Komplexitätstheorie (Martin Hofmann)

Mitgeschrieben von Bernhard Weiß, Sebastian Goebl und anderen

Mitschrift aus dem WiSe 2008/09

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Turingmaschinen, Berechenbarkeit, Komplexitätsklassen

1.1 Turingmaschinen

Definition 1
(k-Band Turingmaschine) Eine k-Band Turingmaschine ist ein Quintupel $T = \langle Q, \Sigma, I, q_0, F\rangle$ wobei

- Q ist endliche Menge von Zuständen
- Σ ist eine endliche Menge von Symbolen
- I ist eine Menge von Quintupeln der Form (q, s, s', d, q') wobei $q, q' \in Q, s, s' \in \Sigma^k, d \in \{L, R, s\}^k$. Für alle q, s existiert höchstens ein Quintupel der Form $(q, s, -, -, -)$ in der Menge I
- $q_0 \in Q$ Anfangszustand
- $F \subseteq Q$ Endzustände

Beispiel 1
$\Sigma = \{0, 1, \square, \#\}, k = 2, Q = \{q_0, q_1, q_2, q_3, q_4\}$

1. $q_0, (\square, \square), (\#, \#), (S, S), q_1$
2. $q_0, (1, \square), (1, \#), (S, S), q_2$
3. $q_0, (0, \square), (0, 0)(R, R), q_3$
4. $q_3, (0, \square), (0, 1)(R, R), q_3$
5. $q_3, (\square, \square), (\#, \#), (S, S), q_2$
6. $q_3, (1, \square), (1, \square), (R, L), q_4$
7. $q_4, (1, 1), (1, 1)(R, L), q_4$
8. $q_4, (\square, 0), (\square, 0), (S, S), q_1$
9. $q_4, (1, 0), (1, 0), (S, S), q_2$
10. $q_4, (\square, 1), (\square, 1), (S, S), q_2$

Definition 2 (Definition (Globaler Zustand, Konfiguration))
Sei $T = (Q, \Sigma, I, q_0, F)$ eine k-Band Turingmaschine. Ein Globaler Zustand (synonym Konfiguration) von T ist ein Tripel $\mathcal{S} = (q, w, w')$ wobei $q \in Q, w \in (\Sigma^*)^k, w' \in (\Sigma^+)^k$ (q ist der aktuelle Zustand, w das Wort vor dem Schreib/Lesekopf und w' das Wort auf und nach dem Schreib/Lesekopf). Seien $\mathcal{S}, \mathcal{S}'$ Konfigurationen. Man schreibt $\mathcal{S} \xrightarrow{T} \mathcal{S}'$ gesprochen $\mathcal{S}'$ ist die Folgekonfiguration von $\mathcal{S}$ wenn gilt: $\mathcal{S} = (q, w, w'), q \notin F$

- Sei $w = (w_1, \dots w_k), w' = (w'_1, \dots w'_k), a_i =$ erstes Symbol von w'_i
  Sei $(q, (a_1, \dots, a_k), (b_1, \dots b_k)(d_1, \dots d_k), q') \in I$ Dann $\mathcal{S}' = (q', v, v')$ $v = (v_1, \dots, v_k), v' = (v'_1, \dots, v'_k)$ wobei $v_i = \begin{cases} w_i & d_i = S \\ u & d_i = L, w_i = uc \\ \epsilon & d_i = L, w_i = \epsilon \\ w_i b_i & d_i = R \end{cases}$ $w'_i = a_i u, a_i \in \Sigma, u \in \Sigma^*$
  $v'_i = \begin{cases} b_i u & d_i = S \\ c b_i u & d_i = L, c = \text{Letztes Symbol von } w_i \text{ bzw } \square \text{ falls } w_i = \epsilon \\ u & d_i = R \end{cases}$

Definition 3 (Start-Konfiguration)
Sei $X \in \Sigma^*$ und $\square$ kommt nicht in x vor. Die Startkonfiguratin für T auf eingabe x ist definiert definiert als $(q_0, (\epsilon, \dots, \epsilon), (x, \square, \dots \square)$

Definition 4 (Berechnung)
Berechnung von T auf Eingabe x ist eine endliche oder unendliche Folge von Konfigurationen, $\mathcal{S}_0, \dots \mathcal{S}_n$ bzw $\mathcal{S}_0, \mathcal{S}_1, \dots$, sodass

- $\mathcal{S}_0$ ist die Startkonfiguration von T für x
- $\mathcal{S}_i \xrightarrow{T} \mathcal{S}_{i+1}$
- Falls die Folge endlich ist mit letztem Element $\mathcal{S}_n$ so existiert keine Folgekonfiguration $\mathcal{S}'$ mit $\mathcal{S}_n \xrightarrow{T} \mathcal{S}'$

Definition 5 (Akzeptormaschine)
Eine Akzeptormaschine ist eine TM T mit $F = \{q_A, q_R\}$(akzeptierender und verwervender Zustand) eine Eingabe x wird von solche einer Maschine akzeptiert, falls die Berechnung von x aus endlich ist und im Zustand q_A endet

Bemerkung 1
Die Beispielmaschine von oben akzeptiert alle Eingaben der Form $0^n 1^n$.

Definition 6 (Erkannte Sprache)
Sei T eine Turingmaschine $L(T) := \{x| \underbrace{T(x)}_{\text{Berechnung von T mit Startkonfiguaration x}}$ hält in Zustand $g_A\}$.

L heißt akzeptierbar, wenn $L = L(T)$ für eine Maschine T

Bemerkung 2
akzeptierbar = semi-entscheidbar = rekursiv entscheidbar

Definition 7
L wird von TM T entschieden, wenn gilt: $x \in L \Rightarrow T(x)$ hällt in Zustand g_A $x \notin L \Rightarrow T(x)$ hält in Zustand g_R

L heißt entscheidbar, wenn $\exists T$, sodass L wird von T entschieden

Bemerkung 3
Wenn keine Verwechslungsgefahr besteht, verwenden wir die Notation $L(T)$ auch für die von T entschiedene Sprache.

Bemerkung 4
Es gibt Sprachen L, die akzeptierbar, aber nicht entscheidbar sind.

Bemerkung 5
Meist betrachtet man eine Teilmenge $\Sigma_{in} \subseteq \Sigma$ als Alphabet und erweitert Akzeptierbarkeit und Entscheidbarkeit auf Sprachen $L \subseteq \Sigma_{in}^*$. Manche Autoren definieren Turingmaschinen als Tupel $(\Sigma, \Sigma_{in}, Q, I, q_0, F)$

1.2 Zeitkomplexität, Speedup und Hierarchiesatz

Definition 8 (Zeitkomplexität)
Sei T eine Turingmaschine $x \in \Sigma^*$ $\text{TIME}_T(x)$ =Länge der Berechnung $T(x)$, undefiniert falls $T(x)$ nicht hält.

Bemerkung 6
$\text{TIME}_T : \Sigma^* \to \mathbb{N}$ ist partielle Funktion
$|x|$:= Länge von x

Satz 1 (Speeduptheorem)
Sei T ein TM, die L entscheidet und $d \in \mathbb{N}$ Dann existiert TM T' die auch L entscheidet und außerdem erfüllt: $\text{TIME}_{T'}(x) = \frac{1}{d}\,\text{TIME}_T(x) + |x| + 1$

Beweis (Idee) T' verwendet als Alphabet $\Sigma' := \Sigma^d$ Berechne alle Übergänge von einem Block zum nächsten im voraus und speichere sie in der (riesigen!) Zustandstafel I' von T' ab. ∎

Ziel: Definition von $\text{TIME}(f)$ wobei f eine Funktion $\mathbb{N} \to \mathbb{N}$, z.B. $f(n) = n^e$

Definition 9 (Zeitkonstruierbar)
Eine Funktion $f : \mathbb{N} \to \mathbb{N}$ heißt zeitkonstruierbar, wenn eine TM T existiert, die bei Eingabe x hält nach $\leq c \cdot f(|x|)$ Schritten für ein festes $c \in \mathbb{N}$ und auf einem Band die Ausgabe $1^{f(|x|)}$ liefert (□□□ $\underbrace{11\ldots11}_{f(|x|)\text{ Stück}}$ □□)

Satz 2
$f(n) = n, f(n) = n \log n$ sind zeitkonstruierbar. Falls f,g zeitkonstruierbar, $k \in \mathbb{N}$ dann auch: $k \cdot f, f + k, f + g, f \cdot g, f^g, f!, f \circ g$. Zu jeder berechenbaren Funktion f existiert eine zeitkonstruierbare Funktion r, sodass $f(n) \leq r(n)$.

Definition 10
Sei f zeitkonstruierbar. $\text{TIME}(f)$ ist definiert als die Menge aller Sprachen L für die eine TM T existiert mit $\text{TIME}_T(n) \leq x \cdot f(|x|)$ für ein $c \in \mathbb{N}$

Man schreibt TIME($f(n)$) statt TIME(f) und auch TIME(n^2) statt TIME(sq) wobei $sq(n) = n^2$

Beispiel 2
$\{0^n1^n | n \geq 0\} \in$ TIME(n) 'Kürzeste Wege' $\in$ TIME(n^2) 'Sudoku lösen' $\in$ TIME(2^n)
$\{0^n1^n | n \geq 0\} \in$ TIME(2^n) ADDITION $\in$ TIME(n) MULTIPLIKATION $\in$ TIME(n^2)

Definition 11
$P := \bigcup_{k\geq 1}$ TIME(n^k)
EXP $:= \bigcup_{k\geq 1}$ TIME(2^{n^k})
$E := \bigcup_{k\geq 1}$ TIME(2^{kn})
ACHTUNG: $2^{5n} \notin O(2^n)$ d.h. für keine Konstante c gilt: $2^{5n} < c2^n$

Satz 3 (Zeithierarchiesatz)
Sei f zeitkonstruierbare Fkt. Dann gilt: TIME($f(n)$) $\subsetneq$ TIME($n^2 f(n)^2$)

BEWEIS Betrachte folgenden Algorithmus:

Eingabe: TM T /* codiert als 01 Folge /*

1 Berechne $N := f(|T|) + |T| + 1$

2 Simuliere T(M) für N Schritte.

3 Ausgabe: Falls diese Berechnung akzeptiert: REJECT sonst: ACCEPT

Setze

$$L_f = \{T \mid T(T) \text{ akzeptiert } \textit{nicht} \text{ nach } f(|T|) + |T| + 1 \text{ Schritten}\}$$

Klar: Der angegebene Algorithmus ist ein Entscheidungsverfahren für L_f.

Sei T_f eine Turingmaschine, die diesen Algorithmus implementiert und insbesondere L_f entscheidet. Wir zeigen zunächst, dass $L_f \notin$ TIME($f(n)$).

Falls doch, so sei T_0 eine Turingmaschine die L_f entscheidet und außerdem $\text{TIME}_{T_0}(x) \leq cf(|x|)$. Nach Speedup Theorem können wir oBdA annehmen dass $\text{TIME}_{T_0}(x) \leq f(|x|) + |x| + 1$. Es gibt zwei Fälle: entweder $T_0 \in L_f$ oder $T_0 \notin L_f$. Falls $T_0 \in L_f$ dann müsste $T_0(T_0)$ nach $\leq f(|T_0|) + |T| + 1$ Schritten verwerfen (also insbesondere überhaupt verwerfen). Dann aber wäre $T_0 \notin L_f$, weil T_0 die Sprache L_f entscheidet Falls $T_0 \notin L_f$ dann müsste T_0 die Eingabe verwerfen, und aufgrund der Annahme an TIME_{T_0} müsste dies nach spätestens $f(|T_0|) + |T_0| + 1$ Schritten geschehen, also wäre $T_0 \in L_f$.

Komplexitätsabschätzung des Algoritmus von oben: 1: $cf(|T|)$ Schritte (weil f zeitkonstruierbar) 2: Man muss die k Bänder der eingegeben Maschine T auf einem Band speichern (durch Hintereinanderschreiben), da man nicht inputabhängig neue Bänder anfordern kann. Außerdem muss das Alphabet der Maschine T mit festem Alphabet codiert werden, da man nicht inputabhängig neue Arbeitssymbole anfordern kann. Ein Arbeitsband von T enthält hoechstens $f(|T|) + |T| + 1$ Symbole. Da f zeitkonstruierbar ist, gilt $f(n) \geq n$, also $f(|T|) + |T| + 1 = O(f(|T|))$. Die Kodierung

einer globalen Konfiguration erfordert somit $O(|T|^2 \cdot f(|T|))$ Symbole, wenn man beachtet, dass sowohl die Zahl der Bänder von T, als auch die Alphabetgröße von T durch $|T|$ beschränkt sind.

