# Komplexitätstheorie

Markus Lohrey

Universität Siegen

Wintersemester 2021/2022

# Teil 1: Grundlagen

Im folgenden behandeln wir einige Grundlagen:

- Turingmaschinen (nicht-deterministisch, deterministisch)
- Konfigurationen
- Rechnungen,...

Das meiste davon können Sie später wieder vergessen, denn:

- Turingmaschinen können auf viele verschiedene äquivalente Weisen definiert werden.
- Wir könnten Turingmaschinen auch durch andere äquivalente Rechenmodelle ersetzen (z. B. Registermaschinen).

# Turingmaschinen: Definition

Notation: Mit $\mathcal{P}_{\neq\emptyset}(A) = 2^A \setminus \{\emptyset\}$ bezeichnen wir die Menge aller nicht-leeren Teilmengen der Menge $A$.

## Definition 1

Eine nichtdeterministische $k$-Band Turingmaschine ist ein Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$

- $Q$ : endliche Menge der Zustände
- $q_0 \in Q$ : Startzustand
- $q_J \in Q$: akzeptierender Zustand
- $q_N \in Q$: ablehnender Zustand, wobei $q_J \neq q_N$
- $\Gamma$ : endliches Bandalphabet
- $\Sigma$: endliches Eingabealphabet mit $\triangleright, \triangleleft \notin \Sigma$
- $\Box \in \Gamma$: Blanksymbol
- $\delta : (Q \setminus \{q_J, q_N\}) \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma^k \to \mathcal{P}_{\neq\emptyset}(Q \times \Gamma^k \times \{-1, 1\}^{k+1})$: Übergangsfunktion. $-1$ $(1)$: bewege Kopf nach links (rechts)

# Turingmaschinen: Definition

Für alle Anweisungen $(p, c_1, \ldots, c_k, d_0, \ldots, d_k) \in \delta(q, a, b_1, \ldots, b_k)$ gilt:

- $a = \triangleright \quad \Rightarrow \quad d_0 = 1$
- $a = \triangleleft \quad \Rightarrow \quad d_0 = -1$

Bei einer deterministischen $k$-Band Turingmaschine $M$ gilt

$$\delta : (Q \setminus \{q_J, q_N\}) \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 1\}^{k+1}$$

Eine Turingmaschine mit Ausgabe ist wie eine deterministische Turingmaschine definiert, außer dass noch ein Ausgabealphabet $\Sigma'$ existiert, und für $\delta$ gilt:

$$\delta : (Q \setminus \{q_J, q_N\}) \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma^k \to Q \times \Gamma^k \times \{-1, 1\}^{k+1} \times (\Sigma' \cup \{\lambda\})$$

($\lambda$ ist das leere Wort).

# Turingmaschinen: Konfigurationen

## Definition 2

Eine Konfiguration $\alpha$ der Turingmaschine $M$ bei Eingabe $w \in \Sigma^*$ ist ein Tupel $\alpha = (q, i, u_1, i_1, \ldots, u_k, i_k)$ mit:

- $q \in Q$ : aktueller Zustand der Turingmaschine
- $1 \leq i \leq |w| + 2$: Der Lesekopf für das Eingabeband steht gerade auf dem $i$-ten Symbol von $\triangleright w \triangleleft$.
- $\forall j \in \{1, \ldots, k\} : u_j \in \Gamma^+, 1 \leq i_j \leq |u_j|$: Das $j$-te Arbeitsband hat den Inhalt $\cdots \Box \Box u_j \Box \Box \cdots$ und der $j$-te Schreib/Lesekopf liest gerade das $i_j$-te Symbol von $u_j$. Falls $i_j < |u_j|$ (bzw. $i_j > 1$) gilt, darf $u_j$ nicht mit $\Box$ enden (bzw. anfangen).

Die Länge $|\alpha|$ der Konfiguration $\alpha = (q, i, u_1, i_1, \ldots, u_k, i_k)$ ist $|\alpha| = max\{|u_j| \mid 1 \leq j \leq k\}$.

# Turingmaschinen: Startkonfiguration, Übergänge, . . .

1. Für die Eingabe $w \in \Sigma^*$ ist
$$\mathrm{Start}(w) = (q_0, 1, \Box, 1, \dots, \Box, 1)$$
die zu $w$ gehördende Startkonfiguration.
Beachte: $|\mathrm{Start}(w)| = 1$.
2. Für ein $\tilde{u} \in Q \times \Gamma^k \times \{-1, 1\}^{k+1}$ und Konfigurationen
$$\alpha = (q, i, u_1, i_1, \dots, u_k, i_k) \text{ und } \beta$$
schreiben wir $\alpha \vdash_{\tilde{u}} \beta$, falls
$$\tilde{u} \in \delta(q, (\triangleright w \triangleleft)[i], u_1[i_1], \dots, u_k[i_k])$$
und die Anwendung der "Anweisung" $\tilde{u}$ auf die Konfiguration $\alpha$ die Konfiguration $\beta$ ergibt.
Übung: Definieren Sie dies formal.
3. Es gelte $\alpha \vdash_M \beta$ falls ein $\tilde{u} \in Q \times \Gamma^k \times \{-1, 1\}^{k+1}$ mit $\alpha \vdash_{\tilde{u}} \beta$ existiert.

# Turingmaschinen: Rechnungen, Protokolle

1. $\mathrm{Accept}_M$ (bzw. $\mathrm{Reject}_M$) ist die Menge aller Konfigurationen mit aktuellen Zustand $q_J$ (bzw. $q_N$).
   Beachte: Für $\alpha$ gibt es keine Konfiguration $\beta$ mit $\alpha \vdash_M \beta$ genau dann, wenn $\alpha \in \mathrm{Accept}_M \cup \mathrm{Reject}_M$.
2. Beachte: $\alpha \vdash_M \beta \Rightarrow |\alpha| - |\beta| \in \{-1, 0, 1\}$
3. Eine Rechnung von $M$ bei Eingabe $w$ ist eine Folge von Konfigurationen $\alpha_0, \alpha_1, \ldots, \alpha_m$ mit
   - $\mathrm{Start}(w) = \alpha_0$
   - $\forall 1 \leq i \leq m : \alpha_{i-1} \vdash_M \alpha_i$

   Die Rechnung ist akzeptierend, falls $\alpha_m \in \mathrm{Accept}_M$.
4. Das Protokoll dieser Rechnung ist die eindeutige Folge

$$\tilde{u}_0 \tilde{u}_1 \cdots \tilde{u}_{m-1} \in (Q \times \Gamma^k \times \{-1, 1\}^{k+1})^*$$

mit $\alpha_i \vdash_{\tilde{u}_i} \alpha_{i+1}$

# Turingmaschinen: akzeptierte Menge, Zeit- und Platzbedarf

1. Der Zeitbedarf (bzw. Platzbedarf) der Rechnung $\alpha_0, \alpha_1, \ldots, \alpha_m$ ist $m$ (bzw. $\max\{|\alpha_i| \mid 0 \leq i \leq m\}$).

2. $M$ hat bei Eingabe $w$ den Zeitbedarf (bzw. Platzbedarf) höchstens $N \in \mathbb{N}$, falls jede Rechnung von $M$ bei Eingabe $w$ Zeitbedarf (bzw. Platzbedarf) $\leq N$ hat.

3. Sei $f : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion.

   $M$ is $f$-zeitbeschränkt, falls $M$ für jede Eingabe $w$ Zeitbedarf höchstens $f(|w|)$ hat.

   $M$ is $f$-platzbeschränkt, falls $M$ für jede Eingabe $w$ Platzbedarf höchstens $f(|w|)$ hat.

4. $L(M) = \{w \in \Sigma^* \mid \exists \text{ akzeptierende Rechnung von } M \text{ bei Eingabe } w\}$ ist die von $M$ akzeptierte Menge.

# Turingmaschinen: Anzahl der Konfigurationen

Das folgende einfache Lemma werden wir häufig verwenden:

## Lemma 3

Sei $M$ eine nichtdeterministische Turingmaschine. Dann existieren Konstanten $c, d$, so dass für alle Eingaben $w$ für $M$ und alle $m \geq 1$ gilt:

- Es gibt höchstens $c \cdot |w| \cdot d^m$ Konfigurationen der Länge $\leq m$ mit $w$ als Eingabe.
- Sei $M$ $f$-platzbeschränkt. Dann ist die Anzahl der von $\mathrm{Start}(w)$ erreichbaren Konfigurationen höchstens $c \cdot |w| \cdot d^{f(|w|)}$.
- Insbesondere: gilt $f \in \Omega(\log(n))$, dann ist die Anzahl der von $\mathrm{Start}(w)$ erreichbaren Konfigurationen höchstens $2^{\mathcal{O}(f(|w|))}$ (falls $|w|$ groß genug ist).

# Komplexitätsklassen

Sei $f : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktionen.

$$
\begin{aligned}
\mathrm{DTIME}(f) &= \{L(M) \mid M \text{ deterministisch } \& \ f\text{-zeitbeschränkt}\} \\
\mathrm{NTIME}(f) &= \{L(M) \mid M \text{ nichtdeterministisch } \& \ f\text{-zeitbeschränkt}\} \\
\mathrm{DSPACE}(f) &= \{L(M) \mid M \text{ deterministisch } \& \ f\text{-platzbeschränkt}\} \\
\mathrm{NSPACE}(f) &= \{L(M) \mid M \text{ nichtdeterministisch } \& \ f\text{-platzbeschränkt}\}
\end{aligned}
$$

Für eine Klasse $\mathcal{C}$ von Sprachen ist $\mathbf{Co}\mathcal{C} = \{L \mid \Sigma^* \setminus L \in \mathcal{C}\}$ die Menge der Komplemente der in $\mathcal{C}$ enthaltenen Sprachen.

# Komplexitätsklassen

Wir werden die Klassen $\mathrm{DTIME}(t)$ und $\mathrm{NTIME}(t)$ nur für Funktionen $t(n)$ mit $\forall n \in \mathbb{N} : t(n) \geq n+1$ betrachten.

Dies erlaubt, die gesamte Eingabe zu lesen.

Wir werden die Klassen $\mathrm{DSPACE}(s)$ und $\mathrm{NSPACE}(s)$ nur für Funktionen $s(n) \in \Omega(\log(n))$ betrachten.

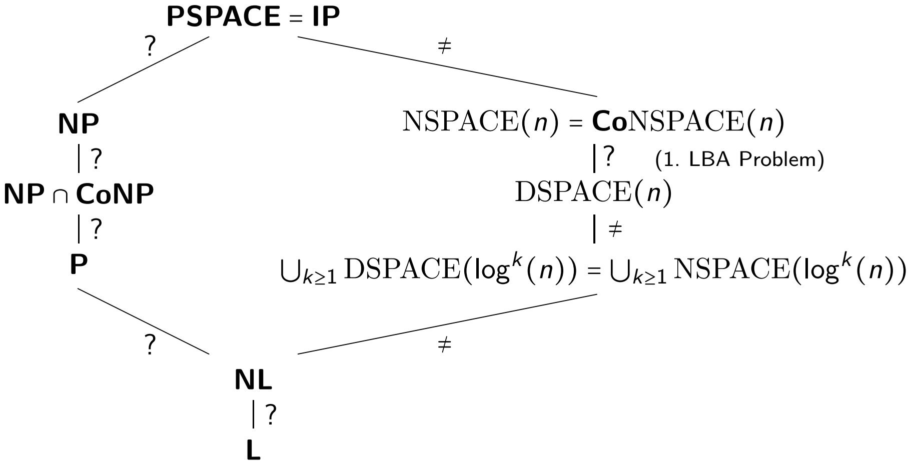
Dies erlaubt, eine Position $i \in \{1, \ldots, n\}$ im Eingabewort auf einem Arbeitsband abzuspeichern.

# Wichtige Komplexitätsklassen

Gebräuchliche Abkürzungen:

$$
\begin{aligned}
\mathbf{L} &= \mathrm{DSPACE}(\log(n)) && (1)\\
\mathbf{NL} &= \mathrm{NSPACE}(\log(n)) && (2)\\
\mathbf{P} &= \bigcup_{k\ge 1} \mathrm{DTIME}(n^k) && (3)\\
\mathbf{NP} &= \bigcup_{k\ge 1} \mathrm{NTIME}(n^k) && (4)\\
\mathbf{PSPACE} &= \bigcup_{k\ge 1} \mathrm{DSPACE}(n^k) = \bigcup_{k\ge 1} \mathrm{NSPACE}(n^k) && (5)
\end{aligned}
$$

Die Gleichung = in (5) ist als Satz von Savitch bekannt (kommt noch).




Es gibt viele weitere Komplexitätsklassen: Besuchen Sie den complexity zoo (`https://complexityzoo.uwaterloo.ca/Complexity_Zoo`)

# Beispiele

- $\{a^n b^n c^n \mid n \geq 1\} \in \mathbf{L}$
- $\{w\$w \mid w \in \Sigma^*\} \in \mathbf{L}$
- Die Menge PRIM $= \{p \in 1\{0,1\}^* \mid p$ ist Binärcodierung einer Primzahl $\}$ ist in $\mathrm{DSPACE}(n)$.

  Agrawal, Kayal und Saxena haben 2002 gezeigt, dass PRIM $\in \mathbf{P}$ gilt, siehe z. B. das Buch *Primality Testing in Polynomial Time* von M. Dietzfelbinger, Springer 2004.

  Beachte: In PRIM wird für eine binär kodierte Zahl gefragt, ob es sich um eine Primzahl handelt. Für eine unär kodierte Zahl $n$ (repräsentiert durch $n$ viele $a$'s) kann man sehr leicht in Polynomialzeit überprüfen, ob es sich um eine Primzahl handelt.

# Varianten algorithmischer Probleme

**Beispiel 1:** Traveling Salesman Problem (TSP)

Ein Reisender will eine gegebene Anzahl von Städten besuchen, ohne dabei an einem Ort zweimal vorbeizukommen, und er will dabei den kürzesten Weg nehmen. Das Wegenetz kann als gerichteter Graph und die Wegstrecken als Gewichte auf den Kanten des Graphen aufgefasst werden. Die Knoten stellen die Städte dar.

