 sind kontextfrei)*
3. *G heißt rechtslinear, falls alle Regeln von der Gestalt* $A \to aB,\ A \to \epsilon$, *für* $A, B \in V,\ a \in T$ *sind*

6.1.1 Chomsky-Hierarchie

$\mathcal{L}_0 = \{L;\ \exists$ Grammatik G mit $L(G) = L\}$
$\mathcal{L}_1 = \{L;\ \exists$ kontextsensitiv Grammatik G mit $L(G) = L\}$
$\mathcal{L}_2 = \{L;\ \exists$ kontextfreie Grammatik G mit $L(G) = L\}$
$\mathcal{L}_3 = \{L;\ \exists$ rechtslineare Grammatik G mit $L(G) = L\}$

Beobachtung 6.2 *Es gilt:* $\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$

Bemerkung 6.1 (zu kontextfrei und -sensitiv)
$A \to aBc$ bedeutet A kann an jeder beliebigen Stelle durch aBc ersetzt werden
$xAy \to xaBcy$ bedeutet A kann nur dann durch aBc ersetzt werden, wenn der Kontext xy gegeben ist

Satz 6.1
Es gilt $\mathcal{L}_0 = ENUM = \{L,\ L$ rekursiv aufzählbar$\}$
d.h. $\exists$ TM M mit M(x)=1 $\iff x \in L$

Beweis:

1. $\mathcal{L}_0 \subseteq ENUM$
   Fixiere $L \in \mathcal{L}_0$ d.h. eine Grammatik G mit L(G)=L, G=(V,T,S,R) und konstruiere einen Algorithmus der L aufzählt
   Idee: $L_k(G) = \{w;\ S \overset{k}{\underset{G}{\to}} w\} \Longrightarrow L = \bigcup_{k\geq 1}(L_k(G) \cap T^*)$
   Algorithmus:
   Erzeuge $L_1(G), L_2(G), L_3(G), \ldots$ und stoppe mit 1 falls Eingabe $x \in T^*$ erzeugt wurde
   Bemerkung: $L_k(G)$ lässt sich aus $L_{k-1}(G)$ berechnen, indem alle Werte erzeugt werden, die sich direkt aus Werten aus $L_{k-1}(G)$ ableiten lassen.

2. $ENUM \subseteq \mathcal{L}_0$
   Fixes beliebiges $L \in ENUM;\ L \subseteq T^* \Longrightarrow \exists$ TM M, die L aufzählt
   $\Longrightarrow \exists$ „spezielle“TM M, die L aufzählt

   - M ist 1-Band TM über $T \cup \{\#, \$\}$
   - M hat genau 1 akzeptierenden Stoppzustand q^*
   - Anfangskonfiguration zu $x = (x_1, \ldots, x_n) \in T^*$ *ist* $q_0 x_1 \ldots x_n \$$
   - M betritt nie Band rechts von \$
   - Akzeptierende Stoppkonfig ist stets Zustand q^* + leeres Band

   (Jede TM M kann in derartige „spezielle“TM umgewandelt werden) Idee: Wir übersetzen die Zustandsüberführungsfunktion $\delta : \Sigma \times Q \Rightarrow \Sigma \times MOVE \times Q$ von M in eine Grammatik G für L; $\Sigma = \{0, 1, \#, \$\}$.
   Konstruktion von G : G simuliert M „von hinten nach vorne“
   $G = (V, T, S, R_G)$, wobei $V = Q \cup \{S\}$; $\quad T = \Sigma = \{0, 1, \#, \$\}$.
   $R_G =$

Gruppe 1: $S \to q^* \qquad q^* \to \#q^* \qquad q^* \to q * \#$ erzeugt Stoppkonfiguration $\# \ldots \#q^*\# \ldots \#$
Gruppe 2: $\forall a, a', b, b' \in \Sigma \quad \forall q, q' \in Q$

- $aq \to q'a'$ falls $\delta(a', q') = (a, R, q)$
- $qba \to bq'a'$ falls $\delta(a', q') = (a, L, q)$
- $qa \to q'a'$ falls $\delta(a', q') = (a, N, q)$

simuliert eine Berechnung von M nach hinten.
Gruppe 3: Bereinigt die Startkonfiguration $q_0 w_1 \ldots w_n$ von überflüssigen Symbolen $q_0 a \to a q_0 \quad \forall\, a \in \{0, 1\} \quad \#q_0 \to q_0 \quad q_0\$ \to \epsilon$
Behauptung: $L(G) = L$
Beweis:
Beobachtung 1: Mittels Gruppe 1+2 können nur solche Worte $x \in (V \cup T)^*$ abgeleitet werden, die solchen Konfigurationen von M entsprechen, die zu einer akzeptierenden Stoppkonfiguration führen. $\Rightarrow$ Es können nur Konfigurationen $k_0(w)$ von solchen w abgeleitet werden, für die $M(w) = 1$ (d.h. $w \in L$).
Beobachtung 2: Es können nur solche Worte $x \in (V \cup T)^*$ zu einem $w \in \{0, 1\}^*$ abgeleitet werden, die der Startkonfiguration von M entsprechen.
Wir betrachten $\mathcal{L}_3 = \{L, \exists$ rechtslineare Grammatik G, $L(G) = L\}$

Satz 6.2 *$\mathcal{L}_3 = REG$*

Beweis:
z.z. $L \in \mathcal{L}_3 \Leftrightarrow \exists$ NFA für L
$L \in \mathcal{L}_3 \Rightarrow \exists$ rechtslineare Grammatik $G = (V, T, S, R_G)$ für L, konstruiere NFA $A = (Q, q_0, F, \delta)$ für L, setze $Q = V \quad q_0 = S \quad \delta(a, A) = \{B$, mit $A \to aB \in R_G\} \quad A \in V \quad a \in T \quad F = \{A; A \to \epsilon \in R_G\}$. Es ist leicht zu sehen dass $L(G) = L = L(A)$
$\mathcal{L}_1 = \{\mathcal{L}; \exists$ kontextsensitive Grammatik $G = (V, T, S, R_G)$ mit $R_G = \{u \to v; |v| \geq |u|; s \neq$, einzige Ausnahme: $s \to \epsilon$ ist erlaubt.$\}\}$

Satz 6.3
$\mathcal{L}_1$ =NSPACE(n), wobei NSPACE(n) die Menge der Sprachen, die durch eine NTM M mit n Speicherzellen berechenbar sind ist.

