Theoretische Informatik
Vorlesungsmitschrift

Sebastian T. Hafner
sonix@uni-saarland.info

11. Februar 2005
last compile: 19. Februar 2005

Inhaltsverzeichnis

1 Rechnen an sich

- Was ist „Rechnen“?
- Welche Rechenmechanismen gibt es?
- Kann man „alles“ „berechnen“?
- Wie „teuer“ ist Rechnen?

2 Rechenmodelle / Rechensituationen

2.1 PC / Mac

ZEICHNUNG CHRISTIAN HOLLER 1

Auf bestimmte Eingaben sollen bestimmte Ausgaben erfolgen.

2.2 AIRBUS

viele vernetzte Computereinheiten

Eingaben: Steuerungen durch Besatzung
Sensoren
- Temperatur
- GPS
- Luftgeschwindigkeit
- Positionsanzeige / Kompass

Ausgabe: Informationen für Besatzung
Aktuatoren

2.3 „Einfachste Situation“: Theoretitischer PC

ZEICHNUNG CHRISTIAN HOLLER 2

soll Funktion F: Zeichenketten ⟵ Zeichenketten berechnen
Eingabe: s Ausgabe F(s) $\{0,1\}$ (Ja / Nein)

2.4 EINSCHUB: Zeichenketten

Alphabet: Σ endliche Menge
String / Zeichenkette endliche Folge von Elementen von Σ
z.B. $v = v_1 v_2 \dots v_k \; v_i \in \Sigma \forall i$
$\underbrace{|v|}_{\text{Länge}} = k$

Σ^k Menge der Strings der Länge k

$$\{\}^k = \{\} \quad \forall k \; ?$$

Σ^0

$$\Sigma^0 = \left\{ \underbrace{\epsilon}_{leererString} \right\}$$

Σ^* Menge der Strings über Σ (mit endlicher Länge)

$$\Sigma^* = \cup_{k \geq 0} \Sigma^k$$

2.4.1 Konkatenation

$$u, v \in \Sigma^*$$
$$u \in \Sigma^l$$
$$v \in \Sigma^k$$
$$u\epsilon = u$$
$$\epsilon u = u$$
$$uv = u_1 u_2 \dots u_l v_1 \dots v_k$$

$$v \in \Sigma^* i \in \mathbb{N} = \{0, 1, \dots\}$$
$$v^0 = \epsilon$$
$$v^i = v v^{i-1}$$
$$L \subset \Sigma^* \dots \text{„Sprache über } \Sigma\text{“}$$

2.4.2 „Maschine definiert Sprache“

$$\{s \in \Sigma^* \mid F(s) = JA\}$$

3 Welche Sprachen können von welchen Arten von „Automaten“ berechnet werden?

3.1 3 Arten von Automaten

1. Endliche Automaten
2. Keller Automat
3. Turing Maschine

3.2 Endlicher Automat

Eingabeband

a	b	b	a	a	a	

ZEICHNUNG CHRISTIAN HOLLER 3

endlich viele Steuerungsregeln folgender Form:

Wenn Zustand $= q$
$\wedge$ Eingabekopf liest Zeichen a
dann:
1) Rücke Eingabekopf um eine Zelle nach rechts
2) wechsle in Zustand q'

$s \in Q$ Startzustand
$F \subset Q$ Endzustände

3.2.1 Anfangskonfiguration

Maschine ist im Startzustand s
Eingabekopf ganz links

3.2.2 Endkonfiguration

Maschine ist in einem der Endzustände
Eingabekopf ganz rechts

Maschine akzeptiert einen Engabestring u, wenn aus der Startkonfiguration durch wiederholtes Anwenden von Regeln eine Endkonfiguration erreicht werden kann.

3.3 Beispiel

3.3.1 Konfiguration

$$\Sigma = \{a, b\} \qquad Q = \{\underbrace{0}_{=s}, 1, 2\} \qquad F = \{2\}$$

3.3.2 Regeln

derzeitiger Zustand	gelesenes Zeichen	neuer Zustand
0	a	1
0	b	0
1	a	2
1	b	0
2	a	2
2	b	0

3.3.3 ungültiger String

a	b	b	a		
↑	↑	↑	↑	↑	
0	1	0	0	$\underline{1}$	ungültiger Endzustand

3.3.4 gültiger String

a	b	a	a		
↑	↑	↑	↑	↑	
0	1	0	0	2	

Das Beispiel akzeptiert alle Strings über $\{a, b\}$, die in aa enden.

3.4 Nicht deterministische Maschine

d.i. mehr als eine Regel in gleichen Zuständen anwendbar.

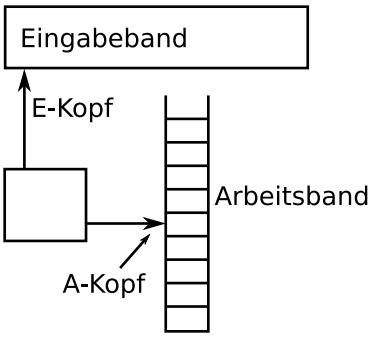
	derzeitiger Zustand	gelesenes Zeichen	neuer Zustand
α	0	a	0
β	0	b	0
γ	0	a	1
δ	1	a	2

a	b	a	a		
↑	↑	↑	↑	↑	
0	1	0	0	2	
α	β	γ	δ		

3.5 Kellerautomat

Eingabeband

a	b	b	a	a	a	



Q Zustandsmenge
$s \in Q$ Startzustand
$F \subset Q$ Endzustand

3.5.1 Regeln:

Wenn Zustand = q
$\wedge$ Eingabekopf liest a
$\wedge$ Arbeitskopf liest A

dann:
1) gehe in Zustand q'
2) Bewege Eingabekopf
i) rechts
ii) nicht
3) Arbeitskopf: i) ersetze A durch A'
ii) Kopf nach unten, A gelöscht
iii) Kopf nach oben, schreibe A'

3.5.2 Anfangskonfiguration

Zustand = s
Eingabekopf ganz links
Arbeitsband

ZEICHNUNG CHRISTIAN HOLLER 5

3.5.3 Endkonfiguration

Maschine in einem der Endzustände
Eingabekopf ganz rechts

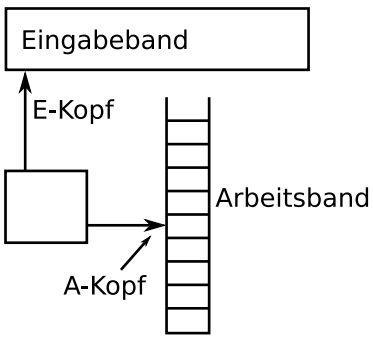
Maschine akzeptiert einen Engabestring u, wenn aus der Startkonfiguration durch wiederholtes Anwenden von Regeln eine Endkonfiguration erreicht werden kann.

$$\{a^n b^n \mid n \geq 1\}$$

Deterministisch immer höchstens 1 Regel anwendbar

Nicht Deterministisch es können mehrere Regeln anwendbar sein

3.6 Keller Maschine



Zustand
Eingabekopf $\longrightarrow$ EK $\longrightarrow$ •
Arbeitskopf

3.6.1 Beispiel

$u \in \{a, b\}$
$u = u_1 \dots u_n$
$u^R = u_n \dots u_1$
$u\$u^R\$$

derzeitiger Zustand	EK	AK	neuer Zustand	EK	AK
$\underbrace{0}_{\text{Anfangszustand}}$	a	Z	1	$\rightarrow$	$\uparrow$ A
0	b	Z	1	$\rightarrow$	$\uparrow$ B
1	a	A	1	$\rightarrow$	$\uparrow$ A
1	a	B	1	$\rightarrow$	$\uparrow$ A
1	b	A	1	$\rightarrow$	$\uparrow$ B
1	b	B	1	$\rightarrow$	$\uparrow$ B
1	\$	A	2	$\rightarrow$	A
1	\$	B	2	$\rightarrow$	B
2	a	A	2	$\rightarrow$	$\downarrow$
2	b	B	2	$\rightarrow$	$\downarrow$
2	\$	Z	$\underbrace{3}_{\text{Endzustand}}$	$\leftarrow$	$\downarrow$

3.7 Turing Maschine

BILD

3.7.1 Regeln

Wenn	Zustand = q		neuer Zustand q'	
	Eingabekopf liest a	dann	Eingabekopf	$\rightarrow, \bullet$
	Arbeitskopf liest A		Arbeitskopf	A',$\leftarrow, \bullet, \rightarrow$

DEA	deterministischer endlicher Automat
NEA	nichtdeterministischer endlicher Automat
DKA	deterministischer Keller Automat
NKA	nichtdeterministischer Keller Automat
DTM	deterministische Turing Maschine
NTM	nichtdeterministische Turing Maschine

3.8 Definition

Eine Sprache L ist eine DEA-Sprache, wenn es einen DEA gibt, der L akzeptiert.
und analog andere Sprachen.

3.9 Lemma

- Jede DEA-Sprache ist eine NEA-Sprache.
- Jede DKA-Sprache ist eine NKA-Sprache.
- Jede DTM-Sprache ist eine NTM-Sprache.

3.10 Lemma

- Jede DEA Sprache ist eine DKA-Sprache.
- Jede NEA Sprache ist eine NKA-Sprache.
- Jede DKA Sprache ist eine DTM-Sprache.
- Jede NKA Sprache ist eine NTM-Sprache.

BILD

3.11 Fragen

1. Welche Inklusionen sind strikt?
2. Gegeben eine NKA:
   Gibt es einen DEA der genau die gleiche Sprache akzeptiert?
3. Gegeben zwei Maschinen:
   akzeptieren sie die gleiche Sprache?
4. ...

4 (Baby) Mengenlehre

$f : A \to B$ Funktion

Injektion:	$\forall a, b \in A: \quad a \neq b \quad \Rightarrow \quad f(a) \neq f(b)$	
Surjektion:	$\forall b \in B \exists a \in A : f(a) = b$	
Bijektion:	Surjektion & Injektion	(eindeutige Zuordnung)

4.1 Definition

A und B heißen genau dann gleichmächtig, wenn es eine Bijektion zwischen A und B gibt.

4.1.1 Beispiel

$\mathbb{G}$ seien die nichtnegativen, geraden Zahlen: $\{0, 2, 4, \ldots\}$
$\mathbb{N}$ und $\mathbb{G}$ sind gleichmächtig.

4.1.2 Beweis

$$f(x) = 2 \cdot x \qquad \text{Bijektion!}$$

4.2 Definition

A heißt endlich, wenn $\exists k \in \mathbb{N}$, sodass A gleichmächtig wie $\{0, \ldots, k-1\}$

$$A = \{a_0, a_1, \ldots, a_{k-1}\}$$

A heißt abzählbar, wenn A endlich, oder A gleich mächtig wie $\mathbb{N}$ - abzählbar unendlich.

4.3 Lemma

$A_1, A_2, \ldots A_k$ endliche Mengen, $k \in \mathbb{N}$

$A_1 \cup A_2 \cup \ldots \cup A_k$	ist endlich
$A_1 \times A_2 \times \ldots \times A_k$	ist endlich
A_1^k	ist endlich

4.4 Lemma

$A_0, A_1, \ldots$ sei unendliche Folge von disjunkten endlichen Mengen
Dann ist $\bigcup_{i \in \mathbb{N}} A_i$ abzählbar (undendlich)

4.5 Lemma

A, B abzählbar unendlich $\quad \Rightarrow \quad$ $A \times B$ abzählbar unendlich

4.5.1 Beweis

	0	1	2	3	4
0	$(0,0)$	$(0,1)$	$(0,2)$	$(0,3)$	$(0,4)$
1	$(1,0)$	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$
2	$(2,0)$	$(2,1)$	$(2,2)$	$\ldots$	
3	$(3,0)$	$(3,1)$	$\ldots$		

$$C_i = \{(a_j, a_k) \mid j + k = i\}$$

$$A \times B = \bigcup_{i \in \mathbb{N}} C_i$$

C_i ist disjunkt und $|C_i| = i + 1$

4.6 Lemma

Sei A abzählbar unendlich.

$$2^A = \{B | B \subset A\}$$

2^A ist NICHT abzählbar

4.6.1 Beweis

$A = \{a_o, a_1, \ldots\}$ $B \subset A$ kann mit „undendlicher“ Bitfolge s_B identifiziert werden.

$$s_B = \begin{cases} 1 & a_i \in B \\ 0 & a_i \notin B \end{cases}$$

$$\begin{array}{ll} B = \{1, 3, 7\} & 010100010\ldots \\ B = \{1, 3, 5, 7, \ldots\} & 010101010101\ldots \end{array}$$

2^A nicht abzählbar, d.i. $f : \mathbb{N} \to 2^A$
$\Rightarrow$ f ist keine Bijektion (z.B. weil f keine Surjektion)

$$\begin{array}{lllllllllll} f(0) & = & B_0 & s_{B_0} & 0 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots & \\ f(1) & = & B_1 & s_{B_1} & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots & 1 \\ f(2) & = & B_2 & s_{B_2} & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \ldots & 0 \end{array}$$

4.7 Cantor'sches Diagonalisierungsverfahren

Möchte zeigen:
$\exists B \subset A$ sodass $\forall n \in \mathbb{N} : f(n) \neq B$
$\Leftrightarrow$
Suche Bitfolge, die in Auflistung nicht vorkommt, sich also von jedem s_{B_i} unterscheidet.

4.8 Satz

A Menge $\Rightarrow$ $\exists$ keine Surjektion $h : A \to 2^A$

4.8.1 Konzept

A Menge $\Rightarrow$ $\exists$ keine Bijektion $h : A \to 2^A$

4.8.2 Beweis

Wollen zeigen:

$$\forall h : A \to 2^A \quad \exists X \in 2^A : \forall a \in A : h(a) \neq X$$
$$X \subseteq A$$

h vorgegeben.

$$\Delta = \{x \in A \mid x \notin h(x)\}$$

Behauptung:

$$\forall a \in A: \ \Delta \neq h(a) \qquad \text{(sie unterscheiden sich in element a)}$$
$$a \in \Delta \Leftrightarrow a \notin h(a)$$

4.9 Satz

$$\underbrace{\text{NTM-Sprachen über } \Sigma}_{L^{\in}} \quad \neq \quad \underbrace{\text{alle Sprachen über } \Sigma}_{2^{\Sigma *}}$$

4.9.1 Beweis

$\Rightarrow$ $\exists$ NTM die genau L akzeptiert.
Es reicht zu zeigen, dass es nur abzählbar viele NTMs gibt.
Σ^* abzählbar unendlich
2^{Σ^*} überabzählbar

Jede NTM als endlichen String über $\Sigma \vee ASCII$ spezifizieren.
Abbildung $(\Sigma \vee ASCII)^*$ ist abzählbar.

5 Endliche Automaten (formale Spezifikation)

$$M = (\Sigma, Q, s, F, \Delta)$$

Σ	Eingabealphabet (endlich)
Q	Zustandsmenge (endlich)
$s \in Q$	Startzustand
$F \subset Q$	Endzustände
$\Delta \subset Q \times \Sigma \times Q$	„Regeln“

Definition

M akzeptiert $w = w_1 w_2 \dots w_n \in \Sigma^*$ genau dann wenn $\exists$ Zustandsfolge

$$q_0 q_1 \dots q_n \text{ mit } \begin{array}{l} q_0 = s \\ q_n \in F \\ \text{für } 0 < i \leq n \, (2_{i-1} w_i, q_i) \in \Delta \end{array}$$

$$L(M) = \{w \in \Sigma^* \mid w \text{ wird von } M \text{ akzeptiert}\}$$

5.1 Beispiel

$$\begin{array}{lll} \Sigma = \{a, b\} & s = 0 & \Delta = \{(0, a, 0), (0, b, 0), (0, a, 1), (1, a, 2)\} \\ Q = \{0, 1, 2\} & F = \{2\} & \end{array}$$

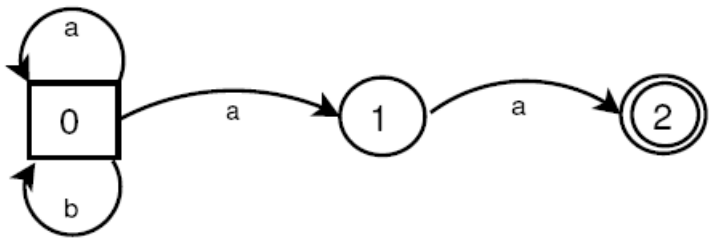
babaa wird von M akzeptiert

w_1	w_2	w_3	w_4	w_5	
0	0	0	0	1	2

aab wird nicht von M akzeptiert.

5.2 Übergangsgraph eines endlichen Automaten M (Transitionsgraph)

Knotenmenge	Q	
gerichtete Kanten mit Beschriftung	$[q, q'\rangle$	mit Beschriftung a ist im Graphen
	$\Leftrightarrow$	$(q, a, q') \in \Delta$



$w = w_1 w_2 \dots w_n$ wird von M akzeptiert

$\Leftrightarrow$ $\exists$ gerichteter Pfad P von s zu einem Knoten $q_n \in F$, so dass die Kantenbeschriftungen entlang P genau $w = w_1 w_2 \dots w_n$ ergeben.

5.3 Definition

M heißt deterministisch, wenn

$$\forall (q,a) \in Q \times \Sigma i \,|\{q' \mid (q,a,q') \in \Delta\}| \leq 1$$

sonst heißt M nicht-deterministisch.

5.4 Alternative Definition von L(M)

für jedes $q \in Q$ definiert $\delta_q : \Sigma^* \to 2^Q$

$$\delta_q(\epsilon) = \{q\}$$

$$\delta_q(ua) = \bigcup_{q' \in \delta_q(u)} \delta_{q'}(a)$$

$\delta_q(w) =$ „Zustände die von q beginnend über w erreichbar sind“

$$L(M) = \{w \; in \Sigma^* \mid \delta_s(w) \cap F \neq \emptyset\}$$

5.5 Gibt es Sprachen, die nicht von NEA's akzeptiert werden?

Behauptung

$$L = \{a^n \$ b^n \$ \mid n \in \mathbb{N}\} \quad \text{wird von keinem NEA akzeptiert.}$$

Beweis

Wollen zeigen $\forall$ NEA's M: $L(M) \neq L$

Sei M NEA, Zustandsmenge Q, $|Q| = m$
Betrachte $a^m \$ b^m \$ \in L$

Fall 1

$$a^m \$ b^m \$ \notin L(M) \quad \Rightarrow L(M) \neq L$$

Fall 2

$$a^m \$ b^m \$ \notin L(M) \quad \Rightarrow \exists \underbrace{q_o q_1 \dots q_m} p_0 p_1 \dots p_m \quad p \in F$$

Folge von $m+1$ Zuständen
$\Rightarrow$ irgenein Zustand wird wiederholt.

ZEICHNUNG CHRISTIAN HOLLER 667

Korrektur

„NEA-Sprachen $\overset{\subset}{\neq}$ NKA-Sprachen “
„DEA-Sprachen $\overset{\subset}{\neq}$ DKA-Sprachen “

5.6 Pumping Lemma

Sei L eine DEA/NEA Sprache
$\Rightarrow \exists m \in \mathbb{N} : \forall x \in L$ mit $|x| > m$
$\exists$ Unterteilung $x = uvw$ mit $|uv| \leq m$ und $0 < |v| \leq m$
$\forall i \in \mathbb{N}: \quad uv^i w \in L$

Beweis

Sei L EA Sprachen
Sei M eine DEA/NEA mit $L(M) = L$
Q sei Zustandsmenge von M
$m = |Q|$
Sei $\mathcal{G}$ Übergangsgraph von M
s Startzustand
$\mathcal{G}$ hat m Knoten
Sei $x \in L,\ |x| > m$
(wenn nicht existent, muss nichts gezeigt werden!)
Sei P der „akzeptierende Pfad“ für x durch $\mathcal{G}$
Im Anfangsteil der Lnänge m von P muss es Knotenwiederholungen geben.

ZEICHNUNG CHRISTIAN HOLLER 668

$$P \Rightarrow Q \quad \Leftrightarrow \quad \neg Q \Rightarrow \neq P$$

$$\forall m \in \mathbb{N}\ \exists x \in L :\ |x| > m\ \forall \text{ Unterteilungen } \begin{array}{l} x = uvw \\ |uv| \leq m \\ 1 \leq |v| \leq m \end{array} \qquad \forall i \in \mathbb{N}\ uv^i w \notin L$$

Verwendung, um zu zeigen, das L nicht NEA-Sprache

1. Gegner gibt m vor
2. „Wir“ geben $x \in L$ an, mit $|x| > m$
3. Gegner gibt Unterteilung $x = uvw$ vor, $|uv| \leq m$, $1 \leq |v| \leq m$
4. „Wir“ geben $i \in \mathbb{N}$ an, sodass $uv^i w \notin L$.

Wenn „wir“ dieses Spiel immer gewinnen können, dann ist L KEINE NEA-Sprache.

Beispiel

Zeige, dass $L = \{a^n \mid n \text{ ist Primzahl}\}$ keine NEA-Sprache ist.

1. Gegner gibt m
2. Wir wählen eine Primzahl $p > m$ und geben $x = q^p \in L$
3. Gegner: $x = uvw \quad a^p = a^j a^k a^l \quad j + k \leq m,\ 1 \leq k \leq m \quad j + k + l = p$
4. Wir wählen i so, dass $a^j \left(a^k\right)^i a^l \notin L$
   d.i. $j + k + l$ soll keine Primzahl sein
   z.B. $i = j + l \quad j + k(j + l) + l = (k + 1)(j + l)$

$$\delta_q(w) = \{q' \in Q \mid q' \text{ von } q \text{ über Pfad mit Beschreibung w erreichbar}\}$$

$$L_q = \{w\ in\Sigma^* \mid \delta_q(w) \cap F \neq \emptyset\}$$

$$L_\delta = L(m)$$

5.7 Definition

$L \subset \Sigma^*$ irgendeine Sprache

$$C_L : \Sigma^* \to 2^{\Sigma^*} \quad C_L(w) = \{x \in \Sigma^* \mid wx \in L\}$$

Forsetzungssprache von w bezüglich L.

Beispiel 1

$$\begin{aligned} L_1 &= \{a^n€b^n€ \mid n \in \mathbb{N}\} \\ C_L(aa€b) &= \{b€\} \\ C_L(aa€ab) &= \{\ \} \\ C_L(aa) &= \{€bb€,\ a€bbb€,\ \ldots\} \\ &= \left\{a^n€b^{n+2}€ \mid n \in \mathbb{N}\right\} \\ C_{L_1}(a^n€) &= \{b^n€\} \end{aligned}$$

Beispiel 2

$$L = \left\{w\ in\{a,b\}^* \mid (\#a'smw)\ mod\ 3 = 0\right\}$$

$$\begin{aligned} C_L(aaa) &= L \\ C_L(ababbaaaaa) &= L \\ C_L(baa) &= \{x\ in\{a,b\}^* \mid (\#a'smx)\ mod\ 3 = 1\} = L_1 \\ C_L(bab) &= \{x \in \{a,b\}^* \mid (\#a'smx)\ mod\ 3 = 2\} = L_2 \\ C_L(abbaaab) &= L_2 \end{aligned}$$

5.8 Satz (Myhill-Nerode)

L ist DEA-Sprache $\Leftrightarrow \mathcal{F}_L = \{C_L(w) \mid w \in \Sigma^*\}$ ist endlich.