Um einen Schritt zu simulieren, muss man auf dieser langen Codierung hin- und her navigieren, sodass sich ein Aufwand von $O(|T|^2 \cdot f(|T|))$ ergibt. Da nun $f(|T|) + |T| + 1$ Schritte zu simulieren sind, kann der Gesamtaufwand durch $O(|T|^2 \cdot f(|T|)^2)$ beschränkt werden. ■

Korollar 4
$P \subsetneq \text{EXP}$

BEWEIS P $\subseteq$ TIME(2^n) weil 2^n schneller wächst als jedes Polynom.
TIME$(2^n) \subsetneq$ TIME$(n^2 \cdot (2^n)^2)$ wegen Zeithierarchiesatz.
TIME$(n^2 \cdot (2^n)^2) \subseteq$ EXP ■

Satz 5 (GAP-Theorem)
Es existiert eine berechenbare Funktion $f : \mathbb{N} \to \mathbb{N}$ T', sodass TIME$(f(n)) =$ TIME$(2^{f(n)})$. *Allerdings ist f nicht zeitkonstruierbar.*

1.3 Nichtdeterminismus, wichtige Komplexitätsklassen und deren Beziehungen

Definition 12 (Nichtdeterministische Turingmaschine)
Wie deterministische Turingmaschine aber I beliebig ($I \subseteq Q \times \Sigma \times \Sigma \times Q \times \{S, L, R\}$)
Zu gegebenem q, a kann es mehrere Quintupel in I der Form $(q, a, -, -, -)$ geben.
Zu einer Eingabe $x \in \Sigma^*$ definiert man den Berechnungsbaum wie folgt:

- Der Binärbaum ist endlich verzweigt aber möglicherweise unendlich.
- Die Wurzel ist mit der Startkonfiguration von T auf x beschriftet.
- Ist ein Knoten mit Konfiguration S beschriftet und sind $S_1, ..., S_n$ die Nachfolgekonfigurationen von S, so hat der Knoten n Kinder die mit diesen beschriftet werden.

Definition 13
Eine nichtdeterministische Turingmaschine T akzeptiert ein Wort $x \in \Sigma^*$ wenn es im Berechnungsbaum von T auf x einen Ast gibt, der mit einer akzeptierenden Endkonfiguration endet und außerdem alle Äste endlich sind. Ein Wort x wird verworfen, wenn es nicht akzeptiert wird, d.h. es gibt einen unendlichen Ast oder alle Äste enden in einer verwerfenden Konfiguration.

Bemerkung 7
Alle Nichtdeterministischen Turingmaschinen die wir betrachten, haben endl. Berechnungsbäume.

Definition 14 (NTIME)
Sei T Nichtdeterministische Turingmaschine $x \in \Sigma^*$

$$\mathrm{NTIME}_T(x) = \begin{cases} \text{Länge des kürzesten akzeptierenden Astes, falls existent} \\ \text{Länge des kürzesten Astes überhaupt, sonst} \end{cases}$$

Bemerkung 8
Meist sind Nichtdeterministische Turingmaschinen so gebaut, dass alle Äste gleich lang sind.

Definition 15 (NTIME($f(n)$))
Sei f zeitkonstruierbar. $\mathrm{NTIME}(f(n)) = \{L \mid$ es existiert Nichtdeterministische Turingmaschine T, $c > 0 : \forall x.\ x \in L \Leftrightarrow T$ akzeptiert x und $\mathrm{NTIME}_T(x) \leq c \cdot f(|x|)\}$

Definition 16 (P, NP, E, EXP, NEXP, NE)
$\mathrm{P} = \bigcup_{k\geq 0} \mathrm{TIME}(n^k)$ /* polynomielle Laufzeit */
$\mathrm{NP} = \bigcup_{k\geq 0} \mathrm{NTIME}(n^k)$ /* nichtdeterministisch, polynomielle Laufzeit */
$\mathrm{E} = \bigcup_{k\geq 0} \mathrm{TIME}(2^{kn})$
$\mathrm{NE} = \bigcup_{k\geq 0} \mathrm{NTIME}(2^{kn})$
$\mathrm{EXP} = \bigcup_{k\geq 0} \mathrm{TIME}(2^{n^k})$
$\mathrm{NEXP} = \bigcup_{k\geq 0} \mathrm{NTIME}(2^{n^k})$

Bemerkung 9
P ='praktisch berechenbar'? Lässt sich ein Graph G knotenfrei im $\mathbb{R}^3$ zeichnen? Dieses Problem ist in P aber kein Algorithmus konnte bisher konkret angegeben werden.

Satz 6 (Bekannte Beziehungen)

$$\begin{array}{ccc} \mathrm{P} & \subseteq^{\dagger} & \mathrm{NP} \\ \wr\cap_{*} & & \wr\cap_{*} \\ \mathrm{E} & \subseteq^{\dagger} & \mathrm{NE} \\ \wr\cap_{*} & & \wr\cap_{*} \\ \mathrm{EXP} & \subseteq^{\dagger} & \mathrm{NEXP} \end{array}$$

**folgt aus Zeithierarchiesätzen*
† *Die Echtheit dieser Inklusionsbeziehungen wird vermutet, konnte aber nicht gezeigt werden.*

Satz 7
$\mathrm{NP} \subseteq \mathrm{EXP}$

Beweis Durch systematische Exploration aller Pfade im Berechnungsbaum. ∎

Satz 8
$\mathrm{P} = \mathrm{NP} \Rightarrow \mathrm{E} = \mathrm{NE}$ *(also* $\mathrm{E} \neq \mathrm{NE} \Rightarrow \mathrm{P} \neq \mathrm{NP}$*)*

BEWEIS (DURCH PADDING) Sei P = NP und $L \in$ NE
Definiere L^* über $\Sigma \uplus \{\#\}$
$L^* = \{w \underbrace{\#...\#}_{2^{|w|}-|w|} \mid w \in L\}$
Wir behaupten: L^* ist in NP:
Eingabe: w'
Prüfe ob $w' = w \underbrace{\#...\#}_{2^{|w|}-|w|}$
Prüfe ob $w \in L$ /* w aus Zerlegung von w' */ mit gegebenem nichtdeterministischem Algorithmus.
Zeitbedarf dieses Algorithmus:
$O(|w'|)$ für Zerlegung
NB: $|w| = O(\log |w'|)$
Zeitbedarf für Test ob $w \in L$
$O(2^{k\cdot|w|}) = O(2^{k\cdot O(\log(w'))}) = O(|w'|^{k\cdot O(1)})$ /* polynomiell */
also Gesamtlaufzeit des (nichtdeterministischen!) Algorithmus: $O(|w'|^{O(1)})$ also in NP.

Nach Annahme gibt es ein polynomielles deterministisches Verfahren für L^*. Wir konstruieren daraus ein Verfahren "in E" wie folgt:

EINGABE $w \in \Sigma^*$
Konstruiere $w' = w\#...\#$
Prüfe mit deterministisch polynomiellem Verfahren ob $w' \in L^*$
Falls ja: Akzeptiere
sonst: Verwirf

Laufzeit dieses Verfahrens:

- Konstruktion von w': $O(2^{O(|w|)})$
- Test ob $w' \in L^*$:
  $O(|w'|^k)$ k fest
  $= O((2^{|w|})^k) = O((2^{k\cdot|w|}))$

$\rightsquigarrow$ Algorithmus läuft in Zeit $O(2^{O(|w|)})$ also $L \in E$ wzbw. ∎

Korollar 9
$E \neq NE \Rightarrow P \neq NP$

Slogan *Gleichheit von Komplexitätsklassen vererbt sich nach oben; Ungleichheit vererbt sich nach unten.*

2 Die Klassen P und NP

2.1 NP-Vollständigkeit

Satz 10 (NP als Verifikation in polynomieller Zeit)
Wenn $L \in$ NP *dann gibt es* $R_L \in P$ *und* $k \in \mathbb{N}$ *so dass* $x \in L \Leftrightarrow$ *es existiert* w *mit* $(x, w) \in R_L$ *und* $|w| \leq |x|^k$

BEWEIS Deute w als 'Beweis' für $x \in L$. Aussage des Satzes ist, dass für Sprachen in NP immer ein Beweisbegriff existiert, sodass Korrektheit eines Beweises in polynomieller Zeit geprüft werden kann.
Sei $L \in$ NP und $\text{NTIME}_T(x) \leq |x|^k$ wobei T eine Nichtdeterministische Turingmaschine für L ist.
ObdA habe der Berechnungsbaum von T auf x Verzweigungsgrad 2.
$R_L = \{(x, w) \mid w \in \{0,1\}^*$, der durch w ausgewiesene Zweig im Berechnungsbaum von T auf x akzeptiert $\}$

Klar ist $R_L \in$ P (nichtdet. Entscheidung werden ja gemäß w vorgenommen)
Falls $x \in L$ so wähle einen akzeptierenden Pfad der Länge $\leq |x|^k$. Setze für w das entsprechende Wort aus $0, 1^*$. Es gilt $(x, w) \in R_L$ und $|w| \leq |x|^k$.

Falls $(x, w) \in R_L$ und $|w| \leq |x|^k$ so gilt $x \in L$ nach Definition von R_L ∎

Definition 17 (NP-schwer)
Eine Sprache $L \subseteq \Sigma^*$ ist NP-schwer, falls für jede Sprache $L' \in$ NP eine in polynomieller Zeit berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ existiert, sodass $x \in L' \Leftrightarrow f(x) \in L$.

Definition 18 (Transducer)
Ein Transducer (Übersetzer) ist eine deterministische TM mit zwei ausgezeichneten Bändern (ein Eingabe- und ein Ausgabeband). Ein Transducer berechnet eine Funktion $f : \Sigma^* \to \Sigma^*$, wenn die Maschine T für alle Eingaben $x \in \Sigma^*$ hält und dann auf dem Ausgabeband das Wort $f(x)$ steht. Man sagt "T berechnet f". DTIME wird analog auch für Transducer definiert.

Definition 19 (FP formal)
$f \in$ FP gdw. es einen Transducer T, der f berechnet, und es existiert ein Polynom p, so dass $\text{TIME}(T(x)) \leq p(|x|)$.

Beispiel 3
$f(x) = x^2$ (x eine Zahl in Binärnotation)
$f(x) =$ Tabelle der kürzesten Wege von einem Startknoten s in einem gerichteten Graphen G, wobei G und s durch x codiert sind.
$f(x) = 2^x$ ist nicht in FP

N.B. Falls $f \in$ FP dann gilt $|f(x)| \leq p(|x|)$ für ein festes Polynom p.