Bemerkung 6.2 *NSPACE(n) enthält NP-vollständige Probleme (z.B. SAT).*

Zusammenfassung: $\mathcal{L}_0, \mathcal{L}_1, \mathcal{L}_3$ sind als passende Klasse für Programmiersprachen nicht geeignet.

6.2 Kontextfreie Sprachen

Definition 6.3
$\mathcal{L}_2 = \{L; L$ kontextfrei$\} = \{L; L = L(G); G$ kontextfrei$\}$
(alle Regeln haben die Gestalt $A \to a;\ A \in V; a \in (V \cup T)^*$).

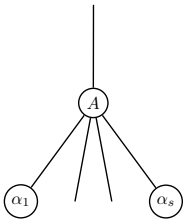
Beispiele

1. $S \to 0S1, S \to \epsilon$ erzeugt $L = \{0^n 1^n, n \geq 0\} \notin \text{REG}$ ($\Rightarrow \mathcal{L}_3 \subset \mathcal{L}_2$)

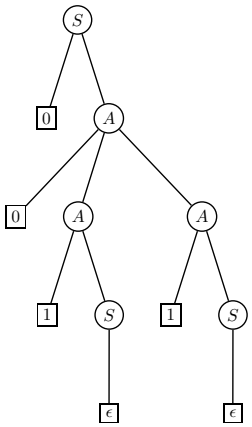
2. beliebiges Alphabet $T : S \to aSa, \forall\, a \in T, S \to \epsilon$ erzeugt Sprache PALINDROME=$\{w \in T^*, \overrightarrow{w}{=}\overleftarrow{w}\}$

3. $S \to 0A, S \to 1B, S \to \epsilon, A \to 0AA, A \to 1S, B \to 1BB, B \to 0S$ erzeugt die Sprache der Worte $w = w_1 \dots w_n \in \{0,1\}^*$ mit n gerade und $\sum_{i=1}^{n} w_i = \frac{n}{2}$
   gegeben: G kontextfrei; wie sehen Ableitungen von Worten mittels G aus?
   Veranschaulichung: Ableitungsbaum, an dessen Wurzel S steht, dessen innere Knoten mit Symbolen $A \in V$ markiert sind und an dessen Blättern Symbole $a \in T \cup \{S\}$ stehen und deren Nachfolgerelation den Ableitungsregeln entspricht.

   Mit $A \to \alpha_1 \dots \alpha_s \in R_G$

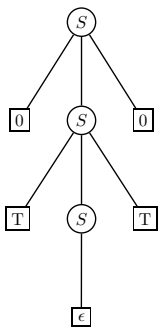


4. $S \to 0A \to 00AA \to 001SA \to 001A \to 0011S \to 0011$



   Die Blätter von links nach rechts ergeben das abgeleitete Wort w.

5. $S \to OSO \to OTSTO \to OTTO$



Definition 6.4
Eine Ableitung heißt Links- (oder Rechtsableitung) wenn stets bezüglich des linkesten (oder des rechtesten) Hilfssymbols abgeleitet wird.

Beobachtung 6.3
G kontextfrei, $L = L(G)$ so läßt sich jedes $w \in L$ durch Linksableitung (bzw. Rechtsableitung) erzeugen.

6.2.1 Die Chomsky-Normalform (CNF) für kontextfreie Grammatiken

Definition 6.5
G heißt in CNF falls alle Regeln die Gestalt $A \to BC$ mit $A, B, C \in V$ oder $A \to a$ mit $A \in V, a \in T$ haben.

Bemerkung 6.3
CNF-Grammatiken können nicht das leere Wort erzeugen. Falls das leere Wort erzeugt werden soll müssen Regeln $\bar{S} \to \epsilon, \bar{S} \to S$ hinzugefügt werden, wobei $\bar{S}$ ein neues Startsymbol ist.

Satz 6.4 *Sei $L \in \mathcal{L}_2, \epsilon \notin L \Rightarrow \exists$ CNF-Grammatik für L.*

Beweis: Zu zeigen: Jede $\mathcal{L}_2-$Grammatik kann effizient in eine äquivalente CNF-Grammatik umgewandelt werden.
gegeben: $G = (V, T, S, R_G)$ kontextfrei, $\epsilon \notin L(G)$
Stufe 1: Es gibt Regeln $\in R_G$, bei denen auf der rechten Seite sowohl Terminal- als auch Hilfssymbole vorkommen. Wir ersetzen in allen diesen Regeln alle vorkommenden Terminalsymbole $a \in T$ durch ein neues Hilfssymbol T_a und fügen Regeln $T_a \to a$ für alle $a \in T$ hinzu. $G \to G_1$; offensichtlich gilt $L(G) = L(G_1)$
Stufe 2: In R_{R1} existieren Regeln $A \to \alpha$ mit $\alpha \in V^+$ und $|a| > 2$. Wir verwandeln R_{G1} in R_{G2} nach folgender Regel: $A \to B_1 B_2 \dots B_s \quad A_i B_i \in V \quad i \in 1, \dots, s$, wir nehmen zusätzliche Hilfssymbole $C_1, \dots, C_{s-2}$ und ersetzen diese Regel durch $A \to B_1 C_1, C_1 \to B_2 C_2, C_2 \to B_3 C_3, \dots, C_{s-2} \to B_{s-1} B_s$ Ł$(G) = L(G_1) = L(G_2)$
Bemerkung: Wir haben in R_{G2} noch 2 Typen von Regeln, die die Vorgaben der CNF verletzen: $A \to \epsilon \qquad A \to B$
Stufe 3: Eliminierung der $A \to \epsilon$ Regeln ist nicht einfach möglich, siehe folgendes Beispiel:
$S \to AB \qquad A \to a, A \to \epsilon, B \to b \equiv L = \{b, ab\}$
aber: $A \to a, B \to b \equiv L = \{a, b\}$
Stufe 3.1: Wir markieren alle $A \in V$ mit $A \overset{*}{\to} \epsilon$
Beobachtung: Es gilt $A \overset{*}{\to} \epsilon \Leftrightarrow A \to \epsilon$ ist Regel oder $\exists$ Regel $A \to BC$ mit $B \overset{*}{\to} \epsilon$ und $C \overset{*}{\to} \epsilon \Rightarrow$ Markierungsalgorithmus, bestehend aus höchstens $|V|$ Runden:
Runde 1: Markiere alle A mit $A \to \epsilon$ Regel
Runde i: Markiere alle $A \in V$ für die eine Regel $A \to BC$ existiert mit B, C bereits markiert.
Stufe 3.2: Wir fügen für alle Regeln $A \to BC$ mit $B \overset{*}{\to} \epsilon$ die Regel $A \to C$ hinzu und falls $C \overset{*}{\to} \epsilon$ die Regel $A \to B$ hinzu und streichen alle Regeln $V \to \epsilon$.