Beweis

Teil (i) „⇒“

L ist DEA-Sprache $\Rightarrow$ $\exists$ DEA M $\begin{matrix}\Sigma \\ Q \\ s \\ F \\ \Delta \text{ deterministisch}\end{matrix}$ sodass $L = L(M)$.

$$w \in \Sigma^* : C_L(w) = \begin{cases} L_p & \text{falls } \delta_q(w) = \{p\} \\ \emptyset & \text{sonst} \end{cases}$$

$$|Q| = n \quad \mathcal{F}_L = \{L_p \mid p \in Q\}$$

(unter der Annahme, dass jeder Zustand $q \in Q$ von s erreichbar ist)

$$|\mathcal{F}| \leq |Q| + 1 \quad \text{endlich}$$

Teil (ii) „⇐“

$$L \subset \Sigma^*, \quad \mathcal{F}_L \text{ endlich}$$

Wir wollen einen DEA bauen, der genau L akzeptiert:

$$\begin{aligned}
&\Sigma \\
&Q = \mathcal{F}_L \\
&s = C_L(\epsilon) = L \\
&F = \{L' \in \mathcal{F}_L \mid \epsilon \in L'\} \\
&\quad = \{C_L(w) \mid w \in L\} \\
&\Delta = \{(C_L(w)), a, C_L(wa) \mid w \in \Sigma^*, a \in \Sigma\}
\end{aligned}$$

Behauptung 1

M ist deterministisch

Behauptung 2

M akzeptiert genau L

Beweis zu Behauptung 1

müssen zeigen:

$$v, w \in \Sigma^*, a \in \Sigma \quad C_L(v) = C_L(w) \Rightarrow C_L(va) = C_L(wa)$$

$$\underbrace{x \in C_L(va)}_{\Leftrightarrow vax \in L} \Leftrightarrow \underbrace{ax \in C_L(v)}_{= C_L(w)} \Leftrightarrow wax \in L \Leftrightarrow x \in C_L(wa)$$

Beweis zu Behauptung 2

$$w = w_1 w_2 \dots w_n \in L$$

Anwendung von Satz (Myhill-Nerode)

$$\begin{aligned}
w_1 w_2 \dots w_n &\in C_L(\epsilon) \\
&\Updownarrow \\
w_2 \dots w_n &\in C_L(\epsilon) \\
&\Updownarrow \\
w_3 \dots w_n &\in C_L(\epsilon) \\
&\vdots \\
w_n &\in C_L(w_1 \dots w_{n-1}) \\
&\Updownarrow \\
\epsilon &\in C_L(w_1 \dots w_n)
\end{aligned}$$

$$\begin{aligned}
w_1 w_2 \dots w_n &\notin C_L(\epsilon) \\
&\Downarrow \\
w_2 \dots w_n &\notin C_L(w_1) \\
&\vdots \\
\epsilon &\notin C_L(w_1 w_2 \dots w_n) \quad \text{kein Endzustand}
\end{aligned}$$

Anmerkung

5.9 Satz

L ist NEA-Sprache $\Rightarrow$ L ist auch eine DEA-Sprache

5.9.1 Beweis

L ist NEA-Sprache

d.i. $\exists$ nicht deterministischer Automat $M \begin{matrix}\Sigma\\Q\\s\\F\\\Delta\end{matrix}$ $L(M) = L$

Nach Myhill-Nerode reicht es zu zeigen, dass $\mathcal{F}_L$ endlich.

$$w \in \Sigma^* : \; C_L(w) = \bigcup \{L_{q'} \;\mid\; q' \in \delta_q(w)\}$$

Q ist endlich $\Rightarrow$ # der verschiedenen $\delta_q(w)$'s endlich.

$$\left\{\delta_q(w) \;\mid\; |w \in \Sigma^*| \leq 2^{|Q|}\right\}$$

$$\Rightarrow \{C_L(w) \;\mid\; w \in \Sigma^*\} \quad \text{endlich}$$

Direkte Konstruktion

Aus NEA M konstruiere DEA M', so dass $L(M') = L(M)$

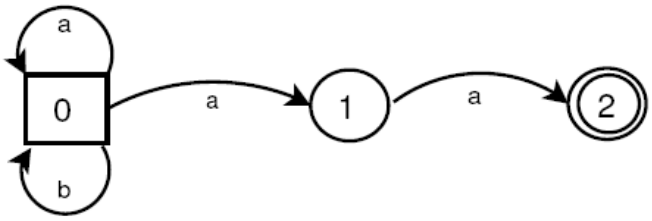
$$\begin{matrix}\Sigma\\Q\\s \in Q\\F \subset Q\\\Delta\end{matrix} \qquad M' \qquad \begin{matrix}\Sigma\\Q' &=& 2^Q\\s' &=& \{s\}\\F' &=& \{A \subset B \;\mid\; A \cap F \not\approx Q\}\\\Delta' &=& \left\{(A,a,B) \;\mid\; A \subseteq Q, B = \bigcup_{\xi \in A} \delta_q(a)\right\}\end{matrix}$$

Idee:

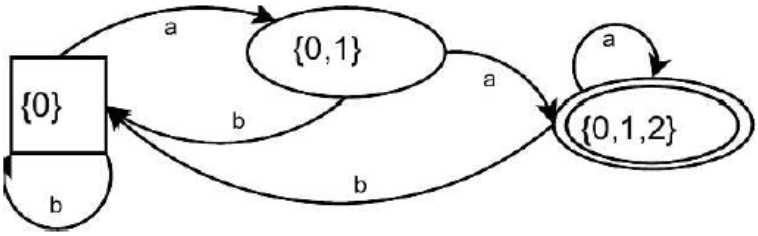
$\delta_q(w) \ldots$ Zustände von M'

Beispiel

M:



M':



5.10 „ϵ-Übergänge “

Idee

Endlicher Automat muss Lesekopf nicht bei jedem Übergang nach rechts rücken, Kopf darf auch verweilen.

$$\begin{aligned} \Sigma & \\ Q & \\ s \in Q, F \subset Q & \\ \Delta \subseteq Q \times \Sigma^{\leq 1} = \Sigma^1 \cup \{\epsilon\} & \end{aligned}$$

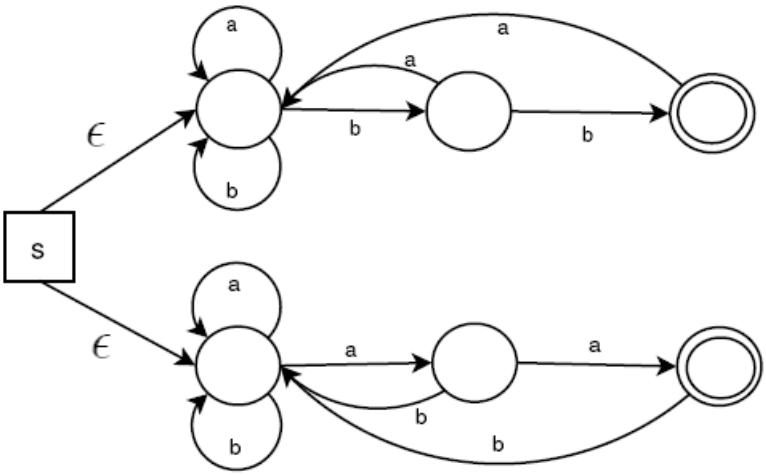
M akzeptiert $x \in \Sigma^*$

$$\exists q_0, \ldots, q_m \in Q,\ \exists x_1, x_2, \ldots, x_m \in \Sigma^{\leq 1}$$

für $1 \leq i \leq m : (q_{i-1}, x_i, q_i) \in \Delta$

Übergangsgraphen

Kanten können auch mit ϵ beschriftet werden.



5.11 Satz

Wenn L durch einen NEA mit ϵ-Übergängen akzeptiert wird, dann ist L regulär.
d.i. L wird auch durch DEA (oder NEA) ohne ϵ-Übergange akzeptiert.

Beweis

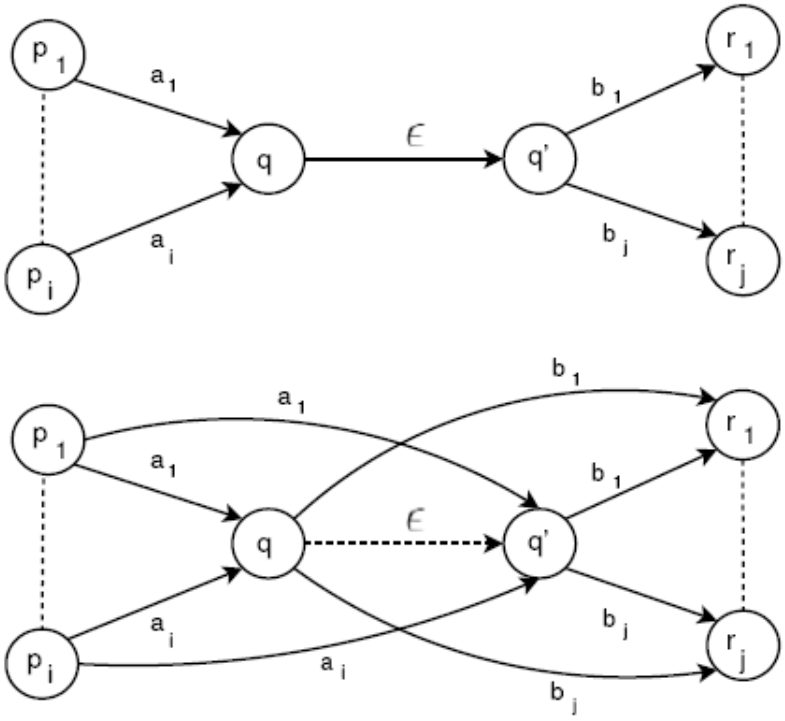
Methode 1

Zeige $\mathcal{F}_L$ ist endlich

Methode 2

Konstruiere aus NEA mit ϵ-Übergängen M, einen NEA ohne ϵ-Übergänge M', so dass $L(M) = L(M')$.

Es reicht zu zeigen, wie man eine ϵ-Kante entfernen kann!


p 1
a 1
q
ϵ
q'
b 1
r 1
p i
a i
b j
r j
p 1
a 1
b 1
r 1
a 1
b 1
q
ϵ
q'
p i
a i
b j
r j
a i
b j

6 2-Wege Automat

Idee

deterministischer endlicher Automat
Lesekopf darf auch zurückrücken.

Übergangsregeln

$$(q, a, \epsilon', \nu) \in Q \times \Sigma \times Q \times \{L, B, R\}$$

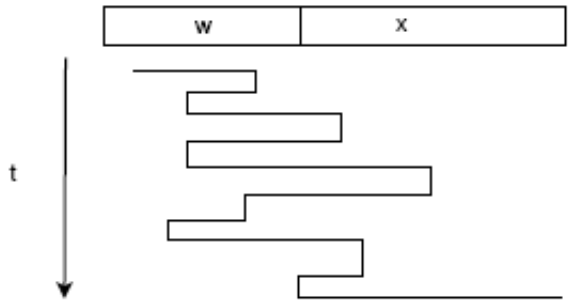
6.1 Satz

Wenn L durch einen deterministischen 2-Wege-Automaten akzeptiert wird, dann ist L regulär.

Beweis

Idee

Zeige, L hat endlich viele Forsetzungssprachen.



$$f_x : \ Q \to Q \cup \{\text{term, div}\}$$

Nehmen an, Maschine ist im Zustand ϵ und hat Lesekopf auf erstem Zeichen von x und Maschine läuft.

1. Maschine rückt Lesekopf irgendwann über linkes Ende von x und ist dann im Zustand ϵ'

$$f_x(\epsilon) = \epsilon'$$

2. Maschine terminiert ohne über linkes Ende von x zu rücken.

$$f_x(\epsilon) = \text{term}$$

3. sonst

$$f_x(\epsilon) = \text{div}$$

$$x, y \in \Sigma^*$$

$$f_x = f_g \quad \Rightarrow \quad \forall m \in \Sigma^* : \quad wx \in L \quad \Leftrightarrow \quad wy \in L$$

$$L_f = \{x \in \Sigma^* \ \mid \ f_x = f\}$$

$$w \in \Sigma^* \quad F_w = \{f_x \mid wx \in L \text{ von M akzeptiert}\}$$
$$C_L(w) = \bigcup \{L_f \mid f \in F_w\}$$

$$F = \{f : Q \to Q \cup \{\text{div, term}\}\} \quad \text{endlich}$$

$\Rightarrow$ es gibt nur endlich viele $C_L(u)$'s

7 Abschlusseigenschaften der regulären Sprachen

7.1 Satz

Es seien L und L' reguläre Sprachen über Σ. Darum sind auch die folgenden Sprachen regulär:

1. $\bar{L} = \{w \in \Sigma^* \mid w \notin L\}$
2. $L\ L' = \{uv \mid u \in L,\ v \in L'\}$
3. $L \cup L'$
4. $L \cap L'$
5. $L^* = \{w_1 w_2 \dots w_k \mid k \in \mathbb{N}, w_i \in L,\ 1 \leq i \leq k\}$

Beweis

$$L \text{ regulär } \rightarrow \exists DEA\ M \text{ mit } L(M) = L$$
$$L' \text{ regulär } \rightarrow \exists DEA\ M' \text{ mit } L(M') = L'$$

o.B.d.A. nehmen wir an, M und M' seien vollständig spezifiziert.

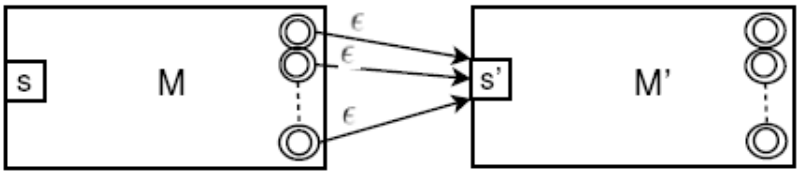
1. $\forall q \in Q,\ \forall a \in \Sigma,\ \exists_1 q' \in Q,\ (q, a, q') \in \Delta$
2. alle $q \in Q$ vom Startzustand erreichbar.

Wenn nicht vollständig spezifiziert:

1. $\tilde{Q} = Q \cup \{\perp\}$
   Es gibt keinen Übergang, der von q mit a wegführt
   $\rightarrow$ erzeuge zusätzlichen Übergang $(q, a, \perp)$
2. entferne nicht von s erreichbare Zustände

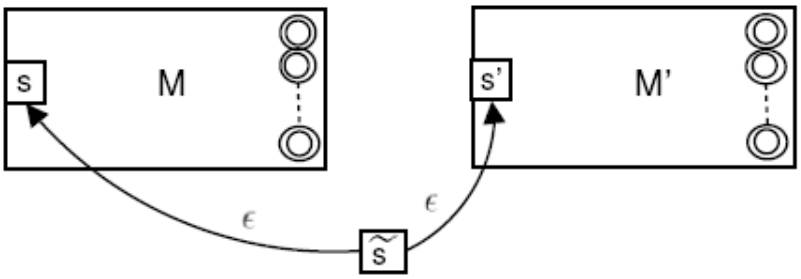
1. Maschine für $\bar{L}$

   Σ
   Q
   s
   $F = Q \setminus F$
   Δ

2. 

   Σ
   $Q \cup Q'$
   s
   F'
   $\Delta \cup \Delta' \cup \{(q, \epsilon, s') \mid q \in F\}$

3.



Σ
$Q \cup Q'$
$\tilde{s}$
$F \cup F'$
$\Delta \cup \Delta' \cup \{(\tilde{s}, \epsilon, s'), (\tilde{s}, \epsilon, s)\}$

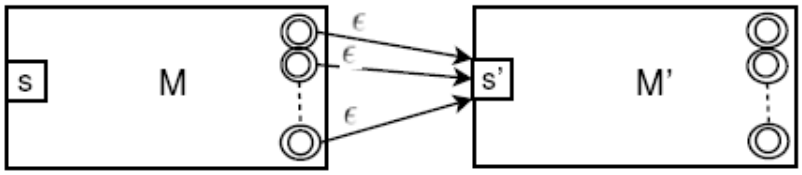
4.

$$A \bar{\cup} B = \bar{A} \cup \bar{B}$$
$$L \cap L' = \overline{\bar{L} \cup \bar{L}'}$$

Lasse M und M' parallel laufen.

Σ
$Q \times Q'$
(s, s')
$\tilde{F} = \{(q, q') \mid q \in F,\ q' \in F'\}$
$\tilde{\Delta} = \{(p, p'), a, (q, q') \mid (p, a, q) \in \Delta,\ (p', a, q') \in \Delta'\}$

5.



8 Entscheidungseigenschaften von regulären Sprachen

8.1 Satz:

„Mann kann entscheiden, ob für eine gegebene reguläre Sprache $L \subset \Sigma^*$ gilt $L = \Sigma^*$."
Gegeben DEA M, kann entschieden werden, ob $L(M) = \Sigma^*$.

8.1.1 Beweis

o.B.d.A M vollständig spezifiziert.
M akzeptiert $\Sigma^* \iff F = Q$

8.2 Satz

„Gegeben DIA M, kann entschieden werden, ob $L(M) = \{\}$, M akzeptiert $\{\} \iff$ alle von s erreichbaren Zustände sind nicht in F"

Frage

$M, M' \quad DEA's$
kann man entscheiden, aber $L(M) = L(M')$?

Antwort

JA! denn $L, L' \subseteq \Sigma^*$

$$L = L' \iff L \cup \bar{L'} = \Sigma^* \text{ und } L \cap \bar{L'} = \{\}$$

wegen Abschlußeigenschaften sind $L \cup \bar{L'}$ und $L \cap \bar{L'}$ regulär ...

9 „Minimierung“ von DEA’n

Problem

Gegeben: DEA $\tilde{M}'$ mit möglichst wenigen Zuständen, sodass $L(M) = L(\tilde{M})$.

$$\delta_q(w) = \{q' \mid q' \text{ von } q \text{ über Pfad mit Beschreibung } w \text{ erreichbar}\}$$
$$L_q = \{w \in \Sigma^* \mid \delta_q(w) \subseteq F\}$$

Falls $q \neq q'$ gilt:
$L_q = L_{q'}$ heißen q, q' ununterscheidbar und man kann auf einen der beiden verzichten.

Falls $L_q \neq L_{q'}$ dann heißen q und q' unterscheidbar.

Beachte: Unterscheidbarkeit ist eine Äquivalenzrelation auf Q

$$q \sim q' \Rightarrow q' \sim q$$
$$q \sim q$$
$$q \sim q' \wedge q' \sim q'' \Rightarrow q \sim q'$$

d.i. Äquivalenzklassen können zusammengefasst werden, und bilden Zustände des minimalen Automaten.

Wie bestimmt man alle unterscheidbaren Zustandspaare U ?

Regel 1

$$q \in F,\ q' \notin F \Rightarrow \{q, q'\} \in U$$

Regel 2

$$q, q' \in Q, a \in \Sigma : \begin{matrix}\{p\} = \delta(a) \\ \{p'\} = \delta_{q'}(a)\end{matrix} \{p, p'\} \in U \Rightarrow \{q, q'\} \in U$$

„Table-filling“ Algorithmus (zur Bestimmung von U)

1. Wende Regel 1 an, solange möglich
2. Wende Regel 2 an, solange möglich

Hinweis vom Autor

Es werden die Zustände verglichen die von dem jeweiligen Punkt erreichbar sind.

Behauptung

Wenn der Algorithmus terminiert, umfasst U genau alle unterscheidbaren Paare von Zuständen. Die restlichen Paare sind ununterscheidbar.

Beispiel

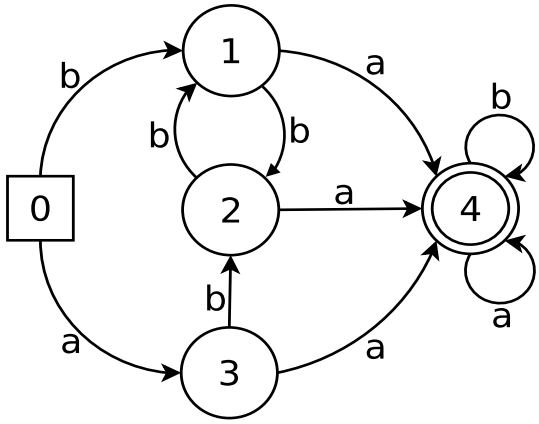
ZEICHNUNG SEBASTIAN WAINER 5.11.2004.4

	3	2	1	0
0	×	×		
1	×	×		
2	×			
3				

$$\{0,1\} \notin U$$

d.i. Zustände 0 und 1 sind ununterscheidbar und bilden eine „Äquivalenzklasse“ (Seite 25)

Beispiel: Keller-Automat



$$U = \{\{2,4\} \mid 0 \leq i \leq 3\}$$

0	■	■	■	■	■
1	×	■	■	■	■
2	×		■	■	■
3	×			■	■
4	×	×	×	×	■
	0	1	2	3	4

$$\{0,1\} \xrightarrow{a} \{3,4\} \in U$$
$$\Rightarrow \{0,1\} \in U$$

$$\{0,2\} \xrightarrow{a} \{3,4\} \in U$$
$$\Rightarrow \{0,2\} \in U$$

$$\{0,3\} \xrightarrow{a} \{3,5\} \in U$$
$$\Rightarrow \{0,3\} \in U$$

$$\{1,2\} \xrightarrow{a} \{4,4\} \notin U$$
$$\{1,2\} \xrightarrow{b} \{1,2\} \notin U$$

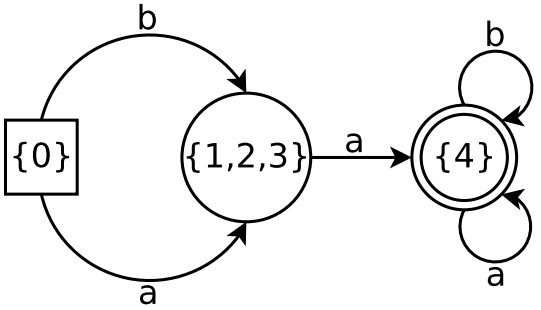
$$\{1,3\} \xrightarrow{a} \{4,4\} \notin U$$
$$\{1,3\} \xrightarrow{b} \{2,2\} \notin U$$

$$\{2,3\} \xrightarrow{a} \{4,4\} \notin U$$
$$\{2,3\} \xrightarrow{b} \{1,2\} \notin U$$

$$2 \quad \equiv \quad 3 \quad \equiv \quad 1$$

Äquivalenzklassen

$\{0\}, \quad \{1,2,3\}, \quad \{4\}$



9.1 Korrektheit

9.1.1 Lemma 0

$\forall\ \{p,q\} \in U$ gilt: p und q sind unterscheidbar; **d.i.**

$$L_p \neq L_q$$

Beweis

Stimmt nach Schritt 2, denn:

$$p \in F \Leftrightarrow \epsilon \in L_p$$
$$q \notin F \Leftrightarrow \epsilon \notin L_q$$

Regelanwendung in Schritt 3:

$$\{q,q'\} \in U \rightarrow L_q \neq L_{q'} \rightarrow \exists w \in L_q \vee w \notin L_{q'}$$

Regel

$\Rightarrow\ aw \in L_p \vee aw \notin L_{p'} \Rightarrow L_p \neq L_{p'}$
$\Rightarrow \{p,p'\} \in U$

9.1.2 Lemma 1

Sobald in Schritt 3 Reegel nicht mehr anwendbar ist, gilt für jedes Paar $p \neq q$ und $\{p,q\} \notin U$, dass $L_p = L_q$, also p und q sind NICHT unterscheidbar.

Beweis

Sei (mit U nach Terminierung des Algorithmus)

$$X = \{\{p,q\} \notin U \ \mid \ L_p \neq L_q\}$$

$X = \emptyset$ stimmt Lemma

$X \neq \emptyset$ Sei $w \in \Sigma^*$ das kürzeste Wort, das für irgenein $\{p,q\} \in X$ folgendes erfüllt: $w \in L_p,\ w \notin L_q$

$$p \overset{w_1}{\curvearrowright} p_1 \overset{w_2}{\curvearrowright} p_2 \overset{w_3}{\curvearrowright} p_3 \curvearrowright \ldots \overset{m}{\curvearrowright} p_m \quad \in F$$

$$q \overset{w_1}{\curvearrowright} q_1 \overset{w_2}{\curvearrowright} q_2 \overset{w_3}{\curvearrowright} q_3 \curvearrowright \ldots \overset{m}{\curvearrowright} q_m \quad \notin F$$

$p_1,\ q_1$ unterscheidbar!

Fall 1

$$\{p_1, q_1\} \in U$$

Algorithmus darf noch nicht terminiert haben: Regel im Schritt 3 noch anwendbar!

Fall 2

$$\{p_1, q_1\} \notin U \rightarrow \{p_1, q_1\} \in X \ (L_{p_1} \neq L_{q_1})$$

kürzer als w im Widerspruch zu Wahl von v.