Definition 20 (SAT Problem)
Gegeben: aussagenlogische Formel q
Gefragt: Ist q erfüllbar (gibt es Belegung der Variablen, die q wahrmacht)

Definition 21 (3-COL)
Die Sprache 3-COL besteht aus allen ungerichteten Graphen $G = (V, E)$ die mti drei Farben gefärbt werden können, d.h. es existiert $c : V \to \{r, g, b\}$ sodass:
$\{v, v'\} \in E \Rightarrow c(v) \neq c(v')$

Satz 11
Es gibt eine Funktion $f \in FP$ sodass $x \in$ 3-COL $\Leftrightarrow f(x) \in SAT$

BEWEIS Sei $x \in (V, E)$ Graph
$f(x)$ ist wie folgt definiert:
Für jeden Knoten $v \in V$, für jede Farbe $c \in \{r, g, b\}$ eine Variable $x_{v,c}$ Das macht insgesamt $3 \cdot |v|$ Variablen
Für jeden Knoten v die Formel $x_{v,r} v x_{v,b} v x_{v,g}$ /* |v| Formeln
Für jeden Knoten v die Formel $\neg(x_{v,r} \wedge x_{v,b}) \wedge \neg(x_{v,r} x_{v,g}) \wedge \neg(x_{v,b} \wedge x_{v,g})$ /* |V| Formeln */
Für $\{v, v'\} \in E : \neg(x_{v,r} \wedge x_{v',r}) \wedge \neg(x_{v,g} \wedge x_{v',g}) \wedge \neg(x_{v,b} \wedge x_{v',b})$
f(x) besteht aus konjunktion all dieser Formeln ($2 \cdot |V| + |E|$ Stück)
$f(x)$ erfülbar $\Leftrightarrow x \in$ 3-Col
Man schreibt 3-Col $\leq_P$ SAT ■

Satz 12 (Satz von Cook-Levin)
SAT ist NP-vollständig D.h. für jede Sprache L $\in$ NP *gibt es eine Übersetzungsfunktion $f \in FP$ welche Instanzen von L auf aussagenlogische Formeln abbildet, sodass $x \in L \Leftrightarrow f(x)$ erfüllbar*

BEWEIS (Skizze)
Sei $L \in$ NP und T eine NTM mit L = L(T) und NTIME($T(x)$) $\leq p(|x|)$ für ein Polynom p
Wir müssen eine Funktion f konstruieren, sodass $x \in L \Leftrightarrow f(x) erfllbar$. Erinnerung: $x \in L \Leftrightarrow$in Berechnungsbaum von T auf x ein akzeptierender Pfad existier, bzw, wenn ausgehend von der Startkonfiguration S_0 von T auf x eine Folge von Konfigurationen $S_0 \to S_1 \ldots \to S_n$ existiert, sodass S_n akzeptierend ist und $' \to'$ Übergangsrelation von T ist
ObdA : $n = p(|x|)$ und T hat nur ein Band
f(x) ist jetzt wie folgt definiert:
Variablen:
Für alle $t \leq p(|x|), q \in Q$ eine Variable $zust_{q,t}$ /* T ist zur Zeit t im Zustand q, O(p(|x|)) Stück */
Für alle $i \leq p(|x|), t \leq p(|x|), a \in \Sigma$ eine Variable $sym_{a,i,t}$ /* zur Zeit t steht an Position i das Symbol a, $O(p(|x|)^2)$ Stück */
Für alle $z, t \leq p(|x|)$ eine Variable $kopf_{i,t}$ /* Zur Zeit t ist der Kopf an Position i */
Formeln welche ausdrücken, dass die Belegung dieser Variablen einer legalen akzeptierenden Berechnung entspricht:

Z.B. $\bigwedge_{i \leq p(|x|)} sym_{x_i,i,0}$ Wobei $x = x_0 \dots x_{m-1}$
$\bigwedge_{i \geq |x|} sym_{\square,i,0}$
$\bigwedge_{i \geq p(|x|), a \neq a', t \leq p(|x|)} \neg(sym_{a,i,t} \bigwedge sym_{a',i,t})$
Für jeden Zustand q und Symbol a Zeitpunkt t, Position i die Formel
$zust_{q,t} \wedge kopf_{i,t} \wedge symbol_{a,i,t} \Rightarrow \vee_{(q,a,q',a',d) \in I} zust_{q',t+1} \wedge kopf_{i+d,t+1} \wedge sym_{a',i,t+1}$
...
Idee: Die Variablen von $f(x)$ beschreiben Folgen von globalen Konfigurationen von T der Länge und Größe $p(|x|)$
Die Formeln von $f(x)$ beschreiben, dass solch eine Folge akzeptierend ist. Die Konjunktion dieser Formeln ist dann erfüllbar, genau dann, wenn eine akzeptierende Berechnung existiert. ■

Weite NP-vollständige Probleme:
3-COL, TSP, Hamiltonsche Kreise, ...

Definition 22 (NP-Schwer)
Ein Problem/Sprache L ist NP-schwer wenn $L' \leq_P L$ für alle $L' \in$ NP

Definition 23 (NP-vollständig)
Eine Sprache L ist NP-vollständig, wenn $L \in$ NP und L NP schwer

Bemerkung 10
Könnte man ein NP-vollständiges Problem in polynomieller Zeit lösen, so wäre NP = P
Levin hat einen Algorithmus angegeben, der zu gegebener erfüllbarer Formel in polynomieller Zeit eine erfüllende Belegung findet, vorausgesetzt, dass $P = NP$ (Levinsearch).

2.2 Satz von Ladner

Satz 13 (Satz von Ladner)
Es gibt eine Sprache $L \in$ NP *die nicht NP-vollständig ist und trotzdem nicht in P liegt.*
Das natürlich unter der Voraussetzung P $\neq$ NP.

Beweis Beginne mit NP-vollständiger Sprache L_0 z.B $L_0 =$ Sat wähle $w_0 \notin L_0$
Ansatz:
$L = \{x \in L_0 \mid |x| \in G\}$ für noch zu bestimmendes G.
$f(x) = \begin{cases} x & \text{falls } |x| \in g \\ w_0 & \text{falls } |\text{x}| \notin G \end{cases}$
Es gilt $x \in L \leftrightarrow f(x) \in L_0$
Ziel:

1. Wähle G so, dass $f \in$ FP , dazu muss $|x| \in G$? in polynomieller Zeit entscheidbar sein. Dann ist zumindestens $L \in$ NP, da $L \leq_P L_0$ vermöge f und L_0 NP-vollständig

2. Wähle G so dass $L \neq$ i-te Sprache in ¶ und außerdem $L \neq$ i-te Sprache die NP-vollständig ist. Dies für alle $i \geq 0$

Sei $T_1, T_2, T_3 \ldots$ eine Aufzählung von Turingmaschinen, so, dass

$$\mathrm{P} = \{L(T_i) \mid i \geq 0\}$$

Außerdem soll $i \mapsto T_i$ berechenbar sein.

Idee:
Gegeben:
Decodiere i als Tupel (T, p) wobei T eine TM ist und p Polynom
bei falschem Format setze $T_i = 0$ /* Default Maschine */ Setze $T_i :=$ Die Maschine, die auf Eingabe x die Berechnung von $T(x)$ für $p(|x|)$ Schritte simuliert und akzeptiert gdw. die Simulation akzeptiert hat.'

Sei $T'_1, T'_2, \ldots$ eine Aufzählung von Turingmaschinen, so dass NPC $= \{L | \exists i : T'_i$ entscheidet $L\}$
Idee: gegeben i:
Decodiere i als Tupel (T,f)
T nichtdet. TM polynomieller Laufzeit
f ist FP-Transducer
Konstruiere aus T, f folgende deterministische TM T'_i
Bei Eingabe x prüfe zunächst, ob $y \in \mathrm{SAT} \iff f(y) \in L(T)$ für alle y mit $|y| \leq |x|$. (durch Brute-Force Simulation aller Berechnungspfade). Falls Ja: Akzeptiere, gdw. $x \in L(T)$
Falls Nein: Akzeptiere, gdw. $x \in$ SAT

Ist $i = (T, f)$ und $L(T)$ NP-vollständig vermöge der Reduktion f, so gilt $L(T'_i) = L(T)$. Anderenfalls unterscheidet sich $L(T'_i)$ nur an endlich vielen Punkten von SAT und ist somit immerhin NP-vollständig.

Sei nunmehr $L_i = L(T_i)$ und $L'_i = L(T'_i)$. Es ist P $= \{L_i \mid i \geq 0\}$ und NPC $= \{L'_i \mid i \geq 0\}$. Hier ist NPC die Menge der NP-vollständigen Sprachen.

Ziel: Konstruiere $G \subseteq \mathbb{N}$ so, dass $\forall i L^* \neq L_i$ und $L^* \neq L'_i$

NB A,B Mengen: $A \neq B$ falls $A \Delta B \neq \emptyset$ wobei $A \Delta B = A \setminus B \cup B \setminus A$

Für jedes $n, i \in \mathbb{N}$ gibt es ein Wort x mit $|x| \geq n$ so dass $x \in L \Delta L_i$

NB x bezeugt, dass $L \neq L_i$

Nach Voraussetzung P $\neq$ NP muss $L \Delta L_i \neq \emptyset$ sein

Mehr noch: für jedes n gibt es $y \in L \Delta L_i$ mit $|y| \geq n$, denn sonst würde sich L nur an endlich vielen Punkten von L_i unterscheiden und wäre damit selbst in P.

Für jedes $i, n \in \mathbb{N}$ gibt es außerdem y mit $|y| \geq n$ und $y \in L_i'$
Sonst wäre nämlich L_i' endlich und deshalb in P. Wegen der Voraussetzung P $\neq$ NP kann eine NP-vollständige Sprache wie L_i' nicht in P sein
Sei nun r zeitkonstruierbare funktion $r : \mathbb{N} \to \mathbb{N}$, sodass für alle $n \in \mathbb{N}$ sodass für jedes $i \leq n$ ein Wörter $x \in L \Delta L_i$ und $y \in L_i'$ mit mit $n \leq |x|, |y| \leq r(n)$ existieren.
Idee:
Gegeben n:
Simuliere alle T_i, T_i' mit $i \leq n$ auf allen Eingaben der Längen , $n, n+1, n+2, \ldots$ solange bis passende Wörter x und y gefunden sind. Nach vorhergehender Diskussion gibt es solche.
Ausgabe: Rechenzeit dieses ganzen Prozesses.

Dahinter steckt die allgemeine Beobachtung:
Zu jeder berechenbaren Funktion f gibt es eine zeitkonstruierbare Funktion g mit $g(n) \geq f(n)$
Setze nun:
$r_0 = 0$
$r_1 = r(0) + 1$
$r_2 = r(r_1) + 1$
$\vdots$
$r_{n+} = r(r_n) + 1$

$$G = \{n | r_i \geq n < r_i + 1 \text{ wobei i gerade } \}$$

Erinnerung $L^* = \{x \in L | |x| \in G\}$

Behauptung: $L^* \notin$ NPC
Wäre $L^* = L_i'$ für ein i, dann wähle n ungerade, sodass $r_n \geq i$
Im Bereich $r_n \ldots r(r_n)$ findet sich ein $y \in L_i'$ Nach Definition von G ist aber $y \notin L^*$ weil $|y| \notin G$
Behauptung: $L^* \notin$ P:
Wäre $L^* = L_i$ für ein i dann wähle n gerade, sodass $r_n \geq i$. Im Bereich $r_n, \ldots, r(r_n) = r_{n+1} - 1$ findet sich ein $x \in L_i \Delta L$. Auf diesem Bereich stimmen L und L^* überein.
Also ist $x \in L_i \Delta L^*$
Es folgt $L_i \neq L^*$

Es bleibt zu zeigen, dass: $\{x | x \in G\} \in P$
Gegeben: x
Berechne sukzessive $r_0, r_1, r_2, \ldots$
bis n gefunden ist mit $r_n \geq |x| < r_{n+1}$
akzeptiere $\Leftrightarrow$n gerade
Die Berechnung eines r_i verbraucht Zeit $\mathcal{O}(r_i)$
also nachdem alle $r_i \leq |x|$ sind: $\mathcal{O}(|x|)$

Außerdem kann n (sehr grob) durch |x| abgeschätzt werden
⇒Gesamtlaufzeit $\mathcal{O}(|x|^2)$ ■

Bis heute wurden keine konkreten Sprachen echt zwischen P und NPC gefunden.
Immerhin:
Definiert man $co - \text{NP} := \{L | \overline{L} \in \text{NP}\}$
Dann ist $\text{NP} = co - \text{NP}$ ein offenes Problem und auch ob:
$\text{P} = \text{NP} \cap co - \text{NP}$
Falls $NP \neq co - \text{NP}$, so ist kein Problem in $co - \text{NP}$ auch NP-vollständig.

2.3 Orakelmaschinen

Definition 24 (Orakelmaschine)
Eine (nicht)deterministische Orakelturingmaschine ist eine (nicht)deterministische TM T mit drei ausgezeichneten Zuständen q_Q, q_Y, q_N
Falls $C \subseteq \Sigma^*$, so definiert man die Berechnung von T auf Eingabe $x \in \Sigma^*$ ($T^C(x)$)wie folgt:
$T^C(x)$ erfolgt wie 'normal' mit der folgenden Ausnahme:
gelangt die Berechnung in den Zustand q_Q (query) so wird die Berechnung im Zustand q_Y fortgesetzt falls 'Bandinhalt' (des ersten Bandes)$\in C$
und in q_N fortgesetzt, falls 'Bandinhalt' $\notin C$

$L(T^C) = \{x | T^C(x)$ akzeptiert $\}$
bzw. $L(T^C) = \{x \mid$ es gibt eine akzeptierende Berechnungsfolge im Baum $T^C(x)\}$

Definition 25
Sei $C \subseteq \Sigma^*$ Wir definieren:

$$\text{P}^C = \{L \mid \begin{array}{l}\text{es existiert det. Orakel TM } T,\ L = L^C(T) \text{ und Polynom } p: \\ \text{die Länge der Berechnung } T^C(x) \text{ ist durch } p(|x|) \text{ beschränkt.}\}\end{array}$$

Bemerkung 11
Falls $L \in$ P, z.B. L = Addition, Matching, ...
so ist $\text{P}^L = \text{P}$
Begründung:
Simuliere Berechnung $T^L(x)$ durch eigenständige Beantwortung der Orakelanfragen.