$\Rightarrow S \to AB, A \to a, B \to b, S \to B$
Stufe 4: Eliminiere Regeln der Gestalt $A \to B$
Wir definieren Graphen, Knotenmenge V, Kantenmenge $\{(A,B); A \to B \text{ ist Regel}\}$
Stufe 4.1: Für jeden Zyklus $B_1 \to B_2 \to B_3 \to \ldots \to B_t \to B_1$ in diesem Graphen ersetzen wir B_i durch B_1 für $i = 2, \ldots, t$. Begründung: $\forall w \in T^*$ gilt: $B_i \overset{*}{\to} w \Leftrightarrow B_j \overset{*}{\to} w$
(Bemerkung:
Zyklen können durch iterierte Tiefensuche-Durchläufe identifiziert werden.)
$\Rightarrow$ azyklischer Graph über V
Wir nummerieren die Hilfssymbole so, dass eine topologische Knotenmarkierung auf diesem Graphen entsteht, d.h. $V = \{A_1, \ldots, A_l\}$, es gilt: Falls Regel $A_i \to A_j$ existiert, dann ist $i < j$.
for i=l downto 1 do:
Falls $A_i \to A_j$ Regel, so fügen wir allen Regeln $A_j \to \alpha$ die Regel $A_i \to \alpha$ hinzu und streichen die Regel $A_i \to A_j$.
$\Rightarrow$ Grammatik in CNF

Beispiel

(nichtleere) Palindrome gerader Länge:
$S \to 0A0, S \to 1A1, A \to 1A1, A \to 0A0, A \to \epsilon$
Stufe 1: $S \to T_0AT_0, S \to T_1AT_1, A \to T_1AT_1, A \to T_0AT_0, A \to \epsilon, T_0 \to 0, T_1 \to 1$
Stufe 2: $S \to T_0C_1, S \to T_1D_1 \ldots$
$C_1 \to AT_0, D_1 \to AT_1$
Stufe 3: Wir streichen $A \to \epsilon$, fügen $C_1 \to T_0, D_1 \to T_1 \ldots$ ein.
Stufe 4: $\ldots$

Bemerkung 6.4
$\Rightarrow \mathcal{L}_2 \subseteq \mathcal{L}_1$ *CNF-Grammatik* $+ \bar{S} \to \epsilon, \bar{S} \to S$ *ist kontextsensitiv.*

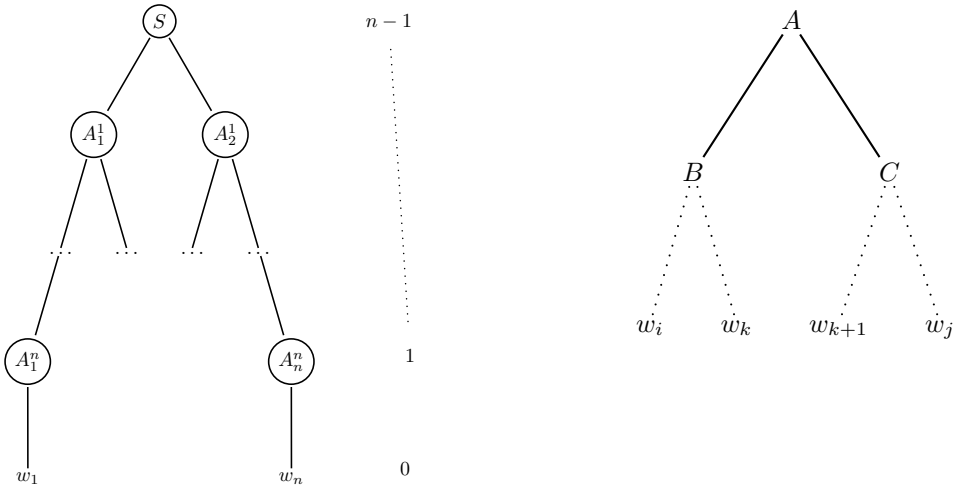
Satz 6.5
Für jede kontextfreie Sprache L gilt: $\exists$ Algorithmus, so dass das Problem $w \in L$ in der Laufzeit $O(|w|^3)$ entschieden werden kann. Folgerung: $\mathcal{L}_2 \subseteq P$

Beweis:
Algorithmus von Crocke, Younger, Kasami (1967)
Wir fixieren ein beliebiges $L \in \mathcal{L}_2$ und eine CNF-Grammatik G mit $L = L(G)$
Eingabe: $w = (w_1, \ldots, w_n) \in T^* \quad G = (V, T, S, R_G)$
Definition: $\forall 1 \leq i \leq j \leq n \quad V_{i,j}(w) = \{A \in V; A \overset{*}{\to} w_i \ldots w_j\}$
Beobachtung:

1. $w \in L \Leftrightarrow S \in V_{1,n}(w)$
2. $\forall\, i = 1, \ldots, n \quad V_{i,i}(w) = \{A; A \to w_i \in R_G\}$

Der Algorithmus berechnet die $V_{i,j}(w)$ in $s = 0, \ldots, n-1$ Stufen, wobei in Stufe 0 die $V_{i,i}(w)$ berechnet werden $(i = 1, \ldots, n)$ und in Stufe $s (s > 0)$ die $V_{i,j}(w)$ berechnet werden mit $j - s = s$, und zwar nach folgender Regel:

(*) $A \in V_{i,j}(w) \Leftrightarrow \exists k, i \leq k < j$ und Regel $A \to BC$ in R_G mit $B \in V_{i,k}(w), C \in V_{k+1,j}(w)$
Veranschaulichung: $w \in L \Leftrightarrow \exists$



Wir sind fertig wenn $V_{1,n}(w)$ berechnet ist.
Laufzeit:
n Stufen, in denen höchstens n Mengen berechnet werden. Das Testen der Gleichung (*) kostet $O(n * |G|) \Rightarrow O(n^3 * |G|)$
Wir wissen das $\mathcal{L}_2 \subseteq$ P. Gesucht ist nun eine Methode zum Nachweis, dass gegebene Sprachen $\notin \mathcal{L}_2$ sind. (Wir zeigen $\mathcal{L}_2 \subset$ P, $\mathcal{L}_2 \subset \mathcal{L}_1$)

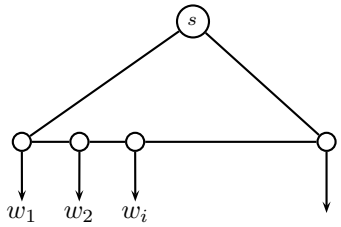
Satz 6.6 (Pumping Lemma für kontextfreie Sprachen)
L kontextfrei $\Rightarrow \exists$ Zahl $n \geq 1$ $\forall w \in L$ mit $|w| \geq n$ existiert eine Zerlegung von w in $w = uvxyz$ mit

1. $1 \leq |vxy| \leq n$ *und*
2. $\forall j \geq 0 \quad uv^j xy^j z \in L$

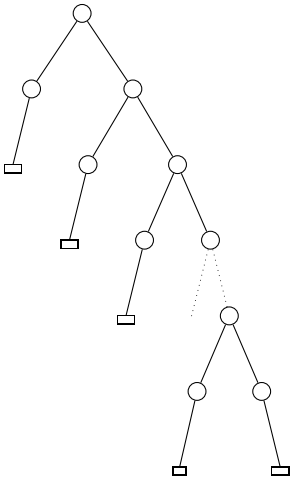
Beispiel:
$L = \{0^n 1^n; n \geq 0\} \notin$ REG $\in \mathcal{L}_2 \quad w \in 0 \ldots 01 \ldots 1$
$\underbrace{0\ldots0}_{u}\underbrace{0\ldots0}_{v}\underbrace{01}_{x}\underbrace{1\ldots1}_{y}\underbrace{1\ldots1}_{z} \quad |u| = |y| \Rightarrow uv^j xy^j z \in L \quad \forall j \geq 0$

Beweis

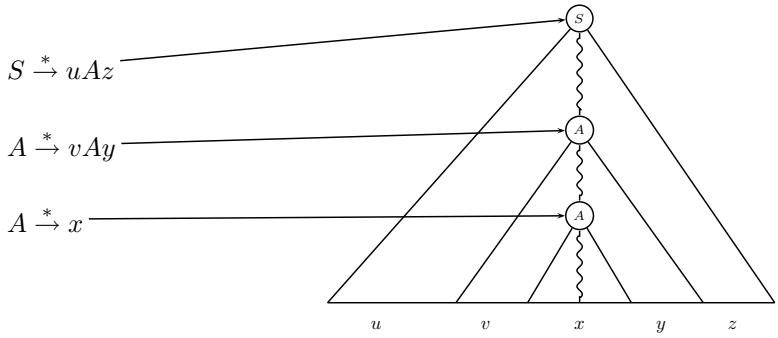
Wir zeigen: $w \in L \Rightarrow \exists w = uvxyz$ und $A \in V \quad G = (V, T, S, R_G)$ CNF-Grammatik für L mit:
$S \overset{*}{\to} uAz, A \overset{*}{\to} vAy, A \overset{*}{\to} x$
Aus dieser Aussage folgt der Beweis, denn:
$j = 0$: $S \overset{*}{\to} uAz \overset{*}{\to} uxz$
$j > 1$: $s \overset{*}{\to} uAz \overset{*}{\to} uvAyz \overset{*}{\to} uvvAyyz \overset{*}{\to} \ldots \overset{*}{\to} uv^j Ay^j z \overset{*}{\to} uv^j xy^j z$
Wir fixieren $n = 2^{|v|+1}$ und fixieren beliebiges $w \in L = L(G) \quad |w| \geq n$
Wir betrachten den Ableitungsbaum für w:



Wir fixieren einen Pfad von der Wurzel zu einem Blatt nach folgender Regel:
Hat der Pfad einen Knoten mit Aus-Grad 2 erreicht, so folgt er dem Teilbaum, der mehr Blätter enthält (bei gleicher Blattzahl egal).
Beispiel:



$\Rightarrow$ Wir bekommen den längsten möglichen Pfad von der Wurzel zu einem Blatt.
Es gilt: Der Baum hat mindestens $2^{|V|+1}$ Blätter $\rightarrow$ Pfad hat mindestens $|V|+1$ innere Knoten.
Wir betrachten das Endstück des Pfads, das genau $|V|+1$ innere Knoten enthält
$\Rightarrow \exists\ A \in V$, so dass A zweimal im Endstück vorkommt.