10 Reguläre Ausrücke [q] Beschreibungsart für Sprachen

q	regulärer Ausdruck	$\underbrace{L(q)}_{\subseteq\Sigma^*}$
$\emptyset$	regulärer Ausdruck	$L(\emptyset) = \{\}$
ϵ	regulärer Ausdruck	$L(\epsilon) = \{\epsilon\}$
a	regulärer Ausdruck	$L(a) = \{a\} \quad \forall a \in \Sigma^*$
α, β	reguläre Ausdrücke	$\Rightarrow \alpha\beta,\ (\alpha + \beta),\ (\alpha)^*$ sind reguläre Ausdrücke

$$L(\alpha\beta) = L(\alpha)L(\beta) = \{uv \mid u \in L(\alpha),\ v \in L(\beta)\}$$
$$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$$
$$L\left((\alpha)^*\right) = L((a))^*$$

10.1 Satz

	$L = L(q)$	für irgendeinen regulären Ausdruck q
$\Leftrightarrow$	$L = L(M)$	für irgendeinen endlichen Automaten M

Beweis

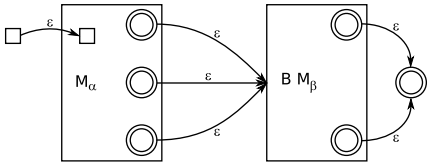
„⇒“

Gegeben: q, wohldefinierter endlicher Automat M_{rho}, mit $L(q) = L(M_q)$

$\emptyset \quad M_\emptyset \quad \square$

$\epsilon \quad M_\epsilon \quad \square$

$a \quad M_a \quad \square \overset{a}{\curvearrowright} \square$



„⇐“

Sei M ein deterministischer Endlicher Automat

$$Q = \{1, \ldots, n\} \qquad \Sigma$$
$$q = 1$$

G sei Übergangsgraph von M, F Endzustände

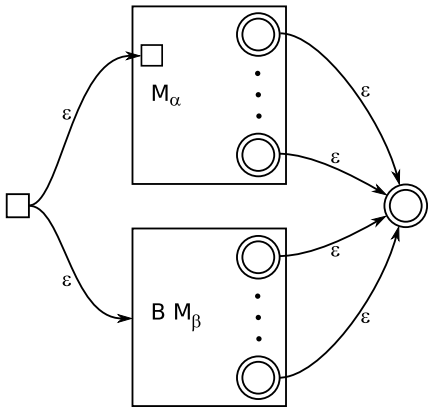
$R_{ij}^k \subset \Sigma^*$ Menge aller Beschriftungen von Pfaden in G mit Anfangszuständen i, Endknoten j und inneren Knoten $\leq k$.
$i \curvearrowright k_1 \curvearrowright \ldots \curvearrowright j$

$$L(M) = \bigcup_{q \in F} R_{iq}^k$$

Also wenn q_{ij}^k regulärer Ausdruck mit $L\left(q_{ij}^k\right) = R_{ij}^k$ dann

$$L(M) = L\left((q_{iq_1}^n + q_{iq_2}^n) + \ldots q_{iq_e}^n\right) \qquad F = \{q_1, \ldots, q_e\}$$

$$R_{ii}^0 = \{\epsilon\} \cup \bigcup \{\{a\}^* \mid (i, a, i) \in \Delta\}$$
$$R_{ij}^0 = \{a \in \Sigma \mid (i, a, j) \in \Delta\}$$



Annahme

R_{ij}^k bekannt für alle i, j

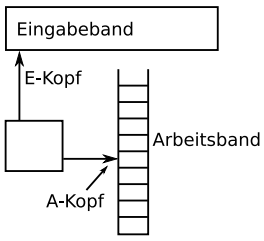
$$R_{ij}^{k+1} = R_{ij}^k \quad \cup \quad R_{i(k+1)}^k \left(R_{(k+1)(k+1)}^k\right)^* R_{i(k+1)}^k$$

$$i \overset{\rightsquigarrow}{\leq} k\ k+1 \overset{\rightsquigarrow}{\leq} k\ k+1 \overset{\rightsquigarrow}{\leq} k\ k+1 \overset{\rightsquigarrow}{\leq} k\ j$$

$$q_{ij}^{k+1} = q_{ij}^k + q_{i(k+1)}^k \left(q_{(k+1)(k+1)}^k\right)^* q_{i(k+1)}^k$$

$$L = \{w \in \{a, b\} \mid \#_a(w) \text{ gerade und } \#_b(w) \text{ ungerade}\}$$

11 Kellerautomaten



Eingabekopf	bleiben
	nach rechts rücken (nichts schreiben!!)
Arbeitskopf	bleiben
	nach unten rücken (löscht Zeichen)
	nach oben rücken und schreiben

11.1 Regeln: je nach Zustand

			neuer Zustand
Eingabekopf	*symbol*	$\longrightarrow$	Eingabekopf Bewegung
Arbeitskopf	*symbol*	$\longrightarrow$	Arbeitskopf Bewegung

11.2 Formalisierung

NKA	*Nichtdeterministischer Kellerautomat M*
Σ	Eingabealphabet
Γ	Arbeitsalphabet
Q	Zustandsmenge (endlich)
$s \in Q$	Startzustände
$F \subseteq Q$	Endzustände
$\in \in \Gamma$	Unterstes Symbol auf Arbeitsband

$$\Delta \subseteq \left(Q \times \underbrace{\Sigma^{\leq 1} \times \Gamma}_{\Sigma^0 \cup \Sigma^1} \right) \times \Gamma^* \times Q \qquad \text{Regeln}$$

11.3 informelle Erklärung

$$(q, a, A, U, q') \in \Delta$$

Wenn in Zustand q

Eingabekopf liest a
Arbeitskopf liest A

neuer Zustand q'

Eingabekopf rückt nach rechts
Arbeitskopf ersetzt A durch U
$U = \epsilon \rightarrow$ Arbeitskopf rückt nach rechts unten
$U = B \rightarrow$ Arbeitskopf ersetzt A durch B_k und bleibt
$U = B_1 B_2 \dots B_k \rightarrow$ Arbeitskopf ersetzt A durch B_1, rückt nach oben und schreibt $B_{k-1} \dots B_1$

11.4 Übereinkunft

Alphabetsanfangsbuchstaben		
klein	a,b,c,...	stehen für einzelne Symbole in Σ
groß	A,B,C,...	stehen für einzelne Symbole in Γ
Alphabetsendbuchstaben		
klein	z,y,x,...	$\in \Sigma^*$
greoß	Z,Y,X,...	$\in \Gamma^*$

11.5 Welche Worte werden von M akzeptiert?

Konfigurationen von M

$$K_M = Q \times \Sigma^* \times \Gamma^*$$

M in Konfiguration (q, u, W) bedeutet:
M und Zustand q
Arbeitsbandinhalt W
Eingabebandinhalt über und rechts vom Lesekopf ist u

11.6 Relation

$$\overset{M}{\longmapsto} \text{ auf } K_M$$

$$k \overset{M}{\longmapsto} k'$$

soll ausdrücken: aus Kopf k kommt M durch 1 Regelanwendung.

$$(q, ax, BY) \overset{M}{\longmapsto} (q', x, UY) \quad \textbf{g.d.w} \quad (q, a, B, U, q') \in \Delta$$

$$(q, x, BY) \overset{M}{\longmapsto} (q', x, UY) \quad \textbf{g.d.w} \quad (q, \epsilon, B, U, q') \in \Delta$$

Anfangskonfiguraton von M für $x \in \Sigma^*$

$$(s, x, \in) = Init_M(x)$$

Endkonfiguration von M

$$Fin_M = \{(q, \in, U) \mid q \in F\}$$

M akzeptiert $x \in \Sigma^*$ genau dann wenn:
$\exists\, k_0, k_1, \ldots, k_m \in K_M$
mit $k_0 = Init_M(x)$ und $k_m \in Fin_M$
und $k_i \overset{M}{\longmapsto} k_{i+1}$ für $0 \leq i < m$

$$Init_M(x) \overset{M}{\longmapsto} k_1 \overset{M}{\longmapsto} k_2 \overset{M}{\longmapsto} \ldots \overset{M}{\longmapsto} k_{m-1} \overset{M}{\longmapsto} k_m \in Fin_M$$

Äquivalent dazu

Sei $\overset{*}{\underset{M}{\longmapsto}}$ die reflexive, transitive Hülle von $\overset{M}{\longmapsto}$.

$$M \text{ akzeptiert } x \quad \Leftrightarrow \quad \exists k \in Fin_M : \; Init_M(x) \overset{*}{\underset{M}{\longmapsto}} k$$

$$L(M) = \{x \in \Sigma^* \mid M \text{ akzeptiert } x\}$$

Beispiel

NKA für die Sprache $\left\{ww^R \mid w \in \{a,b\}^*\right\}$

$$Q = \{s, q, f\} \qquad s = s \qquad F = \{f\}$$

$$\Sigma = \{a, b\} \qquad \Gamma = \{€, A, B\} \qquad €$$

$$\begin{aligned}\Delta = \{ & \\ & (s, a, €, A€, s),\\ & (s, b, €, B€, s),\\ & (s, a, A, AA, s),\\ & (s, a, B, AB, s),\\ & (s, b, A, BA, s),\\ & (s, b, B, BB, s),\\ & (s, \epsilon, A, A, q),\\ & (s, \epsilon, B, B, q),\\ & (s, \epsilon, €, €, q),\\ & (q, a, A, \epsilon, q),\\ & (q, b, B, \epsilon, q),\\ & (q, \epsilon, €, \epsilon, f) \qquad \}\end{aligned}$$

Eingabebeispiel 1

$$baab$$

$$\begin{aligned}(s, baaab, €) &\overset{M}{\longmapsto} (s, aab, B€)\\ &\overset{M}{\longmapsto} \textit{Regel } 4\\ &(s, ab, AB€)\\ &\overset{M}{\longmapsto} \textit{Regel } 7\\ &(q, ab, AB€)\\ &\overset{M}{\longmapsto} \textit{Regel } 10\\ &(q, b, B€)\\ &\overset{M}{\longmapsto} \textit{Regel } 11\\ &(q, \epsilon, €)\\ &\overset{M}{\longmapsto} \textit{Regel } 12\\ &(f, \epsilon, \epsilon)\\ &\in Fin_M\end{aligned}$$

Eingabebeispiel 2

$$baaba$$

$$
\begin{array}{lll}
(s, baaab, \in) & \overset{M}{\longmapsto} & (s, aaba, B\in) \\
& & \overset{M}{\longmapsto} \textit{ Regel } 4 \\
& & (s, aba, AB\in) \\
& & \overset{M}{\longmapsto} \textit{ Regel } 7 \\
& & (q, aba, AB\in) \\
& & \overset{M}{\longmapsto} \textit{ Regel } 10 \\
& & (q, ba, B\in) \\
& & \overset{M}{\longmapsto} \textit{ Regel } 11 \\
& & (q, a, \in) \\
& & \overset{M}{\longmapsto} \textit{ Regel } 12 \\
& & (f, a, \epsilon) \\
& & \notin Fin_M
\end{array}
$$

(f, a, ϵ) ist kein gültiger Endzustand
Der Automat hat nicht die ganze Eingabe gelesen.

11.7 Definition: Einfacher nichtdeterministischer Kellerautomat (NKA)

alle Regeln haben Form

$$\left(q, \begin{matrix} a \\ \epsilon \end{matrix}, B, U, q'\right)$$

mit

$$U = \begin{cases} \epsilon & (POP) \\ A & \text{ersetzen} \\ CB & PUSH\ C \end{cases}$$

11.8 Lemma

Für jeden $NKA\ M$ gib es einen einfachen $NKA\ M'$ mit

$$L(M) = L(M')$$

Beweis

Ersetze jede Regel

$$\left(q, \begin{matrix} a \\ \epsilon \end{matrix}, B, U, q'\right)$$

von M mit $|U| > 1$ durch Regeln

$$
\begin{array}{l}
\left(q, \begin{matrix} a \\ \epsilon \end{matrix}, B, U_k, q_k\right) \\
(q, \epsilon, U_k, U_{k-1}, q_{k-1}) \\
(q_i, \epsilon, U_k, U_{i-1}, q_{i-1}) \quad \text{für } 1 < i \leq k
\end{array}
$$

$$q_2, \ldots, q_{k-1}$$

sind neue Zustände nur für die Regel.

Konfiguration

$$\left(\underbrace{Q}_{\text{Zustand}} \times \underbrace{\Sigma^*}_{\text{Eingabewert}} \times \underbrace{\Gamma^*}_{\text{Kellerinhalt}}\right)$$

$$(q, ax, AU) \overset{M}{\longmapsto} (q', ax, BU) \text{falls } (q, \epsilon, A, B, q')$$

$\overset{M}{\longmapsto}^*$ reflexive transitive Hülle

$$k \overset{M}{\longmapsto}^* k'$$

$$kw \in L(U) \Longleftrightarrow (s, w, \in) \overset{M}{\longmapsto}^* (f, \epsilon, U)$$

12 Pumping Lemma für NKA Sprachen

12.1 Lemma

Jede NKA Sprache hat die folgende „Pumping Eigenschaft“

$$P: \ \exists N \in \mathbb{N} \quad \forall z \in K, \ |z| > N \quad \exists \text{Unterteilung} \quad z = uvwxy$$

$$|vwx| \leq N \qquad |vx| \geq 1$$

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

Um zu zeigen, dass L keine NKA Sprache ist, genügt es zu zeigen, dass L nicht die Eigenschaft P besitzt. Also für L gilt $\neg P$:

$$\forall N \in \mathbb{N} \quad \exists z \in L, \ |z| > N \quad \forall z = uvwxy \quad \exists z \in \mathbb{N}: \ uv^i wx^i y \notin L$$

$$|vwx| \leq N \qquad |vx| \geq 1$$

Spiel

1. Gegner gibt $N \in \mathbb{N}$ vor
2. Wir produzieren $z \in L, \ |z| > N$
3. Gegner produziert Unterteilung $z = uvwxy$
4. Wir produzieren $i \in \mathbb{N}$, sodass $uv^i wx^i y \notin L$

12.2 Lemma

Die Sprache $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist keine NKA Sprache.

Beweis

Zeige L hat nicht Eigenschaft P

1. Gegner produziert N
2. Wir wählen $z = a^N b^N c^N \in L$
3. Gegner produziert Unterteilung $z = uvwxy, \ |vwx| \leq N, \ |vx| \geq 1$
   also vwx enthält keine a's (*Fall 1*)oder keine c's (*Fall 2*).
4. Wir wählen i

   Fall 1

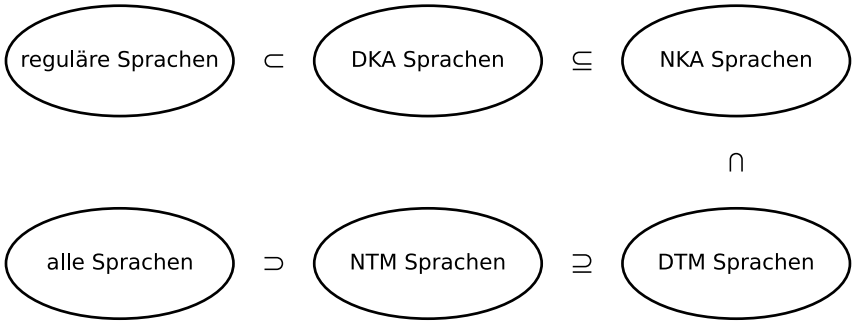
   $uv^i wx^i y = uwy$
   $|uwy| < 3N$ also uwy enthält N a's
   weil $|vx| \geq 1$ also $uwy \notin L$

   Fall 2

   analog

12.3 Korrolar

NKA Sprachen sind echte Teilmengen der DTM Sprachen.
(Wenn man glaubt, dass L von einer DTM erkannt werden kann)



12.3.1 Beweis des Pumping Lemmas

$$L \text{ NKA Sprache}$$

$$\text{NKA } M \quad \begin{matrix} \Sigma \\ Q \\ \Gamma \\ q \in Q \\ F \subset Q \\ \text{€} \in \Sigma \\ \Delta \end{matrix} \quad \text{mit } M(M) = L$$

o.B.d.A
M einfach, M „vollständig“
$|F| = 1$ und jede akzeptierende Berechnung endet mit leerem Keller.

Statusgraph von M ($\mathcal{S}G(M)$)

Status von M $\quad (q, U) \in Q \times \Sigma^*$

$\mathcal{S}G(M)$ Knotenweg $\quad Q \times \Gamma^*$
gerichtete Kanten
beschriftet mit
String aus $\Sigma^{\leq 1}$

$$\begin{pmatrix} q, & A \\ & W \end{pmatrix} \xrightarrow{w} (q', W) \quad \text{g.d.w.} \quad (q, y, A, \epsilon, q') \in \Delta$$

$$\begin{pmatrix} q, & A \\ & W \end{pmatrix} \xrightarrow{y} \begin{pmatrix} q', & B \\ & W \end{pmatrix} \quad \text{g.d.w.} \quad (q, y, A, B, q') \in \Delta$$

$$\begin{pmatrix} q, & A \\ & W \end{pmatrix} \xrightarrow{y} \begin{pmatrix} & B \\ q', & A \\ & W \end{pmatrix} \quad \text{g.d.w.} \quad (q, y, A, BA, q') \in \Delta$$

Wir schreiben

$$(q, W) \overset{u \leq \Sigma^*}{\times \twoheadrightarrow} (q', W')$$

wenn es einen Pfad

$$(q_0, W_0) \overset{u_1}{\to} (q_1, W_1) \overset{u_2}{\to} \dots \overset{u_m}{\to} (q_m, W_m)$$

in $\mathcal{S}G(M)$ gibt mit

$$\begin{matrix} (q_0, W_0) & = & (q, W) \\ (q_m, W_m) & = & (q', W') \end{matrix}$$

und $u_1 u_2 \dots u_m = u$

Beachte

M akzeptiert $u \in \Sigma^*$ genau dann wenn

$$(\underbrace{s, \in}_{\text{Anfangsstatus}}) \times \overset{u}{\twoheadrightarrow} (\underbrace{f, \epsilon}_{\text{Endstatus}})$$

12.4 Definition

Ein Pfad

$$(q_0, W_0) \overset{u_1}{\to} (q_1, W_1) \overset{u_2}{\to} \ldots \overset{u_m}{\to} (q_m, W_m) \quad \in \mathcal{SG}(M)$$

heißt Kellersuffix U erhaltend, wenn fpr $0 \leq i \leq m$ gilt

$$W_i = \begin{matrix} W_i \\ U \end{matrix}$$

$$(q_0, W_0) \overset{\overset{v}{\twoheadrightarrow}}{U} (q_m, W_m)$$

12.5 Hauptbeobachtung

$$\begin{pmatrix} & X & \\ q, & A & \\ & Y & \end{pmatrix} \overset{v}{\twoheadrightarrow} \begin{matrix} A \\ Y \end{matrix} \begin{pmatrix} & X' & \\ q', & A & \\ & Y & \end{pmatrix} \Longrightarrow \forall W \in \Gamma^* : \begin{pmatrix} & X & \\ q, & A & \\ & W & \end{pmatrix} \overset{v}{\twoheadrightarrow} \begin{matrix} A \\ W \end{matrix} \begin{pmatrix} & X' & \\ q', & A & \\ & W & \end{pmatrix}$$

Beweisidee fürs Pumping Lemma

Numm an fur $z \in L$ gibt es akzeptierten (?) Pfad in $\mathcal{SG}(M)$ mit folgender Unterteilung:

$$(s, \in) \overset{u}{\twoheadrightarrow} \begin{pmatrix} & A & \\ q, & U & \end{pmatrix} \overset{v}{\twoheadrightarrow} \begin{matrix} A \\ U \end{matrix} \begin{pmatrix} & A & \\ q, & X & \\ & U & \end{pmatrix} \overset{w}{\twoheadrightarrow} \begin{matrix} A \\ X \\ A \\ U \end{matrix} \begin{pmatrix} & A & \\ q', & X & \\ & U & \end{pmatrix} \overset{x}{\twoheadrightarrow} \begin{matrix} A \\ A \\ U \end{matrix} \begin{pmatrix} & A & \\ q', & U & \end{pmatrix} \overset{y}{\twoheadrightarrow} (f, \epsilon)$$

Tafelbild wird noch nachgereicht, dies ist erst ein Teil!

Muss zeigen: Wenn z lang genug, dann gibt es so eine Konstellation!

12.6 Definition

Ein Kapitel eines akzeptierten (?) Pfades in $\mathcal{SG}(M)$ ist eine Kellersuffixerhalnder Teilpfad maximaler Länge

$$\begin{matrix} C \\ BBB \end{matrix} \boxed{\begin{matrix} A & A \\ B & \\ \in & \end{matrix}} \quad C$$

Beachte

1. Verschiedene Kapitel eines Pfades sind entweder verschachtelt, oder disjunkt.
2. (q, U) Knoten in akzeptierten (?) Pfad P
   $\neq$ **Kapitel**, die (q, U) enthalten, entspricht $|U|$

12.6.1 Typ eines Kapitels

$$\begin{pmatrix} q, & A \\ & U \end{pmatrix} \longrightarrow \begin{pmatrix} q', & A \\ & U \end{pmatrix} \qquad (q, A, q')$$

Die Anzahl der Typen ist $\leq |Q| \cdot |P| \cdot |Q| = t$.

$$PICTURE$$

Was passiert wenn bei Akzepterung eines langen Wertes z Stack immer $\leq t$ bleibt?

$$\Rightarrow \# \text{ der verschiedenen Status}$$
$$|Q| \sum_{0 \leq i \leq t} |\Gamma|^i = N$$

Wenn Stack ¿ t, gibt es Typenwiederholungen auf Stack!
Betrachte Punkt, wo Stack am höchsten
und betrachte oberste Typenwiederholung (muss in obersten $t+1$ Stellen passieren).
$v = x = \epsilon$ wird verhindert, indem man kürzesten akzeptierten (?) Pfad für z betrachtet.

12.7 Satz

$L \subset \Sigma^*$ sei NKA Sprache
$|\Sigma| = 1$
$\Rightarrow L$ ist regulär.

Beweis

Hilfslemma 1

$\exists N \forall z \in L$ mit $|z| > N$: $za^{N!} \in L$

Hilfskorollar

$\exists N \forall z \in L$ mit $|z| > N$: $\forall i \in \mathbb{N}: \; za^{iN!} \in L$

$$PICTURE$$

$z \in L, \; |z| > N_i$ definiere

$$m_z = \min \left\{ m \in \mathbb{N} \;\middle|\; a^m \in L, \; z = a^m \cdot a^{iN!} \text{ für irgendein } i \in \mathbb{N} \right\}$$
$$M = \{ m_z \mid z \in L, \; |z| > N \}$$

Behauptung

$$|M| \leq N!$$
$$z, z' \in L, \; |z| > N, \; |z'| > N$$
$$|z| \bmod N! = |z'| \bmod N!$$
$$\rightarrow m_z = m_{z'}$$
$$\Longleftrightarrow |z| = |z'| + kN! \quad k \in \mathbb{Z}$$

$$L = \underbrace{\underbrace{\{w \in L \mid |w| \leq N\}}_{\text{endlich, also regulär}} \cup \bigcup_{m \in M} \underbrace{\{a^m a^{iN!} \mid u \in \mathbb{N}\}}_{\text{regulär}}}_{\text{regulär}}$$

$$\underbrace{aaa \ldots a}_{m} \underbrace{(a \ldots a)^*}_{N'}$$

Beweis von Hilfslemma 1

L NKA Sprache $\Rightarrow$ L hat Pumping Eigenschaf

$$\exists N \forall z \in L\ |z| > N: \quad z = uvwxy,\ |vx| \geq 1,\ |vwx| \leq N \quad \forall i \in \mathbb{N}:\ uv^i wx^i y \in L$$

$$z = a^{|u|}a^{|v|}a^{|w|}a^{|x|}a^{|y|}$$

Wähle $i = \frac{N!}{p} + 1 \ \in \ \mathbb{N}$

Sorry, dieser Beweis wird hier abgebrochen - Null Struktur, Null Verständnis - wer ihn hat, ich freu mich dann über Post

13 Deterministische Kellerautomaten

Ein NKA ist deterministisch (also ein DKA),
„wenn in jeder Situation höchstens eine Regel anwendbar ist“.

$$\begin{matrix}\Sigma\\ Q\\ \Gamma\\ s\\ F\\ \in\!\!\!\!/\\ \Delta\end{matrix} \qquad \forall(q,a,A) \in Q \times \underbrace{\Sigma^{\leq 1}}_{\{a,\epsilon\}} \times \Gamma$$

gibt es höchstens ein

$$(U,q') \in \Gamma^* \times Q \quad \text{mit} (q,x,A,U,q') \in \Delta$$

und

$$(q,\epsilon,A,U,q') \in \Delta \Rightarrow \forall a \in \Sigma \forall (\bar{U},p) \subset \Gamma^* \times Q : \ (q,a,A,\bar{U},p) \notin \Delta$$

Wollen zeigen

$$\boxed{\begin{matrix}\text{DKA}\\ \text{Sprache}\end{matrix}} \subsetneq \boxed{\begin{matrix}\text{NKA}\\ \text{Sprache}\end{matrix}}$$

also es gibt Sprache L', sodass L' NKA Sprache aber keine DKA Sprache.

Idee

1. Zeige, dass DKA Sprachen unter Komplementbildung abgeschlossen sind. Also L DKA Sprache $\Rightarrow \bar{L}$ DKA Sprache

2. Gib eine NKA Sprache L an mit $\bar{L}$ nicht NKA Sprache.
   (also auch keine DKA Sprache)

$$L_{abc} = \{a^n b^n c^n \ \mid \ n \in \mathbb{N}\}$$

ist keine NKA Sprache

$$\begin{aligned} L &= \bar{L_{abc}} \\ \bar{L} &= L_{abc} \end{aligned}$$

Beweis

$$\begin{aligned} \bar{L_{abc}} &= \{a^i b^j c^k \ \mid \ i \neq j\} && \text{NKA Sprache} \\ &= \{a^i b^j c^k \ \mid \ i \neq k\} && \text{NKA Sprache} \\ &= \{w \text{ hat nicht Form } a^i b^j c^k\} && \text{regulär, also NKA Sprache} \end{aligned}$$

Idee

M DKA für L
tausche Endzustände
funktioniert „fast“

Beispiel

$$\Sigma = \{a, k\} \qquad Q = \{s, p, q, f\} \qquad \Gamma = \{€\}$$

$$\Delta: \quad \begin{array}{l}(s, a, €, €, f)\\ (s, b, €, €, q)\\ (f, \epsilon, €, €, p)\end{array}$$

akzeptiert a
aa wird nicht akzeptiert (nicht vollständig konsumiert)
b wird nicht akzeptiert, weil $q \notin F$
Es gibt Pfad:

$$(s, €) \xrightarrow{a} (f, €) \xrightarrow{\epsilon} (p, €)$$

13.1 Satz

L DKA Sprache $\Rightarrow \bar{L}$ DKA Sprache

13.2 Korrolar

$\bar{L_{abc}} \in$ NKA Sprachen $\setminus$ DKA Sprachen

13.3 Lemma 1

Für jede DKA Sprache L existiert ein DKA $\hat{M}$ mit $L(\hat{M}) = L$ und $\hat{M}$ konsumert jede Eingabe vollständig

13.4 Lemma 2

Es sei $\hat{M}$ ein DKA, der jede Eingabe vollständig akzeptiert. Aus $\hat{M}$ lässt sich ein DKA $\bar{M}$ bauen, mit $L(\bar{M}) = L(\hat{M})$.

13.4.1 Beweis: (Satz)

$$L \text{ DKA Sprache} \overset{\text{Lemma 1}}{\Longrightarrow} \exists \hat{M} \text{ mit } L(\hat{M}) = L \text{ und } \hat{M} \text{ konsumiert jede Eingabe}$$

$$\overset{\text{Lemma 2}}{\Longrightarrow} \exists \bar{M} \text{ mit } L(\bar{M}) = L(\bar{\bar{M}}) = \bar{L} \Longrightarrow \bar{L} \text{ ist DKA Sprache}$$

13.4.2 Beweis Lemma 2

$$\hat{M} = (\Sigma, \Gamma, Q, s, F€, \Delta)$$

DKA konsumfreudig!

Idee

$$\bar{M} = (\Sigma, \Gamma, Q, s, F€, \Delta)$$

Gegenbeispiel

$$\begin{array}{c}\Sigma = \{a, b\}\\ \Gamma = \{€\}\\ Q = \{s, f\}\\ F = \{f\}\end{array} \quad \Delta \quad \begin{array}{c}(s, a, €, €, f)\\ (s, b, €, €, s)\\ (f, \epsilon, €, €, s)\end{array}$$

akzeptiert alle Strings, die in a enden.