Bemerkung 12
Lautzeit von T beschränkt durch p und $L \in \text{TIME}(q)$ für Polynom q
⇒Laufzeit der Simulation beschränkt durch $p(n) \cdot q(\underbrace{p(n)}_{\text{Schranke an die Länge der Anfrage}})$

Bemerkung 13
$\text{NP} \subseteq \text{P}^{SAT}$ ist klar. Die Frage, ob $\text{NP} \supseteq \text{P}^{SAT}$ ist ein offenes Problem.

Setzt man $TAUT = \{\phi \mid \phi$ allgemeingültig$\}$, so folgt TAUT$\in \text{P}^{SAT}$ mit folgender Maschine:

Eingabe: q /* q aussagenlogische Formel */
Frage ob $\neg q \in SAT$ /* Schreibe $\neg q$ aufs erste Band gege in q_Q */
Falls ja: REJECT
Falls nein: ACCEPT

Aber $TAUT \in NP$ ist ein bekanntes offenes Problem.

Definition 26
Sei $C \subseteq \Sigma^*$ Wir definieren:

$$\mathrm{NP}^C = \{L \mid \begin{array}{l}\text{es existiert nichtdet. Orakel TM } T,\ L = L^C(T) \text{ und Polynom } p:\\ \text{die Laufzeit der Berechnung } T^C(x) \text{ ist durch } p(|x|) \text{ beschränkt.}\}\end{array}$$

Bemerkung 14
$P^C \subseteq NP^C$

Bemerkung 15
Die Sprache $\{(\varphi, U) \mid \varphi$ auss.log. Formel, U Teilmenge der Variablen, es existiert Belegung η der Variablen in U, sodass $\varphi[\eta]$ allgemeingültig ist$\}$ ist in NP^{SAT}. (Rate Belegung der Variablen in U, entscheide mit SAT-Orakel, ob entstehende Formel allgemeingültig ist.)

2.4 Relativierungen von P und NP

Es ist ein offenes Problem, ob diese Sprache in P^{SAT} ist.

Satz 14
Es gibt eine Sprache $C \subseteq \Sigma^*$, *sodass* $\mathrm{P}^C = \mathrm{NP}^C$

Bemerkung 16
Es ist unbekannt ob $\mathrm{P}^{SAT} = \mathrm{NP}^{SAT}$

Beweis $C = \{(T, x, 0^k) \mid T$ ist det. TM und $T(x)$ akzeptiert und verbraucht höchstens k Bandpositionen (zusätzlich zur Eingabe)$\}$

NB: C ist entscheidbar. Es gibt nämlich nur $2^{poly(k,|x|)}$ viele mögliche globale Konfigurationen. Hat die Berechnung nach dieser Zeit nicht gehalten, so hält sie überhaupt nicht mehr und kann abgebrochen werden.

NB: Es ist offen ob $C \in \mathrm{P}$
$P^C \subseteq NP^C$ ist klar.

Es bleibt zu zeigen $NP^C \subseteq P^C$
Sei T nichtdet. Orakel TM zeitbeschränkt durch Polynom p. Wir konstruieren eine deterministische polynomialzeitbeschränkte Orakel TM T' mit $L(T^C) = L(T'^C)$ und T' tätigt nur eine einzige Orakelanfrage ganz am Ende.

T' wie folgt:
EINGABE: x
Baue eine det. TM T_0, die $T^C(x)$ ohne Benutzung des Orakels simuliert. Dies durch systematische Exploration aller Berechnungspfade und Beantwortung aller Orakelanfragen durch obenstehende Entscheidunsprozedur
Man kann diese Maschine so konstruieren, dass sie nur $p'(|x|)$ Bandzellen anfordert für ein Polynom p'
Frage nun, ob $(T_0, x, 0^{p'(|x|)}) \in C$
Falls ja, akzeptiere
Falls nein verwerfe.
NB: Laufzeit dieser Maschine ist polynomiell, denn sie setzt sich zusammen aus folgendem: Erzeugung des Codes für T_0, Erzeugung von $0^{p'(|x|)}$ und stellen der Anfrage an C. ■

Bemerkung 17
Eine Beweismethode wie z.B. Diagonalisierung, die in Gegenwart von Orakeln funktioniert kann nicht P von NP trennen.

Satz 15
Es existiert eine Sprache $C \subseteq \Sigma^*$ sodass $P^C \neq NP^C$

Bemerkung 18 (Vorbemerkung zum Beweis)
Ist T eine OTM, $C \subseteq \Sigma^*$ und wird die Anfrage y in der Berechnung $T^C(x)$ nicht gestellt, dann gilt: $T^C(x)$ akzeptiert $\Leftrightarrow T^{C\setminus\{y\}}(x)$ akzeptiert $\Leftrightarrow T^{C\cup\{y\}}(x)$ akzeptiert

BEWEIS Sei $T_0, T_1, \ldots$ eine effektive Aufzählung von polynomialzeitbeschränkten OTM sodass gilt:

1. $P^C = \{L^C(T_i) \mid i \geq 0\}$ für alle C
2. Laufzeit von $T_i^C(x) \leq |x|^i + i$

Übung: Solche Aufzählung existiert
Wähle $B \subseteq \Sigma^*$ in noch zu bestimmender, geeigneter Weise
C = $\{0^n \mid$ es existiert $x \in B$ mit |x| = n$\}$
Es gilt $C \in NP^B$ ganz egal was B am Ende ist:
Gegeben 0^n, rate $x \in \Sigma^*$ mit $|x| = n$. Frage dann das Orakel, ob $x \in B$
Wir werden es so einrichten, dass $C \neq L^B(T_i)$ für jedes i und somit $C \notin P^B$
$B_0 = \emptyset, n_0 = 0$
Seien B_i und n_i schon definiert. Wir legen B_{i+1}, n_{i+1} wie folgt fest:
/* Invariante: B_i enthält kein Wort der Länge $\geq n_i$ */
/* Invariante: $2^{n_i} > n_i^i + i - 1$ */
Rechne $T_i^{B_i}(0^{n_i})$ /* Dauert $n_i^i + i$ Schritte */
Falls akzeptiert wird: $B_{i+1} := B_i$, sodass $0^{n_i} \notin C$
Falls zurückgewiesen wird:
Wähle w mit $|w| = n_i$ und so dass w in obiger Berechnung
nicht gefragt wurde. So ein w muss es wegen der Wahl von n_i geben.
Setze $B_{i+1} := B \cup \{w\}$

Wähle n_{i+1} so dass

- $2^{n_{i+1}} > n_{i+1}^{i+1} + i + 1$
- $n_{i+1} > n_i^i + i - 1$

Setze nun: $B = \bigcup_{i \geq 0} B_i$. Mit dieser Wahl von B gilt:
$C \notin P^B$. Denn wäre $C = L(T_i^B)$ für ein $i \geq 0$, dann betrachten wir die Berechnung $T_i^B(0^{n_i})$. Aufgrund der Konstruktion kommt hierbei dasselbe Ergebnis heraus wie bei der Berechnung $T_i^{B_i}(0^{n_I})$. Es gilt aber $0^{n_i} \in L(T_i^{B_i}) \Leftrightarrow 0^{n_i} \notin C$
Also ist $C \neq L(T_i^B)$

Halbkonkrete Konstruktion der ersten 3 Schritte ($\Sigma = \{0, 1\}$):
Betrachte: $T_0^{B_0}(\epsilon)$ /* $B_0 = 0, n_0 = 0$ */
Falls akzeptiert: $B_1 = B_0 = \emptyset$ hier $\epsilon \notin C$ (wir nehmen an, dass dieser Fall eintritt)
Falls zurückgewiesen wird: $B_1 = \{\epsilon\}$ hier $\epsilon \in C$
n_1 sodass $2^{n_1} > n_1^1 + 1$ und $n_0^0 + 0$
$n_1 = 2$
Betrachte $T_1^{B_1}(00)$ Rechenzeit $2^1 + 1 = 3$ Schritte
Annahme. Die Berechnung weise zurück und das Wort 11 werde nicht angefragt.
$B_2 = \{11\}$
n_2 so dss $2^{n_2} > n_2^2 + 2$ und $n_2 > n_1^1 + 1 = 3$
z.b: $n_2 = 5$
Betrachte $T_2^{B_2}(00000)$
Rechenzeit: 27 Schritte
Annahme: Die Berechnung weise zurück und frage das Wort 11011 nicht an. Dann setzen wir $B_3 = \{11, 11011\}$
n_3 sodass $2^{n_3} > n_3^3 + 3$ und $n_3 > 27$
z.B.: $n_3 = 28$ ■

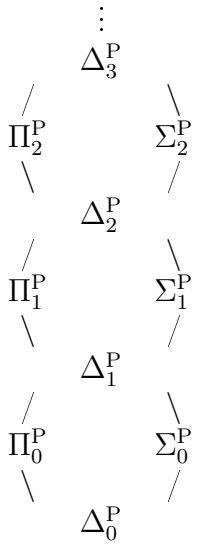
2.5 Die Polynomielle Hierarchie

$\mathrm{NP}^{\mathrm{SAT}}$ ist mehr als NP. Es gilt sogar TAUT $\in \mathrm{P}^{\mathrm{SAT}} \subseteq \mathrm{NP}^{\mathrm{SAT}}$ aber TAUT ist co-NP vollständig.
Die Komplexitätsklasse $\mathrm{NP}^{\mathrm{SAT}}$ wird mit Σ_2^{P} bezeichnet, die Klasse $\mathrm{P}^{\mathrm{SAT}}$ wird mit Δ_2^{P} bezeichnet.

Definition 27 (polynomielle Hierarchie)
$\Sigma_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} := \mathrm{P}$ /* Polynomialzeit */
$\Sigma_{i+1}^{\mathrm{P}} = \mathrm{NP}^{\Sigma_i^{\mathrm{P}}} = \bigcup_{C \in \Sigma_i^{\mathrm{P}}} \mathrm{NP}^C$
$\Delta_{i+1}^{\mathrm{P}} = P^{\Sigma_i^{\mathrm{P}}}$
$\Pi_{i+1}^{\mathrm{P}} = \text{co-NP}^{\Sigma_i^{\mathrm{P}}}$

Bemerkung 19
Es gilt:
$\mathrm{NP} = \Sigma_1^{\mathrm{P}}$

co-NP $= \Pi_1^{\mathrm{P}}$
$\mathrm{P}^{\mathrm{SAT}} = \mathrm{P}^{\mathrm{NP}} = \Delta_2^{\mathrm{P}}$
$\mathrm{NP}^{\mathrm{SAT}} = \Sigma_2^{\mathrm{P}}$
Offensichtlich gelten folgende Inklusionsbezeichnungen als Hasse Diagramm:



Bemerkung 20 (Notation)
Für $n \in \mathbb{N}$ schreiben wir
$\exists_n y.\varphi(y)$ für $\exists y \in \Sigma^*.|y| \leq n \wedge \varphi(y)$
$\forall_n y.\varphi(y)$ für $\forall y \in \Sigma^*.|y| \leq n \Rightarrow \varphi(y)$

Satz 16
$L \in \Sigma_i^{\mathrm{P}}$ genau dann wenn eine Sprache $A \in \mathrm{P}$ und ein Polynom p existiert, sodass:
$x \in L \Leftrightarrow \exists_{p(|x|)} y_1 \forall_{p(|x|)} y_2 \exists_{p(|x|)} y_3 \ldots \exists / \forall_{p(|x|)} y_i (x, y_1, y_2, \ldots, y_i) \in A$
Analog gilt $L \in \Pi_i^{\mathrm{P}}$ $\Leftrightarrow$es gibt Sprache $A \in \mathrm{P}$ und Polynom p mit
$x \in L \Leftrightarrow \forall_{p(|x|)} y_1 \exists_{p(|x|)} y_2 \ldots (x, y_1, \ldots, y_i) \in A$