Folgerung: Die Sprache $L = \{a^i b^i c^i j \quad i \geq 1\} \notin \mathcal{L}_2$ ist auf einer 2-Band TM in Linearzeit berechenbar.
Wir nehmen an: $L \in \mathcal{L}_2 \Rightarrow \exists$ CNF-Grammatik $G = (V, \{a,b,c\}, S, R_G)$ für L, wir fixieren n so, dass $3n \geq 2^{|V|+1} \quad w = \underbrace{a \ldots a}_{n} \underbrace{b \ldots b}_{n} \underbrace{c \ldots c}_{n}$
$L \in \mathcal{L}_2 \Rightarrow \exists\ w = uvxyz$ mit $uv^j xy^j z \in L \quad \forall\ j \geq 0$

Beobachtung 1: $v, y \in \{a^i, b^i, c^i\}$, ansonsten wir ein falsches Muster erzeugt
Beobachtung 2: $\exists$ Buchstabe, der weder in v noch in y vorkommt $\Rightarrow$ beim Aufpumpen wird die Balanciertheit zerstört $\Rightarrow$ Widerspruch!
Beobachtung: L ist durch eine logarithmisch platzbeschränkte 2-Band TM berechenbar $\Rightarrow L \in \mathcal{L}_1 \Rightarrow \mathcal{L}_2 \subset \mathcal{L}_1$
Allgemein gilt: $\mathcal{L}_3 \underbrace{\subset}_{a^n b^n; n \geq 0} \mathcal{L}_2 \underbrace{\subset}_{a^n b^n c^n; n \geq 1} \mathcal{L}_1 \underbrace{\subset}_{Halteproblem} \mathcal{L}_0$ sowie $\mathcal{L}_1 \subseteq \text{REK} \subset \mathcal{L}_0$

6.3 Kellerautomaten und kontextfreie Sprachen

Grammatiken in Greibach-Normalform: G heißt in Greibach NF, falls alle Regeln die Gestalt $A \to a\alpha, \quad \alpha \in V^*$ haben. $G = (V, T, S, R_G) \quad A \in V \quad a \in T$

Satz 6.7 *Jede Sprache $L \subseteq \mathcal{L}_2$ mit $\epsilon \notin L$ hat eine Grammatik in Greibach NF und jede $\mathcal{L}_2$-Grammatik kann effizient in eine Greibach NF transformiert werden.*

Beispiel: $S \to 0A, S \to 1B, A \to 0AA, A \to 1S, B \to 1BB, B \to 0S, B \to 0, A \to 1$ erzeugt alle Worte mit gleich vielen Nullen wie Einsen.

Linksableitungen durch Greibach NF

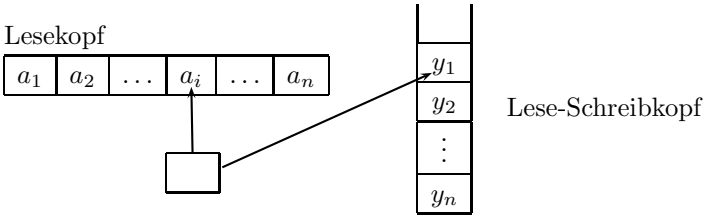
Die Zweichenergebnisse haben die Gestalt $a_1 \dots a_k A_1 \dots A_l$, benutze als nächstes eine Regel der Gestalt $A_1 \to a_{k+1} B_1 \dots B_s$; nächstes Zwischenergebnis ist $a_1 \dots a_{k+1} B_1 \dots B_S A_2 \dots A_l$
$S \to 0A \to 00AA \to 001A \to 0010AA \to 00101SA \to 001011BA \to 0010110A \to 00101101$
Die Ableitung des Wortes $a_1 \dots a_n$ hat n Schritte, nach dem $k-$ten Schritt $a_1 \dots a_k A_1 \dots A_l$
Beobachtung:

1. Eine Linksableitung mittels Greibach NF entspricht der Verwaltung des Suffix an Hilfssymbolen in einem LIFO-Speicher (Last In - First Out) (auch Stack oder Kellerspeicher genannt)
2. Die Berechnung ist inhärent nichtdeterministisch, in jeder Situation $a_1 \dots a_k A_1 \dots A_l$ kann aus allen Regeln mit linker Seite A_1 gewählt werden.

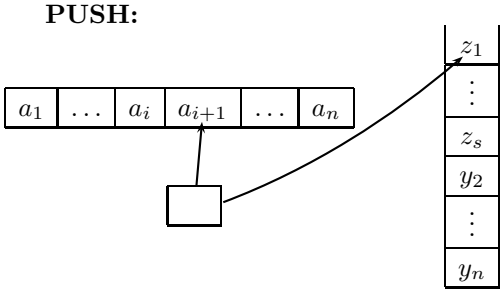
Kellerautomaten

Basisvariante (Kellerautomaten vom Typ 1), englisch: Pushdown-Automata (PDA).
Wir betrachten nichtdeterministische PDAs (NPDAs).
NPDAs vom Typ 1:

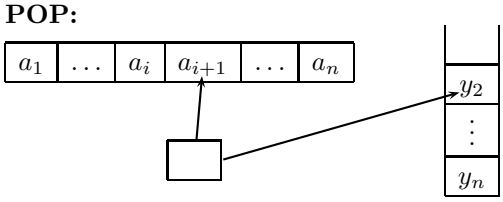
- Eingabeband, auf dem zeichenweise von links nach rechts gelesen wird
- Stack, dessen oberstes Symbol gelesen werden kann und dessen oberstes Symbol durch ein neues Wort überschrieben werden kann.
  Wort $\neq \epsilon$ „Push“ Wort $= \epsilon$ „Pop“



PUSH:



POP:



Formal:
NPDA (Typ 1) ist $N = (T, \Gamma, \Gamma_0, \delta)$, wobei T die Menge der Terminalsymbole, Γ das Kelleralphabet, Γ_0 das Startsymbol und δ die Zustandsüberführungsfunktion bezeichnet.
$\delta : T \times \Gamma \rightarrow \underbrace{2^{\Gamma^*}}_{Teilmenge}$; jedem Paar (a, Y) wird eine Menge von Worten über Γ zugeordnet.
NPDA stoppt bezüglich $a \in \Sigma$ und $\gamma \in \Gamma$ falls $\delta(a, \gamma) = \emptyset$, insbesondere: NPDA stoppt falls der Stack leer ist oder nachdem das letzte Zeichen eingelesen wurde.
gegeben: NPDA A mit $A = \{\Sigma, \Gamma, \Gamma_0, \delta\}$; A akzeptiert $w \in \Sigma^* \Leftrightarrow \exists$ akzeptierende Berechnung von a mit Eingabe w. Eine Berechnung auf w ist akzeptierend wenn sie

1. erst nach Einlesen des letzten Zeichens stoppt und zwar
2. mit leerem Stack

Eine Konfiguration eines NPDAs ist gegeben durch

1. den String der noch einzulesenden Eingabezeichen
2. den aktuellen Stackinhalt

Schreibweise für Konfiguration: $(w, \alpha) \in \Sigma^* \times \Gamma^*$
Für Konfigurationen $(w, \alpha), (w', \alpha')$ schreiben wir $(w, \alpha) \vdash_\alpha (w', \alpha')$ falls (w', α') eine mögliche Nachfolgekonfiguration von (w, α) ist, dies ist für $w = (w_i w_{i+1} \dots w_n)$

und $\alpha = (Y_1 \dots Y_m)$ genau dann der Fall wenn $w' = (w_{i+1} \dots w_n)$

$$\alpha' = \begin{cases} Z_1 \dots Z_r Y_2 \dots Y_m & (Z_1 \dots Z_r) \in \delta_A(w_1, Y_1) \\ Y_2 \dots Y_m & \epsilon \in \delta_A(w_i, Y_1) \end{cases}$$

Wir schreiben $(w, \alpha) \overset{*}{\vdash}_A (w', \alpha')$ falls $\exists$ eine Folge von Konfigurationen $(w^{(1)}, \alpha^{(1)}), \dots, (w^{(s)}, \alpha^{(s)})$
$s \geq 1$ mit $(w^{(1)}, \alpha^{(1)}) = (w, \alpha) \quad (w^{(s)}, \alpha^{(s)}) = (w', \alpha') \quad (w^{(i)}, \alpha^{(i)}) \vdash (w^{(i+1)}, \alpha^{(i+1)})$
$\forall i = 1, \dots, s+1$
$\Rightarrow A$ akzeptiert $w \in \Sigma^* \Leftrightarrow (w, \Gamma_0) \overset{*}{\vdash} (\epsilon, \epsilon)$
Beispiel: $L \subseteq (\Sigma \cup \{\$\})^* \quad L = \{w_1 \dots w_n \$ w_n \dots w_1;\ (w_1 \dots w_n) \in \Sigma^*\}$
$\delta_A(a, \Gamma_0) = \{a\} \quad a, a' \in \Sigma$

$$\delta_A(a, a') = \begin{cases} \{aa', \epsilon\} & a' = a \\ \{a', a\} & a' \neq a \end{cases}$$

$\delta_A(a, \$) = \{a\}$
$\Rightarrow A = (\Sigma \cup \{\$\}, \Sigma \cup \{\Gamma_0\}), \Gamma_0, \delta_A$
$L(A) = L$
Beispiel: $L = \{(w_1 \dots w_n w_n \dots w_1); w = (w_1 \dots w_n) \in \Sigma^*\}$
$\delta_A(a, \Gamma_0) = \{a\}$

$$\delta_A(a, a') = \begin{cases} \{aa'\} & a \neq a' \\ \{a'a, \epsilon\} & a' = a \end{cases}$$

$L = L(A)$

Satz 6.8 *$\forall L \in \mathcal{L}_2$ gilt: $\exists G$ in Greibach NF mit $L = L(G)$*

Satz 6.9 *$L \in \mathcal{L}_2 \Rightarrow \exists$ NPDA mit $L(A) = L$*

Beweis:
Wir fixieren eine Grammatik $G = (V, T, S, R_G)$ für $L \subseteq T^*$ in Greibach NF und konstruieren NPDA (Typ 1) A für L, $A = (T, V, S, \delta_A) \quad \forall a \in T, A \in V$ gelte: $\delta(a, A) = \{\alpha \in V^*; A \to a\alpha \in R_G\}$
Es gilt: Berechnungen von A auf w entsprechen - falls akzeptierend - Linksableitungen von w bezüglich G.
Es gilt: $w \in L(A) \Leftrightarrow \exists$ akzeptierende Berechnung von A auf w
$(w_1 \dots w_n, S) \vdash (w_2 \dots w_n, \alpha^{(1)}) \vdash \dots \vdash (w_n, \alpha^{(n-1)}) \vdash (\epsilon, \epsilon) \quad \alpha^{(n-1)} = A$
$S \to w_1\alpha^{(1)} \to \dots \vdash A \to w_n \qquad \to A_i \to w_i\alpha(i)$
A_i ist das erste Symbol von $\alpha^{(i-1)} \Leftrightarrow S \overset{*}{\to} w_1 \dots w_n$

Beispiel

$G = \{S \to 0A, S \to 1B, A \to 1, A \to 0AA, A \to 1S, B \to 0, B \to 1BB, B \to 0S\} \equiv$
NPDA: $\delta(0, S) = \{A\}, \delta(1, S) = \{B\}, \delta(0, A) = \{AA\}, \delta(1, A) = \{\epsilon, S\}, \delta(0, B) = \{\epsilon, S\}, \delta(1, B) = \{BB\}$
$S \to 0A \to 00AA \to 001A \to 0011 \equiv$ akzeptierende Berechnung $(0011, S) \vdash (011, A) \vdash (11, AA) \vdash (1, A) \vdash (\epsilon, \epsilon)$

Frage: Können NPDAs (Typ 1) auch nichtkontextfreie Sprachen berechnen?
Antwort:Nein.