Beispiel

aba von M akzeptiert
aba wird auch von $\bar{M}$ akzeptiert

$$(s, \in) \xrightarrow{a} (f, \in) \xrightarrow{\epsilon(?)} (s, \in) \xrightarrow{b} (s, \in) \xrightarrow{a} (f, \in) \xrightarrow{\epsilon} (s, \in) \in Q \backslash F$$

also Endzustand von $\bar{M}$

Problem

Am Berechungsende kann es Folge von ϵ Übergängen geben, die dich Endzustände und Nicht-Endzustände führt.

Idee

Merke dir, ob seit Konsum des letzten Eingabezeichens ein Endzustand erreicht wurde

$$\bar{Q} = Q \times \{1, 2, 3\}$$

$(q, 1)$	bedeutet $\hat{M}$	ist im Zustand q seit letztem Eingabekonsum wurde ein Endzustand erreicht.
$(q, 2)$	bedeutet $\hat{M}$	ist im Zustand q seit letztem Eingabekonsum wurde KEIN Endzustand erreicht. also es kann
$(q, 3)$	bedeutet $\hat{M}$	ist im Zustand q seit letztem Eingabekonsum wurde KEIN Endzustand erreicht. und um nächsten Schritt muss ein Eingabezeichen konsumiert werden.

auch kein Endzustand über weitere ϵ Übergänge erreicht werden.

$$\bar{s} = \begin{cases} (s, 1) & s \in F \\ (s, 2) & s \notin F \end{cases} \qquad \bar{M} = \left(\Sigma, \Gamma, \bar{Q}, \bar{s}, \bar{F}, \in, \bar{\Delta}\right)$$

$$\bar{F} = \{(q, 3) \mid q \in Q\}$$

$$\bar{\Delta} = (q, \epsilon, A, U, p) \in \Delta$$

$p \in F$	$((q, 1), \epsilon, A, U, (p, 1)) \in \bar{\Delta}$
	$((q, 2), \epsilon, A, U, (p, 1)) \in \bar{\Delta}$
$p \notin F$	$((q, 1), \epsilon, A, U, (p, 1)) \in \bar{\Delta}$
	$((q, 2), \epsilon, A, U, (p, 1)) \in \bar{\Delta}$
()	$((q, 2), \epsilon, A, A, (q, 3)) \in \bar{\Delta}$
$p \in F$	$((q, 3), a, A, U, (p, 1)) \in \bar{\Delta}$
	$((q, 1), a, A, U, (p, 1)) \in \bar{\Delta}$
	$((q, 2), a, A, U \ldots$ Konflikt mit ()
$p \notin F$	$((q, 3), a, A, U, (p, 2)) \in \bar{\Delta}$
	$((q, 1), a, A, U, (p, 2)) \in \bar{\Delta}$

$\bar{M}$ akzeptiert $z \in \Sigma^*$, genau dann wenn $\hat{M}$ z nicht akzeptiert.

13.4.3 Beweis zu Lemma 1

Es sei $M = (\Sigma, \Gamma, Q, s, F, \in, \Delta)$ ein DKA für L. Wie kann es sein, dass M nicht die gesamte Eingabe konsumiert?

1. es fehlt anwendbare Regel
   „Friedhofzustand“ übergehen, in dem Rest des Wortes gelesen wird

2. Keller leer
   zusätzliches neues Kellerbodensymbol, zu Friedhof, wenn je erreicht!
3. Es gibt Zyklus von ϵ Übergängen
   wenn so ein Zyklus beginnen würde, gehe zu Friedhof

$$\hat{M} = \left(\Sigma, \hat{P}, \hat{Q}, \hat{s}, \hat{F}, \hat{\in}, \hat{\Delta}\right)$$
$$\hat{\Gamma} = \Gamma \cup \left\{\hat{\in}\right\}$$
$$\hat{Q} = Q \cup \left\{\hat{s}, \hat{\in}, \hat{f}\right\}$$
$$\hat{F} = F \cup \left\{\hat{f}\right\}$$
$$\hat{\in} \notin \Gamma$$

$$\hat{\Delta} : \left(\hat{s}, \in, \in\hat{\in}, s\right) \in \hat{\Delta} \text{ zusätlicher Kellerboden}$$

$$\forall q \in Q : \left(q, \epsilon, \hat{\in}, \hat{\in}, c\right) \in \hat{\Delta} \text{ neuer Keller leer zu Friedhof}$$

$\forall q \in Q,\ \forall A \in \Gamma$:Falls in Status (q, AU)keine Regel vorhanden
$(\forall U \in \Gamma^*)$

$$(q, a, A, A, c) \in \hat{\Delta}$$
$$\forall a \in \Sigma\,(c, a, A, A, c) \in \hat{\Delta}$$
$$\forall A \in \Gamma$$

$\forall q \in Q,\ \forall A \in \Gamma$:Falls es aus Status (q, A)beliebig lange folgenden von ϵ Übergängen gibt

mathematisch wohldefiniert, aber nicht offensichtlich „entscheidbar“

dann

$(q, \epsilon, A, A, c) \in \hat{\Delta}$falls in der Folge kein Endzustand vorkommt.
$(q, \epsilon, A, A, \hat{f}) \in \hat{\Delta}$falls in der Folge ein Endzustand erreicht wird.

für Existenz $\hat{M}$ ist „entscheidbar“ irrelevant.

14 Abschlußeigenschaften von NKA Sprachen

14.1 Satz

L, L' NKA Sprachen, R reguläre Sprache

1. $L \cup L'$ NKA
2. $L \cap R$ NKA
3. $L \cdot L'$ NKA
4. L^R NKA
5. $L \cap L'$ i.A. nicht NKA
6. $\bar{L}$ i.A. nicht NKA

14.2 Beweis

M NKA für L
M' NKA für L'
N DEA für R

1. Baue Maschine mit neuem Startzustand $\bar{s}$ aus $\bar{s}$ ϵ Übergängen zu $\begin{matrix} s \\ \bar{s} \end{matrix}$ und M, M'
2. Idee, lasse M und N „parallel“ laufen
3. lasse M bei Akzeptanz Keller leer machen
   ϵ Übergänge von Endzuständen von M zu s'
4. „trivial“
5. $L_{abc} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist keine NKA Sprache
   $\underbrace{\{a^m b^n c^n \mid m, n \in \mathbb{N}\}}_{\text{NKA}}$ $\underbrace{\{a^n b^n c^k \mid n, k \in \mathbb{N}\}}_{\text{NKA}}$
6. schon bewiesen

15 Grammatiken anderer Mechanismus zur Sprachspezifikation

$\mathcal{G}$	Σ	Sprachalphabet	„Terminale“
	V	Variablen	„Nicht Terminale“
	$S \in V$	Startsymbol	

$$P \subset (\Sigma \cup V)^* \vee (\Sigma \cup V)^* \times (\Sigma \cup V)^*$$

Produktionen, Regeln.

15.0.1 Schreibweise

$$(\alpha, \beta) \in P \qquad \text{schreibt man} \qquad \alpha \to \beta$$

15.1 Beispiel

$$\mathcal{G} = (\Sigma, V, S, P) \quad \Sigma = \{a, b, c\} \quad V = \{A, B, C, S\}$$

$$\begin{aligned} P = \{\ & S \to aSBC && 1 \\ & S \to aBC && 2 \\ & CB \to BC && 3 \\ & aB \to ab && 4 \\ & bB \to bb && 5 \\ & bC \to bc && 6 \\ & cC \to cc && 7 \\ \} & \end{aligned}$$

$\mathcal{G}$ definiert Relation auf $(V \cup \Sigma)^*$

$$u \overset{G}{\Longrightarrow} v \qquad \text{falls } u = xyz,\ v = xy'z,\ y \to y' \in P$$

„aus u leutet sich v ab“
„Ableitungsschritt“ - „Derivationsschritt“

$$\overset{G}{\Longrightarrow}^* \quad \text{reflexsiv transitive Hülle von} \quad \overset{G}{\Longrightarrow}$$

$$u \overset{G}{\Longrightarrow}^* v$$

heißt $\exists u_0, u_1, \ldots, u_k$ so dass $u_{i-1} \overset{G}{\Longrightarrow} u_i$ für $1 \le i \le k$

$$u_0 \overset{G}{\Longrightarrow} u_1 \overset{G}{\Longrightarrow} u_2 \overset{G}{\Longrightarrow} \ldots \overset{G}{\Longrightarrow} u_k$$

„Ableitung“ - „Derivation“

nach Grammatik spezifizierte Sprache

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

15.2 Beispiel

$$
\begin{aligned}
S &\overset{G}{\Longrightarrow}_1 a\underline{S}BC \\
&\overset{G}{\Longrightarrow}_1 aa\underline{S}BCBC \\
&\overset{G}{\Longrightarrow}_2 aaa\underline{BC}\underline{BC}BC \\
&\overset{G}{\Longrightarrow}_3 aa\underline{aB}BCCBC \\
&\overset{G}{\Longrightarrow}_4 aa\underline{ab}BC\underline{CB}C \\
&\overset{G}{\Longrightarrow}_3 aaabB\underline{CB}CC \\
&\overset{G}{\Longrightarrow}_3 aaa\underline{bB}BCCC \\
&\overset{G}{\Longrightarrow}_5 aaabb\underline{bB}CCC \\
&\overset{G}{\Longrightarrow}_5 aaabbbCCC \\
&\overset{G}{\Longrightarrow}_6 aaabbb\underline{cC}C \\
&\overset{G}{\Longrightarrow}_7 aaabbbc\underline{cC} \\
&\overset{G}{\Longrightarrow}_7 aaabbbccc
\end{aligned}
$$

Behauptung

$$L(\mathcal{G}) = \{a^n b^n c^n \mid n \geq 1\} = L_{abc}$$

Beweis

i)

$z \in L_{abc} \Rightarrow z \in L(\mathcal{G})$
$z = a^n b^n c^n$
Gib Ableitung für z an

$$
\begin{aligned}
S &\overset{G}{\Longrightarrow}_1 aSBC \\
&\overset{G}{\Longrightarrow}_{n-2\ Regel\ 1} \underbrace{a\ldots a}_{n-1} S \underbrace{BCBC\ldots BC}_{n-1} \\
&\overset{G}{\Longrightarrow}_{Regel\ 2} \underbrace{a\ldots a}_{n} \underbrace{BCBC\ldots BC}_{n} \\
&\overset{G}{\Longrightarrow}_{Regel\ 3} \underbrace{a\ldots a}_{n} \underbrace{B\ldots B}_{n} \underbrace{C\ldots C}_{n} \\
&\overset{G}{\Longrightarrow}_{Regel\ 4} \underbrace{a\ldots a}_{n} b \underbrace{B\ldots B}_{n-1} \underbrace{C\ldots C}_{n} \\
&\overset{G}{\Longrightarrow}_{Regel\ ?} \underbrace{a\ldots a}_{n} \underbrace{b\ldots b}_{n} \underbrace{C\ldots C}_{n} \\
&\overset{G}{\Longrightarrow}_{Regel\ 6} \underbrace{a\ldots a}_{n} \underbrace{b\ldots b}_{n} c \underbrace{C\ldots C}_{n-1} \\
&\overset{G}{\Longrightarrow}_{n-mal\ Regel\ 7} \underbrace{a\ldots a}_{n} \underbrace{b\ldots b}_{n} \underbrace{c\ldots c}_{n}
\end{aligned}
$$

i)

$Z \in L(\mathcal{G}) \Rightarrow z \in L_{abc}$, also z hat Form $a^n b^n c^n$

Behauptung 1

$S \overset{G}{\Longrightarrow}^* \alpha$, dann

$$\#_a(\alpha) = \#_b(\alpha) + \#_B(\alpha) = \#_c(\alpha) + \#_C(\alpha)$$

Beweis
Gilt für $\alpha = S$ und wird durch jeden Ableitungsschritt erhalten! (prüfe jede Regel)

Behauptung 2

$\Rightarrow z \in \Sigma^*,\ S \overset{G}{\Longrightarrow}^* z$, dann

$$\#_a(z) = \#_b(z) = \#_c(z)$$

$S \overset{G}{\Longrightarrow}^* \alpha' B \alpha'' \overset{G}{\Longrightarrow} \alpha' b \alpha''$ dann gilt $\alpha' B \alpha''$ hat Form $\underbrace{a^n b^n}_{\alpha'} B \alpha''$.

Behauptung 3

Analog für c's

15.3 Kontextfreie Grammatiken (KFG/CFG)

linke Seite jeder Produktion besteht aus genau einer Variablen

$$P \subset V \times (\Sigma \cup V)^*$$

15.3.1 Beispiel

kontextfreie Grammatik für arithmetische Ausdrücke

$$\mathcal{G} = (\Sigma, V, E, P) \qquad \Sigma = \{a, (,), *, +\}$$

$$\begin{aligned} E &\to T \\ T &\to E + T \\ T &\to F \mid T * F \\ F &\to a \mid (E) \end{aligned}$$

$$\begin{aligned} E &\Rightarrow \underline{E} + T \\ &\Rightarrow \underline{T} + T \\ &\Rightarrow \underline{T} * F + T \\ &\Rightarrow \underline{F} * F + T \\ &\Rightarrow (\underline{E}) * F + T \\ &\Rightarrow (\underline{E} + T) * F + T \\ &\Rightarrow (\underline{T} + T) * F + T \\ &\Rightarrow (\underline{F} + T) * F + T \\ &\Rightarrow (a + \underline{T}) * F + T \\ &\Rightarrow \dots \\ &\Rightarrow (a + a) * a + a \end{aligned}$$

Links Ableitung

es wird immer Produktion am linkesten nicht Terminal der schon abgeleiteten Satzform angewendet.

15.4 Syntaxbaum (Ableitungsbaum)

Baum
Knoten markiert mit Elementen von $V \cup \Sigma$

$$v \quad \text{markiert} \quad A$$

$v_1, \ldots, v_k$ Kinder von v $\quad$ v_i markiert mit $\alpha_i \in (V \cup \Sigma)$, dann muss es Produktion

$$A \to \alpha_1 \alpha_2 \ldots \alpha_k$$

Der String α der sich aus Markierungen der Blätter des Baumes ergibt (wenn von links nach rechts abgelesen), hat die Eigenschaft, dass $A \Rightarrow^* \alpha$, wobei A Markierung der Wurzel

15.4.1 Satz

L Sprache
$\exists$ KFG $\mathcal{G}$ mit $L(\mathcal{G}) = L \Leftrightarrow \exists$ NKA M mit $L(M) = M$

15.5 Satz

L ist NKA-Sprache $\Leftrightarrow$ L ist kontextfreie Sprache
$\exists$ NKA M mit $L(M) = L \Leftrightarrow \exists$ kontextfreie Grammatik G mit $L(G) = L$

Anmerkung

NKA Sprachen heißen daher üblicherweise kontextfreie Sprachen.

Beweis: „$\Leftarrow$“

Kontextfreie Grammatik

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

wollen NKA M konstruieren, sodass

$$L(M) = L(G)$$

Idee

M soll Linksableitungen „simulieren“

Terminalpräfix $\approx$ konsumierten Eingabe
Rest $\approx$ Kellerinhalt

Vorlesung Mittwoch

Also, diese Vorlesung wird noch nachgetragen ...

16 Turing Maschine

PICTURE

16.1 Formalisierung

$$\begin{aligned} Q &\text{ Zustandsmenge} \\ \Sigma &\text{ Eingabealphabet} \\ \Gamma &\text{ Arbeitsalphabet} \\ \bar{b} \in \Gamma &\text{ Leerzeichen} \\ s \in Q &\text{ Startzustand} \\ F \subset Q &\text{ Endzustand} \end{aligned}$$

$$\Delta \subset (Q \times (\Sigma \cup \{\bar{b}\}) \times \Gamma) \times (Q \times K \times \Gamma \times K)$$

16.2 Konfiguration einer Turing Maschine

- Zustand
- Eingabeband Inhalt
- Position des Eingabe Lesekopfs
- Arbeitsbandinhalt
- Position des Arbeits Lese/Schreibkopf

$$Q \times \Sigma^* \times \mathbb{Z} \times \mathcal{B} \times \mathbb{Z}$$

$$\mathcal{B} = \left\{ B : \mathbb{Z} \to \Gamma \quad | \quad \text{für nur endlich viele } i \in \mathbb{Z} \text{ gilt } B[i] \neq \bar{b} \right\}$$

16.3 Ausgangskonfiguration

$$init\,(x) = (s, x, 1, B_0, 0)$$
$$B_0[i] = \bar{b} \quad \forall i \in \mathbb{Z}$$

16.4 Endkonfiguration

$$(q, \ldots,) \quad q \in F$$

16.5 Rechenschrittrelation zwischen Konfigurationen

$$(q, x, i, B, j) \overset{M}{\longmapsto} (q', x, i', B', j')$$

g.d.w.

$$\exists (q, a, A, q', \mu, A', \lambda) \in \Delta$$

mit

$$x[i] = a \qquad\qquad i' = i + \mu$$
$$B[j] = A \quad B'[l] = \begin{cases} A' & l = j \\ B[l] & l \neq j \end{cases}$$
$$j' = j + \lambda$$

$x[l] = \bar{b}$ für $l \leq 0$ und für

$\overset{M}{\longmapsto}^*$ reflexiver, transitiver Abschluss von $\overset{M}{\longmapsto}$
$x \in \Sigma^*$ wird von M akzeptiert, g.d.w.

$$init\ (x) \overset{M}{\longmapsto}^*\ k \qquad k \ldots \text{Endkonfiguration}$$

$$L(M) = \{x \in \Sigma^* \ \mid \ M \text{ akzetiert } x\}$$

16.6 Beispiel für Turing Maschine

$$L(M)\,\{a^n b^n c^n \ \mid \ n \in \mathbb{N}\}$$

Idee

1. Lese a's von Eingabe und kopier auf Arbeitsband
2. Lese b's von Eingabe und vergleiche mit $\#a$'s auf Arbeitsband (ziehe Arbeitskopf links!)
3. Lese c's von Eingabe und vergleiche mit $\#a$'s auf Arbeitsband (ziehe Arbeitskopf rechts!)

$$\begin{aligned}
\big(s, a, \bar{b} \quad , \quad s, +1, a, +1\big) \\
\big(s, b, \bar{b} \quad , \quad q, 0, \bar{b}, -1\big) \\
(q, b, a \quad , \quad q, +1, a, -1) \\
\big(q, c, \bar{b} \quad , \quad q', 0, \bar{b}, +1\big) \\
(q', c, a \quad , \quad q', +1, a, +1) \\
\big(q', \bar{b}, \bar{b} \quad , \quad h, 0, a, 0\big) \\
\big(s, \bar{b}, \bar{b} \quad , \quad h, 0, \bar{b}, 0\big)
\end{aligned}$$

$$\{h\} = F$$

17 Mehrband Turing Maschinen

$$PICTURE$$

17.1 Übergangsregeln

$$\Delta \subset (Q \times (\Sigma \cup \{\bar{b}\}) \times \Gamma^k) \times (Q \times K \times (\Gamma \times K)^k)$$

17.2 Konfigurationsmenge

$$Q \times (\Sigma^* \times \mathbb{Z}) \times (\mathcal{B} \times \mathbb{Z})^k$$

k-Band Turing Maschine können nicht mehr als 1-Band Turing Maschinen

17.3 Satz

Sei M eine Turing Maschine mit k Arbeitsbändern.
Dann gibt es eine 1-Band Turing Maschine $\hat{M}$ mit $L(\hat{M}) = L(M)$.

Beweis

Idee

Betrachte die k Bänder von M als k Spuren eines Bandes
Merke, wo die einzelnen LEseköpfe sich befinden

$$\hat{\Gamma} = (\gamma \times \{0,1\})^k$$

Für jede Regel $(q, a, A_1, \ldots, a_k, q', \mu, A_1', \lambda_1, \ldots, A_k', \lambda_k)$
Baue ein „Unterprogramm“, das den Effekt dieser Regel simuliert.
Lese/Schreibkopf von $\hat{M}$ steht am linken nde des benutzten Teils des Arbeitsbandes.

	A	B	C	A	A	B	B	C	A	
	0	0	0	0	0	0	0	0	1	
										$\bar{b}$

Für jedess von 1 bis k:
Arbeitskopf von $\hat{M}$ fährt nach rechts, bis Ez Kopfposition von M auf auf Bandstreifen i findet.
Mache Operation der Regel auf Band i nach.
Fahremit Kopf zurück zu linkem Ende.
Beobachte Initialisierung der k-Streifen Zellen am linken und rechten Ende des bearbeiteten Teils.

Beachte

Wenn M be Eingabe x $f(x)$ Rechenschritte macht, dann braucht $\hat{M} \leq \mathcal{O}(f(x)^2)$ Rechenschritte
denn, Simulation jedes der $f(x)$ Schritte von M braucht $k = \mathcal{O}(1)$ Läufe von $\hat{M}$ über den bearbeiteten Teil des Arbeitsbandes. Dieser Besteht aus höchstens $f(x)$ Zellen, da mit jedem Schritt von M nur höchstens eine weitere neue Arbeitszelle dazukommt.

Turing Maschine M heißt deterministisch, wenn in jeder Situation höchstens 1 Regel anwendbar ist.
Sonst nicht deterministisch

17.4 Satz

Sei M eine nicht deterministische Turing Maschine (mit 1 Arbeitsband).
Dann gibt es eine deterministische Turing Maschine M' mit $L(M') = L(M)$.

$$\text{reguläre Sprachen } \subsetneq DKA \subsetneq \text{kontextfreie } \subsetneq DTM = NTM \subsetneq \text{ alle Sprachen}$$

17.5 Beweis

$$M = \left(\Sigma, \Gamma, Q, s, F, \bar{b}, \Delta\right) \qquad \text{nicht deterministische Turing Maschine}$$

$$\begin{array}{rcl} k(q,a,A) & = & \#\text{ Regeln in } \Delta \text{ die in } q,a,A \text{ anwendbar sind} \\ Q \times \Sigma \times \Gamma & & \\ k & = & \max\{k(q,a,A) \mid (q,a,A) \times Q \times \Sigma \times \Gamma\} \end{array}$$

Pro Situation (q, a, A) sind die anwendbaren Regeln durchnummeriert.

Baue Maschine M' mit 2 Arbeitsbändern $\begin{matrix} AB1 \\ AB2 \end{matrix}$ deterministisch.

$AB1$ „simuliert“ Arbeitsband von M
$AB2$ enthält „Leitstring“ $l_1, l_2, l_3, \ldots, l_n$

Idee

Bei Verwendung von Leitstring $l_1, l_2, l_3, \ldots, l_n$ soll in schrittweiser Simulation von M im $i$ten Schritt die $l_i$te anwendbare Regel verwendet werden.
Wenn keine $l_i$te Regel vorhanden oder Leitstring erschöpft, dann

1. generiere nächsten Leitstring auf $AB2$ und bewege Kopf auf linkes Ende
2. Lösche Inhalt von $AB1$
3. Bewege Eingabekopf aufs linke Ende als Eingabe

Beginne Simulation von neuem

M' hält, wenn eine der Sumulationen hält.

Wenn x von M akzeptiert wird
$\Rightarrow \exists$ Rechenschrittfolge der Länge m, die von $Init\ (x)$ zu Haltekonfiguration führt.
$\Rightarrow$ Unter den Leitstrings l der Länge m ist einer dabei, der genau zur Simulation dieser akzeptierenden Rechenschrittfolge führt.

17.6 Satz

L Sprache $\subset \Sigma^*$
$\exists$ TM M mit $L(M) = L \Leftrightarrow \exists$ Grammatik G mit $L(G) = L$

Beweis „$\Leftarrow$“

Brauchen: TM M, die $w \in \Sigma^*$ genau dann akzeptiert, wenn w von G generiert werden kann.
$\exists$ Derivation $S \rightarrow_g \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \ldots \Rightarrow_G x_n = w$
M nicht deterministisch.
erzeugt auf Arbeitsband eine Derivation
simuliert NICHT DETERMINISMUS

nnach der Grammatik, wenn nur mehr (?noch): Terminalsymbole auf dem Arbeitsband stehen, wird Inhalt von Arbeitsband und Eingabe verglichen, wenn gleich, hält M.

Beweis „$\Rightarrow$“

Sei M eine Turing Maschine
Wollen Grammatik G mit $L(G) = L(M)$

EINSCHUB: Einfache Turing Maschine

Nur 1 Band anfangs Eingabe mit Lese/Schreibkopf ganz links
auf diesem Band kann uach geschrieben werden

Beobachtung

M Turing Maschine mit Arbeitsband, dann gibt es eine einfache Turing Maschine $\hat{M}$, mit $L(\hat{M}) = L(M)$.
Beweis analog zur Simulation von k Arbeitsband Turing Maschinen durcfh 1 Arbeitsband Turing Maschine.