Beweis durch Induktion über i. Wir führen nur die Fälle $i = 0, 1, 2$ durch.
$i = 0$ Klar
$i = 1$ Aussage des Satzes:
$L \in \Sigma_1^{\mathrm{P}}$, d.h. $L \in \mathrm{NP}$ $\Leftrightarrow$ eine Sprache $A \in \mathrm{P}$ und ein Polynom p existiert sodass:
$x \in L \Leftrightarrow \exists y_1 : |y_1| \leq p(|x|) \wedge (x, y_1) \in A$
Dies wurde bereits bewiesen.
$i = 2$:
Aussage des Satzes:
$L \in \Sigma_2^P$, d.h. $\mathrm{NP}^{\mathrm{SAT}}$ $\Leftrightarrow$eine Sprache $A \in \mathrm{P}$ und ein Polynom p existieren mit: $L = \{x \mid \exists_{p(|x|)} y_1 . \forall_{p(|x|)} y_2 . (x, y_1, y_2) \in A\}$
$\underline{\Leftarrow}$:
Sei $L = \{x \mid \exists_{p(|x|)} y_1 . \forall_{p(|x|)} y_2 . (x, y_1, y_2) \in A\}$ für ein $A \in P$ und Polynom p
Wir sollen zeigen $L \in \mathrm{NP}^{\mathrm{NP}}$
Es gilt $x \in L \Leftrightarrow$

$$\underbrace{\exists_{p(|x|)} y_1 . \neg \exists_{p(|x|)} y_2 . \underbrace{(x, y_1, y_2) \notin A}_{\in \mathrm{P}}}_{=:B(x,y_1) \in \mathrm{NP}}$$

Offensichtlich:
$B \in \mathrm{NP}$ und $L \in \mathrm{NP}^B \subseteq \mathrm{NP}^{\mathrm{NP}} = \Sigma_2^{\mathrm{P}}$
$\underline{\Rightarrow:}$
Sei eine Orakelmaschine T vorgelegt sodass $L = L(T^{\mathrm{SAT}})$ und sei T nichtdet und polynomiell zeitbeschränkt. Wir müssen eine Sprache $A \in P$ finden sodass
$L = \{x \mid \exists_{p(|x|)} y_1 . \forall_{p(|x|)} y_2 . (x, y_1, y_2) \in A \}$
Es gilt $x \in L \Leftrightarrow$ es existiert eine Folge von Konfigurationen y von T auf Eingabe x, sodass, falls $z = z_1 z_2 \dots z_k$ die in y getätigten und positiv beantworteten Anfragen an das SAT Orakel bezeichnet und falls $w = w_1 w_2 \dots w_k$ die in y_1 getätigten und negativ beantworteten Anfragen an das SAT Orakel bezeichnet, folgendes gilt:

- alle Rechenschritte in y_1 erfolgen gemäß T
- alle $z_1 \dots z_k \in$ SAT /* das ist von der Form $\exists y_1' \dots$ */
- alle $w_1 \dots w_l \notin$ SAT /* das ist von der Form $\forall y_2 \dots$ */

Die gewünschte Charakterisierung lässt sich daraus ablesen.
Beachte: zwei aufeinanderfolgende polynomiell beschränkte Existenzquantoren können durch einen einzigen ersetzt werden. ■

Abstrakte Notation

- φ, ψ bezeichnen Formeln (Boolesch)
- $|\varphi|, |\psi|$ bezeichnet Länge der Formel
- η, η' bezeichnet Belegung (codiert)
- $\eta \models \varphi$: Belegung η erfüllt Formel φ
  Klar: $\{(\eta, \varphi) \mid \eta \models \varphi\} \in P$
  $\mathrm{SAT} = \{\varphi \mid \exists \eta : \eta \models \varphi\} = \{\varphi \mid \exists \eta . |\eta| \leq |\varphi| \wedge \eta \models \varphi\}$
- C, C' bezeichne Schaltkreise
  $|C|, |C'|$ deren Länge bzgl fester sinnvoller Codierung
- $C(\varphi)$ bezeichne die Anwendung von C auf (codierung von) φ
  Klar: $\{(C, \varphi) \mid C(\varphi) = \mathrm{true}\} \in \mathrm{P}$

Betrachte:
Test-SAT $= \{(C, 1^n) \mid C$ hat n Eingabedrähte und für alle Formeln φ der Länge n gilt: $C(\varphi) = \mathrm{true} \Leftrightarrow \varphi \in \mathrm{SAT}\}$
Offensichtlich gilt Test-SAT $\in \Pi_2^{\mathrm{P}}$, denn
$(C, 1^n) \in$ Test-SAT $\Leftrightarrow$
$\forall \varphi : |\varphi| \leq n : (C(\varphi) = \mathrm{true} \vee \forall \eta . |\eta| \leq n \Rightarrow \eta \not\models \varphi) \wedge (C(\varphi) = \mathrm{false} \vee \exists \eta . |\eta| \leq n \Rightarrow \eta \models \varphi)$
$\Leftrightarrow \forall_n \varphi \forall_n \eta \exists_n \eta' (C(\varphi) = \mathrm{true} \vee \eta \not\models \varphi) \wedge (C(\varphi) = \mathrm{false} \vee \eta' \models \varphi)$

Lemma 1 (Selbstreduzierbarkeit von SAT)
TEST-SAT $\in \Pi_1^{\mathrm{P}}$

BEWEIS Betrachte folgenden offensichtlich polynomiellen) Algorithmus:
Eingabe: (C, φ, n)
$x_1, \ldots, x_n$ die Variablen von φ
IF $C(\varphi) =$ false RETURN 'Angeblich unerfüllbar'
ELSE /* $C(\varphi) =$ true*/ wähle $b_1 \in \{\text{true}, \text{false}\}$ sodass $C(\varphi[x_1 := b_1] =$ true
falls nicht existent RETURN 'Widerspruch!'
$\vdots$
wähle $b_k \in \{\text{true}, \text{false}\}$ sodass $C(\varphi[x_1 := b_1, \ldots, x_k := b_k]) =$ true
falls nicht existent RETURN 'Widerspruch!'
IF $[x_1 \mapsto b_1, \ldots, x_k \mapsto b_k] \models \varphi$ return 'erfüllbar'
ELSE RETURN 'Widerspruch!'

Dieser Algorithmus erfüllt folgende Spezifikationen:

- $A(C, \varphi, n) =$ 'Angeblich unerfüllbar' $\Leftrightarrow C(\varphi) =$ false
- $A(C, \varphi, n) =$ 'Widerspruch!' $\Rightarrow \exists_n \psi. \neg(C(\psi) = \text{true} \Leftrightarrow \psi \in \text{SAT}\}$
- $A(C, \varphi, n) =$ 'erfüllbar' $\Rightarrow \varphi \in$ SAT ■

Daraus folgt: $(C, 1^n) \in$ TEST-SAT $\Leftrightarrow \forall_n \varphi$
$(C(\varphi) = \text{false} \wedge \forall_n \eta. \eta \not\models \varphi) \vee (C(\varphi) = \text{true} \wedge A(C, \varphi, n) = '\text{erfüllbar}')$.
Das aber ist nach Satz 16 in Π_1^P.

Satz 17 (Karp-Lipton)
Falls ein Polynom p existiert derart dass $\forall n. \exists C. |C| \leq p(n).$ TEST-SAT$(C, 1^n)$ *(kurz :* SAT $\in$ P / poly*) dann folgt:*
$\Sigma_2^{\mathrm{P}} = \Pi_2^{\mathrm{P}}$

BEWEIS Wir zeigen zunächst: $\Pi_2^{\mathrm{P}} \subseteq \Sigma_2^{\mathrm{P}}$:
Sei also $L \in \Pi_2^{\mathrm{P}}$ und $A \in$ P und r Polynom sodass:
$x \in L \Leftrightarrow \forall_{r(|x|)} y \exists_{r(|x|)} z(x, y, z) \in A$
Da SAT NP-vollständig ist, gibt es eine Funktion $f \in$ FP sodass:
$x \in L \Leftrightarrow \forall_{r(|x|)} y f(x, y) \in$ SAT. Letzteres ist aber äquivalent zu $\exists_{p(q(|x|))} C.$ TEST-SAT$(C, 1^{p(q(|x|))}) \wedge$ $C(f(x, y)) =$ true
Hier ist q so gewählt, dass $|y| \leq r(|x|) = |f(x, y)| \leq g(|x|)$
Nachdem TEST-SAT $\in \Pi_1^{\mathrm{P}}$ ist folgt $L \in \Sigma_2^{\mathrm{P}}$

Für die umgekehrte Richtung $\Sigma_2^{\mathrm{P}} \subseteq \Pi_2^{\mathrm{P}}$ argumentieren wir so: Falls $L \in \Sigma_2^{\mathrm{P}}$ dann ist $\overline{L} \in \Pi_2^{\mathrm{P}}$ also nach dem vorher Gezeigten ist $\overline{L} \in \Sigma_2^{\mathrm{P}}$ und also $L \in \Pi_2^{\mathrm{P}}$ ■

Aus Satz 16 ergibt sich auch unmittelbar, dass wenn $\Sigma_2^P = \Pi_2^P$, dann auch $\Sigma_i^P = \Sigma_2^P$ für alle $i \geq 2$. Man sagt dann: die PH kollabiert auf die zweite Stufe oder kürzer: die PH kollabiert. Der Satz von Karp-Lipton besagt also, dass sofern die PH nicht kollabiert, es keine polynomiell großen Schaltkreise für SAT gibt.

3 Platzkomplexität

3.1 Grundlegende Definitionen

Definition 28
Bei platzbeschränkter Berechnung verwendet man Turingmaschinen mit mindestens zwei Bändern. Auf das erste Band wird die Eingabe geschrieben.
Dieses Band darf während der Berechnung nicht verwendet werden. Der Platzbedarf SPACE($T(x)$) der Berechnung einer solchen Maschine T auf Eingabe x ist definiert als die Anzahl der Felder, die auf den verbleibenden Bändern benutzt werden.

Definition 29 (Platzkonstruierbare Funktion)
$s : \mathbb{N} \to \mathbb{N}$ platzkonstruierbar, falls TM existiert, die bei Eingabe x auf Ausgabeband $1^{s(|x|)}$ ausgibt und dabei Platz $\mathcal{O}(s(|x|))$ verbraucht.

Beispiel 4
$n, n^2, 2^n, \log n, n \log n$

Definition 30
DSPACE$(s(n)) = \{L \mid \exists$ DTM T mit $L = L(T)$ und SPACE$(T(x)) = \mathcal{O}(s(|x|))\}$
NSPACE$(s(n)) = \{L \mid \exists$ NTM T mit $L = L(T)$ und SPACE$(T(x)) = \mathcal{O}(s(|x|))\}$

Bemerkung 21
Analog zur Zeitkomplexität gibt es auch für Platzkomplexität Hierarchiesätze, die in etwa besagen:
Mit mehr Platz lässt sich auch beweisbar mehr berechnen.

Satz 18
DSPACE$(s(n)) \subseteq$ DTIME$(2^{\mathcal{O}(s(n))})$
NSPACE$(s(n)) \subseteq$ DTIME$(2^{\mathcal{O}(s(n))})$

Beweis Eine $s(n)$-platzbeschränkte TM hat $2^{\mathcal{O}(s(n))}$ viele globale Konfigurationen. In Zeit DTIME$(2^{\mathcal{O}(s(n))})$ kann die TM somit simuliert werden. ∎

Bemerkung 22 (triviale Beobachtung)
DTIME$(t(n)) \subseteq$ DSPACE$(t(n))$

Satz 19
DTIME$(t(n)) \subseteq$ DSPACE$(\frac{t(n)}{\log(n)})$
$\frac{t(n)}{\log n}$ *platzkonstruierbar*
$t(n)$ *zeitkonstruierbar*

Beweis siehe z.B. Skriptum von Goldreich ∎

Satz 20
Falls L durch $t(n)$ zeitkonstruierbare Einbandturingmaschine ber. werden kann. so ist $L \in$ DSPACE$(\sqrt{t(n)})$ *falls $t(n)$, $\sqrt{t(n)}$ zeitkonstruierbar sind und $t(n) \geq n^2$.*

BEWEIS siehe z.B. Bovet-Crescenzi ■

Definition 31 (wichtige Platzklassen)
LOGSPACE (auch L) : DSPACE($\log n$)
NLOGSPACE (auch NL) : NSPACE($\log n$)
PSPACE : $\bigcup_{k\geq 1}$ DSPACE(n^k)

Bemerkung 23 (Typische Probleme in LOGSPACE)
1. Enthällt ein Graph eine bestimmte Konfiguration?
2. Ist ein ungerichteter Graph zykelfrei?
3. Addition
4. Multiplikation(schwieriger)
5. Ereichbarkeit im ungerichteten Graphen (schwierig, Satz von Reingold)

Bemerkung 24 (Typisches Probem in NLOGSPACE)
Ereichbarkeit in gerichteten Graphen
Raten eines Pfades unter Vergessen bisher besuchter Knoten. Mitführen eines Zählers von $1 \ldots n$ Abbruch nach maximal n Schritten (n = Zahl der Knoten)

Bemerkung 25 (Typische Probleme in PSPACE)
Verallgemeinerte Spiele $n \times n$ Schach, Dame etc.
Quantifizierte Boolesche Formeln

Bemerkung 26
$(2^{\mathcal{O}(\log(n)} = n^{\mathcal{O}(1)})$
LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ PSPACE
LOGSPACE $\subsetneq$ PSPACE
Eine der drei Inklusionen muss echt sein, da LOGSPACE wegen des Platzhierarchiesatzes echt in PSPACE enthalten ist.
Von keiner der drei ist es derzeit bekannt.