Satz 6.10 *Jede von einem NPDA A berechnete Sprache ist kontextfrei.*

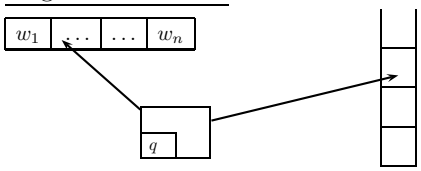
Beweis:
Gegeben: $A = \{\Sigma, \Gamma, \Gamma_0, \delta\}$ berechnet $L \subseteq \Sigma^*$.
Wir konstruieren $G = \{\Sigma, \Gamma, \Gamma_0, R_G\}$ mit $L(G) = L$, für alle $a \in \Sigma, Y \in \Gamma$ fügen wir für alle $a \in \delta(a, Y)$ die Regel $Y \to a\alpha$ zu R_G hinzu. Die ist die umgekehrte Operation zum vorherigen Satz $\Rightarrow G$ berechnet L.
Wir suchen ein NPDA-Modell, das mehr Möglichkeiten der Programmierung bietet.

NPDAs vom Typ 2

PDAs mit inneren Zuständen.
Formal: $A = \{\Sigma, \Gamma, Q, \Gamma_0, q_0, F, \delta\}$ wobei

- Σ das Bandalphabet
- Γ das Kelleralphabet
- Q eine endliche Zustandsmenge
- Γ_0 das Stack-Startsymbol
- q_0 das Startsymbol
- F eine Menge akzeptierender Zustände ($F \subseteq Q$)
- δ die Zustandsüberführungsfunktion bezeichnet.

mögliche Aktionen:



1. A kann Zeichen vom Band und vom Stack einlesen und in Abhängigkeit vom inneren Zustand etwas auf den Stack legen und den Zustand wechseln
2. A kann $\epsilon-$Bewegungen ausführen, d.h. etwas im Stack schreiben und den Zustand wechseln ohne dass ein neues Zeichen gelesen wird.

Formal: $\delta : (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \to 2^{\Gamma^* \times Q}$
$(\alpha, q') \in \delta(a, Y, q)$ heisst: Wenn A in $q \in Q$ ist, a auf Band liest und Y oberstes Stacksymbol $\Rightarrow A$ kann α auf Stack legen und nach q' wechseln $(a, q') \in \delta(\epsilon, Y, q)$.
Wenn A in q und Y oben auf Stack $\Rightarrow$ A kann α auf Stack legen und nach q' wechseln.
Konfigurationen von A haben die Gestalt (q, w, α), wobei q den aktuellen Zustand, w den String mit den noch einzulesenden Zeichen und α den aktuellen Stackinhalt

bezeichnet.
Aktionen:
$(q, w_i, w_{i+1} \dots w_n, Y_1 \dots Y_m) \overset{Leseschritt}{\vdash} (q', w_{i+1} \dots w_n, Z_1 \dots Z_r, Y_2 \dots Y_m)$
$(q, w_i, w_{i+1} \dots w_n, Y_1 \dots Y_m) \vdash (q', w_i \dots w_n, Z_1 \dots Z_r, Y_2 \dots Y_m)$
$q', \alpha = (Z_1 \dots Z_r) \in \delta(q, w_1, Y_1)$ beziehungsweise $\in \delta(q, \epsilon, Y_1)$

Berechnungsmodus

A akzeptiert $w \in \Sigma^* \Leftrightarrow \exists$ akzeptierte Berechnung von A auf $w \Leftrightarrow (q_0, w, \Gamma_0) \overset{*}{\vdash} (q, \epsilon, \alpha)$ mit $q \in F$
Beispiel: $L = \{(w_1 \dots w_n \$ w_n \dots w_1); w_i \in \Sigma, \$ \notin \Sigma\}$ $\quad Q = \{q_0, q_1, q_2\}$
$\Gamma = \{\Gamma_0\} \cup \Sigma \quad F = \{q_2\} \quad$ Bandalphabet $\Sigma \cup \{\$\}$
$\delta(a, \Gamma_0, q_0) = \{a\Gamma_0, q_0\} \quad a, a' \in \Sigma$
$\delta(a, a', q_0) = \{(aa', q_0)\} \quad \delta(\$, a', q_0) = \{(a', q_1)\}$

$$\delta(a, a', q_1) = \begin{cases} \emptyset & a \neq a' \\ (\epsilon, q_1) & \text{sonst} \end{cases}$$

$a, a' \in \Sigma$
$\delta(\epsilon, \Gamma_0, q_1) = \{(\epsilon, q_2)\}$
Es gilt $(q_0, x, \Gamma_0) \overset{*}{\vdash} (q_2, \epsilon, \epsilon)$ falls $x = w_1 \dots w_n \$ w_n \dots w_1$
Beispiel: Klammersprache $D \subseteq \{(,)\}^*$
$A : \Sigma = \{(,)\} \quad \Gamma = \{\Gamma_0, A\} \quad Q = \{q_0, q_1, q_2\} \quad F = \{q_1\}$
$\delta_A((, \Gamma_0, q_0) = \{(A\Gamma_0, q_0)\} \quad \delta_A(), \Gamma_0, q_0) = \{\epsilon, q_2\} \quad \delta_A(), A, q_0) = \{(\epsilon, q_0)\}$
$\delta_A(\epsilon, \Gamma_0, q_0) = \{(\epsilon, q_1)\}$
Bemerkung: Der vereinbarte Berechnungsmodus heißt Akzeptanz mittels „leerem Stack“. In der Literatur existiert auch ein anderer Modus:
$w \in L(A) \Leftrightarrow (q_0, w, \Gamma_0) \overset{*}{\vdash} (q, \epsilon, \alpha) \quad q \in F$
Es ist eine leichte Übungsaufgabe zu zeigen, dass jeder NPDA mit Modus 2 durch einen NPDA (leerer Stack) simuliert werden kann.
Frage: Wie ist die Berechnungskraft von Typ 2 PDAs?