TM $M \Rightarrow \exists$ einfache Turing Maschine $\hat{M}$, mit $L(\hat{M}) = L(M)$
Wollen grammatik G mit $L(G) = L(\hat{M})$

Beachte Gegensatz

$\hat{M}$ beginnt mit $w \in \Sigma^*$
manipuliert es, bis Haltezustand erreicht
G beginnt mit Symbol S
manipuliert es, bis w erreicht ist

Idee

Mache Umkehrung von akzetierten Rechenschrittfolgen von $\hat{M}$ zu Derivationen von G.
o.B.d.A.

a)

Startzustand von $\hat{M}$ wird nach Start nie wieder erreicht.

b)

$\hat{M}$ wird umgebaut, sodass nie ein $\bar{b}$ geschrieben wird.
führe neues Zeichen $\hat{\bar{b}}$ ein, dass statt $\bar{b}$ geschrieben wird und sonst gleichen Effekt wie $\bar{b}$ hat.
Stringkodierung von Konfigurationen von $\hat{M}$.

$$\begin{array}{lll} \bar{b}\bar{b}A_1A_2A_3 & A_4 & \ldots\ldots\ldots A_l\bar{b}\bar{b}\bar{b} \\ & \uparrow & \\ & \epsilon & \text{TM Zustand} \end{array}$$

$$A_1A_2A_3\,(q, A_4)\,A_5 \ldots\ldots\ldots A_l$$

$$\begin{array}{lll} \text{Startzustand} & & \text{Endzustand} \\ (s, w_1) & w_2 \ldots\ldots w_n \ldots\ldots a_n & (h, A_j)? \end{array}$$

Idee für G

1. Erzeuge Endkonfiguration von $\hat{M}$
2. ermögliche genau diese Ableitung, die Rechenschritt von $\hat{M}$ reversiert
3. Erzeuge aus Startkonfiguration das Eingabewort von $\hat{M}$

$$G = (\Gamma \cup \Gamma' \cup \{S\} \cup (q \times \Gamma'), \Sigma, S, P)$$
$$\Gamma' = \{A' \mid A \exists \Gamma\}$$

1)

$$\forall A' \in \Gamma' \qquad A'_1A'_2(h, A'_i)A'_{i+1} \ldots\ldots A'_l$$
$$\begin{array}{rcl} S & \rightarrow & SA' \\ S & \rightarrow & A'S \end{array}$$
$$\forall A' \in \Sigma' \quad \forall h \in F$$
$$S \rightarrow (h, A')$$

2)

$$(q, A, B, p, 0) \in \Delta$$
$$(p.B') \longrightarrow (q, A')$$

...

3)

$$
\begin{array}{lcc}
\forall a' \in \Sigma' & & \forall b' \in \Sigma' \\
(s, a')b' & \rightarrow & a(s, b') \\
(s, a') & \rightarrow & a(s, b') \\
(s, b') & \rightarrow & \epsilon
\end{array}
$$

18 Grammatiken ≈ Turing Maschinen

Anderer Rechenmechanismus

18.1 Register Maschinen

n Register, jeder kann eine natürliche Zahl fassen
$x_1, x_2, \ldots, x_n$

18.1.1 Befehle

$$x_i = x_j \; op \; c \qquad op \in \{+, *, \text{div}\,, \quad \text{mod}\,, \dot{-}\}$$

$$x \dot{-} y = \begin{cases} x - y & \text{falls } x \geq y \\ 0 & \text{sonst} \end{cases}$$

18.1.2 Prädikate

$$x_i \stackrel{?}{=} 0$$

18.1.3 Programm

Label:	Befehl *goto Label*	
Label:	*if* Prädikat *then* Befehl *goto Label else* Befehl *goto Label*	
spezielle Labels:	*START* *STOP*	Jedes Programm für so eine
Eingabe:	Inhalt von $x_1, \ldots, x_e$	Am Rechenanfang
Ausgabe:	Inhalt von $x_1, \ldots, x_e$	Am Rechenende

Maschine berechnet eine (partielle) Funktion

$$f: \; \mathbb{N}^e \to \mathbb{N}^a$$

Beispiel

$$f: \; \mathbb{N}^2 \to \mathbb{N} \qquad f(x, y) = x \cdot y$$

$$e = 2$$
$$a = 1$$
$$x_1 \;\; 5$$
$$x_2 \;\; 7$$

```
   START:    if  x1=0  then  NULL  goto  L1         // x1 x   x2 y   x3 0
                       else  x1=x1-1  goto  L2
      L2:    x3=x3+1  goto  START
      L1:    if  x3=0  then  NULL  goto  STOP       // x1 0   x2 y   x3 x
                       else  NULL  goto ADDYTOX1
ADDYTOX1:    if  x2=0  then  NULL  goto  L3
                       else x2=x2-1  goto  L4
      L4:    x1=x1+1  goto  L5
      L5:    x4=x4+1  goto  ADDYTOX1
      L3:    if  x4=0  then  NULL  goto  L6         // x1 +y   x2 0   x3 x   x4 y
                       else  x4=x4-1  goto  L7
      L7:    x2=x2+1  goto  L3                      // x1 +y   x2 y   x3 x-1   x4 0
      L6:    x3=x3-1  goto  L1
```

18.2 Satz

Sei $M = \big(Q, \Sigma, \Gamma, s, F, \bar{b}, \Delta\big)$ eine deterministische Turing Maschine.
Es gibt ein Programm Π für eine 3 Register Maschine
$\forall w \in \Sigma^*$: M hält auf Eingabe w
$\Leftrightarrow \Pi$ hält auf Eingabe $\langle w \rangle_t$
wobei

$$\Sigma = \{\underbrace{0}_{\bar{b}}, 1, \ldots, t-1\} \quad \langle w_0, w_1, \ldots, w_k \rangle_t = \sum_{0 \le i \le k} w_i t^i$$

Also: Registermaschinen „können“ mindestens so viel wie Turing Maschinen

18.2.1 Beweis

Idee

Π soll M schrittweise simulieren.
o.B.d.A. M ist Turing Maschine ohne Arbeitsband und verwendet Eingabeband als Arbeitsband.

0 0 0	5	3	2	1	7	3	9	6	5	1	0	3	0 0 . . . 0
	l	l	l	$\uparrow$	r	r	r	r	r	r	r	r	
				$q \in Q$									

Kodierung der Konfiguration

$$x_1 = \langle r \rangle_t$$
$$x_1 = \langle e^R \rangle_t$$

Zustand wird durch Programmstück „kodiert“.
Für jeden Zustand $q \in Q$ enhält Π ein Programmstück.

L_q: 1) extrahiere das Symbol a unter dem Kopf aus x_1
und speichere es in x_3
2) je nach a, simuliere Kopfbewegung
gehe zu L_p mit (q, a, A, μ, p)

L_q :	$x_3 = x_1 \mod t$
	$if\ x_3 = 0\ then\ AKTION\ für\ (q,0) \in Q \times \Gamma$
	$else\ x_3 = x_3 - 1$
	$if\ x_3 = 0\ then\ AKTION\ für\ (q,1)$
$t\ mal$	$else\ x_3 = x_3 - 1$
	$if\ x_3 = 0\ then\ AKTION\ für\ (q,2)$
	$\vdots$

AKTION **für** (q, a)

i)		$x_3 = x_1 \mod t$ $x_1 = x_1 - x_3$ $x_1 = x_1 + A$	*existriert a durch A*
ii)	$Fall: \ \mu = 0$	$GOTO \ L_p$	
	$Fall: \ \mu = -1$ $x_3 = x_2 \mod t$ $x_2 = x_2 \text{div } t$ $x_1 = x_1 + x_3$ $GOTO \ L_p$	$x_1 = x_1 \cdot t$	
	$Fall: \ \mu = +1$ $x_3 = x_1 \mod t$ $x_1 = x_1 \text{div } t$ $x_2 = x_2 + x_3$ $GOTO \ L_p$	$x_2 = x_2 \cdot t$	

19 Einfache k Register Maschine

Wie k Registermaschine, aber als Operationen

$$x_k = x_k + 1$$
$$x_k = x_k \dot{-} 1$$

Prädikator

$$x_k \stackrel{?}{=} 0$$

19.1 Satz

Für jedes Programm Π für eine normale k Registermaschine RM existiert eine Programm Π' für eine einfache 2 Register Maschine SRM, sodass Π und Π' bei geeignet Eingabekoordierung das gleiche Akztanzverhalten zeigen.

Kodierung der k Registerinhalte von RM in einem Register von SRM

$$2^{x_1} \cdot 3^{x_2} \cdot 5^{x_3} \cdot 7^{x_4} \cdot 11^{x_5} \ldots p_k^{x_k} \qquad p_i \ i\text{te Primzahl}$$

Idee

Baue k' Register SRM für Π.
nur Operation $x_i + 1$, $x_i - 1$, $x_i \stackrel{?}{=} 0$
Simuliere diese einfache Programm $\hat{\Pi}$ auf 2 Registern!

Brauche simulierende Programmstücke für

k' Register SRM		2 Register SRM
$x_i = x_i + 1$	$\longrightarrow$	$x_1 = x_1 \cdot p_i$
$x_i = y_i - 1$	$\longrightarrow$	$x_1 = if \ x_1 \mod p_i \neq 0 \ then x_1 = x_1 \text{div } p_i$
$x_i \stackrel{?}{=} 0$	$\longrightarrow$	$x_1 \mod p_i \neq 0$

Durchführung mit $+1$, $\overset{\bullet}{-}1$, $\overset{?}{=}0$ mir 2 Register

$$
\begin{array}{lll}
x_1 = x_1 - p_i: & L: & if\ x_1 \overset{?}{=} 0\ then\ gotoLL \\
 & & else\ x_1 = x_1 - 1\ goto\ L1 \\
 & L1: & x_2 = x_2 + 1\ goto\ L2 \\
 & L2: & x_2 = x_2 + 1\ goto\ L3 \\
 & \vdots & \\
 & LL: & if\ x_2 = 0\ then\ goto\ LL \\
 & & x_2 = x_2 - 1,\ x_1 = x_1 + 1
\end{array}
$$

19.2 Register Maschinen

k Register $x_1, \ldots, x_k$, jeder hält eine Zahl aus $\mathbb{N}$
Eingabe auf $x_1, \ldots, x_e$ Resit ist „0“

19.2.1 Operationen

$$x_j = x_i \text{ op } \underline{c}_{\text{Konstante}}$$

$$\text{op } \in \left\{ +, \overset{\bullet}{-}, *, /, \quad \text{mod} \right\}$$

19.2.2 Tests

$$x_i \overset{?}{=} 0$$

19.2.3 Programme

```
label:      Operation  Goto  label
if  Test  then  Operation  Goto  label  else  Operationen  goto  label
```

Jede Turing Maschine lässt sich durch 4 Register Maschinen simulieren.

Maschine M lässt sich durch M' simulieren

1. M akzeptiert $L \subset \epsilon$
2. M' akzeptiert $L' \subset \epsilon'$

$$\exists \text{ Kodierung } K: \ \epsilon \to \epsilon'$$

$$\forall x \in \epsilon: \ M \text{ akzeptiert } x \quad \Leftrightarrow \quad M' \text{ akzeptiert } K(x)$$

19.3 Einfache Registermaschinen

19.3.1 Operationen

$$x_i = x_i + 1$$
$$x_i = x_i \overset{\bullet}{-} 1$$

19.3.2 Test

$$x_i \overset{?}{=} 0$$

19.4 Satz

Jede k Register Maschine lasst sich durch eine einfache $(k+2)$ Register Maschine simulieren.

19.5 Korrolar

Jede Turing Maschine lässt sich durch eine einfache Register Maschine simulieren

19.6 Satz

„Jede einfache k Register Maschine M lässt sich durch eine fache 2 Register Maschine M' simulieren."

19.6.1 Beweis

M Register $x_i, \ldots, x_k$
M' Register y, x
M' speichert Registerinhalt von M als

$$y = p_1^{x_1} p_2^{x_2} \ldots p_p^{x_k} \qquad p_i \text{ ite Primzahl}$$

Wollen zeigen: M' kann Effekt der Operationen

$$x_i = x_i + 1$$
$$x_i = x_i \dot{-} 1$$

und Test

$$x_i \stackrel{?}{=} 0$$

simulieren.

M	M'	
$x_i = x_i + 1$	$y = y + p_i$	
	$\{y = \prod p_j^{x_j},\ z = 0\}$	
	while $y \neq 0$ *do*	
	$y = y \dot{-} 1$	
	$z = z + 1$	
	$\ldots$	p_i *mal*
	$z = z + 1$	
	od	
	$\{y = 0,\ z = p_i \cdot \prod p_j^{x_j}\}$	
	while $z \neq 0$ *do*	
	$z = \dot{-} 1$	
	$y = y + 1$	
	od	
	$\{y = p_j \cdot \prod p_j^{x_j},\ z = 0\}$	
$x_i \stackrel{?}{=} 0$	p_i test y nicht	

Brauch Programmstück, das testet, ob p Register y teilt, aber den Inhalt von y und z unberührt lässt.

$$\{y = Y,\ z = 0\}$$

$$
\begin{array}{l}
\textit{while } y \neq 0 \textit{ do} \\
\quad y = y \stackrel{\bullet}{-} 1 \textit{if } y = 0 \textit{ then goto } Restist1 \quad p_i \textit{ mal} \\
\quad y = y \stackrel{\bullet}{-} 1 \textit{if } y = 0 \textit{ then goto } Restist2 \qquad \vdots \\
\qquad \vdots \\
\quad y = y \stackrel{\bullet}{-} 1 \textit{if } y = 0 \textit{ then goto } Restist2 \qquad \textit{end} \\
\quad y = y \stackrel{\bullet}{-} 1 \\
\quad z = z \stackrel{\bullet}{-} 1 \\
\qquad \textit{od} \\
\qquad\qquad \left\{ y = 0,\ z = \dfrac{Y}{P} \right\} \\
\qquad\quad \textit{while } z \neq 0 \ldots \\
\qquad\quad Restistp_i - 1y = y + 1 \\
\qquad\quad \{ y = 0,\ z = \lfloor Y/P \rfloor \} \\
\quad y = p_i \cdot z \stackrel{\bullet}{-} (p_i - 1) \\
\qquad Restistp_i - 2y = y + 1 \\
\qquad\qquad \vdots \\
\qquad\quad Restist1y = y + 1 \\
\qquad \textit{while } z \neq 0 \textit{ do} \\
\qquad\qquad y = y + 1 \quad p_i \textit{ mal} \\
\qquad\qquad y = y + 1 \\
\qquad\qquad \vdots \\
\qquad\qquad \vdots \\
\qquad\qquad z = z \stackrel{\bullet}{-} 1 \qquad \textit{end} \\
\qquad\quad \textit{od} \\
\qquad\quad \ldots
\end{array}
$$

M	M'
$x_i = x_i \stackrel{\bullet}{-} 1$	$if\ p_i\ teilt\ y\ then\ y = y/p_i$
	$Programmstück\ p_i\ teilt\ y?$
$\ldots$	

19.7 Satz

Jede einfache 2 Register Maschine R lässt sich durch Turing Maschine M simulieren.

19.7.1 Beweis

	R	M	
y	Y	2 Arbeitsbänder	$\bar{b}\overbrace{1\ldots1}^{Y\ mal}\uparrow\bar{b}\bar{b}$
z	Z		$\bar{b}\bar{b}\overbrace{1\ldots1}^{Z\ mal}\uparrow\bar{b}$

Eingabe auf dem Arbeitsband 1 kopieren
offentsichtliche schrittweise Simulation

19.8 Korrolar

Jede Register Maschine lässt sich durch Turing Maschine simulieren.

19.9 Satz

Turing Maschinen und Registermaschinen sind „gleichmächtig“, d.i.

$$L \subset \{1, \ldots, t-1\}^* \qquad \langle L \rangle_t = \{\langle w \rangle_t \mid w \subset L\} \subset \mathbb{N}$$

$\exists$ Turing Maschine M die L akzeptiert $\Leftrightarrow$ $\exists$ Programm für Register Maschine das $\langle L \rangle_t$ akzeptiert.

$$\begin{array}{rcc} \text{Turing Maschine} & \approx & \text{Grammatiken} \\ & \approx & \text{„rekursive Funktionen“ (Klasse)} \\ & \approx & \text{Markov, Kolmogoroff} \\ & \approx & \lambda \text{ Kalkül (Church)} \\ & \approx & \text{Register Maschine} \end{array}$$

19.10 Church Turing These

„Alles, was (intuitiv) berechenbar ist, lässt sich durch eine Turing Maschine berechnen.“
„Wenn etwas von einer Turing Maschine nicht berechnet werden kann, dann kann es überhaupt nicht berechnet werden.“

20 Universalität von Turing Maschinen

Idee

Programmierbare Turing Maschinen.

Wir wollen eine Turing Maschine U: Eingabe: 1) „Programm“: als String codierte Turing Maschine M $\langle M \rangle$
2) „Eingabe“: String w $\langle w \rangle$

U soll $\langle M \rangle$ $\langle w \rangle$ akzeptieren $\Leftrightarrow$ M w akzeptiert
Kodierung $\langle w \rangle$ von w notwendig, damit Eingabealphabet Σ_M von M ins Eingabealphabet Σ_U von U eingebettet werden kann.

$$M = \left(Q_M, \Sigma_M, \Gamma_M, \underbrace{s}_{\in Q_M}, \underbrace{F}_{\in Q_M}, \bar{b}, \Delta \right)$$

$$\Delta \subset \left(Q \times \Sigma \cup \{\bar{b}\} \times \Gamma\right) \times (Q \times K \times \Gamma \times K)$$

$$\begin{array}{rr} Q_M = \{q_1, \ldots, q_n\} & s = q_1 \\ \Sigma_M = \{a_1, \ldots, a_n\} & \bar{b} = a_0 = b_0 \\ \Gamma_M = \{b_1, \ldots, b_n\} & \bar{b} = a_0 = b_0 \end{array}$$

$$\langle q_i \rangle \underbrace{01^i 0^{n-1} 0}_{n+2}$$

$$\langle a_i \rangle \underbrace{01^i 0^{\sigma-1} 0}_{\sigma+2}$$

$$\langle b_i \rangle \underbrace{01^i 0^{\gamma-1} 0}_{\gamma+2}$$

$$F = \left\{q_{i_1}, q_{i_2}, \ldots, q_{i_f}\right\}$$

20.1 Kodierung von M

$$\langle M\rangle = \underset{\langle n\rangle}{01^n0}\ \underset{\langle\sigma\rangle}{01^\sigma0}\ \underset{\langle\gamma\rangle}{01^\gamma0}$$

$$0\langle q_{i_1}\rangle\langle q_{i_2}\rangle\dots\langle q_{i_f}\rangle 00$$

$$\langle\delta_1\rangle\langle\delta_2\rangle\dots\langle\delta_D\rangle$$

Kodierung von $w = w_1\dots w_m \quad \langle w\rangle = \langle w_1\rangle\langle w_2\rangle\dots\langle w_m\rangle$

20.2 Satz

Es gibt eine universelle Turing Maschine U, d.i. auf Eingabe $\langle N\rangle\ \langle w\rangle$ terminiert U genau dann wenn M mit Eingabe w terminiert (mit gleichem Akzeptanzverhalten).

20.3 Beweis

Eingabeband:	$\langle M\rangle\langle w\rangle$
$A1$	simuliert Eingabeband von M
$A2$	simuliert Arbeitsband von M
$A3$	enthält aktuellen Zustand von M
$A4$	$\langle n\rangle \quad \dots 1^n$
$A5$	$\langle\sigma\rangle \quad \dots 1^\sigma$
$A6$	$\langle\gamma\rangle \quad \dots 1^\gamma$

20.3.1 Arbeitsweise von U

Initialisierung

- $\langle w\rangle$ von Eingabeband auf $A1$ kopieren und Kopf ganz links
- $\langle q_1\rangle$ auf $A3$ schreiben
- $\langle n\rangle$ auf $A4$ schreiben
- $\langle\sigma\rangle$ auf $A5$ schreiben
- $\langle\gamma\rangle$ auf $A5$ schreiben

Loop Invariante

- EK: Steht am linken Ende der Endzustandskodierungen
- K-$A1$: steht auf linkem Ende eines kodierten Symbols
- $A2$: enthält Kodierung des Inhalts von *Arbeitsband von M*
  Kopf steht auf linkem Ende, der Kopf des entsprechenden Symbols
- $A3$: Kopf ganz links

Loop Durchlauf soll einen Schritt von M simulieren

1. Teste, ob der(?) Zustand $q \in F$

   wenn *„Ja!“*, akzeptiere

2. Suche eine anwendbare Regel in Δ

   wenn gefunden: $\longrightarrow$ neuer Zustand auf $A3$, neues Symbol auf $A2$, Bewege Köpfe

$$\underbrace{SAM}_{\text{Selbst akzeptierende Maschine}} = \left\{w \in \{0,1\}^1 \mid w = \langle M\rangle \text{ und } M \text{ akzeptiert } w\right\}$$

$$\{\langle M\rangle \mid \langle M\rangle \in L(M)\}$$

20.3.2 Korrolar

SAM ist eine Turing Maschine Sprache

20.3.3 Definition

Turing Maschine	$\approx$	rekursiv aufzählbar
	$\approx$	„r.e.“ - „recursive enumerable“

20.4 Satz

$\overline{SAM}$ ist nicht rekursiv aufzählbar, also keine Turing Maschine

20.4.1 Beweis

Müssen zeigen, es gibt keine Turing Maschine M mit $L(M) = \overline{SAM}$ für alle Turing Maschinen M gilt $L(M) \neq \overline{SAM}$

Turing Maschine M	$\langle M\rangle \in L(M) \Rightarrow \langle M\rangle \in SAM \Rightarrow \langle M\rangle \notin \overline{SAM}$, also $L(M)$ und $\overline{SAM}$ unterschiedlich
	$\langle M\rangle \notin L(M) \Rightarrow \langle M\rangle \notin SAM \Rightarrow \langle M\rangle \in \overline{SAM}$, also $L(M)$ und $\overline{SAM}$ unterschiedlich

Nebenbehauptung

Das gleiche Ergebnis gilt auch für andere, analoge Rechenmechanismen, z.B. SML Programme, etc

20.5 Modifikation von Turing Maschinen

Es gibt „beschriftete“ Endzustände: (JA / NEIN)

20.5.1 Definition

Eine Turing Maschine M entscheidet eine Sprache $L \subset \Sigma^*$.
Wenn M für jedes $w \in \Sigma^*$ hält, und $w \in L \iff M$ hält an einem JA Endzustand.

20.5.2 Definition

$L \subset \Sigma^*$ heißt entscheidbar (bzw. rekursiv), wenn es eine Turing Maschine gibt, die L entscheidet.

20.5.3 Lemma

L entscheidbar $\iff$ L ist r.e. und $\overline{L}$ ist r.e.

20.5.4 Beweis „$\Rightarrow$“

Nimm M, die L entscheidet und modifiziere JA (bzw. NEIN) Endzustände zu nicht terminierenden Loops

20.5.5 Beweis „⇐“

$$\begin{array}{rcl} L \text{ r.e.} & \Longrightarrow & \exists \text{ Turing Maschine } M^+, \text{ die } L \text{ akzeptiert} \\ \overline{L} \text{ r.e.} & \Longrightarrow & \exists \text{ Turing Maschine } M^-, \text{ die } \overline{L} \text{ akzeptiert} \end{array}$$

Baue Maschine M, die M^+ und M^- schrittweise simuliert.
Stoppe, wenn einer der beiden Maschinene einen Endzustand erreicht.

$$\text{Endzustände von } \begin{cases} M^+ & \text{JA} \\ M^- & \text{NEIN} \end{cases}$$

Korrolar

SAM ist NICHT entscheidbar

$$\boxed{\text{entscheidbare Sprachen}} \subsetneq \boxed{\text{Turing Maschinen Sprachen}}$$

$$UNIV = \{\langle M\rangle\,\langle w\rangle \;\mid\; M \text{ akzeptiert } w\}$$

20.6 Satz

$UNIV$ ist r.e. aber nicht entscheidbar.

20.6.1 Beweis

r.e. wegen vorherigem Satz

Behauptung

Wäre $UNIV$ entscheidbar, dann wäre auch SAM entscheidbar.

Nebenbehauptung

$$UNIV \quad \approx \quad \text{Halteproblem von Turing Maschine}$$

21 Automaten für unendliche Werte

$$L \subset \Sigma^* \qquad \Sigma \text{ endliches Alphabet}$$

$$\Sigma^\omega \circ \{\sigma : \mathbb{N} \to \Sigma\}$$

$$r \in \Sigma : \ \sigma(0), \sigma(1), \sigma(2), \ldots$$

$$\Sigma = \{a, b, c\} \quad \sigma = aabaacaabaa \ldots \quad \omega \text{ Wort}$$

Möchten mit endlichen Automaten eine MEnge von ω Worten spezifizieren.
(„ω Sprache“)

$$M = \left\{ Q, \Sigma, \underbrace{s}_{\subset Q}, \underbrace{F}_{\subset Q}, \underbrace{\Delta}_{\subset Q \times \Sigma \times Q} \right\}$$

$$l \in Q^\omega$$

L auf für σ, wenn $l(\sigma) = s$ und $\forall \in \mathbb{N} : \ (l(i), \sigma(i), l(i+1)) \in \Delta$ also,l ist ein endloser Pfad durch Übergangsgraphen von $\mathcal{G}$ und l ist mit σ beschriftet.
l ist erfolgreich für σ, wenn $Inf\ (l) \cap F \neq \emptyset$, $\underline{M \text{ akzeptiert } \sigma}$, $\exists$ ein erfolgreicher Pfad für σ

21.1 Büchi Automat

$$L(M) = \{\sigma \in \Sigma^\omega \ \mid \ M \text{ akzeptiert } \sigma\} \ldots \omega \text{ Sprache}$$

$L \subset \Sigma^\omega$ heißt ω regulär, wenn es Automat M ibt, mit $L(M) = L$.

21.2 Beispiel

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

$$L_1 = \{\sigma \in \Sigma^\omega \ \mid \ \text{nach jedem Auftreten von } a \text{ in } \sigma\text{gibt es irgendwann ein } b \ \}$$

$$PICTURE\ MAXI\ AUTOMAT\ L_1$$

$$L_2 = \{\sigma \in \Sigma^\omega \ \mid \ \text{nach jedem Auftreten von } a \text{ in } \sigma\text{gibt es irgendwann ein } b \ \} = L_1$$

$$PICTURE\ MAXI\ AUTOMAT\ L_2$$

$$L_2 = \{\sigma \in \Sigma \ \mid \ \text{zwischen jeden 2 } a\text{'s gibt es eine gerade Anzahl von } b\text{'s ohne } c's \ \} = L_1$$

$$PICTURE\ MAXI\ AUTOMAT\ L_3$$

$$q \xrightarrow{w} q'$$

q' von q über Pfad mit Beschriftung w erreichbar.

$$W_{q,q'} = \left\{ w \in \Sigma^* \ \mid \ q \xrightarrow{w} q' \right\}$$

reguläre Sprache

$$L(M) = \bigcup_{q \in F} W_{sq} W_{qq}$$

21.3 Satz

L ω reguläre Sprache $\iff$ L ist endliche Vereinigung von $U(V)^\omega$ mit U, V regulär.