Sei $G = (V, E, \gamma : E \to \mathbb{N})$ ein gerichteter Graph mit Knotenmenge $V = \{1, \dots, n\}$, Kantenmenge $E \subseteq V \times V$ und den Kantengewichten $\gamma(e) \in \mathbb{N} \setminus \{0\}$ für alle $e \in E$.

Ein Rundweg $W$ ist gegeben durch eine Folge $W = (x_0, \dots, x_n)$, $x_0 = x_n$, $x_i \neq x_j$ für $1 \leq i < j \leq n$ und $(x_{i-1}, x_i) \in E$ für $1 \leq i \leq n$.

Die Kosten $\gamma(W)$ des Rundweges $W$ sind durch die Summe der Kantengewichte gegeben: $\gamma(W) = \sum_{i=1}^{n} \gamma(x_{i-1}, x_i)$.

# Varianten algorithmischer Probleme

(A) Entscheidungsvariante:

**Eingabe:** $G = (V, E, \gamma : E \to \mathbb{N})$ und ein $k \geq 0$.

**Frage:** Existiert Rundweg mit Kosten $\leq k$? D.h., existiert ein Weg der alle Knoten genau einmal besucht und dessen Kosten höchstens $k$ sind?

(B) Berechnungsvariante:

**Eingabe:** $G = (V, E, \gamma : E \to \mathbb{N})$ und ein $k \geq 0$.

**Ziel:** Falls Rundweg $W$ mit $\gamma(W) \leq k$ existiert, berechne solche ein $W$.

(C) Optimierungsproblem:

**Eingabe:** $G = (V, E, \gamma : E \to \mathbb{N})$.

**Ziel:** Berechne kostenoptimalen Rundweg, falls ein Rundweg existiert.

Die Inputgröße ist (bis auf einen konstanten Faktor)
$|V|^2 + \sum_{e \in E}(\lfloor \log \gamma(e) \rfloor + 1) + \lceil \log(k) \rceil$ für (A) und (B) bzw.
$|V|^2 + \sum_{e \in E}(\lfloor \log \gamma(e) \rfloor + 1)$ für (C).

# Varianten algorithmischer Probleme

Aus praktischer Sicht ist Variante (C) (Optimierungsproblem) am wichtigsten.

Aber: (A) in Polynomialzeit lösbar $\Longrightarrow$ (C) in Polynomialzeit lösbar.

**Beweis:**

1. Schritt:
Überprüfe, ob überhaupt ein Rundweg existiert:

Rufe hierzu (A) mit $k_{\max} = |V| \cdot \max\{\gamma(e) \mid e \in E\}$ auf.

Beachte: Es existiert ein Rundweg genau dann, wenn ein Rundweg mit Kosten $\leq k_{\max}$ existiert.

Im folgenden nehmen wir an, dass ein Rundweg existiert.

# Varianten algorithmischer Probleme

2. Schritt:

Berechne $k_{opt} = \min\{\gamma(W) \mid W \text{ ist ein Rundweg}\}$ mittels binärer Suche:

**FUNCTION** $k_{opt}$
  $k_{\text{min}} := 1$ (oder alternativ $k_{\text{min}} := |V|$)
  **while** $k_{\text{min}} < k_{\text{max}}$ **do**
    $k_{\text{mitte}} := k_{\text{min}} + \lceil \frac{k_{\text{max}} - k_{\text{min}}}{2} \rceil$
    **if** $\exists$ Rundweg $W$ mit $\gamma(W) \leq k_{\text{mitte}}$ **then** $k_{\text{max}} := k_{\text{mitte}}$
    **else** $k_{\text{min}} := k_{\text{mitte}} + 1$
    **endif**
  **endwhile**
  **return** $k_{\text{min}}$
**ENDFUNC**

Beachte: Die Anzahl der Durchläufe durch die **while**-Schleife ist beschränkt durch $\log_2(k_{\text{max}}) = \log_2(|V| \cdot \max\{\gamma(e) \mid e \in E\}) = \log_2(|V|) + \log_2(\max\{\gamma(e) \mid e \in E\}) \leq$ Inputgröße.

# Varianten algorithmischer Probleme

3. Schritt:
Berechne optimalen Rundweg wie folgt:

**FUNCTION** optimaler Rundweg
  Sei $e_1, e_2, \ldots, e_m$ beliebige Auflistung von $E$
  $G_0 := G$
  **for** $i := 1$ **to** $m$ **do**
    **if** $\exists$ Rundweg $W$ in $G_{i-1} \setminus \{e_i\}$ mit $\gamma(W) \leq k_{opt}$ **then**
      $G_i := G_{i-1} \setminus \{e_i\}$
    **else**
      $G_i := G_{i-1}$
    **endif**
  **endfor**
  **return** $G_m$
**ENDFUNC**

# Varianten algorithmischer Probleme

**Behauptung:** Für alle $i \in \{0, \ldots, m\}$ gilt:

1. In $G_i$ existiert ein Rundweg $W$ mit $\gamma(W) = k_{opt}$.
2. Jeder Rundweg $W$ in $G_i$ mit $\gamma(W) = k_{opt}$ benutzt alle Kanten aus $\{e_1, \ldots, e_i\} \cap E[G_i]$ ($E[G_i]$ = Menge der Kanten von $G_i$).

**Beweis:**

1. Folgt sofort durch Induktion über $i$.
2. Angenommen es gibt einen Rundweg $W$ in $G_i$ mit $\gamma(W) = k_{opt}$ sowie eine Kante $e_j$ $(1 \leq j \leq i)$ mit:
   - $e_j$ gehört zum Graphen $G_i$
   - $e_j$ gehört nicht zum Weg $W$

   $W$ ist auch ein Rundweg in $G_{j-1}$. $\Rightarrow$
   $W$ ist ein Rundweg in $G_{j-1} \setminus \{e_j\}$. $\Rightarrow$
   $e_j$ gehört nicht zu $G_j$ und damit nicht zu $G_i$. Widerspruch!

Konsequenz: $G_m$ hat einen Rundweg $W$ mit $\gamma(W) = k_{opt}$ und jede Kante von $G_m$ gehört zu $W$.

$\Rightarrow G_m = W$ $\square$

# Varianten algorithmischer Probleme

**Beispiel 2:** Vertex Cover (VC)

Sei $G = (V, E)$ ein ungerichteter Graph (d.h. $E \subseteq \binom{V}{2}$).
Eine Teilmenge $C \subseteq V$ ist eine Knotenüberdeckung von $G$ falls für jede Kante $\{u, v\} \in E$ gilt: $\{u, v\} \cap C \neq \emptyset$

(A) Entscheidungsvariante:

**Eingabe:** $G = (V, E)$ und ein $k \geq 0$.

**Frage:** Hat $G$ Knotenüberdeckung $C$ mit $|C| \leq k$?

(B) Berechnungsvariante:

**Eingabe:** $G = (V, E)$ und ein $k \geq 0$.

**Ziel:** Falls eine Knotenüberdeckung $C$ mit $|C| \leq k$ existiert, berechne ein solches $C$.

(C) Optimierungsproblem:

**Eingabe:** $G = (V, E)$.

**Ziel:** Berechne eine möglichst kleine Knotenüberdeckung von $G$.

# Varianten algorithmischer Probleme

Wieder gilt: (A) in Polynomialzeit lösbar $\Longrightarrow$ (C) in Polynomialzeit lösbar.

Zeigen Sie dies als Übung.

# Das Grapherreichbarkeitsproblem

Das Grapherreichbarkeitsproblem (GAP — für graph accessibility problem) ist ein zentrales Entscheidungsproblem in der Komplexitätstheorie:

**INPUT:** Ein gerichteter Graph $G = (V, E)$ und zwei Knoten $s, t \in V$.
**FRAGE:** Existiert in $G$ ein Pfad von $s$ nach $t$?

GAP gehört zur Klasse **P**: GAP kann in Zeit $\mathcal{O}(|V|)$ mittels Breitensuche gelöst werden.

Verschärfung: GAP gehört zur Klasse **NL** (wir zeigen noch $\mathbf{NL} \subseteq \mathbf{P}$):

**FUNCTION** Grapherreichbarkeit
  **var** $v := s$
  **while** $v \neq t$ **do**
    wähle einen Knoten $w \in V$ mit $(v, w) \in E$
    $v := w$
  **endwhile**
  **return** „Es gibt einen Pfad von $s$ nach $t$."
**ENDFUNC**

# Das Grapherreichbarkeitsproblem

Dieser nichtdeterministische Algorithmus kann leicht auf einer nichtdeterministischen Turingmaschine implementiert werden.

Warum benötigt obiger Algorithmus nur logarithmischen Platz?

- Zu jedem Zeitpunkt muss sich der Algorithmus nur einen Knoten $v \in V$ merken.
- Wenn es $n$ Knoten gibt, so können die Knoten mit den Zahlen $1, \ldots, n$ identifiziert werden. Die Variable $v$ benötigt somit $\log_2(n) = \log_2(|V|)$ viele Bits.

Bemerkungen:

- Aus dem Satz von Savitch (kommt noch) wird folgen: $\mathrm{GAP} \in \mathrm{DSPACE}(\log^2(n))$.
- Omer Reingold konnte 2004 zeigen: Das Grapherreichbarkeitsproblem für ungerichtete Graphen (UGAP) gehört zur Klasse **L**, siehe `http://www.wisdom.weizmann.ac.il/~reingold/publications/sl.ps`

# Teil 2: Beziehungen zwischen den Komplexitätsklassen

Die Beweise für die Beziehungen in diesem Abschnitt finden sich in Standardlehrbüchern (z. B. Hopcroft, Ullman; *Introduction to Automata Theory, Languages and Computation*, Addison Wesley 1979).
Wir werden Beweise hier nur andeuten.

Für eine Funktion $f : \mathbb{N} \to \mathbb{N}$ sei $\mathrm{DTIME}(\mathcal{O}(f)) = \bigcup_{c \in \mathbb{N}} \mathrm{DTIME}(c \cdot f)$, und analog für $\mathrm{NTIME}, \mathrm{DSPACE}, \mathrm{NSPACE}$.

## Satz 4

Sei $f : \mathbb{N} \to \mathbb{N}$.

1. Sei $X \in \{D, N\}$, dann gilt $\mathrm{XSPACE}(\mathcal{O}(f)) = \mathrm{XSPACE}_{1\text{-Band}}(f)$.
2. $\exists \epsilon > 0\ \forall n : f(n) \geq (1 + \epsilon)n \quad \Longrightarrow \quad \mathrm{DTIME}(\mathcal{O}(f)) = \mathrm{DTIME}(f)$.
3. $\mathrm{NTIME}(\mathcal{O}(f)) = \mathrm{NTIME}(f)$.
4. $\mathrm{DTIME}(n) \subsetneq \mathrm{DTIME}(\mathcal{O}(n))$.

Der Punkt 1 kombiniert Bandreduktion mit Bandkompression.
Die Punkte 2 und 3 bezeichnet man als Zeitkompression.

# Der Satz von Hennie und Stearns (1966)

Der Satz von Hennie und Stearns ist ein Bandreduktionssatz für Zeitkomplexitätsklassen.

**Satz 5**

Sei $k \geq 1$ und gelte $\exists \varepsilon > 0 \; \forall n : f(n) \geq (1 + \varepsilon)n$. Dann gilt: $\mathrm{DTIME}_{k\text{-Band}}(f) \subseteq \mathrm{DTIME}_{2\text{-Band}}(f \cdot \log(f))$.

# $\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$

**Satz 6**

Gelte $\forall n : f(n) \geq n$. Dann gilt $\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$.

**Beweis:** Zu zeigen ist nur $\mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$.

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$ eine nichtdeterministische $f$-zeitbeschränkte Turingmaschine.

Eine Eingabe $w \in \Sigma^*$ der Länge $n$ wird genau dann von $M$ akzeptiert, falls es ein Protokoll $\tilde{u}_1 \tilde{u}_2 \cdots \tilde{u}_m$ gibt mit $m \leq f(n)$ und

$$\mathsf{Start}(w) \vdash_{\tilde{u}_1} c_1 \vdash_{\tilde{u}_2} c_2 \cdots \vdash_{\tilde{u}_m} c_m \in \mathrm{Accept}_M .$$

Wir durchsuchen (z. B. in längenlexikographischer Reihenfolge) alle Protokolle der Länge höchstens $f(n)$ und überprüfen, ob solch ein Protokoll zu einer akzeptierenden Konfiguration führt.

# $\mathrm{DTIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{DSPACE}(f)$

Beachte:

- Jede von $\mathsf{Start}(w)$ erreichbare Konfiguration benötigt Platz $f(n)$.
- Ein Protokoll der Länge höchstens $f(n)$ kann in Platz $\mathcal{O}(f(n))$ gespeichert werden.

Gesamter Platzbedarf: $\mathcal{O}(f) + \mathcal{O}(f) = \mathcal{O}(f)$.

**FUNCTION** Protokollsuche($w$)
    **for all** Protokolle $\tilde{u}_1\tilde{u}_2\cdots\tilde{u}_m$ mit $m \le f(|w|)$ **do**
        Berechne die eindeutige Konfiguration $c_m$ (falls existent) mit
        $\mathsf{Start}(w) \vdash_{\tilde{u}_1} c_1 \vdash_{\tilde{u}_2} c_2 \cdots \vdash_{\tilde{u}_m} c_m$
        **if** $c_m \in \mathrm{Accept}_M$ **then**
            **return** $M$ akzeptiert $w$
    **endfor**
    **return** $M$ akzeptiert $w$ nicht
**ENDFUNC**

**Satz 7**

Sei $f(n) \in \Omega(\log(n))$. Dann gilt:
$\mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)})$.

**Beweis:** Zu zeigen ist nur $\mathrm{NSPACE}(f) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)})$.

Sei $M$ eine $f$-platzbeschränkte nichtdeterministische Turingmaschine und $w \in \Sigma^*$ eine Eingabe der Länge $n$.