3.2 Platzkomplexität und Nichtdeterminismus

Satz 21 (Satz von Savitch)
$s(n)$ *platzkonstruierbar* $s(n) \geq \log n$
NSPACE($s(n)$) $\subseteq$ DSPACE($s(n)^2$)

BEWEIS Sei $L \in$ NSPACE($s(n)$) und T eine nichtdeterministische TM. mit $L = L(T)$ und SPACE($T(x)$) $\leq s(n) \quad n = |x|$
Sei V die Menge der globalen Konfigurationen von T deren Bänder Größe $s(n)$ haben. Deren Zahl ist durch $2^{c \cdot s(n)}$ für ein c beschränkt.
Beachte: hier geht die Voraussetzung $s(n) \geq \log(n)$ ein, denn der Lesezeiger auf das Eingabeband verbraucht Platz $\log(n)$.
Setze $N := 2^{c \cdot s(n)}$ = Anzahl dieser globalen Konfigurationen.

Für $g, g' \in V$ schreibe $(g, g') \in E$ falls g' mögliche Ein-Schritt Folgekonfiguration in T ist.
Schreibe s für Startkonfiguration von T auf x
Schreibe t für die akzeptierende Endkonfiguration (oBdA können wir annehmen, dass es genau eine solche gibt)
Es gilt dann $x \in L \Leftrightarrow$im Graphen (V, E) ein Pfad von s nach t existiert
Definiere rekursive Prozedur reach(g, g', i) sodass reach$(g, g', i) =$ true $\Leftrightarrow$in (V, E) ein Pfad von g nach g' der Länge $\leq 2^i$ existiert.
reach : $V \times V \times \mathbb{N} \to$ bool
Es gilt dann:
$x \in L \Leftrightarrow$ reach$(s, t, i) =$ true für $i = c \cdot s(n)$
reach$(g, g', i) =$
  if$(i = 0)$
    if$(g = g' \vee (g, g') \in E)$
      return true
    else return false
  else :
    for$(g'' \in V)$
      if(reach$(g, g'', i-1) \wedge$ reach$(g'', g', i-1))$ return : true
    return : false
Ein Stackframe ist größenmäßig durch $\mathcal{O}(s(n))$ beschränkt.
Rekursionstiefe = Stackhöhe $\leq cs(n)$
Die gesamte Prozedur lässt sich also in Platz $\mathcal{O}(s(n)^2)$ auswerten.
Alternativ: funktionale Version von reach:

$$\text{reach}(g, g', 0) = \begin{cases} \text{true} & \text{if } g = g' \text{ or} (g, g') \in E \\ \text{false} & \text{sonst} \end{cases}$$

reach$(g, g', i+1) =$ checkfrom(g_0, g, g', i)
checkfrom$(g'', g, g', i) =$
  reach$(g, g'', i) \wedge$ reach$(g'', g', i) \vee$ checkfrom(next$(g''), g, g', i)$ Falls g'' nicht die letzte Konfiguration ist
checkfrom$(g'', g, g', i) =$ reach$(g, g'', i) \wedge$ reach(g'', g', i) sonst
Hierbei sei $g_0, \text{next}(g_0), \text{next}(\text{next}(g_o)), \ldots$ eine beliebige Aufzählung von V.
Auswerten von reach$(s, t, cs(n))$ führt auf Ausdrücke der Größe $\mathcal{O}(s(n)^2)$ ∎

Korollar 22
Ereichbarkeit in Graphen $\in$ DSPACE$((\log n)^2)$
NLOGSPACE $\subseteq$ DSPACE$((\log n)^2)$
$\underbrace{\text{NSPACE}(n^{\mathcal{O}(1)})}_{=\text{NPSPACE}} =$ PSPACE

Satz 23 (Immerman-Szelepscényi)
Sei $s(n) \geq \log n$ platzkonstruierbar.
co-NSPACE$(s(n)) =$ NSPACE$(s(n))$
Hierbei gilt:
co-NSPACE$(s(n)) = \{L | \overline{L} \in \text{NSPACE}(s(n))\}$

BEWEIS Sei T eine $s(n)$ platzbeschränkte NTM für Σ. Sei V die Menge ihrer globalen Konfigurationen $|V| \leq 2^{cs(n)}$
s: Startkonfiguration bei Eingabe x. $|x| = n$
t: akzeptierende Endkonfiguration
$E = \{(g, g') | g'$ ist 1-Schritt Folgekonfiguration von $g\}$
$x \in L \Leftrightarrow t$ ist von s aus in (V, E) nicht erreichbar.

Folgender nichtdeterministischer Algorithmus *kann* ACCEPT ausgeben gdw. t von s aus *nicht* erreichbar ist, also wenn $x \notin L(T)$ ist. Ist $x \in L(T)$, so gibt der Algorithmus auf jeden Fall FAIL aus. Ist aber $x \notin L(T)$, so kann man die nichtdeterministischen Entscheidungen des Algorithmus so treffen, dass ACCEPT ausgegeben wird. Bei ungeschickter Wahl dieser Entscheidungen kann (und wird in den meisten Fällen) trotzdem FAIL ausgegeben.

```
/* Invariant: count = # Config reachable from s in <= i steps */
i = 0;count = 1; /* Invariant ok */
while (i <= N) {
      count' = 0; /* count' is to be the new value of count for i+1 */
      for (v:V) { /* for each v we see whether it can be reached from s in
<=i+1 steps.*/
         if (v==s) {count'++;} else {
            c = 0;found = false;
            for (u:V) { /* for each u we check whether v is reachable from
s with last step via u */
                guess path pi of length <=i starting from s;
                if (pi ends in u) {
                    c++;
                    if( (u,v):E ) {found=true; /* found a path of length
<= i+1 from s to v (via u)*/}
                }
            }
            if (found) count'=count'+1;
            else if (c<count)  FAIL; /* We missed some of the u's
reachable in <=i steps */
            else ;
            /* If !found and we haven't FAILed then indeed v isn't
reachable from s in <=i+1 steps so
               we don't need to update count'.*/
        }
     }
     count = count';i++;
}
/* At this point count equals the number of configs reachable from s */
c = 0;found = false;
for (v:V) {
    guess path pi of length <=N starting from s;
```

```
    if (pi ends in v) {
       if (v==t) found=true;
       c++;
    }
}
if (c<count || !found) FAIL;
else ACCEPT;
```

Beispiel 5
Berechnung von count:
In ≤ 3 Schritten seien 5 Knoten ereichbar
in ≤ 4 Schritten seien 7 Knoten erreichbar
Zu Begin der Schleife sei i=3 und count=5

- Wir können jetzt den Durchlauf der Schleife so steuern, dass am Ende des Rumpfes count =7 ist.
- Egal wie der Durchlauf gesteuert wird, ist am ende count = 7 oder es wurde abgebrochen (FAIL)

count = 5

1. 1. Fall: t nicht erreichbar Wir können dann T' nach Accept lenken:
   Rate für jeden ereichbaren Knoten genau den Pfad der ihn erreicht.
2. 2. Fall: t ist erreichbar
   Egal wie T' gesteuert wird erreichen wir immer FAIL und nich accept.
   (a) Für t wurde der richtige Fall geraten ⇒FAIL
   (b) Für t wurde nicht der richtige Fall geraten ⇒count $>$ r ⇒FAIL

3.3 Ausdruckskraft eines Stacks

Definition 32 (s(n)-platzbeschränkte TM mit STACK)
Eine $s(n)$-platzbeschränkte TM mit STACK hat neben einem read only Eingabebande eine feste Zahl von $s(n)$ platzbeschränkte Arbeitsbänder
Zusätzlich ein weiteres Arbeitsband S (STACK) welches wie folgt eingeschränkt ist:

1. Wird durch ein Quintupel das Band S verändert, so werden in diesem Schritt alle anderen Bänder nicht verändert.
2. Wird in einem Quintupel der S-Kopf nach rechts bewegt, so wird das aktuelle S-Symbol nicht verändert. Im unmittelbar nächsten Schritt wird ein S-Symbol beschrieben.

3. Wird in einem Quintupel der S-Kopf nach Links bewegt, so wird das aktuelle S-Symbol mit $\square$ überschrieben.

Sei x eine Eingabe. Mit V bezeichnen wir die Menge aller Beschriftungen der Arbeitsbänder außer S und die Positionen der Köpfe auf all diesen Bändern.
Für $g, g' \in V$ und $s \in \Sigma$ schreiben wir:
$(g, s) \xrightarrow{\text{TOP}} g'$ falls g in g' übergeht bei Lesen von s vom S-Kopf
$(g, s) \xrightarrow{\text{POP}} g'$ falls g übergeht in g' bei lesen von s beim S-Bandes und bewegung des S-Kopfes nach links (nach obrigen Regeln)
$(g, s) \xrightarrow{\text{PUSH}(s')} g'$ falls g in g' übergeht bei Lesen von s beim S-Band und Bewegung des S-Kopfes nach rechts mit anschließenden Schreiben von s'
Es sollte klar sein, wie aus der Maschinenbeschreibung die Relationen $\xrightarrow{\text{TOP}}, \xrightarrow{\text{POP}}, \xrightarrow{\text{PUSH}(s')}$ abgelesen werden können.
Eine solche Maschine akzeptiert ein Wort x, falls von der Startkonfiguration $g_0(x)$ mit Stack enthällt nur $\$, s_0$die Konfiguration g_A mit Stack \$ ereichbar ist ($s_0, \$$ fest gewählt), wobei

1. g_0 die Startbeschriftung zur Eingabe x ist
2. g_A akzeptierende Endbeschriftung fest gewählt ist (Alle Bänder leer, Lesekopf für die Eingabe am Anfang)
3. Im Verlauf der Berechnung wird das Feld mit Inhalt \$nie besucht. Erst ganz zum Schluss

Definition 33
NSPACE($s(n)$) + STACK ist die Menge aller Sprachen die von solchen Maschinen erkannt werden.