21.4 Frage

Gegeben Büchi Automat M.
Gilt $L(M) = \emptyset$?

Ist entscheidbar:

Betrachte den Übergangsgraphen $\mathcal{G}$ von M. Bestimme die stark-zusammenhängende Komponenten von $\mathcal{G}$.

x, y Knoten zu $\mathcal{G}$.
x, y in der gleichen starken Zusammenhangskomponoente

$$x = y \quad \text{oder} \quad (x \rightsquigarrow y \text{ und } y \rightsquigarrow x)$$

Bestimme ob es Komponenten gibt, die nicht beim Schnitt mit F haben und von s erreichbar sind.

Im Fall $L(M) = \emptyset$ wird ω Wort produziert: $u\, v^\omega$

21.5 Frage

Wenn $L(M)$ ω regulär, gilt $\overline{L(M)}$ ω regulär?

21.6 Achtung

Es gibt ω Sprachen, die ω regulär sind, aber von keinem deterministischen Büchi Automaten erkannt werden.

21.7 Beispiel

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

$$L_4 = \{\sigma \in \Sigma^\omega \mid \sigma \text{ enthält endlich viele } a\text{'s}\}$$

$$PICTURE\ MAXI\ AUTOMAT\ L_4$$

Beweise

M deterministischer Büchi Automat

$$L(M) \neq L_4$$

Hinweis: Betrache ω Werte

$$\{a^n k^\omega \mid n \in \mathbb{N}\}$$

21.8 Satz

L ω regulär $\quad \Rightarrow \quad$ ω regulär
M Büchi Automat für L (mit n Zuständen)
$\overline{M}$ Büchi Automat für $\overline{L}$ $\quad \overline{M}$ hat $2^{\mathcal{O}(n \log n)}$ Zustände

21.8.1 Lemma

$$L, L' \wr \text{ regulär} \quad \Rightarrow \quad L \cup L' \ \omega \text{ regulär}$$

21.8.2 Korrolar

$$L, L' \wr \text{ regulär} \quad \Rightarrow \quad L \cap L' \ \omega \text{ regulär}$$

21.8.3 Korrolar

Gegeben L, L' jeweils durch Büchi Automaten M, M' kann man testen ob $L \cap L' = \emptyset$

21.9 Anwendungen: „Temporale Logik“

Objekte
$\wr$ Werte σ
Suffix von σ $\quad \sigma(0), \sigma(1), \sigma(2), \ldots$ wieder $\wr$ Werte

Prädikate über $\wr$ Werte

$$P: \ \Sigma^{\wr} \to \{ \ true \ , \ false \ \}$$

z.B.

$$a(\sigma) = \begin{cases} true & s(\sigma) = a \\ false & \text{sonst} \end{cases}$$

$$b(\sigma), \ c(\sigma)$$

Operationen

$\Box$	„Immer“	d.i. für alle Suffices
$\Diamond$	„Irgendwann“	d.i. für irgendein Suffix
$\circ$	„nächste“	d.i. für den ersten Suffix

Beispiel

$\Diamond b$		Irgendwann b
		d.i. $\wr$ Werte mit irgendeinem Suffix, den mit b beginnt.
		d.i. $\wr$ Werte die ein b enthalten
$\Diamond\Box b$		Irgendwann Immer nur b's
		d.i. nach endlich vielen Stellen in σ kommen nur noch b's
$\Diamond\Box(b \vee c)$	$\approx L_4$	es können nur endlich viele a's
$\Box(a \to \Diamond b$	$\approx L_1$	

22 Formalismus zum kategorischen Aufschreiben von Eigenschaften von (Zeit-)Folgen

Idee

Unendlicher Prozess wird durch temporale Logik Formal F spezifiziert.
Hat dieser so (durch F) spezifizierte Prozess eine gewisse Eigenschaft E?

$$F \stackrel{?}{\Longrightarrow} E$$

nicht sein darf

$$F \wedge \neg E$$

22.1 Problem

Brauchen Test für: „Ist temporale Formel G“ falsch?

Lösung

Konstruiere Büchi Automat M, der äquivalent zu G ist, und teste ob $L(M) = \emptyset$.

$$F \wedge \neg E$$

23 Wiederholung

Church-Turing These

- „überhaupt berechenbar“ $\Longrightarrow$ von TM berechenbar
- von TM nicht berechenbar $\Longrightarrow$ überhaupt nicht berechenbar

24 TM Erweiterung

- normale TM: akzeptiert Worte
- TM: akzeptiert oder verwirft Eingabe (JEDE)
- funktionale TM: berechnet Funktion $f : \Sigma^* \longrightarrow T^*$, hat Ausgabeband, enthält nach „Ende“ der Berechnung für Eingabe $x \in \Sigma^*$ die Ausgabe $f(x) \in T^*$.

$L \subset \Sigma^*$ TM-Sprache, akzeptierbar, rekursiv aufzählbar, recursively enumerable, r.e.
$L \subset \Sigma^*$ entscheidbare, rekursive, decidable recursive $\exists$ entscheidende TM, die bei

24.1 Lemma 1

L entscheidbar $\Longrightarrow$ L r.e.

24.2 Lemma 2

L entscheidbar $\Longleftrightarrow$ L ist r.e. UND $\bar{L}$ ist r.e.

24.3 Kodierungen von TMen und Strings

Σ	Alphabet	Jeder String $w \in \Sigma^*$ lässt sich als binärer String $\langle w \rangle$ kodieren.
$M = (Q, \Sigma, \Gamma, \ldots)$	TM	Jede TM M lässt sich als binärer String $\langle M \rangle$ kodieren, und zwar so, dass jeden binäre

25 Universelle TM

Es gib eine TM U, die die Eingabe $\langle M\rangle\,\langle w\rangle$ genau dann akzeptiert, wenn TM M Eingabe w akzeptiert.

(U simuliert M auf Eingabe w)

$$\langle M\rangle\,\langle w\rangle \in L(U) \Longleftrightarrow w \in L(M)$$

Universelle Sprache:
$UNIV = \{\langle M\rangle\,\langle w\rangle \mid w \in L(M)\} \subset \{0,1\}^*$

$UNIV$ ist r.e. (wegen U)

25.1 Satz

$UNIV$ ist NICHT entscheidbar.

„Halteproblem“
Es lässt sich nicht automatisch entscheiden, ob ein Programm (TM) bei gegebener Eingabe halten wird.

$SAM = \{\langle M\rangle \mid \langle M\rangle \in L(M)\}$ „selbst akzeptierende Maschinen“

Beachte

SAM ist r.e.

Hauptsatz

SAM ist NICHT entscheidbar.

Beweis

Nach Lemma 2 reicht es zu zeigen $\overline{SAM}$ nicht r.e.
d.i. $\exists$ keine TM N mit $L(N) = \overline{SAM}$
d.i. $\forall$ TMen N gilt $L(N) \neq \overline{SAM}$
$\langle N\rangle \in L(N) \Longrightarrow \langle N\rangle \notin \overline{SAM}$
$\langle N\rangle \notin L(N) \Longrightarrow \langle N\rangle \in \overline{SAM}$

$\overline{SAM}$ und $L(N)$ unterscheiden sich in $\langle N\rangle$

Beweis

Annahme X ist TM, die $UNIV$ entscheidet.
Wollen zeigen, dass es eine TM Y gibt, die SAM entscheidet.(Widerspruch zum Hauptsatz)

Was macht Y

Bei Eingabe $\langle M\rangle$ lässt sie X mit Eingabe $\langle M\rangle\,\langle M\rangle$ laufen.

$$\langle M \rangle \in SAM \iff \langle M \rangle \langle M \rangle \in UNIV$$

u String
$ACC_u = \{\langle M \rangle \mid u \in L(M)\}$

25.1.1 Satz

Für jedes u gilt: ACC_u ist nicht entscheidbar.

25.1.2 Beweis

Annahme: $\exists$ TM X, die ACC_u entscheidet.
Wollen damit TM Y bauen, die $UNIV$ entscheidet.

Widerspruch zur Unentscheidbarkeit von $UNIV$ also ist die Annahme falsch, s ein X gibt es nicht.

Y: Eingabe $\langle M \rangle \langle w \rangle$

1. Baue Beschreibung $\left\langle Z_{\langle M \rangle \langle w \rangle u} \right\rangle$ von TM $Z_{\langle M \rangle \langle w \rangle u}$

   $Z_{\langle M \rangle \langle w \rangle u}$: Eingabe v

   Lasse universelle TM U auf $\langle M \rangle \langle w \rangle$ laufen.

   Falls $v = u$ terminiere, sonst divergiere.

   Beachte: $Z_{\langle M \rangle \langle w \rangle u}$ akzeptiert u g.d.w. M akzeptiert w.

2. Lasse X auf $\left\langle Z_{\langle M \rangle \langle w \rangle u} \right\rangle$ laufen.

   X akzeptiert $\left\langle Z_{\langle M \rangle \langle w \rangle u} \right\rangle$ g.d.w. $Z_{\langle M \rangle \langle w \rangle u}$ akzeptiert u
   $\langle M \rangle \langle w \rangle \in UNIV \iff \left\langle Z_{\langle M \rangle \langle w \rangle u} \right\rangle \in ACC_u$

$LEER = \{\langle M \rangle L(M == \emptyset\}$

25.2 Satz

$LEER$ ist nicht entscheidbar.

25.2.1 Beweis

Annahme $\exists$ TM X, die $\overline{LEER}$ ($ETWAS$) entscheidet, wollen damit TM Y bauen, die UNIV entscheidet. $\lightning$

Y: Eingabe $\langle M \rangle \langle w \rangle$

1. Baue Beschreibung $\left\langle ZZ_{\langle M \rangle \langle w \rangle} \right\rangle$ von TM $ZZ_{\langle M \rangle \langle w \rangle}$

   $ZZ_{\langle M \rangle \langle w \rangle}$: Eingabe v
   Lasse U auf $\langle M \rangle \langle w \rangle$ laufen.

akzeptiere

2. Lasse X auf $\langle ZZ_{\langle M\rangle\langle w\rangle}\rangle$ laufen.

   $ZZ_{\langle M\rangle\langle w\rangle}$ akzeptiert irgendwas $\Longleftrightarrow$ M akzeptiert w.

$$\boxed{\langle M\rangle\,\langle w\rangle \in UNIV \Longleftrightarrow \langle ZZ_{\langle M\rangle\langle w\rangle}\rangle \in \overline{LEER}}$$

26 Reduktionen zwischen Sprachen

26.1 Definition

$L, S \subset \Sigma^*$ rückführbar
L heißt auf S reduzierbar, wenn es eine durch eine Turingmaschine berechenbare Funktion

$$f:\ \Sigma^* \longrightarrow \Sigma^*$$

gibt, so dass

$$x \in L \Longleftrightarrow f(x) \in S$$
$$x \notin L \Longleftrightarrow f(x) \notin S$$

Man schreibt:
$L \leq S$ „L ist leichter als S“

26.2 Satz

1. $L \leq S$ und S entscheidbar $\Rightarrow L$ entscheidbar
2. $L \leq S$ und L nicht entscheidbar $\Rightarrow S$ nicht entscheidbar

Beweis 1)

$L \leq S$ $\exists$ Turingmaschine F berechnet Funktion f

$$x \in L \Longleftrightarrow f(x) \in S$$

S entscheidbar: $\exists$ Turingmaschine M die S entscheidet.

N Turingmaschine	Eingabe x	N entscheidet
	lässt F auf x laufen, berechnet $f(x)$	L
	lässt M auf $f(x)$ laufen	

26.3 Satz von Rice

$\Re\mathcal{E}$ Menge aller reelle Sprachen
$\mathcal{S} \subset \Re\mathcal{E}$ sodass $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \Re\mathcal{E}$
Die Sprache

$$C(\mathcal{S}) = \{\langle M\rangle \mid L(M) \in \mathcal{S}\}$$

ist NICHT entscheidbar.

26.3.1 Anmerkung

Jede Sprache der Form

$$\text{„}\{\langle M\rangle \mid L(M) \text{ besitzt irgendeine nicht-trivialen Eigenschaft}\}\text{“}$$

ist nicht entscheidbar:

$$\{\langle M\rangle \mid u \in L(M)\} = ACCU$$
$$\{\langle M\rangle \mid L(M) \text{ endlich}\}$$

$$\{\langle M\rangle \mid L(M) \text{ ist Turingmaschine akzeptierbar}\} = \Re\mathcal{E}$$

26.3.2 Beweis

$S \subset \Re\mathcal{E}$ o.B.d.A $\emptyset \notin \mathcal{S}$
$L \in \mathcal{S}$ Turingmaschine M_L akzeptiert L
Wollen zeigen: $UNIV \leq C(\mathcal{S})$
d.i. wir brauchen berechenbare Funktion $f: \{0,1\}^* \longrightarrow \{0,1\}^*$, sodass

$$\langle M\rangle \mod w \in UNIV \iff \underbrace{f(\langle M\rangle\langle w\rangle)}_{\langle zzz_{\langle M\rangle\langle w\rangle}\rangle} \in C(\mathcal{S})$$

$$L\left(zzz_{\langle M\rangle\langle w\rangle}\right) \in \mathcal{S}$$

Turingmaschine $zzz_{\langle M\rangle\langle w\rangle}$: Eingabe v

1. Lasse U auf $\langle M\rangle\langle w\rangle$ laufen
2. Lasse M_L auf v laufen

Wie verhält sich $zzz_{\langle M\rangle\langle w\rangle}$?
Wenn M w nicht akzeptiert, dann akzeptiert $ZZZ_{\langle M\rangle\langle w\rangle}$ gar nichts

$$L\left(ZZZ_{\langle M\rangle\langle w\rangle}\right) = \emptyset \iff \left\langle ZZZ_{\langle M\rangle\langle w\rangle}\right\rangle \notin C(\mathcal{S})$$

Wenn M w akzeptiert, dann verhält sich $ZZZ_{\langle M\rangle\langle w\rangle}$ wie M_L, also

$$L\left(ZZZ_{\langle M\rangle\langle w\rangle}\right) = L \in \mathcal{S} \iff \left\langle ZZZ_{\langle M\rangle\langle w\rangle}\right\rangle \in C(\mathcal{S})$$

26.4 Post'sches Korrespondenz Problem

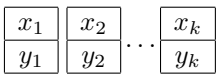
Σ beliebiges Alphabet
Gegeben

$$(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k) \quad x_i, y_i \in \Sigma^*$$

Frage

$\exists$ Indexfolge $I = (i_1, i_2, \ldots, i_k) \in \{1,\ldots,k\}^*$ mit

$$x_{i_1}x_{i_2}x_{i_3}\ldots x_{i_n} = y_{i_1}y_{i_2}y_{i_3}\ldots y_{i_n}$$



x_2	x_1	x_1	x_7	x_8	x_2	x_1
y_2	y_1	y_1	y_7	y_8	y_2	y_1

26.4.1 Beispiel 1

$$\Sigma = \{0, 1\}$$

$$(1, 101), (10, 00), (011, 11)$$

1	10	011
101	00	11

Lösung

$$I = (1, 3, 2, 3)$$

1	011	10	011
101	11	00	11

26.4.2 Beispiel 2

001	01	01	10
0	011	101	001

Lösung

$$\text{Kürzeste: } |I| = 66$$

$$PCP_\Sigma = \left\{((x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)) \in \left(\Sigma^+ \times \Sigma^+\right)^+ \;\middle|\; \exists \text{ korrespondierende Lösung}\right\}$$

$$\begin{aligned} X &= (x_1, \ldots, x_k) \in \left(\Sigma^+\right)^k \\ I &= (i_1, \ldots, i_k) \in \{1, \ldots, k\}^k \\ \bar{X}[I] &= x_{i_1} x_{i_2} \ldots x_{i_n} \end{aligned}$$

Alternative

$$PCP_\Sigma = \left\{(k, X, Y) \in \mathbb{N}_{>0} \times \left(\Sigma^+\right)^k \times \left(\Sigma^+\right)^k \;\middle|\; \bar{X}[I] = \bar{Y}[I]\right\}$$

26.5 Satz

PCP_Σ ist nicht entscheidbar.

26.6 Idee

$$UNIV \leq MPCP_{\Sigma'} \leq PCP_\Sigma$$

$$MPCP_{\Sigma'} = \left\{(k, X, Y) \;\middle|\; \exists I = (1, i_2, \ldots, i_n) \in \{1, \ldots, k\}^+ \text{ sodass } \bar{X}[I] = \bar{Y}[I]\right\}$$

26.7 Lemma

$$MPCP_\Sigma \leq PCP_{\Sigma \cup \{\$, \#\}}$$

26.8 Beweis

$$w = a_1 a_2 \dots a_m \in \Sigma^+$$

$$\begin{aligned} \bar{w} &= \#a_1\#a_2\#a_3\#\dots\#a_m\# \\ \grave{w} &= \#a_1\#a_2\#a_3\#\dots\#a_m \\ \acute{w} &= a_1a_2\#a_3\#\dots\#a_m\# \end{aligned}$$

$$T = (k, (x_1, x_2, \dots, x_k), (y_1, \dots, y_k)) \in \mathbb{N} \times \left(\Sigma^{+k}\right)^2$$

$$f(T) = \left(k+2, (\bar{x_1}, \acute{x_1}, \acute{x_2}, \acute{x_3}, \dots, \acute{x_k}, \$), (\grave{y_1}, \grave{y_1}, \grave{y_2}, \grave{y_3}, \dots, \grave{y_k}, \#\$)\right)$$

T besitzt Lösung mit $i_1 = 1 \iff$ f(T) besitzt Lösung

„⇒“

$$x_1 x_{i_2} x_{i_3} \dots x_{i_n} = y_1 y_{i_2} y_{i_3} \dots y_{i_n}$$

„⇐“

Jede Lösung von $f(t)$ muss mit $\{\bar{x_1}, \grave{y_1}\}$ beginnen, sonst # Mismatch

27 Das Wortproblem für allgemeine Grammatiken

27.1 Instanz

$$\begin{aligned} \text{Grammatik } \mathcal{G} &= (\Sigma, V, S, P) \\ \text{Wort } &= w \in \Sigma^* \end{aligned}$$

27.2 Frage

Gilt $w \in L(\mathcal{G})$?
d.i.

$$S \overset{\mathcal{G}}{\Rightarrow}^* w$$

$$\{\langle \mathcal{G} \rangle \langle w \rangle \mid w \in L(\mathcal{G})\}$$

27.3 Lemma

27.3.1 Beweis

Müssen zeigen, aus jeder Turing Maschine M lässt sich Grammatik $\mathcal{G}$ erzeugen mit

$$w \in L(M) \iff w \in L(\mathcal{G})$$

(schon gezeigt!)

27.4 Lemma

$$WORT \leq MPCP$$

27.4.1 Beweis

Wollen aus Instanz $\langle G\rangle\,\langle w\rangle$ von $WORT$ eine Instanz $F(\langle G\rangle\,\langle w\rangle)$ von $MPCP$ konstruieren, sodass

$$\langle G\rangle\,\langle w\rangle \in WORT \quad \Longleftrightarrow \quad F(\langle G\rangle\,\langle w\rangle) \in MPCP$$

$$w \in L(\mathcal{G})$$

$\exists$ Derivation

$$S \overset{\mathcal{G}}{\Rightarrow} U_1 \overset{\mathcal{G}}{\Rightarrow} U_2 \overset{\mathcal{G}}{\Rightarrow} \ldots \overset{\mathcal{G}}{\Rightarrow} U_n \overset{\mathcal{G}}{\Rightarrow} w$$

$$w \overset{\mathcal{G}}{\Leftarrow} U_m \overset{\mathcal{G}}{\Leftarrow} \ldots \overset{\mathcal{G}}{\Leftarrow} U_3 \overset{\mathcal{G}}{\Leftarrow} U_2 \overset{\mathcal{G}}{\Leftarrow} U_1 \overset{\mathcal{G}}{\Leftarrow} S$$

27.5 Idee

Baue Instanz von $MPCP$ deren Lösungen Derivationen von w darstellen.

27.6 Beispiel

$$CAb \longrightarrow acb$$

ist Produktion in $\mathcal{G}$

$$\$abacbba \quad \Leftarrow \quad abCAbba \quad \Leftarrow \quad \$abacbba \quad \Leftarrow$$

$$\mathcal{G} = (\Sigma, V, P, S)$$
$$P = \{x_j \to \beta_j \mid 1 \le j \le |P|\}$$
$$V = \{A_j \mid \ldots\}$$
$$\Sigma = \{a_j \mid \ldots\}$$

$$w \in \Sigma^*$$

27.7 Instanz von $MPCP$

$$k =$$
$$X = \left(\$w \Leftarrow, A_1, \ldots, A_{|V|}, a_1, \ldots, a_{|\Sigma|}, \Leftarrow, \alpha_1 \ldots, \alpha_{|P|}, \notin\right\}$$
$$Y = \left(\$, A_1, \ldots, A_{|V|}, a_1, \ldots, a_{|\Sigma|}, \Leftarrow, \beta_1 \ldots, \beta_{|P|}, \Leftarrow S \notin\right\}$$

Jede Lösung der Instanz entspricht einer Derivation w

27.8 Satz

Die folgenden Probleme sind NICHT entscheidbar.

Gegeben

Kontextfreie Grammatik $\mathcal{G}_1$, $\mathcal{G}_2$

1. Frage: Gilt $L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$?
2. Frage: Gilt $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ ist unendlich?
3. Frage: Gilt $L(\mathcal{G}_1) \subseteq L(\mathcal{G}_2)$?
4. Frage: Gilt $L(\mathcal{G}_1) = L(\mathcal{G}_2)$?
5. Frage: Gilt $L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ ist kontextfrei?

27.9 Anmerkung

Die gleichen Fragen sind für rechtslineare Grammatiken (reguläre Sprachen) alle entscheidbar.

27.9.1 Beweis

Wollen zeigen, das sich durch Beantwortung dieses Fragen PCP lösen lässt.

27.9.2 Beweis 1

k, X, Y Instanz von PCP
Betrachte Sprachen

$$S_X = \left\{ I^R \$ \bar{X}[I] \mid I \in \{1, \ldots, k\}^+ \right\}$$
$$S_Y = \left\{ J^R \$ \bar{Y}[J] \mid J \in \{1, \ldots, k\}^+ \right\}$$

1. $S_X \cap S_Y \neq \emptyset \iff$ *quad* PCP Instanz besitzt Lösung
2. S_X ist kontextfrei $\quad G_X = (\{1, \ldots, k\} \cup \Sigma, [S], S, S \to S_{x_i})$
3. S_Y ist kontextfrei

27.9.3 Beweis 2

Beachte:

$$\bar{X}[I] = \bar{Y}[I]$$

bedeutet

$$\bar{X}[II] = \bar{Y}[II] \quad \bar{X}[I^n] = \bar{Y}[I^n] \quad \forall n > 0$$

Also

$$S_X \cap S_y = \emptyset \Rightarrow S_X \cap S_Y \text{ ist endlich}$$

$$S_X \cap S_y \neq \emptyset \Rightarrow |S_X \cap S_Y| \text{ist endlich}$$

27.9.4 Beweis 3

Beachte:

$$A \subseteq B \iff A \cap \bar{B} = \emptyset$$

$$S \cap B = \emptyset \iff A \subseteq \bar{B}$$

S_Y ist deterministisch kontextfrei, also gilt $\overline{S_Y}$ ist deterministisch kontextfrei.

Frage

$$L(\mathcal{G}_1) \cap L(\mathcal{G}_2) = \emptyset$$

lässt sich in diesem Fall reduzieren auf

$$L(\mathcal{G}_1) \subseteq L(\overline{\mathcal{G}_2}) \left(= \overline{L(\mathcal{G}_2)} \right)$$

27.9.5 Beweis 4

$$A \subseteq B \iff A \cup B = B$$

$$PCP \quad k, X, Y$$

$$\downarrow$$

$$\mathcal{G}_x \ , \ \mathcal{G}_y$$

$$\downarrow$$

$$G_X \ , \ \overline{G_Y}$$

$$\downarrow$$

$$\mathcal{G} \ , \ \overline{G_Y} \downarrow$$

$$L(\mathcal{G}) \stackrel{?}{=} L(\overline{\mathcal{G}_y})$$

27.9.6 Beweis 5

k, X, Y Instanz von PCP
betrachte:

$$L_X = \left\{ I^R \# K \$ K^R \# \# \bar{X}[I] \ \mid \ I, K \in \{1, \ldots, k\}^+ \right\}$$

$$L_Y = \left\{ J^R \# K \$ K^R \# \# \bar{Y}[J] \ \mid \ J, K \in \{1, \ldots, k\}^+ \right\}$$

Jedes Wort in $L_X \cap L_Y$ hat Form

$$I^R \# I \$ I^R \# \bar{X}[I] \text{ mit } \bar{X}[I] = \bar{X}[I]$$

ergibt eine Lösung des PCP.
weil dann $\bar{X}[I^n] = \bar{Y}[I^n]$ für alle $n > 0$ gilt;
ist $I^{n^R} \neq I^n \# \bar{X}[I^n]$ in $S_X \cup S_Y \ \forall n > 0$

Problem

Entwerfen S eine SML [1] Funktion

```
self: void -> string
```

```
self () = der eigene Programmtext ...
          "fn() -> ..."
```

Problem 2

Hätte gerne ein SML Konstrukt

```
f = fn x -> ...
       let
           s = programmtextdieserfunktion()
       in
           ...
       end; (* ni <- lol ;) *)
```

[1] MosML, SML/NY, ...