Wegen Lemma 3 ist die Anzahl der von $\mathrm{Start}(w)$ erreichbaren Konfigurationen durch $2^{\mathcal{O}(f(n))}$ beschränkt.

Wir berechnen jetzt die Menge $R$ aller von $\mathrm{Start}(w)$ erreichbaren Konfigurationen:

**FUNCTION** Menge-der-erreichbaren-Konfigurationen
  **var** $R := \{\mathsf{Start}(w)\}$
  **while** $\exists$ Konfigurationen $\alpha, \beta : \alpha \in R \wedge \beta \notin R \wedge \alpha \vdash_M \beta$ **do**
    $R := R \cup \{\beta\}$
  **endwhile**
  **if** $\mathrm{Accept}_M \cap R \neq \varnothing$ **then return** $M$ akzeptiert $w$
**ENDFUNC**

Zeitbedarf:

- $R$ enthält maximal $2^{\mathcal{O}(f(n))}$ Konfigurationen der Länge $\leq f(n)$.
- Der Test $\exists$ Konfigurationen $\alpha, \beta : \alpha \in R \wedge \beta \notin R \wedge \alpha \vdash_M \beta$ kann somit in Zeit $2^{\mathcal{O}(f(n))} \cdot 2^{\mathcal{O}(f(n))} \cdot \mathcal{O}(f(n)) \subseteq 2^{\mathcal{O}(f(n))}$ implementiert werden.
- Gesamter Zeitbedarf: $2^{\mathcal{O}(f(n))}$ $\square$

# Folgerungen

- $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathrm{DTIME}(2^{\mathcal{O}(\log(n))}) = \mathbf{P}$
- $\mathbf{CS} = \mathbf{LBA} = \mathrm{NSPACE}(n) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(n)})$
  Hierbei bezeichnet **CS** die Klasse der kontextsensitiven und **LBA** die Klasse der durch linear beschränkte Automaten akzeptierten Sprachen.
- $\mathrm{DSPACE}(n^2) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(n^2)})$

# Der Satz von Savitch (1970)

**Satz 8**

Sei $s \in \Omega(\log(n))$. Dann gilt $\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2)$.

Wir beweisen den Satz von Savitch unter der Annahme, dass die Funktion $s$ platzkonstruierbar ist:

- Eine Funktion $s : \mathbb{N} \to \mathbb{N}$ mit $s \in \Omega(\log(n))$ heißt platzkonstruierbar, falls es eine deterministische $s$-platzbeschränkte Turingmaschine gibt, die bei Eingabe $a^n$ (d.h. $n$ ist unär kodiert) $a^{s(n)}$ auf dem Ausgabeband berechnet.
- Eine Funktion $t : \mathbb{N} \to \mathbb{N}$ mit $t \in \Omega(n)$ heißt zeitkonstruierbar, falls es eine deterministische Turingmaschine gibt, die bei Eingabe $a^n$ nach genau $t(n)$ Schritten hält.

**Beweis des Satzes von Savitch:**

Sei $M$ eine $s$-platzbeschränkte nichtdeterministische Turingmaschine und $w$ eine Eingabe für $M$.

Sei $\mathsf{Conf}(M, w)$ die Menge aller Konfigurationen $\alpha$ mit:

- Auf dem Eingabeband steht die Eingabe $w$.
- $|\alpha| \leq s(|w|)$.

Dann enthält $\mathsf{Conf}(M, w)$ alle von $\mathrm{Start}(w)$ aus erreichbaren Konfigurationen.

O.B.d.A. enthalte $\mathrm{Accept}_M$ nur eine einzige Konfiguration $\alpha_f$, die von $\mathrm{Start}(w)$ aus erreichbar sein kann.

Für $\alpha, \beta \in \mathsf{Conf}(M, w)$ und $i \geq 0$ definieren wir:

$$\mathsf{Reach}(\alpha, \beta, i) \iff \exists k \leq 2^i, \alpha_0, \alpha_1, \ldots, \alpha_k \in \mathsf{Conf}(M, w) : \alpha_0 = \alpha, \alpha_k = \beta, \bigwedge_{i=1}^{k} \alpha_{i-1} \vdash_M \alpha_i$$

Wegen Lemma 3 und $s(n) \in \Omega(\log(n))$ existiert eine Konstante $c$, so dass für alle Eingaben $w$ gilt:

$$w \in L(M) \quad \Longleftrightarrow \quad \mathrm{Reach}(\mathrm{Start}(w), \alpha_f, c \cdot s(|w|)).$$

Unser Ziel ist, das Prädikat $\mathrm{Reach}(\alpha, \beta, i)$ für $\alpha, \beta \in \mathsf{Conf}(M, w)$ und $0 \leq i \leq c \cdot s(|w|)$ in Platz $\mathcal{O}(s^2)$ auf einer deterministischen Maschine zu berechnen.

Für $i > 0$ verwenden wir folgendes Rekursionschemata:

$$\begin{aligned}\mathrm{Reach}(\alpha, \beta, i) \quad \Longleftrightarrow \quad \exists \gamma \in \mathsf{Conf}(M, w) : & \mathrm{Reach}(\alpha, \gamma, i-1) \wedge \\ & \mathrm{Reach}(\gamma, \beta, i-1).\end{aligned}$$

Umsetzung durch einen deterministischen Algorithmus:

# Beweis des Satzes von Savitch

**FUNCTION** Reach$(\alpha, \beta, i)$ (wobei $\alpha, \beta \in \mathsf{Conf}(M, w)$ und $i \le c \cdot s(|w|)$)

  **var** $b := \text{FALSE}$

  **if** $i = 0$ **then**

    $b := [(\alpha = \beta) \ \lor \ (\alpha \vdash_M \beta)]$

  **else**

    **forall** $\gamma \in \mathsf{Conf}(M, w)$ **do**

      **if not** $b$ **and** Reach$(\alpha, \gamma, i-1)$ **then**

        $b :=$ Reach$(\gamma, \beta, i-1)$

      **endif**

    **endfor**

  **endif**

  **return** $b$

**ENDFUNC**

# Beweis des Satzes von Savitch

**Behauptung:** Es gibt eine Konstane $\varrho$, so dass ein Aufruf von Reach$(\alpha, \beta, i)$ Platz $\varrho \cdot (i+1) \cdot s(|w|)$.

Wir beweisen die Behauptung durch Induktion über $i \geq 0$:

$i = 0$: Die Bedingung $[(\alpha = \beta) \ \vee \ (\alpha \vdash_M \beta)]$ kann offensichtlich in Platz $\varrho \cdot s(|w|)$ für eine geeignte Konstante $\varrho$ überprüft werden.

$i > 0$: Der 1. Aufruf Reach$(\alpha, \gamma, i-1)$ benötigt nach Induktion Platz $\varrho \cdot i \cdot s(|w|)$. Das gleiche gilt für den 2. Aufruf Reach$(\gamma, \beta, i-1)$.

Beachte: Beim 2. Aufruf Reach$(\gamma, \beta, i-1)$ kann der Platz, der beim 1. Aufruf Reach$(\alpha, \gamma, i-1)$ benötigt wurde, wiederverwendet werden.

Zusätzlich wird noch Speicherplatz $3 \cdot s(|w|) + c \cdot s(|w|) \leq \varrho \cdot s(|w|)$ (falls $\varrho \geq c+3$) für die Konfigurationen $\alpha, \beta, \gamma$ und die Zahl $i$ (unär kodiert) benötigt. Dies beweist die Behauptung.

# Beweis des Satzes von Savitch

Um $w \in L(M)$ zu entscheiden, rufen wir Reach$(\mathrm{Start}(w), \alpha_f, c \cdot s(|w|))$ auf.

Beachte: Hierzu müssen wir $s(|w|)$ (unär kodiert) berechnen, was möglich ist, da $s$ nach Annahme platzkonstruierbar ist.

Gesamter Platzbedarf: $\mathcal{O}(c \cdot s(|w|) \cdot s(|w|)) = \mathcal{O}(s(|w|)^2)$. □

# Bemerkungen zum Satz von Savitch

Der Satz von Savitch besagt, dass eine nichtdeterministische platzbeschränkte Turingmaschine unter quadratischem Mehraufwand deterministisch simuliert werden kann. Diese platzeffiziente Simulation wird durch einen extremen Mehraufwand an Rechenzeit realisiert.

**Übung:** Wieviel Zeit benötigt der Algorithmus im obigen Beweis, um $w \in L(M)$ zu entscheiden?

Um sich von der Forderung der Platzkonstruierbarkeit von $s$ zu befreien, zeige man mit dem Ansatz von Savitch, dass sich der tatsächliche Platzbedarf einer $s$-platzbeschränkten nichtdeterministischen Turingmaschine in $\mathrm{DSPACE}(s^2)$ berechnen lässt.

# Folgerungen aus dem Satz von Savitch

**Satz 9**

GAP gehört zu $\mathrm{DSPACE}(\log^2(n))$.

Folgt unmittelbar aus GAP $\in$ **NL** und dem Satz von Savitch.

**Satz 10**

$\mathbf{PSPACE} = \bigcup_{k\geq 1} \mathrm{DSPACE}(n^k) = \bigcup_{k\geq 1} \mathrm{NSPACE}(n^k)$

Folgt aus $\mathrm{NSPACE}(n^k) \subseteq \mathrm{DSPACE}(n^{2k})$

**Satz 11 (Platzhierarchiesatz)**

Seien $s_1, s_2 : \mathbb{N} \to \mathbb{N}$ Funktionen, $s_1 \notin \Omega(s_2)$, $s_2 \in \Omega(\log(n))$ und $s_2$ sei platzkonstruierbar. Dann gilt $\mathrm{DSPACE}(s_2) \setminus \mathrm{DSPACE}(s_1) \neq \emptyset$.

**Bemerkungen:**

- $s_1 \notin \Omega(s_2)$ bedeutet $\forall \epsilon > 0\ \exists$ unendlich viele $n$ mit $s_1(n) < \epsilon \cdot s_2(n)$.

  Seien etwa $s_1(n) = n$ und $s_2(n) = \begin{cases} n^2, & \text{falls } n \text{ gerade} \\ \log n, & \text{sonst} \end{cases}$,

  dann gilt: $s_2 \notin \Omega(s_1)$, $s_1 \notin \Omega(s_2)$.
- Aus dem Platzhierarchiesatz folgt etwa:

$$\begin{aligned} \mathbf{L} &\subsetneq \mathrm{DSPACE}(\log^2(n)) \subsetneq \mathrm{DSPACE}(n) \\ &\subseteq \mathrm{NSPACE}(n) \subsetneq \mathrm{DSPACE}(n^{2,1}) \subsetneq \mathbf{PSPACE} \end{aligned}$$

# Beweis des Platzhierarchiesatzes

Der Beweis des Platzhierarchiesatzes ist ähnlich zum Beweis für die Unentscheidbarkeit des Halteproblems: Diagonalisierung

Wähle zunächst eine geeignete binäre Kodierung von deterministischen 1-Band Turingmaschinen, die eine effiziente Simulation erlaubt (wir sagen gleich, was das bedeutet).

Jedes Wort $x \in \{0,1\}^*$ soll als Kodierung einer Turingmaschine $M_x$ interpretiert werden können (für nicht "wohlgeformtes" $x$ kodiert $x$ eine Default-Turingmaschine).

**Wichtige Konvention:** Für alle $x \in \{0,1\}^*$ und $k \in \mathbb{N}$ gelte $M_x = M_{0^k x}$, d. h. $x$ und $0^k x$ kodieren die gleiche Maschine.

**Folgerung:** jede Turingmaschine hat eine Kodierung in fast jeder Länge.

**Ziel:** eine deterministische $s_2$-platzbeschränkte Turingmaschine $M$ mit $L(M) \notin \mathrm{DSPACE}(s_1)$.

$s_2 \in \Omega(\log(n)) \rightsquigarrow \exists \delta > 0 \; \exists m \; \forall n \geq m : \log_2(n) \leq \delta \cdot s_2(n)$

# Beweis des Platzhierarchiesatzes

Wir beginnen mit einer (deterministischen) universellen Turingmaschine $U$.

Diese erhält als Eingabe die Binärkodierung $x$ einer 1-Band Turingmaschine $M_x$ sowie eine Eingabe $w \in \{0,1\}^*$ für $M_x$.

$U$ simuliert $M_x$ auf der Eingabe $w$.

Wir können die Kodierung von Turingmaschinen und $U$ so wählen, dass es für jedes $x \in \{0,1\}^*$ eine nur von $M_x$ abhängige Konstante $k$ gibt mit:

Wenn $M_x$ $s$-platzbeschränkt ist, dann hat $U$ bei Eingabe $\langle x, w\rangle$ Platzbedarf höchstens $k \cdot s(|w|) + \frac{1}{1+\delta} \log_2(|w|)$.

Wegen Lemma 3 existiert eine Konstante $c$, so dass es $\leq n \cdot c^m$ Konfigurationen von $U$ mit Platzbedarf $m$ und einer festen Eingabe der Länge $n$ gibt.

Unsere Maschine $M$ arbeitet für eine Eingabe $y = 0^\ell x$ (wobei $x$ nicht mit 0 beginnt) der Länge $n = |y|$ wie folgt:

# Beweis des Platzhierarchiesatzes

1. Markiere Platz $s_2(n)$ auf den Arbeitsbändern und installiere einen Zähler $C$ mit initialen Wert $2n \cdot c^{s_2(n)} + 1$ (benötigt Platz $\leq s_2(n)$ nach ausreichender Bandkompression).

   Dies geht, da $s_2$ platzkonstruierbar ist.
2. Sobald im folgenden der markierte Platz verlassen wird, stoppt $M$ im nicht-akzeptierenden Zustand $q_N$.

   Damit ist $M$ $s_2$-platzbeschränkt.
3. Führe die universelle Maschine $U$ mit Eingabe $\langle y, y \rangle$ (hat Länge $2n$) aus und setze $C := C - 1$ nach jeder Transition von $U$.
4. Falls $C$ den Wert 0 erreicht hat und die Simulation von $M_y = M_x$ noch nicht terminiert hat, muss $U$ in einen Zyklus gelangt sein.