Satz 24 (Satz von Cook)
$s(n) \geq \log n$ *platzkonstruierbar*
$\text{NSPACE}\, s(n) + \text{STACK} = \text{DTIME}(2^{\mathcal{O}(s(n))})$

Beweis $\subseteq$:
Sei T solch eine Maschine und eine Eingabe x vorgegeben.
Wir definieren die Relation M wie folgt:
$(g, s, g') \in M$ genau dann wenn
$g, s \rightarrow^* g', \$$
und im Verlauf dieser Berechnung wurde \$ nicht besucht.
Klar. $x \in L(T) \Leftrightarrow (g_0(x), s_0, G_A) \in M$
Wir werden jetzt die vollständige Wertetabelle von M ausrechnen und dann die Frage durch Nachgucken beantworten.
$(g, s, g') \in M$ gilt genau dann, wenn das mithilfe folgender Regeln hergeleitet werden kann.
$g, s \xrightarrow{\text{POP}} g' \Rightarrow (g, s, g') \in M$
$g, s \xrightarrow{\text{TOP}} g', (g', sg'') \in M \Longrightarrow (g, s, g'') \in M$

$g, s \overset{\text{PUSH}(s')}{\longrightarrow} g', (g', s', g'') \in M, (g'', s, g'') \in M \Rightarrow (g, s, g''') \in M$
Durch sukzessive Anwendung dieser drei Regeln kann in Zeit $2^{\mathcal{O}(s(n))}$ die Wertetabelle aufgefüllt werden.
$\supseteq$:
Sei T eine DTM, die $2^{c \cdot s(n)}$ zeitbeschränkt ist. Definiere folgende Funktionen:
$\text{symb}(x, t, p) := (a_1, \ldots a_k)$ sodass a_i steht nach t Schritten von T auf Eingabe x auf dem i-ten Band an Position p.
$\text{head}(x, t) := (n_1, \ldots, n_k)$ sodass bei Eingabe x steht der Schreib/Lesekopf auf Band i an der Position n_i nach t Schritten
$\text{state}(x, t) :=$ Zustand von T bei Eingabe x nach t Schritten
Rekursive Definition:
$\text{smyb}(x, 0, p) = \text{if } 0 \leq p \leq |x| \text{ then } (x_p, \square, \ldots, \square) \text{ else } (\square, \ldots, \square)$
$\text{head}(x, 0) = (0, \ldots, 0)$
$\text{state}(x, 0) = q_0$
$\text{symb}(x, t{+}1, p) = F(\text{symb}(x, t, p), \text{symb}(x, t, p{+}1), \text{symb}(x, t, p{-}1), \text{head}(x, t), \text{state}(x, t))$
$\text{state}(x, t+1) = H(\text{symb}(x, t, \text{head}(x, t)), \text{head}(x, t), \text{state}(x, t))$
$\text{state}(x, t+1) = G(\text{state}(x, t, \text{head}(x, t)), \text{state}(x, t))$
Für F, G, H gegeben durch δ
T ist $2^{c \cdot s(n)}$ Zeitbeschränkt.
$\Rightarrow$die Argumente der Fkt. symb, head, state sind durch $\mathcal{O}(s(x))$ beschränkt
$\Rightarrow$Die Funktionen können daher auf einer (sogar det.) TM mit Stack und Platz $\mathcal{O}(s(n))$ berechnet werden. ∎

3.4 LOGSPACE-Reduktionen und vollständige Probleme für P und NLOGSPACE

Definition 34 (E/A-TM)
Eine (N)TM mit Ein- und Ausgabe ist eine (N)TM mit $k \geq 2$ Bändern, das Band 1 das Eingabeband (E) ist und Band k das Ausgabeband (A)

- E bleibt bei der Berechnung unverändert
- A wird nur einmal von links nach rechts beschrieben.

Sei T eine E/A-(N)TM und $x \in \Sigma^*$. Definiere:
$\text{(N)SPACE}_T(x) =$ Platzverbrauch auf den Bändern 2 bis $k-1$
(der sparsamsten Berechnung im Falle von Nichtdeterminismus.)

Definition 35 (FSPACE(f(n)))
$g : \Sigma^* \to \Sigma^* \in \text{FSPACE}(f(n)) :\Leftrightarrow$
es existiert det. E/A TM T, die g berechnet (bei Eingabe x steht $g(x)$ auf dem Ausgabeband wenn T hält), so dass $\text{SPACE}_T(x) = \mathcal{O}(f(|x|))$
$\text{FLOGSPACE} = \text{FSPACE}(\log n)$

Satz 25
Sind $f, g \in \text{FLOGSPACE}$ *so auf* $g \circ f$

BEWEIS Problem: g braucht $\log(|f(x)|)$ Platz
Okay da $|f(x) \leq p(|x|)$ Für ein Polynom.
(es gibt $2^{\mathcal{O}(\log(|x|))}$ viele Konfigurationen. Daher $2^{\mathcal{O}(\log(|x|)}$ Schritten: halten oder Zyklus.)
Aber man kann $f(x)$ nicht zwischenspeichern.
Lösung: on-the-fly berechnung von $f(x)$
Gegeben E/A TMs T_G udn T_f für g und f Simuliere die Hintereinanderschaltung von T_f und T_g mit E/A TM T wie folgt:

- Baue T_f so um, dass das Ausgabeband nur auf einem Feld beschrieben wird. Auf diesem Feld steht das oberste Stacksymbol.
- T verhält sich nun wie T_g. T liest jedoch nicht vom Eingabeband, sondern vom modifizierten Ausgabeband von T_f
  T :
  Eingabeband:
  Bänder von T_f $\mathcal{O}(\log(|x|))$
  Puffer = modifizierte Ausgabeband von T_f $\mathcal{O}(1)$
  Pufferzähler $\mathcal{O}(\log(f(x))) = \mathcal{O}(\log(p(x))) = \mathcal{O}(\log(x))$
  Bänder von T_g $\mathcal{O}(\log(f(x))) = \mathcal{O}(\log(x))$
  Ausgabeband

Wie simuliert T die Maschine T_g:
z.B. T_g bewegt Eingabekopf nach rechts:
T erhält Pufferzähler um 1
T berechnet mit dem mod T_f $f(x)$ bis # Pufferzähler-mal in den Puffer geschrieben wurde
T liest vom Puffer und simuliert T_g weiter ■

Definition 36 (LOGSPACE-Reduktion)
Seien $A, B \subseteq \Sigma^*$
$A \leq_L B :\Leftrightarrow \exists f \in$ FLOGSPACE mit $(x \in A \Leftrightarrow f(x) \in B)$

Satz 26
$\forall A, B \subseteq \Sigma^*$ *gilt:*

1. $A \leq_L A$ *(Reflexivität)*
2. $A \leq_L B$ *und* $B \leq_L C$ *dann* $A \leq_L C$ *(Transitivität*

BEWEIS 1. ist klar

2. Folgt aus vorherigem Satz

Definition 37 (NLOGSPACE-vollständig)
$A \subseteq \Sigma^*$ ist vollständig für NLOGSPACE falls

1. $A \in$ NLOGSPACE
2. $B \leq_L A$ für alle $B \in$ NLOGSPACE

Definition 38 (P-vollständig)
$A \subseteq \Sigma^*$ ist vollständig für P falls

1. $A \in$ P
2. $B \leq_L A$ für alle $B \in$ P

Definition 39 (REACH)
Gegeben: Gerichteter Graph $G = (V, E), s, t \in V$
Frage: Gibt es einen Pfad von s nach t in G?

Bemerkung 27
UREACH $\in$ LOGSPACE Rechte-Hand-Regel.

Satz 27
REACH *ist* NLOGSPACE-*vollständig*

BEWEIS 1. REACH $\in$ NLOGSPACE:
Speichere folgendes:
Einen akt. Knoten $v \in V$
einen Zählen

Starte mit $v = s$ und $n = 0$
Rate Nachfolger v' von v mit $(v, v') \in E$
etze $v := v'$ und $n = n + 1$
Akzeptiere falls $v = t$ erwerfe falls $n = |V|$

2. Sei $A \in$ NLOGSPACE
$\Rightarrow$es existiert NTM T mit $L(T) = A$ und $\text{NSPACE}_T(x) = c \cdot \log(|x|)$
OBdA hat T nur eine globale Endkonfiguration E T besitzt bei Eingabe x $2^{\mathcal{O}(\log(|x|))} = \mathcal{O}(p|x|))$ viele globale Konfigurationen.
Betrachte $G_x = (V, E)$ mit $V =$ Globale Konfigurationen von T bei eingabe x
$E = \{(k, k') |\ T$ kommt in einem Schritt von k nach $k'\}$
Jetzt gilt:
$x \in A \Leftrightarrow (G_x, S, E) \in$ REACH
G_x kann aber aus x in LOGSPACE berechnet werden: zähle systematisch alle Paar von (k, k') von Konfigurationen auf und Prüfe ob $k \to_1 k'$ ∎

Definition 40 (HORN-SAT)
Gegeben: KNF F mit maximal einem positiven Literal pro Klausel. (oBdA: 3 literale)
Frage: Ist F erfüllbar?

Satz 28
HORN-SAT *ist vollständig für* P

BEWEIS Sei eine polynomialzeitbeschränkte DTM M vorgegeben und Eingabe $x \in \Sigma^*$ Wir konstruieren eine Menge HORN Formeln $\mathcal{H}$, sodass $\mathcal{H}$ unerfüllbar ist $\Leftrightarrow M$ die Eingabe x akzeptiert.
Variablen:
Band(i,t,a) = für $i \leq T, t \leq T, a \in \Sigma, T$ = max Laufzeit von M auf x (polynomiell beschränkt in $|x|$)
Kopf(i,t) für $i,t \leq T$
Zustand(t,q) für $t \leq T q \in Q_M$
$X(i,t,b,q',d)$ für $i,t \leq T$, $b \in \Sigma, q' \in Q_m, d \in \{-1,0,1\}$ Klauseln:
$\rightarrow$ Kopf$(0,0)$ /* Am Anfang steht der Kopf an Pos. O
$\rightarrow$ Zustand$(0,q_0)$
$\rightarrow$ Band$(i,0,x_i)$ für $i = 0,\ldots,|x|-1, x = x_0x_1\ldots \rightarrow$ Band$(i,0,\square)$ für $i \geq |x|$
falls $\delta(a,q) = (b,q',d) \quad d \in \{-1,0,1\}$:
Kopf(i,t), Band(i,t,a), Zustand$(t,q) \rightarrow X(i,t,b,q',d)$
$X(i,t,b,q',d) \rightarrow$ Zustand$(t+1,q')$
$X(i,t,b,q',d) \rightarrow$ Band$(i,t+1,b)$
$X(i,t,b,q',d) \rightarrow$ Kopf$(i+d,t+1)$
Zust$(T,q_x) \rightarrow$ false
Kopf(i,t), Band$(j,t,a) \rightarrow$ Band$(j,t+1,a)$ für $i,j,t \leq T, i \neq j, a \in \Sigma$
Bemerkung: false ist genau dann herleitbar, wenn M auf $|x|$ akzeptiert.
Beobachtung: die obige Klauselmenge kann aus M und x in LOGSPACE berechnet werden.

Bemerkung 28 (Erinnerung)
PSPACE = DSAPCE$(n^{\mathcal{O}(1)})$ = NSPACE$(n^{\mathcal{O}(1)})$ (wegen Satz von Savitch)
NSPACE(n^2) ist wahrscheinlich nicht in DSPACE(n^2) enthalten aber sehr wohl in DSPACE(n^4)

Definition 41 (Quantifizierte Boolesche Formeln)
Quantifizierte Boolesche Formeln sind durch folgende Gramatik gegeben: $\varphi, \psi ::== X|\varphi \wedge \psi \ |\varphi \vee \psi|\neg\varphi|\forall X.\varphi|\exists X.\varphi$
Eine QBF ist geschlossen, wenn sie keine freien Variablen enthällt.
(alle vorkommenden Variablen sind durch $\forall$ oder $\exists$ gebunden)
Das Problem QBF besteht darin, zu gegebener geschlossener QBF φ den Wahrheitswert zu bestimmen.