28 Der Rekursionssatz

28.1 Gesucht: $SELF$

Eine Turingmaschine $SELF$ mit

1. löscht Eingabe
2. schreibt eigene Kodierung aufs Band
3. hält

28.2 Anmerkung

1. In dieser Vorlesung haben alle Turing Maschinen nur 1 Band:
   gleichzeitig Eingabe, Arbeits- und Ausgabeband.
2. Komposition von Turing Maschinen
   $$\langle A\rangle \# \langle B\rangle$$
   Kodierung der Turing Maschinen die Zuerst A auf Eingabe x laufen lässt, produziert Ausgabe y und dann B auf y laufen lässt.
3. Wenn Turing Maschine M eine Funktion $f(x, y)$ berechnen soll:
   $$\text{Eingabekonvention: } x\$y$$

28.3 Behauptung

$\exists$ Turing Maschine Q, die bei Eingabe $w \in \Sigma^*$ die Kodierung $\langle PRINT_w\rangle$ einer Turing Maschine $PRINT_w$ produziert. Q berechnet Funktion

$$x \overset{PRINT_w}{\longrightarrow} w$$
$$q:\ \Sigma^* \longrightarrow \Sigma^*$$

28.4 Konstruktion von $SELF$

$$\langle SELF\rangle = \langle A\rangle \# \langle B\rangle$$

B Turing Maschine mit Ein- und Ausgabeverhalten: $\begin{matrix} z \\ \downarrow \\ \langle PRINT_z\rangle \# z \end{matrix}$

A Turing Maschine $A = PRINT_{\langle B\rangle}$

28.5 Was macht A gefolgt von B?

Eingabe

$$x$$
$$A\ \downarrow$$
$$\langle B\rangle$$
$$B\ \downarrow$$
$$\left\langle \underbrace{PRINZ_{\langle B\rangle}}_{A} \right\rangle \# \langle B\rangle$$
$$= \langle A\rangle \# \langle B\rangle$$

Lemma

Es gibt Turing Maschine $SELF$, die Eingabe löscht und eigene Kopdierung ausgibt.

28.6 Rekursionssatz

Sei $t:\ \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$ von Turing Maschine T berechenbare Funktion.
Eingabe hat Form $u\$v$.
$\exists$ Turing Maschine R, die die Funktion $r:\ \Sigma^* \longrightarrow \Sigma^*$ berechnet mit $\forall w \in \Sigma^*:\ r(w) = t(\langle R\rangle, w)$.

28.7 Beweis

Für $z \in \Sigma^*$ sei $PREPRINT_z$ Turing Maschine:

$$\begin{array}{c} x \\ \downarrow \\ zx \end{array}$$

Gegeben z ist $\langle PREPRINT_z\rangle$ berechenbar.
Sei T Turing Maschine

$$\begin{array}{c} u\$v \\ \downarrow \\ t(u,v) \end{array}$$

$$\langle R\rangle = \langle A\rangle \# \langle B\rangle \# \langle T\rangle$$

B:

$$\begin{array}{c} z\$w \\ \downarrow \\ \langle PREPRINT_{z\$}\rangle \# z\$w \end{array}$$

A: ($= PREPRINT_{\langle B\rangle\#\langle T\rangle\$}$)

$$\begin{array}{c} x \\ \downarrow \\ \langle B\rangle \# \langle T\rangle \$x \end{array}$$

$$w$$

$$A\downarrow$$

$$\langle B\rangle \# \langle T\rangle \$w$$

$$B\downarrow$$

$$\left\langle \underbrace{PREPRINT_{\langle B\rangle\#\langle T\rangle\$}}_{A} \right\rangle \# \langle B\rangle \# \langle T\rangle \$$$

$$\langle R\rangle = \langle A\rangle \# \langle B\rangle \# \langle T\rangle \$w$$

28.8 Problem

Ist eine SML Funktion, die mit dem kürzesten Programmtext für vorgeschriebenes Verhalten

$$MIN - TM = \{\langle M\rangle \in \{0,1\}^* \ \mid\ \forall \langle N\rangle:\ L(N) = L(M) \Rightarrow |\langle M\rangle| \leq |\langle N\rangle|\}$$

Instanz

$$\langle M \rangle \qquad M \text{ Turing Maschine}$$

Frage

Gibt es eine Turing Maschine mit $|\langle M \rangle| \leq |\langle N \rangle|$ und $L(N) = L(M)$?

28.9 Satz

$MIN - TM$ ist NICHT entscheidbar.

Beweis

Annahme $MIN - TM$ ist entscheidbar, und zwar durch Turing Maschine E.

Baue neue Turing Maschine A:
Eingabe x

1. Bestimme $w = \langle A \rangle$ (geht wegen Rekursionssatz)
2. $l = |w|$
3. $for\ i = l + 1\ to\ \infty$
   für jedes $\langle N \rangle \in \{0,1\}^i$ teste mit E, ob $\langle N \rangle \in MIN - TM$, wenn ja, gehe weiter
4. simuliere N auf x

Loop in 3) terminiert, da es ∞ viele Turing Maschinen Sprachen gibt und dabei ∞ viele kürzeste Turing Maschinen Kodierungen, aber nur endlich viele Turing Maschinen Kodierungen ausgeschlossen werden.

Beachte

$$L(A) = L(N)$$

aber

$$|\langle A \rangle| = l < |\langle N \rangle|$$

in Widerspruch zu $\langle N \rangle \in MIN - TM$
Widerspruch zu Existenzannahme von E. ↯

28.10 Problem

Ist es möglich, dass *a.out.gz* exekutierbar ist und dasselbe Verhalten wie *a.out* hat.
JA!

28.11 Fixpunktsatz

Sei $t: \Sigma^* \longrightarrow \Sigma^*$ durch Turing Maschine T berechenbare Funktion.

$$\exists \langle M \rangle \in \Sigma^* : \ t(\langle M \rangle) = \langle N \rangle \ \text{ und } L(N) = L(M)$$

Beweis

Betrachte Turing Maschine F
Eingabe x

1. Sei $\langle F \rangle$ eigene Kodierung (OK wegen Rekusionssatz)
2. Berechne $\langle M \rangle = t(\langle F \rangle)$ (von T berechnet)
3. Simuliere M auf x

$$L(F) = L(M) \quad \text{und} \quad \langle M \rangle = t(\langle F \rangle)$$

Bild s. Schöning

Negativergebnis

Sprache X ist nicht entscheidbar

Was wenn

X ist entscheidbar, aber

- **jede** TM braucht dafür ’ßehr viel„ Zeit (Rechenschritte)
- oder ’ßehr viel„ Platz (Bandzellen)

SCHWARZ	GRAU	WEISS
unentscheidbar $\cdots$	$\cdots$	$\cdots$ entscheidbar

28.12 Definition

$$f, g: \mathbb{N} \longrightarrow \mathbb{R}$$

$f \geq_{\text{fü}} g$: $\forall n \in \mathbb{N}$, bisauf endlich viele Ausnahmen, gilt $f(n) \geq g(n)$
$f \leq_{\text{fü}} g$: $\exists n \in \mathbb{N}, \forall n \geq n_0 : f(n) \geq g(n)$, analog

28.13 Definition

$$f, g \geq_{\text{fü}} 0$$

$$\mathcal{O}(f) = \{g \mid \exists c > 0 : g \leq_{\text{fü}} c \cdot f\} \tag{1}$$
$$\wr(f) = \{g \mid \forall c > 0 : g \leq_{\text{fü}} c \cdot f\} \tag{2}$$
$$\Omega(f) = \{g \mid \exists c > 0 : g \geq_{\text{fü}} c \cdot f\} \tag{3}$$
$$\wr(f) = \{g \mid \forall c > 0 : g \geq_{\text{fü}} c \cdot f\} \tag{4}$$
$$\Theta(f) ? \mathcal{O}(f) \cap \Omega(f) \tag{5}$$

Beispiel

$$g(n) = 2n^2 + 5n - 7$$

Behauptung:$g \in \mathcal{O}(n^2)$

$$2n^2 + 5n - 7 < 2n^2 + 5n \leq 2n^2 + 5n^2 = 7n^2 \quad \forall n \in \mathbb{N}$$
$$\forall n \in \mathbb{N} : g(n) \leq 7n^2$$

Behauptung:$g \notin \mathcal{O}(n)$

$$2n^2 + 5n - 7 < 2n^2 + 5n \leq_{\text{fü}} c \cdot n \quad \forall n \geq n_0$$
$$2n + 5 - \frac{7}{n} \leq c \quad \forall n \geq n_0$$

28.14 Definition

1. Wir sagn, eine deterministische Turingmaschine M hat eine wosrt-case Laufzeit von $g(n)$, wenn (Laufzeit im schlechtesten Fall)

   M für jedes Eingabewort w nach höchstens $g(|w|)$ Schritte hält.

2. Wir sagen, eine deterministische TM M hat einen worst-case Platzverbrauch von $g(n)$, wenn M für jedes Eingabewort w höchstens $g(|w|)$ Speicherzellen pro Arbeitsband verwendet.

3. M nicht-deterministische TM
   M hat worst-case Laufzeit $g(n)$, wenn für jedes $w \in L(M)$ es eine akzeptierende Berechnung gibt, die aus höchstens $g(|w|)$ vielen Rechenschritten besteht.

4. M nicht-deterministische TM

   M hat worst-case Platzverbrauch $g(n)$, wenn für jedes $w \in L(M)$ es eine akzeptierende Berechnung gibt, die pro Arbeitsband höchstens $g(|w|)$ viele Zellen verwendet.

28.15 Definition Komplexitätsklassen

$f : \mathbb{N} \longrightarrow \mathbb{R}^+$

- $DTIME(f)$

  ist die Familie aller Sprachen, die durch irgendeine deterministische TM mit worst-case Laufzeit in $\mathcal{O}(f)$

- $DPSACE(f)$

  ist die Familie aller Sprachen, die durch irgendeine deterministische TM mit worst-case Platzverbrauch in $\mathcal{O}(f)$

- $NTIME(f)$

  ist die Familie aller Sprachen, die durch irgendeine NICHT-deterministische TM mit worst-case Laufzeit in $\mathcal{O}(f)$ akzeptiert werden können.

- $NSPACES(f)$

  ist die Familie aller Sprachen, die durch irgendeine NICHT-deterministische TM mit worst-case Platzverbrauch in $\mathcal{O}(f)$ akzeptiert werden können.

Beispiel

$S = \{a^i b^i \mid i \in \mathbb{N}\}$

1. $S \in DTIME(n^2)$

   TM : lasse Kopf zwischen linkem und rechtem Ende hin- und herwandern und streiche jeweils ein a oder b,...

   Laufzeit: $n = 2i$

   $\sum_{1 \leq j \leq i} 2j = 2\binom{i+1}{2} = \mathcal{O}(i^2) = \mathcal{O}(n^2)$

2. $S \in DTIME(n \cdot \log n)$
   a ~~a~~ ~~ab~~ ~~b~~ ~~b~~

   $\log i$ Phasen in jeder Phase streihe jedes 2. a und jeedes 2. b wg. Parität.

3. $S \in DTIME(n)$
   Verwende 2-tes Band!

4. $S \in DSPACE(\log n)$
   Man braucht nur 2 Zähler, die bis höchstens n zählen können müssen.

Wie zeigt man $S \in DTIME(f)$?Wie zeigt man $S \in DSPACE(f)$?Gib geeignete TM an!Wie zeigt man $S \notin DTIME$

28.16 Lemma 1

$$\forall f : DTIME(f) \subseteq \left\{ \begin{array}{c} DSPACE(f) \\ NTIME(f) \end{array} \right\} \subseteq NSPACE(f)$$

28.17 Frage

Warum gilt $\subsetneq DTIME(n) \subsetneq NTIME(n)$

28.18 Lemma 2

$$f \in \wr(g) \implies \begin{array}{c} DTIME(f) \\ DSPACE(f) NTIME(f) NSPACE(f) \end{array} \subseteq DTIME(g) \begin{array}{c} DTIME(g) \\ DSPACE(g) NTIME(g) NSPACE(g) \end{array}$$

28.19 Frage

Unter welchen Bedingungn für f und g gilt $DTIME(f) \subsetneq DTIME(g)$

28.20 Lemma 3

$$L \in DTIME(f) \implies \overline{L} \in DTIME(f)$$
$$L \in DSPACE(f) \implies \overline{L} \in DSPACE(f)$$

28.21 Frage

Gilt Analoges auch für $NTIME$ und $NSPACE$?

28.22 Beispiel

$L?\{w \in \{0,1\}^* \mid \langle w \rangle_2 \text{ ist zusammengesetzte Zahl}\}$

Lemma

$$\begin{array}{ccc} DTIME(f) & \subseteq & DSPACE(f) \\ \subseteq & & \subseteq \\ NTIME(f) & \subseteq & NSPACE(f) \end{array}$$

Lemma A

1. $L \in DTIME(f) \quad \Rightarrow \quad L \in DSPACE(f)$
2. $L \in DSPACE(f) \quad \Rightarrow \quad L \in DTIME(a^{f(n)})$

für irgendeine von L abhängige Konstante a.
Vorraussetzung: $f(n) \geq \log n$

Lemma B

1. $L \in DTIME(f(n)) \quad \Rightarrow \quad L \in NTIME(f(n))$
2. $L \in NTIME(f(n)) \quad \Rightarrow \quad L \in DTIME(b^{f(n)})$

für irgendeine von L abhängige Konstante b.

Lemma C

1. $L \in DSPACE(f(n)) \quad \Rightarrow \quad L \in NSPACE(f(n))$
2. $L \in NSPACE(f(n)) \quad \Rightarrow \quad L \in DSPACE(f^2(n))$

Vorraussetzung: f kann unter Verwendung von $f^2(n)$Platz berechnet werden.

Korrolar

$\gamma : \mathbb{N} \to \mathbb{R}^+$, nicht fallend, $\lim\limits_{n\to\infty} f(n) = \infty$.
$\log^* n \quad \alpha(\alpha(\alpha(n)))$

$$\begin{array}{ccc} DSPACE(f(n)) & \subseteq & DTIME(2^{\gamma(n)f(n)}) \\ NTIME(f(n)) & \subseteq & DTIME(2^{\gamma(n)f(n)}) \\ NSPACE(f(n)) & \subseteq & DSPACE(f^2(n)) \end{array}$$

Beweis: Lemma A

1. trivial
   in X Schritten können höchstens X Bandzellen pro Band berührt werden.
2. $L \in DSPACE(f(n))$, d.i. $\exists$ Turing Maschine M, die L mit worst-case Platzverbrauch $cf(n)$ entscheidet, c Konstante

   M : $Q \ldots$# der Zustände
   $B \ldots$Größe des Arbeitsalphabets
   $k \ldots$# Arbeitsbänder

Eingabe w, $n = |w|$, $N = c \cdot f(|w|)$
Wieviele verschiedene Konfigurationen kann M bei Eingabe w erreichen?

$$Z = Q \cdot n \cdot \left(B^N\right)^k \cdot N^k$$

Jede Konfiguration kann während Berechnung höchstens 1 mal besucht werden (sonst LOOP!).
Also terminiert Berechnung nach $\leq Z$ Schritten.
$Q \leq N$ wenn n groß genug

$$\begin{aligned} Q \cdot n \cdot \left(B^N\right)^k \cdot N^k &\leq \underbrace{N}_{\leq Z^N} \cdot Z^{f(n)} \cdot \left(B^N\right)^k \cdot \underbrace{N^k}_{Z^N} \\ &\leq 2^{f(n)} \left(2^{k+1}\right)^N \cdot \left(B^N\right)^k \qquad N = c \cdot f(n) \\ &= \left(\underbrace{2 \cdot \left(2^{k+1}\right)^N \cdot \left(B^k\right)^c}_{=a} \right)^{f(n)} \end{aligned}$$

Beweis: Lemma B

1. trivial

2. M sei nicht-deterministische Turing Maschine, die L in $c \cdot f(n)$ Zeit akzeptiert.
   Baue deterministische Maschine M' mit einem extra Band, das LEitstring enthält, der die nicht-deterministische Entscheidung in jedem Schritt bestimmt.
   Eingabe w, $n = |w|$
   es gibt akzeptierenden Pfad der Länge $\leq c \cdot f(n)$
   Es reicht ein LEitstring der Länge $c \cdot f(n)$
   M' pribiert M für jedden der $\alpha^{c \cdot f(n)}$ vielen Leitstrings, $\alpha = \max$ # Wahlmöglichkeiten pro Schritt.
   Laufzeit von M' ist

$$\leq \alpha^{c \cdot f(n)} \underbrace{c \cdot f(n)}_{\leq 2^{f(n)}} \leq c \cdot \left(\underbrace{2\alpha^b}_{=b} \right)^{f(n)}$$

   $w \notin L$: Wird festgestellt, wenn jeder Leitstring scheitert.

Beweis: Lemma C

1. trivial

2. „**Satz von Savitck**“
   M nicht-deterministische Turing Maschine, die L mit $c \cdot f(n)$ Platz akzeptiert.
   Eingabe w, $n = |w|$, $N = c \cdot f(|w|)$
   Idee:
   Betrachte den Konfigurationsgraphen, und stelle fest, ob es einen Pfad von $INIT(w)$ zu einer Endkonfiguration gibt.
   Sei $G_w = (C_w, E_w)$ „relevanter Teil“ des Konfigurationsgraphen.
   C_w alle Konfigurationen mit w auf Eingabeband und $\leq N$ verwendeten Zellen pro Arbeitsband.

$$|C_w| \leq a^{f(n)}$$

   E_w gerichtete Kanten zwischen Konfigurationen und C_w
   Beachte: i) Jede Konfiguration $K \in C_w$ lässt sich in $\mathcal{O}(f(n))$ Platz darstellen.
   ii) Für $K, K' \in C_w$ lässt sich feststellen, ob $[K, K'\rangle \in E_w$
   iii) Man kann alle Konfigurationen in C_w aufzählen.
   iv) Man kann feststellen, ob $K \in C_w$ eine akzeptierte Endkonfiguration ist.
   Algorithmus (deterministisch!) fürs Testen, ob M Eingabe w akzeptiert:

```
for each  F in C_w  do
     if  F  Endkonfiguration  AND  Pfad(init(w), F, |C_w|)
         then akzeptiere  w
              stop
verwerfe  w
stop
```

Pfad (A, B, l) testet, ob es einen Pfad von A nach B der Länge höchstens l in G_w gibt.
Pfad (A, B, l) vie tiefensuche zu realisieren braucht viel zu viel Zeit.
Divide and Conquer!
Pfad (A, B, l)

```
if  l=0  and  A=B
    then  YES
else
    if  l=1  and [A,B> ist Kante in G_w
        then  YES
    else
        for each  X in C_w  do
            if  Pfad( A, X, floor(l/2) )  and  Pfad( X, B, ceil(l/2) )
                then  YES
NO
```

$S(l)$ Platz, die von Pfad (A, B, l) verwendet wird.
$S(0) = S(1) = \mathcal{O}(f(n))$

$$s(l) = \mathcal{O}(f(n)) + S(l/2)$$

$\Rightarrow S(l) = \mathcal{O}(f(n) \log l)$

Lemma D: Komplementbildung

1. $L \in DTIME(f) \quad \Rightarrow \quad \bar{L} \in DTIME(f)$
2. $L \in DSPACE(f) \quad \Rightarrow \quad \bar{L} \in DSPACE(f)$
3. $L \in NSPACE(f) \quad \Rightarrow \quad \bar{L} \in NSPACE(f)$

„Satz von Immermann / Szelepcsényi“

Beweis zu 3)

M nicht-deterministisch Turing Maschine für L, Platzverbrauch ist $\mathcal{O}(f(n))$.
$\forall w \in L$
existiert ein Pfad in $G_w = (C_w, E_w)$ von $init_w$ zu irgendeinem Knoten in $Finit_w \subset C_w$ der Länge $\leq c \cdot a^{f(n)} = \Lambda$, $n = |w|$.

Gesucht:
NICHT-deterministische Methode, die für $w \notin L$ beweißt, dass es so einen Pfad nicht gibt und dabei nur $\mathcal{O}(f(n))$ Platz verwendet.

$$c_w(l) = \{K \in C_w \mid \text{es gibt Pfad des Länge} < l \text{ von } init_w \text{ nach } K\}$$

$$N_w(l) = |C_w(l)| < c \cdot a^{f(n)}$$

Ziel:
Zeige für alle $K \in Finit_w$, dass $K \notin C_w(Lambda)$
denn das bedeutet, es gibt keinen Pfad von $init_w$ nach $Finit_w$.

Behauptung 1

Mann kann nicht-deterministisch mit $\mathcal{O}((f(n))$ Platz beweisen, dass gegebene Konfiguraton K in $C_w(l)$.
Beweis
Rate Pfad durch G_w (zähle Länge mit) ≤ 2 Knoten müsssen gleichzeitig gesucht werden.

Behauptung 2

Wenn $N_w(l)$ bekannt ist, dann kann für gegebene K mit $\mathcal{O}(f(n))$ Platz bewiesen werden, dass $K \notin C_w(l)$ nicht-deterministisch.
Beweis

```
z = 0
for each  X in C_w \ {K}  do
    falls  X in C_w(l)  dann beweise dies mit Behauptung 1
    und z = z+1
if z=N_w(l-1)  mit  O(f(n)) Platz
```

Behauptung 3

Gegeben $N_w(l-1)$ kann man $N_w(l)$ mit $\mathcal{O}(f(n))$ Platz nicht-deterministisch berechnen.
Beweis

```
y = 0
for each  X in C_w  do
    for each  Y in Vorg"ange( X ) U {K}  do
        if  Y in C_w(l-1)  then  y = y + 1;
        break
        Y - loop
N_w(l) = y
```

Behauptung 4

$$N_w(0) = 1$$

28.23 Algorithmus: Test auf Akzeptierbarkeit

Algorithmus zum Test, dass w nicht von M akzeptiert wird:

```
for  i=1  to  LAMBDA  do
    berechne N_w(i) von N_w(i-1)    ( Nach Behauptung 3)
for each  X in C_w  do
    if  X in Finit_w  and  X in C_w(LAMBDA)  then  return true;
```

Lemma E

- $DTIME(f) \quad \subset \quad DTIME(g)$
- $DSPACE(f) \quad \subset \quad DSPACE(g)$

trivial

Lemma F: „Hierarchiesätze“

1. $f \in \wr(g)$
   g in $\mathcal{O}(g)$ Platz
   $$SPACE(f(n)) \quad \subsetneq \quad DPSACE(g(n))$$

2. $f(n) \in \wr\left(\frac{g(n)}{\log g(n)}\right)$
   g in $\mathcal{O}(g(n))$ Zeit berechenbar
   $$DTIME(f(n)) \quad \subsetneq \quad DTIME(g(n))$$

28.24 Einschub: „Borolin's Gap Theorem“

Es gilt eine berechenbare Funktion $f: \mathbb{N} \longrightarrow \mathbb{N}$ mit $\overline{\lim_{n\to\infty}} = \infty$ so dass

$$DSPACE(f(N)) = DSPACE\left(2^{2^{2^{f(n)}}}\right)$$

Beweis Lemma F

1. Wollen zeigen:
   $\exists$ Sprache a, die mit $\mathcal{O}(g(n))$ Platz entschieden werden kann, aber nicht mit $(g(n))$ Platz.
   Spezifizieren A über Turing Maschine D
   D: Eingabe ✓

   a) $n = |w|$

   b) berechne $g(n)$

   c) berechne $2^{g(n)}$

   d) w nicht von Form $\langle M \rangle \$0^* \longrightarrow$ antworte NEIN

   e) Simuliere M auf Eingabe w
   Zähle # Schritte von M;
   $> 2^{g(n)} \longrightarrow$ antworte NEIN
   Zähle verwendete Bandzellen
   $> g(n) \longrightarrow$ antworte NEIN

   f) wenn M das w akzeptiert, dann antworte NEIN, sonst antworte JA

2. siehe Sipser, Hopecroft

Behauptung 1

Platzverbrauch von D ist $\mathcal{O}(g(n))$ also

$$A \in DPSACE(g(n))$$

Behauptung 2

Sei N Turing Maschine mit Platzverbrauch $f(n)$, $f(n) \in \wr(g(n))$

$$\Rightarrow L(N) \neq A$$

Idee

Zeige $\exists\, m \in \mathbb{N}:\ \langle N\rangle\, \$0^m \in L(n)$

$$\iff \langle N\rangle\, \$O^m \notin A$$

Betrachte N:
Bei Eingabe u, $|u| = n$
$\leq f(n)$ Platz
$\leq a^{f(n)}$ Schritte für irgendein a

$$f(n) \in a(g(n)) \Rightarrow f(n) \leq g(n) \text{für } n \geq n_0$$
$$a^{f(n)} \leq 2^{g(n)} \text{für } n \geq n_1$$

wähle $m = \max\{n_0, n_1\}$
Bei Eingabe $\underbrace{\langle N\rangle\, \$0^m}_{=w}$ erreicht D Schritt 6, und dann gilt

$$w \in L(D) \iff w \notin L(N)$$

29 Komplexitätsklassen

$$P = \bigcup_{k\geq 0} DTIME(n^k)$$

Probleme, die sich in polynomieller Laufzeit deterministisch lösen lassen.

$$NP = \bigcup_{n\geq 0} NTIME(n^k)$$

Probleme, deren Lösungen sich in polynomieller Zeit verifizieren lassen.

$$PSPACE = \bigcup_{K\geq 0} DSPACE(u^k)$$
$$NSPACE = \bigcup_{K\geq 0} NSPACE(u^k)$$
$$L = DSPACE(\log n)$$
$$NL = NSPACE(\log n)$$
$$L \subseteq NL \subseteq P \subseteq NP \subseteq \underbrace{PSPACE \subseteq NPSPACE}_{=}$$

Behauptung: $PSPACE = NPSPACE$

Beweis

1. $L \in PSPACE \Longrightarrow L \in NPSPACE$ ✓
2. $L \in \underbrace{NPSPACE}_{=\bigcup_{k\geq 0} NSPACE(n^k)} \Longrightarrow L \in \underbrace{NSPACE(n^k)}_{\text{für irgendein } k} \Longrightarrow L \in DSPACE((n^k)^2)$
   $\Longrightarrow L \in PSPACE$

Behauptung $NL \neq PSPACE$

Beweis

$$NL = NSPACE(\log n) \subseteq DSPACE(\log^2 n) \subsetneq DSPACE(n) \subseteq PSPACE$$

P ... Probleme, die in $\underbrace{\text{polynomieller Zeit}}_{\text{‘effizient lösbar“}}$ lösbar sind.

$$\textit{polynomiell} \leftrightsquigarrow \text{„gut“}$$
$$\textit{superpolynomiell} \leftrightsquigarrow \text{„schlecht“}$$

Warum diese Definition?