   Dies bedeutet, dass $M_x$ auf $y$ nicht terminiert.

   $M$ akzeptiert dann die Eingabe $y$.
5. Falls die Simulation vorher terminiert, akzeptiert $M$ die Eingabe $y$ genau dann, wenn $M_x$ die Eingabe $y$ nicht akzeptiert.

# Beweis des Platzhierarchiesatzes

**Behauptung:** $L(M) \notin \mathrm{DSPACE}(s_1)$

**Beweis durch Widerspruch:** Angenommen, es gilt $L(M) \in \mathrm{DSPACE}(s_1)$.

Sei $M'$ eine $s_1$-platzbeschränkte deterministische 1-Band (existiert!) Turingmaschine mit $L(M') = L(M)$.

Sei $M' = M_x$.

Dann simuliert $U$ die Maschine $M' = M_x$ auf einer Eingabe der Länge $n$ in Platz $k \cdot s_1(n) + \frac{1}{1+\delta} \log_2(n)$.

Hierbei ist $k$ eine Konstante, die nur von $M'$ (aber nicht von $n$) abhängt.

Da $s_1 \notin \Omega(s_2)$ existiert ein $n \geq |x|$ mit

$$k(1+\delta) \cdot s_1(n) + \log_2(n) \leq s_2(n) + \log_2(n) \leq (1+\delta) \cdot s_2(n)$$

und somit

$$k \cdot s_1(n) + \frac{1}{1+\delta} \log_2(n) \leq s_2(n).$$

Während der Simulation von $M' = M_x = M_{0^{n-|x|}x}$ auf der Eingabe $0^{n-|x|}x$ (der Länge $n$) wird also der im 1. Schritt von der Maschine $M$ markierte Platz nicht verlassen.

Also:

$$\begin{aligned} 0^{n-|x|}x \in L(M) \quad &\iff \quad M \text{ akzeptiert } 0^{n-|x|}x \\ &\iff \quad M_x \text{ akzeptiert } 0^{n-|x|}x \text{ nicht} \\ &\iff \quad M' \text{ akzeptiert } 0^{n-|x|}x \text{ nicht} \\ &\iff \quad 0^{n-|x|}x \notin L(M') = L(M) \end{aligned}$$

$\square$

# Zeithierarchiesatz

Gemäß der Technik von Hennie und Stearns ist die Simulation von beliebig vielen Bändern zeiteffizient auf zwei Bändern möglich.

Analog zum Platzhierarchiesatz ergibt sich der deterministische Zeithierarchiesatz.

**Satz 12 (Deterministischer Zeithierarchiesatz (ohne Beweis))**

Seien $t_1, t_2 : \mathbb{N} \to \mathbb{N}$ Funktionen, $t_1 \cdot \log(t_1) \notin \Omega(t_2)$, $t_2 \in \Omega(n \log(n))$ und $t_2$ sei zeitkonstruierbar. Dann gilt $\mathrm{DTIME}(t_2) \setminus \mathrm{DTIME}(t_1) \neq \emptyset$ .

Als Folgerung hieraus ergibt sich:

$$\begin{aligned}\mathrm{DTIME}(\mathcal{O}(n)) \subsetneq \mathrm{DTIME}(\mathcal{O}(n^2)) \subsetneq \mathbf{P} \\ \subsetneq \mathrm{DTIME}(\mathcal{O}(2^n)) \subsetneq \mathrm{DTIME}(\mathcal{O}((2+\varepsilon)^n))\end{aligned}$$

# Der Satz von Borodin

Bei den oben erwähnten Hierarchiesätzen haben wir stets eine Konstruierbarkeitsvoraussetzung mitgeführt. Dies lässt sich nach dem folgenden Lückensatz nicht umgehen.

**Satz 13 (Satz von Borodin (1972))**

Sei $r$ eine totale, berechenbare, monotone Funktion, $r(n) \geq n$ für alle $n$. Dann existiert effektiv eine totale, berechenbare Funktion $s : \mathbb{N} \to \mathbb{N}$ mit der Eigenschaft $s(n) \geq n+1$ für alle $n$ und $\mathrm{DTIME}(s) = \mathrm{DTIME}(r \circ s)$.

**Bemerkungen:**

- Die Komposition $r \circ s$ ist definiert durch $r \circ s(x) = r(s(x))$.
- Dass die totale, berechenbare Funktion $s : \mathbb{N} \to \mathbb{N}$ effektiv existiert, bedeutet, dass man aus einer Turingmaschine, die $r$ berechnet, eine Turingmaschine, die $s$ berechnet, konstruieren kann.

# Beweis des Satzes von Borodin

Sei $M_1, M_2, \ldots$ eine Aufzählung aller deterministischen Turingmaschinen.

Sei $t_k(n) \in \mathbb{N} \cup \{\infty\}$ der tatsächliche maximale Zeitbedarf einer Rechnung von $M_k$ auf einer Eingabe der Länge kleiner oder gleich $n$.

Betrachte nun die Menge

$$N_n = \{t_k(n) \mid 1 \leq k \leq n\} \subseteq \mathbb{N} \cup \{\infty\}.$$

Diese Menge ist endlich, also existiert für jedes $n$ eine Zahl $s(n)$ mit

$$N_n \cap [s(n), r(s(n))] = \emptyset.$$

Ein $s(n)$, welches diese Bedingung erfüllt, wäre

$$s(n) = 1 + \max\{t_k(n) \mid 1 \leq k \leq n,\ t_k(n) < \infty\}.$$

Dieser Wert wird jedoch i. allg. zu groß (und auch nicht berechenbar) sein.

# Beweis des Satzes von Borodin

Einen passenden und berechenbaren Wert $s(n)$ finden wir bei Eingabe $n$ wie folgt:

**FUNCTION** $s(n)$
  $s := \max\{n+1, s(n-1)\}$
  **repeat**
    $s := s + 1$
  **until** $\forall k \leq n : [t_k(n) < s$ **or** $t_k(n) > r(s)]$
  **return** $s$
**ENDFUNC**

**Bemerkung:** Die Funktion $s$ ist berechenbar und wächst monoton.
Im allgemeinen kann $s$ jedoch nicht zeitkonstruierbar sein.

**Behauptung:** $\mathrm{DTIME}(s) = \mathrm{DTIME}(r \circ s)$

# Beweis des Satzes von Borodin

**Beweis der Behauptung:**

Da $r(n) \geq n$ für alle $n$, ist $\mathrm{DTIME}(s) \subseteq \mathrm{DTIME}(r \circ s)$ klar.

Sei nun $L \in \mathrm{DTIME}(r \circ s)$.

Sei $M_k$ eine $(r \circ s)$-zeitbeschränkte deterministische Turingmaschine mit $L = L(M_k)$.

Dann gilt: $\forall n : t_k(n) \leq r(s(n))$.

Für alle $n \geq k$ gilt daher nach Berechnung von $s$: $t_k(n) < s(n)$.

Damit gilt $L \in \mathrm{DTIME}(s)$, denn für alle Eingaben der Länge $< k$ (eine Konstante) kann eine Turingmaschine direkt nach Lesen der Eingabe (dies benötigt $n + 1 \leq s(n)$ Schritte) die richtige Antwort ausgeben.

$\square$

# Der Satz von Immerman und Szelepcsényi (1987)

Die Klassen $\mathrm{DTIME}(f)$ und $\mathrm{DSPACE}(f)$ sind unter Komplement abgeschlossen. Ob dies auch für Klassen $\mathrm{NSPACE}(f)$ gilt, war für lange Zeit offen.

Bereits 1964 stellte Kuroda die Frage, ob die Familie der kontextsensitiven Sprachen unter Komplementbildung abgeschlossen ist (2. LBA-Problem). Äquivalent: $\mathrm{NSPACE}(n) = \mathbf{Co}\mathrm{NSPACE}(n)$?

Nach über 20 Jahren wurde diese Frage unabhängig von R. Szelepcsényi und N. Immerman beantwortet:

**Satz 14 (Satz von Immerman und Szelepcsényi)**

Sei $f \in \Omega(\log(n))$ monoton. Dann gilt $\mathrm{NSPACE}(f) = \mathbf{Co}\mathrm{NSPACE}(f)$.

**Beweismethode:** Induktives Zählen

Sei $M$ eine nichtdeterministische $f$-platzbeschränkte 1-Band Turingmaschine und $w \in \Sigma^*$ ein Eingabewort der Länge $n$.

**Ziel:** Überprüfe nichtdeterministisch in Platz $\mathcal{O}(f(n))$, ob $w \notin L(M)$ gilt.

O.B.d.A. sei $\alpha_0$ die einzige akzeptierende Konfiguration; etwa $\alpha_0 = (q_J, 1, \square, 1)$ (insbesondere $|\alpha_0| = 1$).

Wir benötigen eine Auflistung $\alpha_0 \prec \alpha_1 \prec \alpha_2 \prec \cdots$ aller Konfigurationen von $M$ mit Eingabe $w$, so dass gilt:

- $\alpha_0$ ist die kleinste Konfiguration bezüglich $\prec$.
- $\alpha \prec \alpha'$ impliziert $|\alpha| \leq |\alpha'|$.
- $\alpha \prec \alpha'$ kann in Platz $|\alpha| + |\alpha'|$ überprüft werden.

# Beweis des Satzes von Immerman und Szelepcsényi

Wir können $\prec$ z. B. wie folgt definieren, wobei $\alpha = (q, i, u, j)$, $\alpha' = (q', i', u', j')$ Konfigurationen von $M$ mit Eingabe $w$ sind:

- Wenn $|u| < |u'|$, dann $\alpha \prec \alpha'$.
- Wenn $|u| = |u'|$ und $u <_{\text{lex}} u'$, dann $\alpha \prec \alpha'$.
- Wenn $u = u'$ und $j < j'$, dann $\alpha \prec \alpha'$.
- Wenn $u = u'$, $j = j'$ und $i < i'$, dann $\alpha \prec \alpha'$.
- Wenn $u = u'$, $j = j'$, $i = i'$ und $q < q'$, dann $\alpha \prec \alpha'$.

  Hierbei fixieren wir eine beliebige Ordnung $\leq$ auf der Zustandsmenge von $M$, wobei $q_J$ der kleinste Zustand ist.

Sei $k \geq 0$:

$$\begin{aligned}
R(k) &= \{\alpha \mid \exists i \leq k : \mathrm{Start}(w) \vdash_M^i \alpha\} \\
r(k) &= |R(k)| \quad \text{(Anzahl der von } \mathrm{Start}(w) \text{ in } \leq k \text{ Schritten} \\
&\qquad\qquad \text{erreichbaren Konfigurationen)} \\
r(*) &= \max\{r(k) \mid k \geq 0\} \\
&\quad \text{(Anzahl der von } \mathrm{Start}(w) \text{ erreichbaren Konfigurationen)}
\end{aligned}$$

**Beachte:** Nach Lemma 3 gilt

$$r(k) \leq r(*) \in 2^{\mathcal{O}(f(n))}.$$

Da $f$ nicht platzkonstruierbar sein muss, benötigen wir noch den Wert

$$m(k) = \max\{|\alpha| \mid \alpha \in R(k)\}.$$

# Beweis des Satzes von Immerman und Szelepcsényi

Wir geben eine nichtdeterministische $\mathcal{O}(f(n))$-platzbeschränkte Maschine an mit:

- Wenn $w \notin L(M)$, dann wird die Maschine auf mindestens einem Berechnungspfad den korrekten Wert $r(*)$ ausgegeben. Auf anderen Berechnungspfaden kann die Maschine ohne Ergebnis abbrechen.
- Wenn $w \in L(M)$ wird auf allen Berechnungspfaden ohne Ergebnis abgebrochen.

# Beweis des Satzes von Immerman und Szelepcsényi

Berechnung von $r(*)$ unter der Annahme, dass $r(k+1)$ aus $r = r(k)$ mittels Funktion berechne-r$(k+1, r)$ in Platz $\mathcal{O}(f(n))$ berechnet werden kann:

**FUNCTION** $r(*)$
  $k := 0$
  $r := 1$  **(**$*$ speichert $r(k)$ $*$**)**
  **while true do**
    $r' :=$ berechne-r$(k+1, r)$
    **if** $r = r'$ **then return** $r$
    **else** $k := k+1$; $r := r'$
  **endwhile**
**ENDFUNC**

**Platzbedarf:** Wegen $r(*) \in 2^{\mathcal{O}(f(n))}$ wird zur Speicherung von $k, r$, und $r'$ Platz $\mathcal{O}(f(n))$ benötigt.

Die Berechnung der Funktion berechne-r$(k+1, r)$ erfolgt in drei Schritten.

**1. Schritt:** Berechne $m(k)$ aus $r = r(k)$ mittels Funktion berechne-m$(k, r)$

**FUNCTION** berechne-m$(k, r)$
 $\alpha := \alpha_0$;  $m := 1 (= |\alpha_0|)$
 **repeat** $r$ **times**
  berechne ein beliebiges $\alpha' \in R(k)$
  **if** $\alpha \prec \alpha'$ **then**
   $\alpha := \alpha'$
   $m := |\alpha'|$  (* $= \max\{m, |\alpha'|\}$ aufgr. der Ordnung *)
  **else**
   „FEHLER“ $\Rightarrow$ Programmabbruch
  **endif**
 **endrepeat**
 **return** $m$
**ENDFUNC**

**Beachte:**

- Wenn berechne-m$(k, r)$ nicht mit „Fehler“ abbricht (und $r = r(k)$ gilt), dann wird der korrekte Wert $m(k)$ berechnet.
- Wenn $\alpha_0 \in R(k)$ (und damit $w \in L(M)$), dann muss berechne-m$(k, r)$ mit „Fehler“ abbrechen, da in $R(k)$ nicht $r$ viele Konfigurationen zur Verfügung stehen, die alle echt größer als $\alpha_0$ sind.