Bemerkung 29
SAT und TAUT sind Speziallfälle von QBF

Definition 42 (Formale Semantik von QBF)
Sei η eine Belegung der Variablen.
Wie definieren rekursiv:
$\eta \models \varphi \quad (\varphi$ QBF
$\eta \models X \Leftrightarrow \eta(X) =$ true
$\eta \models \varphi \wedge \psi \Leftrightarrow \eta \models \varphi$ und $\eta \models \psi$
$\eta \models \varphi \vee \psi \Leftrightarrow \eta \models \varphi$ oder $\eta \models \psi$

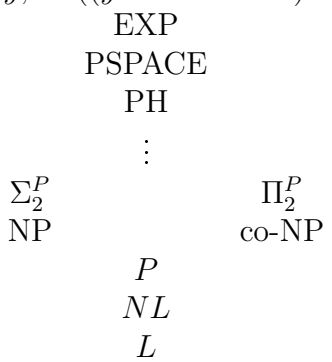
$\eta \models \neq \varphi \Leftrightarrow \eta \not\models \varphi$
$\eta \models \forall X.\psi \Leftrightarrow \eta[X \mapsto \text{true}] \models \psi$ und $\eta[X \mapsto \text{false}] \models \psi$
$\eta \models \exists X.\psi \Leftrightarrow \eta[X \mapsto \text{true}] \models \psi$ oder $\eta[X \mapsto \text{false}] \models \psi$

Satz 29
QBF $\in$ PSPACE

BEWEIS Die Klauseln rekursiv ausprogrammieren. Rekursionstiefe, sowie Stackframes haben lineare Größe ■

Satz 30
GBF *ist* PSACE *vollständig*

BEWEIS Sei M polynomiell platzbeschränkte TM, Eingabe x vorgegeben.
$V =$ Menge der globalen Konfigurationen
$E = \{(v, v') | v, v' \in V, v'$ Ein-Schritt-Nachfolger von $v\}$
s Startkonfiguration, t akzeptierende Endkonfiguration
$x \in L_M \Leftrightarrow (s,t) \in E^*$
Erinnerung:
$x \in L_m \Leftrightarrow \text{REACH}(s,t,T)$ T sodass $2^T > |V|$
$\text{REACH}(u,v,0) \Leftrightarrow (u,v) \in E$ oder $u = v$
$\text{REACH}(e,v,t+1) \Leftrightarrow \exists w.\, \text{REACH}(u,w,t) \wedge \text{REACH}(w,r,t)$
Codiere globale Konfiguration durch boolesche Variablen (T Stück)
Definiere für $i = 0, \ldots T$ QBF Formel:
$\varphi_i(u,v)$ in $2T$ Variablen, sodass $\varphi_i(u,v) \Leftrightarrow \text{REACH}(u,v,i)$
$\varphi_0(u,v) :\Leftrightarrow u = v$ oder $(u,v) \in E$
$\varphi_i + 1(u,v) :\Leftrightarrow \exists w.\varphi_i(u,w) \wedge \varphi_i(w,v)$ $\quad (*)$ gib aus $\varphi_T(i,t)$
Dies ist noch keine Polynomielle Reduktion da $|\varphi_i(u,v)| = \Omega(2^i)$
Ersetze $(*)$ durch:
$\exists w.\forall y,t(y = u \wedge z = w \rightarrow \varphi_i(y,z)) \wedge (y = w \wedge z = v \rightarrow \varphi_i(y,z)$
Besser noch:
$\exists w \forall y,z : ((y = u \wedge z = w) \vee (y = w \wedge z = v)) \Rightarrow \varphi_i(y,z)$



■

Bemerkung 30
Ermittlung von Gewinnposition in Spielen, deren Positionsgröße und Spieldauer polynomiell beschränkt ist, ist in PSPACE, denn man kann das Problem in QBF formulieren
Oft sind solche Spiele auch PSPACE vollständig

4 Programmiersprachen und Komplexität

4.1 Beschränkte Notationsrekursion, Satz von Cobham (1965)

Definition 43 (Notationsrekursion)
Sei g eine n-Stellige Funktion auf $\mathbb{N}$ und h eine $n+2$-Stellige Funktion
Aus g und h wird die $n+1$-stellige Funktion f durch Notationsrekursion definiert, wenn gilt:
$f(0, \vec{x}) = g(\vec{x})$
$f(x, \vec{x}) = h(x, f(\frac{x}{2}, \vec{x})$ falls $x > 0$
Man schreibt: $f = \text{rec}(g, h)$

Definition 44 (Grundfunktionen)

1. 'Nachfolger': $S_0(x) = 2x \quad S_1(x) = 2x + 1$
2. 'Vorgänger': $p(x) = \frac{x}{2}$
3. 'Null' : 0
4. 'Fallunterscheidung' : $\text{coud}(x, y, z, w) = \begin{cases} y & \text{falls} x = 0 \\ z & \text{falls } x \text{ gerade und } > 0 \\ w & \text{falls } x \text{ ungerade} \end{cases}$
5. 'Projektion': $\text{proj}_i^n(\vec{x}) = x_i$

Bemerkung 31
Rekursionstiefe einer durch Notationsrekusion definierten Funktion $f = \text{rec}(g, h)$
$f(x, \vec{x})$ ist $\log x \approx |x|$
Hier: $|x|$ = Länge der Binärdarstellung von $x = \lceil \log_2(x+1) \rceil$
$S_1(\underbrace{1101_2}_{=13_{10}}) = \underbrace{1101}_{=27_{10}}$
$S_0(1101) = 11010$
$p(11011) = 1101$
$p(1101101) = 110110$
$\text{concat}(x, y) = x2^{|y|} + y$
$\text{concat}(x, 0) = x$
$\text{concat}(x, S_0(y)) = S_0(\text{concat}(x, y))$ Falls $y > 0$
$\text{concat}(x, S_1(y)) = S_1(\text{concat}(x, y))$
Formal:
concat' $= rec(g, h)$ wobei:
$g(x) = x$
$h(y, z, x) = \text{cond}(y, 0, S_0(z), S_1(z))$
$\text{concat}(x, y) = \text{concat'}(y, x)$

Satz 31
Die Klasse der Funktionen, die aus den Grundfunktionen mit Notationsrekursion und Komposition gebildet werden kann, ist gleich der Klasse der primitiv rekursiven Funktionen

BEWEIS Definiere:
$\text{code}(x) = 2^x$
$\text{decode}(x) = |x|$
Ist $f(\vec{x})$ primitiv rekursiv, so gibt es eine Funktion $f^{\#}$ die mit Notationsrekursion definierbar ist, sodass gilt:
$\text{decode}\left(f^{\#}(\text{code}(x_1), \ldots, \text{code}(x_n))\right) = f(x_1, \ldots x_n)$
Decode und code sind ebenfalls definierbar und daraus erhält man Def. von f
Intuittion:
$f(0) = 1$
$f(x) = \text{concat}(f(\frac{x}{2}), f(\frac{x}{2}))$
$|f(0)| = 1$
$|f(x)| = 2|f(\frac{x}{2})|$
$|f(x)| = 2^{|x|}$ ■

Definition 45 (beschränkte Notationsrekursion (Cobham))
g: n-stellig, $h : n+2$ Stellig, $k : n+1$ Stellig
$f = \text{brec}(g, h, k)$ (f entsteht aus g,h, k durch beschränkte Notationsrekursion) wenn gilt:
$f(0, \vec{x}) = g(\vec{x})$
$f(x, \vec{x}) = h(x, f(\frac{x}{2}, \vec{x}), \vec{x})$
$f(x, \vec{x}) \leq k(x, \vec{x})$

Bemerkung 32
$\text{brec}(g, h, k)$ ist nur wohlgeformt, falls die dritte Formel gilt

Bemerkung 33 (Alternative Definition)
$f(0, \vec{x}) = g(\vec{x})$
$f(x, \vec{x}) = \min(k(x, \vec{x}), h(x, f(\frac{x}{2}, \vec{x}), \vec{x}))$

Definition 46
$x\#y = 2^{|x| \cdot |y|}$
es gilt: $|x\#y| = |x| \ \cdot |y|$

Satz 32 (Cobham)
Die Klasse der Funktionen, die mit den Grundfunktionen $0, S_0, S_1, p, \text{cond}, \text{proj}, \#$ *durch beschränkte Notationsrekursion und Komposition definierbar sind ist gleich* FP *(Polynomielle Zeit)*
Also brec = FP

BEWEIS brec ⊆ FP Durch Induktion über Darstellung.
Grundfunktionen ∈ FP
Komposition, da FP unter Komposition abgeschlossen ist.
brec:
ind $g, h, k \in$ FP so auch $\text{brec}(g, h, k)$ falls wohlgeformt:

Rekursionstiefe für $f(x, \vec{x})$ ist $|x|$
Größe der Zwischenergebnisse ist beschränkt durch $|k(x, \vec{x})$ (oBdA k monoton)

FP $\subseteq$ brec Mithilfe von # als Schranke lässt sich concat definieren.
$|\operatorname{concat}(x, y)| = |x| + |y| \leq (|x| + 1)(|y| + 1) = |S_0(x) \# S_1(y)|$
Es gilt sogar $\operatorname{concat}(x, y) \leq S_1(x) \# S_1((x)$
Mit concat und # erhällt für jedes Polynom p eine Funktion, sodass
$f(|x|) = |f_p(x)|$
Ein Schritt Funktion einer TM kann somit mit brec und f_p polynomiell lang iteriert werden. ■

Bemerkung 34 (Beschränkte Arithmetik)
Grundfunktionen $S_0, S_1, \text{cond}, \#, \ldots$
Notationsinduktion:
$\varphi(0) \wedge \left(\forall x.\varphi\left(\frac{x}{2}\right) \rightarrow \varphi(x)\right) \rightarrow \forall x.\varphi(x)$
Für Formeln der Form:
$\varphi(x) = \exists y \leq t.\psi(x, y)$
in $\psi(x, y)$ sind alle Quantoren der Form:
$\exists z \leq |t| \qquad \forall z \leq |t|$

Satz 33 (Buss)
Ist $\forall x.\exists y.\psi(x, y)$ beweisbar, so gibt es FP *Funktion, die zu jedem x ein passendes y findet.*

4.2 Sichere Rekursion nach Bellantoni-Cook

1992 stellen Bellantoni, Cook eine Variante von Cobhams System vor, die ohne explizite Schranken auskommt.

Definition 47
Die Variablen einer Funktion werden in 'sicher' und 'normal' eingeteilt.
Je mehr Variablen sicher sind, umso besser.
Das System BC (Belantoni Cook) besteht aus allen Funktionen die mit den folgenden Regeln definiert werden können:
Notation: $f(\vec{x}; \vec{y})$ bedeutet: die $\vec{x}$ sind normal, die $\vec{y}$ sind sicher.

- Grundfunktionen: $0, S_0(; x), S_1(; x), p(; x), \text{cond}(; x, y, z, w)$
  Alle Argumente der Grundfunktionen sind sicher.
- Projektionen: $\text{proj}(\vec{x}; \vec{y}) = x_i, \text{proj}(\vec{x}; \vec{y}) = y_i$
- Komposition:
  $f(\vec{x}; \vec{y}) = g(u_1(\vec{x};), \ldots, u_m(\vec{x};); v_1(\vec{x}; \vec{y}), \ldots, v_n(\vec{x}; \vec{y}))$
  Falls $y, \vec{u}, \vec{v} \in$ BC so auch f
  Will man eine Funktion in eine normale Argumentationsposition einsetzen, so müssen all ihre Argumente vorher auf normal herabgestuft werden (Formal durch Komposition mit einer Projektion)

- Sichere Notationsrekursion:
  $f(x, \vec{x}; \vec{y}) = \begin{cases} g(\vec{x}; \vec{y}) & \text{falls } x = 0 \\ h(x, \vec{x}; f\left(\frac{x}{2}, \vec{x}, \vec{y}\right) & \text{falls } x > 0 \end{cases}$
  Sind g, h, aus BC so auch f
  Rekurriert wird nur über normale Argumente.
  Zugriff auf Ergebnisse rekursiver Aufrufe kann nur über sichere Argumente erfolgen.

Man schreibt: $f = \text{srec}(g, h)$

Beispiel 6
$\text{concat}(y, x) = x \cdot 2^{|y|} + y$
$\text{concat}(0, x) = x$
$\text{concat}(y, x) = \text{cond}\left(y, S_0\left(\text{concat}\left(\frac{y}{2}, x\right)\right), S_1\left(\text{concat'}\left(\frac{y}{2}, x\right)\right)\right)$
Da über y rekurriert wird muss y als normal eingestuft werden.
x kann als sicher eingestuft werden.
Verwendung der rekursiven Aufrufe erfolgt über sichere Positionen.
Formal:
$g(; x) = x$
$h(y; z, x) = \text{cond}\left(; y, 0, S_0\left(; z\right), S_1\left(; z\right)\right)$
Formal ist h eine sichere Komposition aus $\text{cond}(; u, v, w, z)$ und $\text{proj}(y; z, x) = y$ und $O(y; z, x) = 0$ und $w(y; z, x) = S_0(z)$ und $v(y; z, x) = S_1(; z)$.

$\text{concat} = \text{srec}(g, h)$

Ist das folgende legal in BC?:
$f(0;) = S_1(; 0)$
$f(x;) = \text{concat}(f(x/2;); f(x/2;))$

Hier wäre: $h(x; z) = \text{concat}(z; z)$
$h(x; z) =$ Komposition von $\text{concat}(u; v)$ mit
$u(x; z) = z$ und $v(x; z) = z$ ist nicht legal, da sichere Variable in normale Position u eingesetzt wurde.

Folgende Definition ist aber legal:

$f(0, y;) = y$
$f(x, y;) = \text{concat}(y; f(x/2, y;$
Was ist $|f(x, y;)|$ als Funktion von $|x|$ und $|y|$?