- mathematisch angenehm (Polynome sind abgeschlossen unter Addition, Multiplikation und Komposition)
- verschiede Arten von Lösbarkeit fallen zusammen.

Beispiel

Definition
$G = (V, E)$ ungerichteter Graph
$W \subset V$ heißt Clique, wenn $\forall\, u, v \in W : u \neq v\, [u, v\} \in E$

29.1 3 Varianten des Clique-Problems

1. Variante (Entscheidbarkeit)
   gegeben: Graph $G = (V, E), k \in \mathbb{N}$ Frage: $\exists$ *Clique* W in G, mit $|W| = k$

2. Variante (Optimierungsproblem) gegeben: Graph $G = (V, E)$ gesucht: Finde größtes k, sodass G eine *Clique* der Größe k besitzt.

3. Variante (Optimierungs-/Berechnungsproblem) gegeben Graph $G = (V, E)$
   Berechne Clique W von maximaler Größe.

maximal clique $\Longleftarrow$ „nicht vergößerbar“
maximum clique $\Longleftarrow$ „größt-möglich“

Klar

Von 3 in polynomieller Zeit lösbar $\Longrightarrow$ Var 2 in polyomieller Zeit lösbar.
Von 2 in polynomieller Zeit lösbar $\Longrightarrow$ Var 1 in polyomieller Zeit lösbar.

29.2 Lemma

(i) Var 1 in poly. Zeit lösbar $\Longrightarrow$ Var 2 in poly. Zeit lösbar.

(i) Var 2 in poly. Zeit lösbar $\Longrightarrow$ Var 3 in poly. Zeit lösbar.

Beweis für (i)

Lösung für Variante 2:

Probiere für jedes k zwischen 1 und $n = |V|$, ob es Clique der Größe k gibt.

Beweis für (i)

Bestimme k = Größe der maximum Clique G

Nimm sukzessive Knoten weg und teste, ob Graph weiterhin maximale Clique der Größe k hat.

29.3 Frage: Gibt es einen effizienten (also poly. Zeit) det. Algorithmus für das Clique Problem?

Vermutung NEIN!!

Achtung: Das Clique Problem (Entscheidungs-Var) ist in NPd.h. $\exists$ nicht det. Methode, die in poly. Zeit verifiziert, dass G eine Clique der Größe h hat.

Das ist ein Beispiel von vielen wichtigen Problemen, für die eine Lösung effizient verifiziert werden kann, man aber nicht weiß, wie man die Lösung effizient findet.

Weiter Beispiele

BIN PACKING:
Gegeben: $a_1, \ldots, a_n, b \in \mathbb{N}$ binär kodiert

Frage

$\exists f: \{1, \ldots, n\} \longrightarrow \{1, \ldots, k\}$, so dass für jedes j, $1 \leq j \leq k$ gilt

$$\sum_{i:f(i)=j} a_i \leq b$$

29.4 TRAVELLING SALESPERSON

Gegeben

vollständig gerichteter Graph $G = (V, E)$ mit Kantengewichten.

Frage

$\exists$ Permutation $\Pi \in S_n$, so dass

$$c([\Pi(1), \Pi(2)\rangle) + c([\Pi(2), \Pi(3)\rangle) + \ldots c([\Pi(n-1), \Pi(n)\rangle) + c([\Pi(n), \Pi(1)\rangle) \leq B$$

„kürzeste Rundreise“

29.5 Polygon Wachmann Problem

Gegeben

Einfaches Polygon in der Ebene.
$k \in \mathbb{N}$

Frage

Kann man k Wachmänner so im Polygon stationieren, das jeder Punkt im Polygon von mindestens einem der Wachmänner „gesehen“ wird.

29.6 INTEGER PROGRAMMING

Gegeben

$$a_{ij} \in \mathbb{Z}$$

$1 \leq i \leq m$, $0 \leq j \leq n$, binär kodiert.

Frage

$$\exists (x_1, \ldots, x_n) \in \mathbb{Z}^n$$

so dass für $1 \leq i \leq m$:

$$a_{in} x_1 + a_{<} a_{i0}$$

30 Polynomzeit Reduktionen

Definition

Seinen $L, L' \subset \Sigma^*$ Sprachen.
L heißt polynominell leichter als L' (L polynominell reduziertbar auf L')

$$L \leq_p L'$$

wenn es eine Funktion $f: \Sigma^* \longrightarrow \Sigma^*$ gibt mit

1. $\forall w \in \Sigma^*$: $w \in L \iff f(w) \in L'$
2. f ist in polynomieller Zeit berechenbar

Lemma 0

$$L \leq_p L'$$

und

$$L' \in P \Rightarrow L \in P$$

Beweis

$$L \leq_p L'$$

$\exists$ Turing Maschine F, Polynomialzeit $\mathcal{O}(n^k)$ berechnet Funktion $f: w \in L \iff f(w) \in L'$

$$L' \in P$$

$\exists$ Turing Maschine M', Polynomialzeit $\mathcal{O}(n^c)$ M' entscheidet L'

Baue Turing Maschine M für L:
Eingabe w
Berechne $f(w)$ mit F
Teste ob $f(w) \in L'$ mit M'

Lemma 1

$$L \leq_p L'$$

und

$$L' \leq_p L''$$

$$\Rightarrow \quad L \leq_p L''$$

Beweis

siehe Übung

30.1 Definition

Eine Sprache S heißt $\underline{NP\text{-schwer}}$, wenn fpr $L \in NP$ gilt

$$L \leq_p S$$

S heißt $\underline{\text{NP-vollständig}}$, wenn S NP-schwer und $S \in NP$

30.2 Satz 1

S NP-vollständig und $S \in P \Rightarrow P = NP$
($P \neq NP$ und S NP-vollständig $\Rightarrow S \notin P$)

Beweis

Wissen $P \subseteq NP$
Müssen noch zeigen Annahmen $\Rightarrow NP \subset P$

$$L \in NP \Rightarrow L \in P$$

$$\begin{array}{l} L \in NP \\ S\ NP\text{-vollständig} \end{array} \Rightarrow L \leq_p S \Rightarrow L \in P$$

Wenn man von einer Sprache L zeigt, dass L NP-vollständig, dann ist dies auf Grund der Vermutung $P \neq NP$ ein sarker Grund zur Annahme, dass $L \notin P$, also, dass L nicht effizient lösbar ist.

Wie zeigt man. dass L NP-schwer?

1. direkt
2. mit Lemma 2
   Suche dir ein NP-schweres Problem L'

Lemma 2

$$\left.\begin{array}{l} L'\ NP\text{-schwer} \\ L' \leq_p L \end{array}\right\} \Rightarrow L\ NP\text{-schwer}$$

$$\left.\begin{array}{l} S \in NP \\ L'\ NP\text{-schwer} \end{array}\right\} \Rightarrow L\ S \leq_p L' \Bigg\} \Rightarrow S \leq_p L$$

Gibt es NP-vollständige Sprachen?

$$JA$$

Beispiel 1

$P-UNIV = \{\langle M\rangle\,\$\,\langle w\rangle\,\$1^m \mid \text{nicht-deterministische Turing Maschine } M \text{ akzeptiert } w \text{ in} \leq m \text{ Schritten}\}$

Behauptung

$P-UNIV$ ist NP-vollständig.

1. $P-UNIV \in NP$
   simuliere M ein Eingabe w für m richtig geratene Schritte
2. $L \in NP$, müssen zeigen $L \leq_p P-UNIV$
   $\Rightarrow \exists$ nicht-deterministische Turing Maschine M mit $L(M) = L$ und Laufzeit $c \cdot n^k$
   $w \in \Sigma^*$
   Erzeuge

$$\underbrace{\langle M\rangle}_{\mathcal{O}(1)}\,\$\,\underbrace{\langle w\rangle}_{\mathcal{O}(n)}\,\$\,\underbrace{1^{c\cdot|w|^k}}_{\mathcal{O}(n^k)} \quad \in P-UNIV$$

Beispiel 2

Mehrwertige Erfüllbarkeit $MERF$
Instanz:
Variablen $Y_1, \ldots, Y_n$, endliche Wertebereiche $B_1, \ldots, B_n$ jede Variable Y_i kann einen der Werte auf B_i annehmen.
Formel F aufgebaut als $\wedge$ und $\vee$ von Termen:
Terme:

$$Y_i = a \qquad Y_i \neq a \qquad Y_i = Y_j \qquad Y_i \neq Y_j$$

Frage

Gibt es eine Variablenbelegung, die F wahr macht?

Beispiel

$$Y_1, Y_2, Y_3 \quad B_i = \{1, 2, 3, 4, \} \; 1 \leq i \leq 4$$

$$F = (Y_1 = 1 \vee Y_2 = 3) \wedge (Y_2 = 2 \vee X_3 = 1) \wedge (Y_1 = 2 \vee Y_3 = 2) \wedge X_2 = X_3$$

wird war mit Belegung

$$Y_1 = 1$$
$$Y_2 = 2$$
$$Y_3 = 2$$

$$MERF = \{\langle Y_1 \rangle \langle Y_2 \rangle \ldots \langle Y_n \rangle \, \$ \, \langle B_1 \rangle \ldots \langle B_n \rangle \, \$ \, \langle F \rangle \;\mid\; F \text{ erfüllbar } \}$$

Behauptung

$MERF$ ist NP-vollständig.

1. $MERF \in NP$
   errate vorraussichtliche Belegungen, verifiziere F
2. $MERF$ $NP-$schwer
   für jedes $L \in NP$ git $L \leq_p MERF$

$$L \in NP \Rightarrow \exists \text{ Turing Maschine } M = (Q, \Sigma, \Gamma, S, F, \bar{b}, \Delta) \text{ mit } L(M) = L$$
$$\subset Q \times \Sigma \times Gamma \times Q \times B \times \Gamma \times B$$

mit Laufzeit $g(n)$ polynomiell!

gegeben:
$w \; in \Sigma^*$
Konstruiere $MERF$ Formel F, so dass

$$F \text{ erfüllbar} \iff M \text{ akzeptiert } w \; (m \leq p = |g(w)| \text{ Schritte})$$

Idee für F
Simuliere Schritt von M

			Zustand	EK	AK
0			Z_0	I_0	J_0
$z-1$			Z_{z-1}	I_{z-1}	J_{z-1}
z			Z_t	I_t	J_t
$\ldots$					
P			Z_P	I_P	J_P

Instanz von $MERF$

	E_i	$-p \leq i \leq p$	Eingabebandzellen Wertebereich $\Sigma \cup \{bslash\}$	
	A_j^t	$-p \leq j \leq p$, $0 \leq t \leq p$	Arbeitsbandzellen zur Zeit t, Wertebereich Γ	
Variablen:	Z^t	$0 \leq t \leq p$	Zustand zur Zeit t, Wertebereich Q	# Varia-
	I^t	$0 \leq t \leq p$	Position des Eingabekopfes, Wertebereich $\{-p, \ldots, p\}$	
	J^t	$0 \leq t \leq p$	Position des Arbeitskopfes, Wertebereich $\{-p, \ldots, p\}$	

blen

$$\mathcal{O}(p) + \mathcal{O}(p^2) + \mathcal{O}(p) = \mathcal{O}(p^2)$$

Größe der Wertebereiche:
Maximal $\mathcal{O}(p)$ fpr I_t, J_t, sonst $\mathcal{O}(1)$

Formel

$$\text{F} = \text{Anfang} \wedge \bigwedge_{1 \leq t \leq p} (\text{ legaler Übergang von Konfiguration } t-1) \; land \; \text{Ende}$$

Anfang

$$\bigwedge_{-p \leq t \leq 0} (E_i = t_0) \wedge \bigwedge_{0 \leq t \leq |w|} (E_i = w_i) \wedge \bigwedge_{|w| \leq t \leq p} (E_i = t_0) \wedge \bigwedge_{-p \leq t \leq p} (A_j^0 = t_0) \wedge (Z^0 = s) \wedge (I^0 = 0) \wedge (J^0 = 0)$$

Ende

$$\bigvee_{f \in F} (Z^p = f)$$

Annahme:
Maschine macht „leer“ weitere Schritte in jedem Endzustand

$$\ddot{U}^t = \bigvee_{-p \leq i \leq p, \; -p \leq j \leq p} \ddot{U}_{ij}^t$$

$\ddot{U}$ ist der Übergang, wenn Eingabekopf in Position i und Arbeitskopf in Position j ist.

$$\ddot{U}_j^t (I^{t-1} = i) \wedge (J^{t-1} = j) \wedge \bigwedge_{-p \leq l \leq p, \; l \neq j} (A_l^t = A_l^{t-1}) \wedge \bigvee_{(q,a,B,q',\lambda,B',\mu) \in \Delta} (Z^{t-1} = z) \wedge (E_i = a) \wedge (Z^t = q') \wedge (I^t = i + \lambda) \wedge (A_j^t = B$$

Beachte

1. F erfüllbar $\iff$ M akzeptiert w $\iff$ $w \in L$
2. F ist polynomiell groß on $|w|$
3. F kann in polynomieller Zeit aus w und M berechnet werden.

Beispiel 3

$SIMPLE$ Mehrwehrtige Erfüllbarkeit
$SMERF$ wie $MERF$
nur dürfen in Formel nur Terme der Form $Y_i = a$ vorkommen (Also kein $Y_i \neq 0$, $Y_i = Y_j$, $Y_i \neq Y_j$)

Behauptung $SMERF$ ist NP-vollständig

Beweis Übung

Beispiel: SAT „Bool'sche Erfüllbarkeit"

Gegeben

Bool'sche Formel F in Bool'schen Variablen, z.B.:

$$(X_1 \wedge X_2 \vee \bar{X}_3) \wedge (X_3 \vee \bar{X}_4)$$

Gefragt:

Ist F erfüllbar? Gibt es Wahrheitsverteilungen für die X_i, sodass F wahr.

Behauptung SAT ist NP-vollständig

Beweis

1. $SAT \in NP$ ✓ Variablenbelegung erraten, F verifizieren
2. $SMERF \leq_p SAT \longrightarrow SAT$ ist NP-vollständig, da $SMERF$ NP-vollständig

Für jede Instanz

$$\langle Y_1 \rangle \ldots \langle Y_n \rangle \,\$\, \langle B_1 \rangle \ldots \langle B_n \rangle \,\$F'$$

von $SMERF$ brauchen wir Bool'sche Formel F mit F' erfüllbar $\Longleftrightarrow$ F erfüllbar.

Variablen von F

$$\underbrace{X_{ib}}_{\text{kodiert } Y_i = b} \qquad 1 \leq i \leq n \quad b \in B_i$$

Bilde aus F' neue Formel, in der Terme der Form $Y_i = b$ durch X_{ib} ersetzt werden.
und verwende mit

$$\bigwedge_{1 \leq i \leq n} D_i$$

$D_i \approx Y_i$ nimmt nur einen der möglichen Werte aus B_i an

$$D_i = \bigvee_{b \in B_i} \left(X_{ib} \wedge \bigwedge_{b' \in B_i,\ b' \neq b} \overline{X_{ib'}} \right)$$

$$|B_i|^2 \leq \mathcal{O}(m^2)$$

mit m, der Größe der $SMERF$ Instanz.

Beispiel 5: $3 - SAT$

`http://de.wikipedia.org/wiki/3-SAT`

Gegeben Bool'sche Formel in Konjunktiver Normalform (KNF) mit ≤ 3 Literale pro Klausel

Beispiel

$$(X_1 \vee \bar{X}_2 \vee X_4) \wedge (\bar{X}_1 \vee X_3 \vee X_4) \wedge (X_2 \vee \bar{X}_3 \vee \bar{X}_4)$$

Frage Ist Formel erfüllbar?

Behauptung $3 - SAT$ ist NP-vollständig

Beweis

1. $\mathcal{S} - SAT \in NP$ ✓
2. $SAT \leq_p 3 - SAT$

F SAT Formel (vollständig geklammert)
wollen: $3 - SAT$ Formel F' mit

$$F' \text{ erfüllbar} \quad \Longleftrightarrow \quad F \text{ erfüllbar}$$

Beispiel

$$F = \neg((X_1 \vee \neg(X_2 \vee \neg X_3)) \wedge X_2)$$

Verwende DeMorgan'sche Regeln um Negationen zu Blättern zu bringen.
Führt bei Blättern Literale ein

$$BAUMSTRUKTUR\ 1$$

$$BAUMSTRUKTUR\ 2$$

$$BAUMSTRUKTUR\ 3$$

Für jeden inneren Knoten v des Baumes führe Variable Y_v ein.
Baue Konjunktion von Formeln der Art

$$Y_v \Longleftrightarrow Z_n\ op\ Z_w$$

Z_1: Variable beim linken Kind von v
Z_w: Variable beim rechten Kind
$op\ \in \{\wedge, \vee\}$

$$\begin{aligned} Y_{11} &\Longleftrightarrow X_2 \vee \bar{X}_3 \\ \wedge\ Y_{12} &\Longleftrightarrow \bar{X}_1 \wedge Y_{11} \\ \wedge\ Y_{root} &\Longleftrightarrow Y_{11} \vee \bar{X}_2 \\ \wedge\ Y_{root} & \end{aligned}$$

$$A \Longleftrightarrow B$$

$$(A \wedge B) \vee (\bar{A} \wedge \bar{B})$$

$$(A \vee \bar{A}) \wedge (A \vee \bar{B}) \wedge (B \vee \bar{A}) \wedge (B \vee \bar{B})$$

$$Y \Longleftrightarrow X_1 \vee X_2$$

$$(Y \vee \overline{(X_1 \vee X_2)}) \wedge (\bar{Y} \vee X_1 \vee X_2)$$

$$(Y \vee (\bar{X}_1 \wedge \bar{X}_2))$$

$$(Y \vee X_1) \wedge (Y \vee \bar{X}_2) \wedge (\bar{Y} \vee X_1 \vee X_2)$$

$$Y \Longleftrightarrow X_1 \wedge X_2$$

analog

Beispiel 6: $CLIQUE$

Gegeben ungerichteter Graph $G = (V, E)$, $k \in \mathbb{N}$

Frage Hat g eine Clique der Größe k

Behauptung $CLIQUE$ ist NP-vollständig

Beweis

1. $CLIQUE \in NP$ ✓
2. $3 - SAT \leq_p CLIQUE$

$$F = (Z_{11} \vee Z_{12} \vee Z_{13}) \wedge (Z_{21} \vee Z_{22} \vee Z_{23}) \wedge \ldots \wedge (Z_{m1} \vee Z_{m2} \vee Z_{m3})$$

Z_{ij} Literale

$$G = (V, E)$$

$$V = \{Z_{ij} \mid 1 \leq i \leq m,\ 1 \leq j \leq 3\}$$

$$E = \{(Z_{ij}, Z_{kl}) \mid i \neq h \text{ und } Z_{ij} \neq \neg Z_{kl}\}$$

Z_{ij} und Z_{kl} können gleichzeitig wahr werden.

Behauptung F erfüllbar $\Longleftrightarrow$ G hat Clique der Größe m

„$\Rightarrow$“ F erfüllbar
betrachte erfüllende Variablenbelegung
macht in jeder Klausel in Literal Z_{ij}, wahr $\{Z_{ij} \mid 1 \leq i \leq m\}$ bildet m Clique

„$\Leftarrow$“ G hat m Clique C
enthält Lieratel aus jeder Klausel
auf Grund der Kantendefinition können alle diese Literale gleizeitig wahr gemacht werden
$\Rightarrow$ Jede Klausel wird wahr $\Rightarrow$ F wird wahr.

30.2.1 Beispiel 7: Hamilton Kreis

Gegeben Gerichteter Graph $G = (V, E)$

Frage Existiert in G ein Hamilton'scher Kreis?
i.e. $\exists$ Anordnung der Kanten in G

$$v_0, v_1, \ldots, v_{n-1} \qquad n = |V|$$

mit

$$(v_i, v_{i+1}\rangle \in E \qquad \text{Indizes modulo } n$$

Behauptung Hamilton'scher Kreis ist NP-vollständig

Beweis

1. $HaKr \in NP$
2. $3-SAT \leq_p HaKr$
3. $F = K_1 \wedge K_2 \wedge \ldots \wedge K_m$
   KNF Formel in $x_1, \ldots, x_n$
   oBdA genau 3 Literale/Klausel
   Wählen: Graph G_f : G_f hat Hamilton'scher Kreis $\iff$ F erfüllbar
   G_f: $n + G_m$ Knoten
   $3n + 9m + 3m$ Kanten
   Füge jede Klausel K_j gibt es Klauselstück Baustein

Behauptung (modulo Symetrie) gibt es nur 3 Arten, wie ein Klauselbaustein von einem Hamilton'schen Kreis durchquert werden kann.

PICTURE

Daraus folgt, das ein etwaiger Hamilton'scher Kreis den Baustein immer bei dem zum Eingang korrespondierenden Ausgang verlässt.

PICTURES KNOTEN

Einen „Ring" von n Knoten, einen für jedes x_i.
Jeden dieser Knoten verlassen 2 Kanten
Jede Kante beginnt mit einem Pfad; „Pfad" für x_i führt durch alle Klauselbausteine, in denen x_i vorkommt.

PICTURE

G_f kann aus F in polynomieller Zeit konstruiert werden.

Behauptung

G_F hat Hamilton'schen Kreis $\iff$ F erfüllbar

Beweis „$\Longrightarrow$"

Beweis „$\Longleftarrow$"

$$G = (V, E)$$

Instanz von Hamilton'schen Kreis

$$\tilde{G} = (V, V \times V) \quad l([u,v\rangle) = \begin{cases} 1 & [u,v\rangle \in E \\ 2 & [u,v\rangle \notin E \end{cases}$$

$$M = |V|$$

$\tilde{G}$ hat Hamilton'schen Kreis der Länge $M \cdot |v|$ $\iff$ G hat Hamilton'schen Kreis

Beispiel 9: $SNEG$

Simultanz $0-1$ Gleichungen

Gegeben m lineare Gleichungen in n Unterkanten $x_1, \ldots, x_n$ mit Koeffizienten in $\mathbb{N}$

$$a_{j1}x_1 + a_{j2}x_2 + \ldots + a_{jn}x_n = a_{j0} \qquad j = 1, \ldots, m$$

Frage $\exists\, x_1, \ldots, x_n \in \{0,1\}$, so dass alle m Gleichungen wahr sind.

Behauptung $SNEG$ ist NP-vollständig

Beweis

1. $SNEG \in NP$
2. $3-SAT \leq_p SNEG$

F $3-KNF$ Formel
wollen lineares Gleichungssystem mit Koeffizienten aus $\mathbb{N}$ so dass

$$\exists 0-1\text{Lösung} \Longleftrightarrow F \text{ erfüllbar}$$

$$K_1 \wedge K_2 \ldots \wedge K_m \text{ in Vor. } z_1, \ldots, z_n$$

$$\ldots$$

30.3 Beispiel 10: $ENEG$

einfache $0-1$ Gleichung

Gegeben Eine Gleichung

$$x_1 x_1 + a_2 x_2 + \ldots + a_n x_n = a_0$$

mit $a_i \in \mathbb{N}$ (binär dargestellt)

Frage $\exists x_1, \ldots, x_n \in \{0,1\}$ so dass Gleichung wahr

Behauptung $ENEG$ ist NP-vollständig

Beweis

1. $ENEG \in NP$
2. $SNEG \leq_p ENEG$

Beispiel

$$\begin{array}{rrrl|l} 2x_1 & +3x_2 & +x_3 & = 7 & \cdot 10000 \\ x_1 & +2x_2 & +2x_3 & = 8 & \cdot 100 \\ x_1 & & x_3 & = 4 & \end{array}$$

$$\begin{array}{rrrr} 20000x_1 & +30000x_2 & +10000x_3 & = 70000 \\ 100x_1 & +2x_2 & +200x_3 & = 800 \\ x_1 & & x_3 & = 4 \end{array}$$

$$20101_1 + 30200x_2 + 10201x_3 = 70804$$

A_j: Summe der Koeffizienten von Gleichung j
A: $\max A_j$
$T = 2^s > A$
Multiplizieren Gleichung j mit T^{j-1} und addiere alle Gleichungen

Achtung Zahlen müssen binär dargestellt sein; sonst Transformation nicht polynomiell

30.4 Beispiel 11: $SUBSET - SUM$

Gegeben:

$$(a_1, \ldots, a_N) \in \mathbb{N}^n,\ b \in \mathbb{N}$$

Frage:

$$\exists I \subset \{1, \ldots, n\} \cdot \sum_{i \in I} a_i = b$$

Behauptung $SUBSET - SUM$ ist NP-vollstädnig, denn $SUBSET - SUM$ ist $ENEG$, nur in anderer Schreibweise.