  Insbesondere: Gilt $w \in L(M)$, so gibt es ein $k$, so dass berechne-m$(k, r)$ mit „Fehler“ abbricht. Dies führt dann zum Abbruch der Berechnung von $r(*)$.
- Falls $w \notin L(M)$, so hat der Algorithmus die Chance, nicht mit Fehler abzubrechen, dann wird also $m(k)$ korrekt berechnet.

**Platzbedarf von berechne-m**$(k,r)$**:** Wir müssen speichern:

- Konfigurationen $\alpha$, $\alpha'$ mit $|\alpha|, |\alpha'| \leq f(n)$.
- $m \leq f(n)$
- Binärzähler bis $k$ (um beliebiges $\alpha' \in R(k)$ nichtdeterministisch zu generieren)
- Binärzähler bis $r = r(k)$ (für **repeat** $r$ **times**).

Hierfür ist Platz $\mathcal{O}(f(n))$ ausreichend.

**2. Schritt:** Sei $\beta$ eine beliebige Konfiguration. Prozedur Reach$(r, k+1, \beta)$ testet nichtdeterministisch mit Hilfe des Werts $r = r(k)$, ob $\beta \in R(k+1)$ gilt oder nicht:

**FUNCTION** Reach$(r, k+1, \beta)$
  $\alpha := \alpha_0$
  **repeat** $r$ **times**
    berechne ein beliebiges $\alpha' \in R(k)$
    **if** $\alpha' \prec \alpha \vee \alpha' = \alpha$ **then** „FEHLER“ $\Rightarrow$ Programmabbruch
    **elseif** $\alpha' = \beta \ \vee \ \alpha' \vdash_M \beta$ **then return** true **(**$*$ d.h. $\beta \in R(k+1)$ $*$**)**
    **else** $\alpha := \alpha'$
    **endif**
  **endrepeat**
  **return** false **(**$*$ d.h. $\beta \notin R(k+1)$ $*$**)**
**ENDFUNC**

**Beachte:**

- Falls Reach$(r(k), k+1, \beta)$ nicht mit „FEHLER“ abbricht, wird eine korrekte Antwort ausgegeben.
- Gilt $w \notin L(M)$ (und damit $\alpha_0 \notin R(k)$), so hat Reach$(r(k), k+1, \beta)$ die Chance, nicht mit „FEHLER“ abzubrechen.

**Platzbedarf:** Wir müssen speichern:

- Konfigurationen $\alpha$, $\alpha'$ mit $|\alpha|, |\alpha'| \leq f(n)$.
- Binärzähler bis $k$ (um beliebiges $\alpha' \in R(k)$ nichtdeterministisch zu generieren)
- Binärzähler bis $r = r(k)$ (für **repeat** $r$ **times**).

Hierfür ist Platz $\mathcal{O}(f(n))$ ausreichend.

**3. Schritt:** Berechne $r(k+1)$ mittels der Funktion berechne-r$(k+1, r)$ aus $r = r(k)$.

**FUNCTION** berechne-r$(k+1, r)$
  $r' := 0$  (∗ ist am Ende $r(k+1)$ ∗)
  $m :=$ berechne-m$(k, r)$
  **forall** Konfigurationen $\beta$ mit $|\beta| \leq m+1$ **do**
    **if** Reach$(r, k+1, \beta)$ **then**
      $r' := r' + 1$
    **endif**
  **endforall**
  **return** $r'$
**ENDFUNC**

Wir betrachten nur Konfigurationen $\beta$ mit $|\beta| \leq m(k)+1$, da $m(k+1) \leq m(k)+1$.

# Beweis des Satzes von Immerman und Szelepcsényi

Eine erfolgreiche Berechnung von $r(*)$ ist genau dann möglich, wenn $w \notin L(M)$.

Man beachte dazu: Wenn $w \in L(M)$, so bricht *jede* Rechnung mit einer Fehlermeldung ab, da die Funktion $m(k)$, sobald sie die akzeptierende Konfiguration $\alpha_0$ in $R(k)$ vorfindet, nicht mehr erfolgreich durchlaufen werden kann.

Sowie also der Wert $r(*)$ berechnet wurde, kann $w$ als zum Komplement von $L(M)$ gehörig akzeptiert werden.

**Analyse des Platzbedarfs:** Aus den vorherigen Platzbetrachtungen folgt, dass der gesamte Algorithmus mit Platz $\mathcal{O}(f(n))$ auskommt. $\square$

# Die Translationstechnik

Der Translationssatzes erlaubt, aus einer Inklusion zwischen kleinen Komplexitätsklassen eine Inklusion zwischen großen Klassen abzuleiten.

Idee: Ausstopfen (Padding) von Sprachen.

Sei

- $L \subseteq \Sigma^*$ eine Sprache,
- $f : \mathbb{N} \to \mathbb{N}$ eine Funktion mit $\forall n \geq 0 : f(n) \geq n$, und
- $\$ \notin \Sigma$ ein neues Symbol.

Definiere die Sprache

$$\mathrm{Pad}_f(L) = \{w\$^{f(|w|)-|w|} \mid w \in L\} \subseteq (\Sigma \cup \{\$\})^*.$$

Beachte: jedem Wort aus $L$ der Länge $n$ wird ein Wort aus $L\$^*$ der Länge $f(n)$ zugeordnet.

# Der Translationssatz für Zeitklassen

**Satz 15 (Translationssatz für Zeitklassen)**

Seien $f, g$ monotone Funktionen mit $\forall n \geq 0 : f(n), g(n) \geq n$. Sei $g$ zeitkonstruierbar, und bei unärer Eingabe $1^n$ sei $1^{f(n)}$ in Zeit $g(f(n))$ berechenbar. Für $L \subseteq \Sigma^*$ gilt dann:

1. $\mathrm{Pad}_f(L) \in \mathrm{DTIME}(\mathcal{O}(g)) \iff L \in \mathrm{DTIME}(\mathcal{O}(g \circ f))$,
2. $\mathrm{Pad}_f(L) \in \mathrm{NTIME}(\mathcal{O}(g)) \iff L \in \mathrm{NTIME}(\mathcal{O}(g \circ f))$.

**Beweis:** Der Beweis wird für $\mathrm{DTIME}$ geführt; der Beweis für $\mathrm{NTIME}$ verläuft analog.

„$\Rightarrow$“: Sei $\mathrm{Pad}_f(L) \in \mathrm{DTIME}(\mathcal{O}(g))$ und $w \in \Sigma^*$ eine Eingabe, $|w| = n$.

Wir entscheiden $w \in L$ in Zeit $\mathcal{O}(g(f(n)))$ wie folgt:

1. Berechne das Wort $w\$^{f(|w|)-|w|}$ in der Zeit $g(f(|w|))$.
2. Teste in Zeit $\mathcal{O}(g(f(|w|)))$, ob $w\$^{f(|w|)-|w|} \in \mathrm{Pad}_f(L)$ gilt.

   Nach Definition gilt: $w\$^{f(|w|)-|w|} \in \mathrm{Pad}_f(L) \iff w \in L$.

# Beweis des Translationssatzes

„$\Leftarrow$“: Sei $L \in \mathrm{DTIME}(\mathcal{O}(g \circ f))$ und sei $x \in (\Sigma \cup \{\$\})^*$ eine Eingabe der Länge $m$.

Wir testen in Zeit $\mathcal{O}(g(m))$, ob $x \in \mathrm{Pad}_f(L)$ gilt, wie folgt:

1. Teste in Zeit $m \leq g(m)$ ob $x \in w\$^*$ für ein $w \in \Sigma^*$ gilt.
   Sei $x = w\$^{m-n}$ mit $w \in \Sigma^*$, $|w| = n$.
2. Berechne $1^{g(m)}$ in Zeit $g(m)$ ($g$ ist zeitkonstruierbar).
3. Teste nun wie folgt in Zeit $g(m)$, ob $f(n) = m$ gilt:
   Berechne $1^{f(n)}$ in Zeit $g(f(n))$. Falls die Maschine dabei mehr als $g(m)$ Schritte rechnen will, lehnen wir dabei ab (da $g$ monoton ist, gilt $g(f(n)) > g(m) \to f(n) > m$).
   Falls $1^{f(n)}$ berechnet wird, können wir $1^{f(n)}$ mit $1^m$ vergleichen.
   Sei nun $x = w\$^{f(n)-n}$.
4. Teste in Zeit $\mathcal{O}(g(f(n))) = \mathcal{O}(g(m))$, ob $w \in L$ gilt. □

# Der Translationssatz für Platzklassen

**Satz 16 (Translationssatz für Platzklassen (ohne Beweis))**

Sei $g \in \Omega(\log(n))$ platzkonstruierbar und $f(n) \geq n$ für alle $n \geq 0$. Auf eine unäre Eingabe $1^n$ sei die Binärdarstellung von $f(n)$ in Platz $g(f(n))$ berechenbar. Für $L \subseteq \Sigma^*$ gilt dann:

1. $\mathrm{Pad}_f(L) \in \mathrm{DSPACE}(g) \iff L \in \mathrm{DSPACE}(g \circ f)$,
2. $\mathrm{Pad}_f(L) \in \mathrm{NSPACE}(g) \iff L \in \mathrm{NSPACE}(g \circ f)$.

**Konsequenz:** Der Zusammenfall einer Hierarchie von Komplexitätsklassen ist am ehesten weit oben zu erwarten. Wollen wir Separationsresultate zeigen, so bestehen hierfür die besten Aussichten am unteren Ende einer Hierarchie.

# Folgerungen aus dem Translationssatz für Platzklassen

**Satz 17 (Korollar aus Translationssatz für Platzklassen)**

$\mathrm{DSPACE}(n) \neq \mathrm{NSPACE}(n) \implies \mathbf{L} \neq \mathbf{NL}$.

**Beweis:** Angenommen $\mathbf{L} = \mathbf{NL}$.

Sei $L \in \mathrm{NSPACE}(n) = \mathrm{NSPACE}(\log \circ \exp)$.

Dann gilt $\mathrm{Pad}_{\exp}(L) \in \mathrm{NSPACE}(\log(n)) = \mathbf{NL} = \mathbf{L} = \mathrm{DSPACE}(\log(n))$.

Aus dem Translationssatz für Platzklassen ergibt sich
$L \in \mathrm{DSPACE}(\log \circ \exp) = \mathrm{DSPACE}(n)$.

# Folgerungen aus den Translationssätzen

Mit Hilfe der Translationstechnik lässt sich in einigen Fällen die Verschiedenheit von Komplexitätsklassen nachweisen:

**Satz 18 (Korollar aus den Translationssätzen)**

$\mathbf{P} \neq \mathrm{DSPACE}(n)$.

**Beweis:** Wählen Sprache $L \in \mathrm{DSPACE}(n^2) \setminus \mathrm{DSPACE}(n)$ (existiert nach Platzhierarchiesatz) und die Padding–Funktion $f(n) = n^2$.

Dann gilt $\mathrm{Pad}_f(L) \in \mathrm{DSPACE}(n)$.

Wäre nun $\mathrm{DSPACE}(n) = \mathbf{P}$, so wäre $\mathrm{Pad}_f(L) \in \mathrm{DTIME}(n^k)$ für ein $k \geq 1$ und $L \in \mathrm{DTIME}(\mathcal{O}(n^{2k})) \subseteq \mathbf{P} = \mathrm{DSPACE}(n)$.

Dies ist ein Widerspruch.

**Bemerkung:**

- Insbesondere gilt, dass **P** nicht gleich der Sprachklasse der deterministisch kontextsensitiven Sprachen ist.
- Sowohl $\mathrm{DSPACE}(\log(n)) = \mathbf{P}$, $\mathrm{DSPACE}(n) \subset \mathbf{P}$ oder $\mathbf{P} \subset \mathrm{DSPACE}(n)$ sind nach heutigem Wissen möglich.

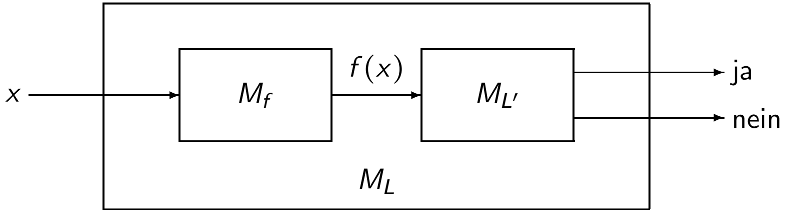
# Teil 3: Reduktionen und vollständige Probleme

Seien $L \subseteq \Sigma^*$ und $L' \subseteq \Sigma'^*$ zwei Entscheidungs-Probleme.

Eine Reduktion von $L$ auf $L'$ ist eine totale berechenbare Abbildung $f : \Sigma^* \to \Sigma'^*$ mit: $x \in L \iff f(x) \in L'$.

Angenommen, wir kennen bereits einen Algorithmus zur Lösung von $L'$. Dann können wir die Frage $x \in L$ wie folgt entscheiden:

1. Berechne den Wert $f(x) \in \Sigma'^*$
2. Entscheide mittels des Algorithmus für $L'$ ob $f(x) \in L'$ gilt.



# Polynomialzeitreduktionen

Eine Reduktion $f : \Sigma^* \to \Sigma'^*$ von $L$ auf $L'$ ist eine Polynomialzeitreduktion, falls sich $f$ durch eine deterministische polynomialzeitbeschränkte Turingmaschine berechnen lässt.

**Proposition 19**

$L' \in \mathbf{P}$ und $\exists$ Polynomialzeitreduktion von $L$ auf $L'$ $\implies$ $L \in \mathbf{P}$.

**Beweis:** Angenommen $L'$ gehört zu $\mathrm{DTIME}(n^k)$ und $f$ kann in Zeit $n^\ell$ berechnet werden.

Für ein Eingabe $x \in \Sigma^*$ der Länge $n$ kann $f(x)$ in Zeit $n^\ell$ berechnet werden.

Damit muss $|f(x)| \le n^\ell$ gelten, und es kann in Zeit $(n^\ell)^k = n^{k\cdot\ell}$ entschieden werden, ob $f(x) \in L'$ (d.h. $x \in L$) gilt.