Satz

A ist Typ 2 NPDA $\Rightarrow L(A)$ ist kontextfrei.
Beweis: gegeben: $A = (\Sigma, \Gamma, \Gamma_0, Q, q_0, F, \delta)$, wir konstruieren eine kontextfreie Grammatik $G, L(G) = L(A)$. „Tripelkonstruktion“
$G = (\Sigma, \{[qYq'] : q, q' \in Q, Y \in \Gamma\} \cup \{S\}, R) \quad R = R_0 \cup R_\Sigma \cup R_\epsilon \quad R_0 = \{S \to [q_0\Gamma_0 q], q \in F\}$
Regeln in R_Σ und R_ϵ sind so zu konstruieren, dass für alle $Y \in \Gamma, q, q' \in Q$ gilt: $[qYq'] \overset{*}{\to} w$ genau dann wenn $[q, w, Y] \overset{*}{\vdash} [q', \epsilon, \epsilon] \Rightarrow$ wenn dies gelingt, dann ist $L(A) = L(G)$ wegen R_G.
$[q_0\Gamma_0 q] \overset{*}{\to} w \in L(A) \quad q \in F \to w \in L(G) \Leftrightarrow S \overset{*}{\to} w \Leftrightarrow \exists\, q \in F$ mit $[q_0\Gamma_0 q] \overset{*}{\to} w \Leftrightarrow \exists\, q \in F [q_0, w, \Gamma_0] \overset{*}{\vdash} [q, \epsilon, \epsilon] \Leftrightarrow w \in L(A)$
R_ϵ enthält für alle $a \in \Sigma, q \in Q, Y \in \Gamma$ und $(q', \alpha) \in \delta(a, Y, q)$ folgende Regeln:
Fall 1: „pop“ $\quad \alpha = \epsilon \quad [qYq'] \to a$

Fall 2: „push“ $\alpha = Y_1 \dots Y_m$
$\{[qYq_{m+1}] \to a[q_1Y_1q_2][q_2Y_2q_3]\dots[q_mY_mq_{m+1}] \quad q_1 = q'$ und alle $q_2 \dots q_{m+1} \in Q\}$ bildet folgendes Verhalten ab:
Wir raten in Gestalt $q_2 \dots q_{m+1}$ die Zustände, in denen A ist wenn Y_1 bzw. Y_2 bzw. $\dots$ bzw. Y_m vom Stack genommen wird.
R_ϵ enthält für alle $Y \in \Gamma, q \in Q$ und für alle $(\alpha, q') \in \delta(\epsilon, Y, q)$ folgende Regeln:
Fall 1: $\alpha = \epsilon \quad [qYq'] \to \epsilon$
Fall 2: $\alpha = Y_1 \dots Y_m \quad \{[qYq_m] \to [q'Y_1q_2][q_2Y_2q_3]\dots[q_mY_mq_{m+1}];\ \forall q_2 \dots q_{m+1} \in Q\}$
Beobachtung: G ist nicht in Greibach NF
Beweis: Obige Betrachtung per Induktion.
Ein Typ 2 PDA heißt deterministisch falls $\forall a \in \Sigma, Y \in \Gamma, q \in Q$ gilt: $|\delta(a, Y, q) \cup \delta(\epsilon, Y, q)| \leq 1 \Rightarrow D\mathcal{L}_2 \subseteq \mathcal{L}_2$, wobei $D\mathcal{L}_2$ die Menge der durch DPDA berechenbare Sprachen bezeichnet.
$D\mathcal{L}_2$ ist interessant für Programmiersprachen.

Anhang A

Komplexitätsklassen

A.1 Sprachen

- L : $\Sigma^* \to \{0,1\}$
- $\mathcal{P} = \{L \subseteq \Sigma^*\}$
- ENUM = $\{L \subseteq \{0,1\}^*$; Es existiert TM M die L aufzählt$\}$
  (d.h. $x \in L \iff M(x) = 1$ und $x \notin L \iff M(x) = 0 \vee$ M hält nicht auf x)
- R $\{g : \{0,1\}^* \to \{0,1,*\}$; Es existiert TM M, die g berechnet$)$
- REK = $\{L \subseteq \Sigma^*$; L berechenbar$\}$
  (d.h es existiert eine 1-Band TM M die L berechnet)
- EXPTIME = $\{L \subseteq \Sigma^*$; $\exists\, 2^{n^{O(1)}}$ zeitbeschränkte TM für L$\}$
- NP = $\{L \subseteq \Sigma^*$; Es existiert NTM M die L in Polynominalzeit berechnet$\}$
  (NP = $\{L \subseteq \Sigma^*$; Es existiert Polynom p = p(n) und Sprache $L' \in P$ mit L gdw. $\exists\, y \in \{0,1\}^{p\{|x|\}}$ mit $x\#y \in L'\}$)
- P = $\{L \subseteq \Sigma^*$; L in Polynominalzeit berechenbar$\}$
  (d.h. es existiert pol. zeitbeschränkte 1-Band TM die L berechnet)
- REG = $\{L \subseteq \Sigma^*$; Es existiert DFA mit L(A) = L$\}$
  (Für alle regulären Ausdrücke gilt $L(R) \in REG$)

$REG \subset P \subseteq NP \subseteq EXPTIME \subset REK \subseteq R \subset ENUM \subseteq \mathcal{P}(\{0,1\}^*)$

A.2 Berechnungsprobleme

- $\mathcal{REK}$ = {Π; Π Berechnungsproblem das berechenbar ist}
- PTIME = {Π; Π Berechnungsproblem mit Polynominalzeitalgorithmus}

$PTIME \subset \mathcal{REK}$