Gesamter Zeitbedarf: $n^\ell + n^{k\cdot\ell}$ $\square$

# Reduktionen in logarithmischen Platz

Viele praktisch wichtige Reduktionen lassen sich in logarithmischem Platz berechnen. $\Rightarrow$ Logspace-Reduktionen

## Definition Logspace-Transducer

Ein logarithmisch platzbeschränkter Transduktor (Logspace-Transducer) ist eine deterministische Turingmaschine $M$ mit

- einem Eingabeband, von dem nur gelesen werden kann,
- einem logarithmisch in der Eingabelänge platzbeschränkten Arbeitsband, und
- einem separaten Ausgabeband, auf das nur geschrieben werden kann.

  In jedem Rechenschritt von $M$ wird
  - entweder ein neues Zeichen auf das Ausgabeband geschrieben und der Schreibkopf wandert ein Feld nach rechts, oder
  - es wird kein neues Zeichen auf das Ausgabeband geschrieben und der Schreibkopf bewegt sich nicht.

## Definition

1. Eine Abbildung $f : \Sigma^* \to \Sigma'^*$ heißt logspace berechenbar, falls gilt:
   $\exists$ Logspace-Transducer $M$ $\forall x \in \Sigma^*$ :
   $M$ hält bei Eingabe $x$ an mit $f(x) \in \Sigma'^*$ auf dem Ausgabeband
2. Ein Problem $L \subseteq \Sigma^*$ heißt logspace reduzierbar auf $L' \subseteq \Sigma'^*$, falls es eine logspace berechenbare Abbildung $f : \Sigma^* \to \Sigma'^*$ gibt mit

$$\forall x \in \Sigma^* : x \in L \iff f(x) \in L'.$$

Kurzschreibweise: $L \leq_m^{\log} L'$.

Der untere Index $m$ steht hier für many-one, dies bedeutet, dass mehrere Worte aus $\Sigma^*$ auf das selbe Wort in $\Sigma'^*$ abgebildet werden können.

**Bemerkungen:**

- Seien $L, L' \in \mathbf{P}$, $L \subseteq \Sigma^*$, $L' \in \Sigma'^*$, $\emptyset \neq L \neq \Sigma^*$ und $\emptyset \neq L' \neq \Sigma'^*$.

  Dann gibt es eine Polynomialzeitreduktion von $L$ auf $L'$ sowie eine Polynomialzeitreduktion von $L'$ auf $L$:

  Sei $x_0 \in \Sigma'^* \setminus L'$ und $x_1 \in L'$.

  Definiere eine Abbildung $f : \Sigma^* \to \Sigma'^*$ durch

  $$f(x) = \begin{cases} x_0 & \text{falls } x \in \Sigma^* \setminus L \\ x_1 & \text{falls } x \in L \end{cases}$$

  Dann ist $f$ eine Polynomialzeitreduktion von $L$ auf $L'$.

**Bemerkungen:**

- Logspace-Reduktionen lassen sich auch auf Klassen unterhalb von **P** anwenden und erlaubt eine feinere Einteilung als unter Verwendung von Polynomialzeitreduktionen.
- Jede logspace berechenbare Abbildung $f : \Sigma^* \to \Sigma'^*$ ist in polynomialer Zeit berechenbar.

  Insbesondere: $\exists k \geq 0 \; \forall x \in \Sigma^* : |f(x)| \leq |x|^k$.
- Logspace-Reduktionen und Polynomialzeitreduktionen sind genau dann gleichmächtig, wenn $\mathbf{L} = \mathbf{P}$ gilt.

**Proposition 20**

$L \leq_m^{\log} L' \leq_m^{\log} L'' \quad \Longrightarrow \quad L \leq_m^{\log} L'' \quad (\leq_m^{\log}$ *ist transitiv*$)$

**Beachte:** Die analoge Aussage für Polynomialzeitreduktionen ist trivial.

Bei der Hintereinanderausführung von Logspace-Reduktionen $f : \Sigma^* \to \Sigma'^*$ und $g : \Sigma'^* \to \Sigma''^*$ gibt es jedoch ein Problem:

- Für Eingabe $w \in \Sigma^*$ mit $|w| = n$ gilt $|f(w)| \leq n^k$ ($k$ Konstante).
- Die Anwendung von $g$ auf $f(w)$ erfordert damit Platz $\mathcal{O}(\log(n^k)) = \mathcal{O}(\log(n))$.
- Aber: In logarithmischen Platz kann $f(w)$ nicht auf das Arbeitsband geschrieben werden.

# $\leq_m^{\log}$ ist transitiv

**Beweis von Proposition 20:**

Wir berechnen $g(f(w))$ in Platz $\mathcal{O}(\log(|w|)$ wie folgt:

- Starte den Logspace-Transducer zur Berechnung von $g$ (ohne $f(w)$ vorher zu berechnen).
- Wenn während der Berechnung von $g$ das $i-$te Bit von $f(w)$ benötigt wird, wird der Logspace-Transducer zur Berechnung von $f(w)$ neu gestartet, bis schließlich das $i$-te Bit von $f(w)$ ausgegeben ist.

  Dabei werden die Bits $1, \ldots, i-1$ von $f(w)$ nicht ausgegeben.

  Hierzu wird ein Binärzähler jedesmal, wenn der Logspace-Transducer für $f$ ein Ausgabebit produziert, hochgezählt.
- Beachte: Binärzähler benötigt Platz $\mathcal{O}(\log(|f(w)|)) = \mathcal{O}(\log(|w|)$ $\square$

# $\leq_m^{\log}$ ist transitiv

**Beispiel:** Sei $f(n) = n^k$.

Dann ist die Abbildung $\$^n \mapsto \$^{f(n)}$ logspace berechenbar.

Damit folgt dass auch die Abbildung $w \mapsto w\$^{|w|^k - |w|}$ für $w \in \Sigma^*$ logspace berechenbar ist.

Konsequenz: $L \leq_m^{\log} \mathrm{Pad}_f(L)$ für $L \subseteq \Sigma^*$ ($\$ \notin \Sigma$)

Umgekehrt gilt auch $\mathrm{Pad}_f(L) \leq_m^{\log} L$ für $L \neq \Sigma^*$.

# Vollständige Probleme

## Definition

Sei $\mathcal{C}$ eine Komplexitätsklasse und sei $L \subseteq \Sigma^*$ ein Problem.

1. $L$ heißt *schwierig* für $\mathcal{C}$ oder kurz $\mathcal{C}$-schwierig (bzgl. logspace-Reduktionen), falls gilt: $\forall K \in \mathcal{C} : K \leq_m^{\log} L$.
2. $L$ heißt *$\mathcal{C}$-vollständig* (bzgl. logspace-Reduktionen), falls $L$ schwierig für $\mathcal{C}$ ist und zusätzlich $L \in \mathcal{C}$ gilt.

# GAP ist **NL**-vollständig

Wir geben ein erstes Beispiel:

**Satz 21**

Das Grapherreichbarkeitsproblem GAP ist **NL**-vollständig.

**Beweis:** GAP $\in$ **NL** wurde bereits gezeigt.

Sei $L \in$ **NL**, sei $M$ eine nichtdeterministische $\log(n)$-platzbeschränkte Turingmaschine mit $L = L(M)$.

Wir definieren eine Reduktion $f$ wie folgt: Für $w \in \Sigma^*$ sei $f(w) = (G, s, t)$ mit:

- $G = (V, E)$ ist der gerichtete Graph mit:

$$
\begin{aligned}
V &= \{c \mid c \text{ ist Konfig. von } M \text{ bei Eingabe } w, |c| \leq \log(|w|)\} \\
E &= \{(c, d) \mid c, d \in V, c \vdash_M d\}
\end{aligned}
$$

- $s = \mathrm{Start}(w)$
- $t$ = die (o.B.d.A) eindeutige akzeptierende Konfiguration von $M$.

# GAP ist **NL**-vollständig

Der Graph $G$ wird dabei durch seine Adjazenzmatrix repräsentiert.

Offensichtlich gilt:

$$w \in L(M) \iff \text{in } G \text{ gibt es einen gerichteten Pfad von } s \text{ nach } t.$$

$f$ kann schließlich in logarithmischen Platz berechnet werden.

Folgender Algorithmus berechnet die Adjazenzmatrix von $G$ in logarithmischen Platz.

**forall** $c \in V$ in längenlexikographischer Ordnung **do**
  **forall** $d \in V$ in längenlexikographischer Ordnung **do**
    **if** $c \vdash_M d$ **then write** 1
    **else write** 0
    **endif**
  **endfor**
  **write** #
**endfor**

$\square$

## Satz 22

Falls es eine **NP**-vollständige Sprache gibt, so auch eine in $\mathrm{NTIME}(n)$:

$$\exists L : L \text{ ist } \mathbf{NP}\text{-vollständig} \;\Rightarrow\; \exists \tilde{L} \in \mathrm{NTIME}(n) : \tilde{L} \text{ ist } \mathbf{NP}\text{-vollständig}.$$

**Beweis:** Sei $L$ ein **NP**-vollständiges Problem.

Es existiert eine Konstante $k > 0$ mit $L \in \mathrm{NTIME}(n^k)$.

Aus dem Translationssatz für Zeitklassen folgt $\mathrm{Pad}_{n^k}(L) \in \mathrm{NTIME}(n)$.

Sei nun $L' \in \mathbf{NP}$ beliebig.

$\Rightarrow L' \leq_m^{\log} L \leq_m^{\log} \mathrm{Pad}_{n^k}(L)$

Da $\leq_m^{\log}$ transitiv ist, folgt $L' \leq_m^{\log} \mathrm{Pad}_{n^k}(L)$.

$\Rightarrow \mathrm{Pad}_{n^k}(L)$ ist **NP**-vollständig. $\square$

# Das generische **NP**-vollständige Problem

Sei $\langle w, M\rangle$ eine Codierung eines Wortes $w \in \Sigma^*$ und einer nichtdeterministischen Turingmaschine $M$.

$$L_{\text{Gen}} = \{\langle w, M\rangle \,\$^m \mid w \in \Sigma^*, M \text{ nichtdeterministische Turingmaschine}, \\ m \in \mathbb{N}, M \text{ hat bei Eingabe } w \text{ eine akzeptierende} \\ \text{Berechnung der Länge} \leq m\}$$

## Satz 23

$L_{\text{Gen}}$ ist **NP**-vollständig.

**Beweis:**

$L_{\text{Gen}} \in$ **NP**:

Für eine Eingabe $\langle w, M\rangle \,\$^m$ simuliere $M$ bei Eingabe $w$ nichtdeterministisch für maximal $m$ Schritte.

Dies ist ein nichtdeterministischer Polynomialzeitalgorithmus für $L_{\text{Gen}}$.

# Das generische **NP**-vollständige Problem

$L_{\text{Gen}}$ ist **NP**-schwierig:

Sei $L \in \mathbf{NP}$ beliebig und $M$ eine $n^k$-zeitbeschränkte nichtdeterministische Turingmaschine mit $L = L(M)$ ($k$ ist eine Konstante).

Die Reduktion von $L$ auf $L_{Gen}$ berechnet nun in logarithmischem Platz auf eine Eingabe $w \in \Sigma^*$ die Ausgabe

$$f(w) = \langle w, M \rangle \$^{|w|^k}.$$

Es gilt: $w \in L(M) \iff f(w) \in L_{Gen}$. $\square$

# Der Satz von Cook

Sei $\Sigma_0 = \{\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, 0, 1, (, ), x\}$.

Sei $\mathbb{A} \subseteq \Sigma_0^*$ die Menge aller aussagenlogischen Formeln über der Variablenmenge $V = x1\{0,1\}^*$.

$\mathbb{A} \subseteq \Sigma_0^*$ ist deterministisch kontextfrei und gehört damit zu $\mathrm{DTIME}(n)$.

Sei $\mathrm{SAT} = \{F \in \mathbb{A} \mid F \text{ ist erfüllbar}\}$.

(Eine aussagenlogische Formel $F$ ist erfüllbar, wenn es eine Belegung $\mathcal{B} : \mathrm{Var}(F) \to \{\mathbf{true}, \mathbf{false}\}$ der in $F$ vorkommenden Variablen mit Wahrheitswerten gibt, unter der sich $F$ zu **true** auswertet.)

**Satz 24 (Satz von Cook)**

$\mathrm{SAT}$ ist **NP**-vollständig.

**(A)** $\mathrm{SAT} \in \mathbf{NP}$: Für ein $F \in \Sigma_0^*$ überprüfen wir "$F \in \mathrm{SAT}$" wie folgt:

1. Teste in Zeit $\mathcal{O}(|F|)$ ob $F \in \mathbb{A}$ gilt.
2. Falls "JA", rate eine Belegung $\mathcal{B} : \mathsf{Var}(F) \to \{\mathbf{true}, \mathbf{false}\}$.
3. Akzeptiere, falls $F$ sich unter der Belegung $\mathcal{B}$ zu **true** auswertet.

**(B)** $\mathrm{SAT}$ ist **NP**-schwierig.

Sei $L \in \mathbf{NP}$.

Zu $w \in \Sigma^*$ konstruieren wir eine Formel $f(w)$ mit

$$w \in L \quad \Longleftrightarrow \quad f(w) \text{ erfüllbar} .$$

Die Abbildung $f$ wird logspace berechenbar sein.

# Beweis des Satzes von Cook

Sei $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$ eine $p(n)$-zeitbeschränkte nichtdeterministische Turingmaschine mit $L = L(M)$ ($p(n) > n$ ist ein Polynom).

Sei $w = w_1 w_2 \cdots w_n \in \Sigma^*$ eine Eingabe der Länge $n$ (o.B.d.A. $n \geq 1$).

O.B.d.A. hat $M$ folgende Eigenschaften:

1. $M$ hat nur ein Band, dessen Inhalt zu Beginn $\cdots \Box \Box w \Box \Box \cdots$ ist, und dessen Zellen während der Berechnung beliebig beschrieben werden dürfen.
2. Die Endmarker $\triangleright$ und $\triangleleft$ brauchen wir nicht.
3. $M$ akzeptiert $w$ genau dann, wenn sich $M$ nach genau $p(n)$ vielen Schritten im Zustand $q_J$ befindet, und der Schreib-Lesekopf wieder in der Ausgangsposition ist, wo ein $\Box$ steht.
4. Aus $(p_1, a_1, d_1), (p_2, a_2, d_2) \in \delta(q, a)$ folgt $a_1 = a_2$ und $d_1 = d_2$.

# Beweis des Satzes von Cook

Punkt 4 von Folie 88 kann wie folgt erzwungen werden: Definiere eine neue nichtdeterministische Turingmaschine

$$M' = (Q', \Sigma, \Gamma, \delta', q_0, q_J, q_N, \Box)$$

wie folgt:

- $Q' = Q \cup (Q \times \Gamma \times \{-1, 0, 1\})$
- Für alle $q \in Q, a \in \Gamma$ setze

$$\delta'(q, a) = \{((p, b, d), a, 0) \mid (p, b, d) \in \delta(q, a)\}.$$

- Für alle $(p, b, d) \in Q \times \Gamma \times \{-1, 0, 1\}$ und alle $a \in \Gamma$ setze

$$\delta'((p, b, d), a) = \{(p, b, d)\}.$$

Dann gilt $L(M) = L(M')$ und auch $M'$ ist polynomiell zeitbeschränkt.

# Beweis des Satzes von Cook

Jede von der Startkonfiguration erreichbare Konfiguration kann durch ein Wort aus

$$\text{Conf} = \{\Box u(q,a)v\Box \mid (q,a) \in Q \times \Gamma,\ uv \in \Gamma^{2p(n)}\}$$

beschrieben werden.

Die Startkonfiguration ist $\Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$.

Sei $\Omega = (Q \times \Gamma) \cup \Gamma$.

**Notation:** Für ein $\alpha \in \text{Conf}$ schreiben wir

$$\alpha = \alpha[-p(n)-1]\alpha[-p(n)]\cdots\alpha[p(n)]\alpha[p(n)+1]$$

wobei

- $\alpha[-p(n)-1] = \alpha[p(n)+1] = \Box$ und
- $\alpha[-p(n)], \ldots, \alpha[p(n)] \in \Omega$.

Seien nun $\alpha, \alpha' \in \mathrm{Conf}$ mit $\alpha \vdash_M \alpha'$ und $-p(n) \leq i \leq p(n)$.

$\alpha[i-1], \alpha[i]$ und $\alpha[i+1]$ legen fest, welche Symbole für $\alpha'[i]$ in Frage kommen.

**Beispiel:**

Falls $(p, a', -1) \in \delta(q, a)$ ist folgende lokale Bandänderung möglich:

| Position | | | $i$–2 | $i$–1 | $i$ | $i$+1 | $i$+2 | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha =$ | $\cdots$ | $\cdots$ | $b'$ | $b$ | $q, a$ | $c$ | $c'$ | $\cdots$ | $\cdots$ |
| $\alpha' =$ | $\cdots$ | $\cdots$ | $b'$ | $p, b$ | $a'$ | $c$ | $c'$ | $\cdots$ | $\cdots$ |

Falls $(p, a', +1) \in \delta(q, a)$ ist folgende lokale Bandänderung möglich:

| Position | | | $i$–2 | $i$–1 | $i$ | $i$+1 | $i$+2 | | |
|---|---|---|---|---|---|---|---|---|---|
| $\alpha =$ | $\cdots$ | $\cdots$ | $b'$ | $b$ | $q, a$ | $c$ | $c'$ | $\cdots$ | $\cdots$ |
| $\alpha' =$ | $\cdots$ | $\cdots$ | $b'$ | $b$ | $a'$ | $p, c$ | $c'$ | $\cdots$ | $\cdots$ |

# Beweis des Satzes von Cook

Wir definieren nun eine Menge $\Delta \subseteq \Omega^4$, die alle möglichen 4-Tupel $(\alpha[i-1], \alpha[i], \alpha[i+1], \alpha'[i])$ dieser Art enthält:

- $(a, b, c, b)$ wobei $a, b, c \in \Gamma$
- $(b, c, (q, a), (p, c))$, $(b, (q, a), c, a')$, $((q, a), b, c, b)$, wobei $(p, a', -1) \in \delta(q, a)$, $b, c \in \Gamma$
- $(b, c, (q, a), c)$, $(b, (q, a), c, a')$, $((q, a), b, c, (p, b))$, wobei $(p, a', +1) \in \delta(q, a)$, $b, c \in \Gamma$

Dann gilt für alle $\alpha, \alpha' \in \Box\Omega^*\Box$ mit $|\alpha| = |\alpha'|$:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$

$$\Longleftrightarrow$$

$$\alpha \in \text{Conf und } \forall i \in \{-p(n), \ldots, p(n)\} : (\alpha[i-1], \alpha[i], \alpha[i+1], \alpha'[i]) \in \Delta.$$

Hierfür ist Punkt 4 von Folie 88 wichtig!

Eine Rechnung von $M$ können wir nun als Matrix beschreiben:

$$
\begin{array}{ccccccccc}
\alpha_0 & = & \Box & \alpha_{0,-p(n)} & \alpha_{0,-p(n)+1} & \cdots & \alpha_{0,p(n)} & \Box \\
\alpha_1 & = & \Box & \alpha_{1,-p(n)} & \alpha_{1,-p(n)+1} & \cdots & \alpha_{1,p(n)} & \Box \\
 & & & & \vdots & & & \\
\alpha_{p(n)} & = & \Box & \alpha_{p(n),0} & \alpha_{p(n),1} & \cdots & \alpha_{p(n),p(n)} & \Box
\end{array}
$$

Für jedes Tripel $(a, i, t)$ $(a \in \Omega,\ -p(n) - 1 \leq i \leq p(n) + 1,\ 0 \leq t \leq p(n))$ sei $x(a, i, t)$ eine aussagenlogische Variable.

**Interpretation:** $x(a, i, t) =$ **true** genau dann, wenn zum Zeitpunkt $t$ das $i$-te Zeichen der aktuellen Konfiguration ein $a$ ist.

# Beweis des Satzes von Cook

An den Positionen $-p(n)-1$ und $p(n)+1$ steht immer $\Box$:

$$G(n) = \bigwedge_{0 \le t \le p(n)} \Big( x(\Box, -p(n)-1, t) \wedge x(\Box, p(n)+1, t) \Big)$$

Für jedes Paar $(i,t)$ ist genau eine Variable $x(a,i,t)$ wahr (zu jedem Zeitpunkt kann auf einem Bandfeld nur ein Zeichen stehen):

$$X(n) = \bigwedge_{\substack{0 \le t \le p(n) \\ -p(n)-1 \le i \le p(n)+1}} \left( \bigvee_{a \in \Omega} \left( x(a,i,t) \wedge \bigwedge_{b \in \Omega \setminus \{a\}} \neg x(b,i,t) \right) \right)$$

# Beweis des Satzes von Cook

Zum Zeitpunkt $t = 0$ ist die Konfiguration gleich $\Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$:

$$S(w) = \bigwedge_{i=1}^{p(n)} x(\Box, -i, 0) \wedge x((q_0, w_1), 0, 0) \wedge \bigwedge_{i=1}^{n-1} x(w_{i+1}, i, 0) \wedge \bigwedge_{i=n}^{p(n)} x(\Box, i, 0)$$

Die Berechnung respektiert die lokale Relation $\Delta$:

$$D(n) = \bigwedge_{\substack{-p(n) \le i \le p(n) \\ 0 \le t \le p(n)-1}} \bigvee_{(a,b,c,d) \in \Delta} \begin{pmatrix} x(a, i-1, t) & \wedge & x(b, i, t) & \wedge \\ x(c, i+1, t) & \wedge & x(d, i, t+1) & \end{pmatrix}$$

Sei schließlich

$$f(w) = G(n) \land X(n) \land S(w) \land D(n) \land x((q_J, \Box), 0, p(n)).$$

Es ergibt sich eine natürliche Bijektion zwischen der Menge der $f(w)$ erfüllenden Belegungen und der Menge der akzeptierenden Rechnungen von $M$ auf die Eingabe $w$.

Es gilt also:

$$f(w) \text{ erfüllbar} \iff w \in L.$$

Zahl der Variablen von $f(w) \in \mathcal{O}(p(n)^2)$

Länge von $f(w) \in \mathcal{O}(p(n)^2 \log p(n))$

Der Faktor $\mathcal{O}(\log p(n))$ ist notwendig, da zum Aufschreiben der Indizes $\log p(n)$ viele Bits benötigt werden. $\Box$

# Weitere **NP**-vollständige Probleme: (1) SAT ∩ KNF

## Definition: Literale, KNF

Ein Literal $\tilde{x}$ ist eine aussagenlogische Variable oder die Negation einer aussagenlogischen Variablen.

Statt $\neg x$ schreiben wir auch $\overline{x}$. Außerdem sei $\overline{\overline{x}} = x$.

Sei $\mathrm{KNF}$ (bzw. $\mathrm{DNF}$) die Menge der aussagenlogischen Ausdrücke in konjunktiver Normalform (bzw. disjunktiver Normalform):

$$\begin{aligned}\mathrm{DNF} &= \{F \mid F \text{ ist Disjunktion von Konjunktionen von Literalen}\}\\ \mathrm{KNF} &= \{F \mid F \text{ ist Konjunktion von Disjunktionen von Literalen}\}\end{aligned}$$

**Fakt:** Für jede aussagenlogische Formel $F$ gibt es äquivalente Formeln $\mathrm{DNF}(F) \in \mathrm{DNF}$ und $\mathrm{KNF}(F) \in \mathrm{KNF}$.

# Weitere **NP**-vollständige Probleme: (1) SAT ∩ KNF

**Beispiel:**

$$F = \bigwedge_{i=1,\dots,k} \left( \bigvee_{j=1,\dots,m} \tilde{x}_{i,j} \right) \equiv \bigvee_{f \in \{1,\dots,m\}^{\{1,\dots,k\}}} \left( \bigwedge_{i=1,\dots,k} \tilde{x}_{i,f(i)} \right) = F'$$

Beachte:

- $|F| = m \cdot k$ während $|F'| = m^k \cdot k$, d.h. eine KNF-Formel mit $k$ Disjunktionen der Länge $m$ kann in eine äquivalente DNF-Formel bestehend aus $m^k$ Konjunktionen der Länge $k$ umgewandelt werden.
- Für Formeln in DNF kann Erfüllbarkeit deterministisch in quadratischer Zeit überprüft werden.
- Wir werden gleich sehen, dass Erfüllbarkeit für Formeln in KNF **NP**-vollständig ist.
  Deswegen ist der exponentielle Blow-Up bei der Umwandlung von KNF in DNF nicht überraschend.

# SAT ∩ KNF ist **NP**-vollständig

**Satz 25**

$\mathrm{SAT} \cap \mathrm{KNF}$ ist **NP**-vollständig.

**Beweis:**

(1) $\mathrm{SAT} \cap \mathrm{KNF} \in$ **NP**: trivial, denn (i) $\mathrm{SAT} \in$ **NP** und (ii) für eine Formel kann in Zeit $\mathcal{O}(n)$ getestet werden, ob sie in KNF ist.

(2) $\mathrm{SAT} \cap \mathrm{KNF}$ ist **NP**-schwierig:

1\. Beweis: Im Beweis der **NP**-Vollständigkeit von $\mathrm{SAT}$ haben wir eine Formel konstruiert, die bis auf innere Teilformeln konstanter Länge bereits in $\mathrm{KNF}$ war.

Wir können also durch eine weitere logspace-berechenbare Abbildung die Formel in KNF bringen und sind fertig.

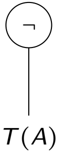
2. Beweis: Wir zeigen $\mathrm{SAT} \leq_m^{\log} \mathrm{SAT} \cap \mathrm{KNF}$.

Hierzu müssen wir eine logspace-berechenbare Abbildung $f : \mathbb{A} \to \mathrm{KNF}$ angeben mit:

$$F \in \mathrm{SAT} \iff f(F) \in \mathrm{SAT} \cap \mathrm{KNF}.$$

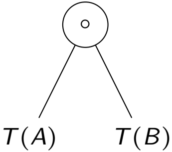
Wir können eine Formel $F \in \mathbb{A}$ als einen Baum $T(F)$ auffassen, der sich rekursiv folgendermaßen aufbauen lässt:

1. Für eine Variable $x$ sei $T(x) = x$.
2. Ist $F$ die Negation einer Formel $A$, also $F = \neg A$, so habe $T(F)$ folgende Gestalt:



# SAT ∩ KNF ist **NP**-vollständig

3. Besteht $F$ aus der Verknüpfung zweier Teilformeln $A, B$, d.h. $F = A \circ B$ mit $\circ \in \{\Longleftrightarrow, \Rightarrow, \wedge, \vee\}$, so habe $T(F)$ folgende Gestalt:

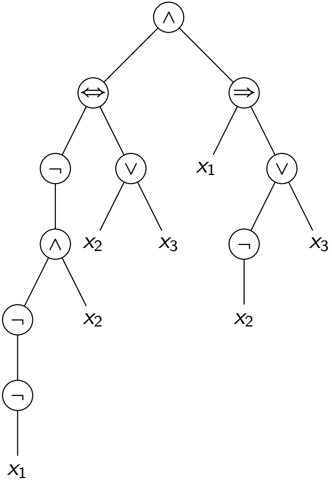



Für die Formel

$$F = \Big(\big((\neg(\neg\neg x_1 \wedge x_2)) \Longleftrightarrow (x_2 \vee x_3)\big) \wedge \big(x_1 \Rightarrow (\neg x_2 \vee x_3)\big)\Big)$$

erhalten wir beispielsweise den Baum $T(F)$ auf der nächsten Folie:

# SAT ∩ KNF ist **NP**-vollständig




Wir ordnen nun jedem Knoten von $T(F)$ eine neue Variable $v(A)$ zu, wobei $A$ die durch den Knoten repräsentierte Teilformel von $F$ ist.

# SAT ∩ KNF ist **NP**-vollständig

**Definiere Hilfsfunktion $f' : \mathbb{A} \to \mathrm{SAT} \cap \mathrm{KNF}$ rekursiv wie folgt:**

1. Ist $F = x$, so ist $f'(F) := \mathrm{KNF}\big(v(x) \Leftrightarrow x\big)$.
2. Ist $F = A \circ B$ mit $\circ \in \{\Leftrightarrow, \Leftarrow, \wedge, \vee\}$, so ist

$$f'(F) := \Big(\mathrm{KNF}\big(v(F) \Leftrightarrow (v(A) \circ v(B))\big) \wedge f'(A) \wedge f'(B)\Big).$$

3. Ist $F = \neg A$, so ist

$$f'(F) := \Big(\mathrm{KNF}\big(v(F) \Leftrightarrow \neg v(A)\big) \wedge f'(A)\Big).$$

Hier erhalten wir nach entsprechender Umformung:

$$f'(F) = \Big(\big(\, v(F) \vee v(A)\big) \wedge \big(\, \neg v(F) \vee \neg v(A) \,\big) \wedge f'(A)\Big).$$

# SAT ∩ KNF ist **NP**-vollständig

**Beachte:** In der Definition von $f'$ (noch nicht die eigentliche Reduktion) wenden wir $\mathrm{KNF}$ nur auf Formeln konstanter Länge an.

Im folgenden sei $V(G)$ die Menge aller Variablen, die in der Formel $G \in \mathbb{A}$ vorkommen.

Beachte: $V(G) \subseteq V(f'(G))$

**Lemma**

1. $f'(F)$ ist stets erfüllbar.
2. Sei $\sigma : V(f'(F)) \to \{0,1\}$ eine Belegung mit $\sigma(f'(F)) = 1$ und $\sigma'$ die Restriktion von $\sigma$ auf $V(F)$. Dann gilt: $\sigma'(F) = \sigma(v(F))$.
3. Für jede Belegung $\sigma' : V(F) \to \{0,1\}$ gibt es eine Belegung $\sigma : V(f'(F)) \to \{0,1\}$ mit $\sigma(f'(F)) = 1$ und $\sigma'(x) = \sigma(x)$ für alle $x \in V(F)$.

**Beweis von (2)**: Sei $\sigma$ eine erfüllende Belegung von $f'(F)$ und $\sigma'$ die Restriktion von $\sigma$ auf $V(F)$.

Strukturelle Induktion über den Aufbau von $F$:

1. Fall: $F = x \in V(F)$. Es gilt

$$\sigma(f'(F)) = \sigma(\mathrm{KNF}(v(x) \Leftrightarrow x)) = \sigma(v(x) \Leftrightarrow x) = 1$$

Also gilt $\sigma(v(F)) = \sigma(v(x)) = \sigma(x) = \sigma'(x) = \sigma'(F)$.

2. Fall: $F = A \circ B$ mit $\circ \in \{\Leftrightarrow, \Leftarrow, \wedge, \vee\}$. Es gilt

$$\begin{aligned}
\sigma(f'(F)) &= \sigma(\mathrm{KNF}\big(v(F) \Leftrightarrow (v(A) \circ v(B))\big) \wedge f'(A) \wedge f'(B)) \\
&= \sigma(v(F) \Leftrightarrow (v(A) \circ v(B))) \wedge \sigma(f'(A)) \wedge \sigma(f'(B)) \\
&= 1
\end{aligned}$$

Nach Induktion gilt $\sigma'(A) = \sigma(v(A))$ und $\sigma'(B) = \sigma(v(B))$.

Außerdem gilt $\sigma(v(F)) = \sigma(v(A) \circ v(B))$.

Also erhalten wir $\sigma(v(F)) = \sigma(v(A) \circ v(B)) = \sigma'(A \circ B) = \sigma'(F)$.

2. Fall: $F = \neg A$: analog zu Fall 2.

**Beweis von (3)**: Sei $\sigma' : V(F) \to \{0,1\}$ eine beliebige Belegung.

Definiere $\sigma : V(f'(F)) \to \{0,1\}$ wie folgt induktiv:

$$
\begin{aligned}
\sigma(x) &= \sigma'(x) \text{ für alle } x \in V(F) \\
\sigma(v(x)) &= \sigma'(x) \text{ für alle } x \in V(F) \\
\sigma(v(G)) &= \sigma(v(A) \circ v(B)) \text{ falls } G = A \circ B \\
\sigma(v(G)) &= \sigma(\neg v(A)) \text{ falls } G = \neg A
\end{aligned}
$$

Mit Induktion über den Aufbau von $F$ folgt dann sofort $\sigma(f'(F)) = 1$.

Punkt (1) folgt unmittelbar aus (3).

# SAT ∩ KNF ist **NP**-vollständig

Die eigentliche Reduktion $f : \mathbb{A} \to \mathrm{KNF}$ können wir jetzt definieren durch

$$f(F) := (f'(F) \wedge v(F)).$$

**Behauptung:** $f(F)$ erfüllbar genau dann, wenn $F$ erfüllbar.

**Beweis der Behauptung:**

**(A)** Sei $\sigma'$ eine Belegung von $F$ mit $\sigma'(F) = 1$.

Nach (3) aus dem Lemma existiert eine Belegung $\sigma$ für $f'(F)$ mit $\sigma(f'(F)) = 1$ und $\sigma(x) = \sigma'(x)$ für alle $x \in V(F)$.

Aus (2) folgt $\sigma(v(F)) = \sigma'(F) = 1$.

Also: $\sigma(f'(F) \wedge v(F)) = 1$.

**(B)** Sei $\sigma$ eine Belegung von $f'(F) \wedge v(F)$ mit $\sigma(f'(F) \wedge v(F)) = 1$.

Für die Restriktion $\sigma'$ auf die Variablen von $F$ gilt nach (2):
$\sigma'(F) = \sigma(v(F)) = 1$. $\square$

# 3-SAT ist **NP**-vollständig

## Definition: 3-SAT

Sei $\mathrm{3\text{-}KNF}$ die Menge der Formeln in konjunktiver Form mit genau drei Literalen je Klausel:

$$\mathrm{3\text{-}KNF} := \{F \in \mathrm{KNF} \mid \text{Jede Klausel in } F \text{ enthält genau drei Literale}\}$$

$\mathrm{3\text{-}SAT}$ sei die Teilmenge der davon erfüllbaren Formeln:

$$\mathrm{3\text{-}SAT} := \mathrm{3\text{-}KNF} \cap \mathrm{SAT}$$

## Satz 26

$\mathrm{3\text{-}SAT}$ ist **NP**-vollständig.

**Beweis:** Nur die **NP**-Schwierigkeit ist nicht trivial.

Wir zeigen: $\mathrm{SAT} \cap \mathrm{KNF} \leq_m^{\log} \mathrm{3\text{-}SAT}$.

Sei $F$ eine $\mathrm{KNF}$-Formel. Wir unterscheiden drei Fälle:

1. $F$ enthält eine Klausel $(\tilde{x})$ mit nur einem Literal.
   Führe neue Variable $y$ ein und ersetze $(\tilde{x})$ durch $(\tilde{x} \vee y) \wedge (\tilde{x} \vee \overline{y})$.
   Dies hat auf die Erfüllbarkeit von $F$ keine Auswirkung.
2. $F$ enthält eine Klausel $(\tilde{x} \vee \tilde{y})$ mit zwei Literalen.
   Führe neue Variable $z$ ein und ersetzte $(\tilde{x} \vee \tilde{y})$ durch
   $(\tilde{x} \vee \tilde{y} \vee z) \wedge (\tilde{x} \vee \tilde{y} \vee \overline{z})$.
3. $F$ enthält Klauseln mit mehr als drei Literalen.
   Sei also $c = (\tilde{x}_1 \vee \tilde{x}_2 \vee \cdots \vee \tilde{x}_k)$ eine solche Klausel mit $k \geq 4$ Literalen.
   Führe $k-3$ neue Variablen $v(\tilde{x}_3), v(\tilde{x}_4), \ldots, v(\tilde{x}_{k-2}), v(\tilde{x}_{k-1})$ ein und ersetzen $c$ durch

$$
\begin{aligned}
c' \quad = \quad & \big(\tilde{x}_1 \vee \tilde{x}_2 \vee v(\tilde{x}_3)\big) \wedge \bigwedge_{j=3}^{k-2} \big(\neg v(\tilde{x}_j) \vee \tilde{x}_j \vee v(\tilde{x}_{j+1})\big) \\
& \wedge \big(\neg v(\tilde{x}_{k-1}) \vee \tilde{x}_{k-1} \vee \tilde{x}_k\big).
\end{aligned}
$$

Beachte: $c'$ kann auch geschrieben werden als

$$\begin{aligned} c' \;=\; & \big(\tilde{x}_1 \vee \tilde{x}_2 \vee v(\tilde{x}_3)\big) \wedge \bigwedge_{j=3}^{k-2} \big( v(\tilde{x}_j) \;\Rightarrow\; \tilde{x}_j \vee v(\tilde{x}_{j+1})\big) \\ & \wedge \big( v(\tilde{x}_{k-1}) \;\Rightarrow\; \tilde{x}_{k-1} \vee \tilde{x}_k \big). \end{aligned}$$

Dass (3) nichts an der Erfüllbarkeit ändert folgt aus folgenden Punkten:

**(A)** Sei $\sigma$ eine erfüllende Belegung für $c$.

Dann muss $\sigma(\tilde{x}_l) = 1$ für ein $1 \le l \le k$ gelten.

Erweitere $\sigma$ zu einer erfüllenden Belegung von $c'$ durch:

$$\sigma'(v(\tilde{x}_p)) = \begin{cases} 1 & \text{falls } p \le l \\ 0 & \text{falls } p > l \end{cases}$$

# 3-SAT ist **NP**-vollständig

Die eindeutige Klausel, in der $\tilde{x}_l$ vorkommt, ist natürlich erfüllt.

In allen anderen Klauseln kommt entweder $v(\tilde{x}_p)$ mit $p \leq l$ oder $\neg v(\tilde{x}_p)$ mit $p > l$ vor.

**(B)** Sei $\sigma'$ eine erfüllende Belegung für $c'$.

Angenommen $\sigma'(\tilde{x}_i) = 0$ für alle $1 \leq i \leq k$.

Es muss $\sigma'(v(\tilde{x}_3)) = 1$ gelten (da $\sigma'(\tilde{x}_1 \vee \tilde{x}_2 \vee v(\tilde{x}_3)) = 1$).

Mit Induktion folgt: $\sigma'(v(\tilde{x}_i)) = 1$ für alle $3 \leq i \leq k-1$.

Wir erhalten $\sigma'(\neg v(\tilde{x}_{k-1}) \vee \tilde{x}_{k-1} \vee \tilde{x}_k)) = 0$. Widerspruch! $\square$

Es sei

$$\mathrm{LinProg}(\mathbb{Z}) := \{\langle A, b\rangle \mid A \in \mathbb{Z}^{m\times n},\ b \in \mathbb{Z}^{m\times 1},\ \exists x \in \mathbb{Z}^{n\times 1} : Ax \geq b\}$$

Zahlen aus $\mathbb{Z}$ werden hier binär kodiert.

**Satz 27**

$\mathrm{LinProg}(\mathbb{Z})$ ist **NP**-vollständig.

**Beweis:**

(1) $\mathrm{LinProg}(\mathbb{Z}) \in$ **NP**:

Dies ist der schwierige Teil des Beweises, siehe z. B. Hopcroft, Ullman; *Introduction to Automata Theory, Languages and Computation*, Addison Wesley 1979

# Integer Programming

(2) $\mathrm{LinProg}(\mathbb{Z})$ ist **NP**-schwierig.

Wir zeigen $3\text{-}\mathrm{SAT} \leq_m^{\log} \mathsf{LinProg}(\mathbb{Z})$.

Sei $F = c_1 \wedge c_2 \wedge \cdots \wedge c_q$ eine Formel in $3\text{-}\mathrm{KNF}$.

Seien $x_1, \ldots, x_n$ die Variablen in $F$.

Wir bilden das folgende System $S$ von $\mathbb{Z}$-Ungleichungen über den Variablen $x_i, \overline{x_i}, 1 \leq i \leq n$:

1. $x_i \geq 0, \;\; 1 \leq i \leq n$
2. $\overline{x_i} \geq 0, \;\; 1 \leq i \leq n$
3. $x_i + \overline{x_i} \geq 1, \;\; 1 \leq i \leq n$
4. $-x_i - \overline{x_i} \geq -1, \;\; 1 \leq i \leq n$
5. $\tilde{x}_{j1} + \tilde{x}_{j2} + \tilde{x}_{j3} \geq 1$, für jede Klausel $c_j = (\tilde{x}_{j1} \vee \tilde{x}_{j2} \vee \tilde{x}_{j3})$.

$$
\begin{aligned}
(3) \text{ und } (4) \quad &\Longrightarrow \quad x_i + \overline{x_i} = 1 \\
(1) \text{ und } (2) \quad &\Longrightarrow \quad x_i = 1, \overline{x_i} = 0 \text{ oder } x_i = 0, \overline{x_i} = 1 \\
(5) \quad &\Longrightarrow \quad \text{in jeder Klausel } c_j \text{ hat mindestens ein Literal} \\
&\phantom{\Longrightarrow} \quad \tilde{x}_{ij} \text{ den Wert } 1
\end{aligned}
$$

Also: $S$ lösbar genau dann, wenn $F$ erfüllbar.

Größe von $S$: $4n+q$ Ungleichungen, $2n$ Variablen.

Schreiben wir $S$ in Matrixform $Ax \geq b$, so hat $A$ (bzw. $b$) $(4n+q) \times 2n$ (bzw. $4n+q$) Einträge von Absolutbetrag $\leq 1$. □

**Bemerkungen:**

- Obiger Beweis zeigt, dass $\mathrm{LinProg}(\mathbb{Z})$ bereits bei unärer Kodierung **NP**-schwierig ist.
- $\mathrm{LinProg}(\mathbb{Q}) \in$ **P**. Dieser Nachweis ist sehr schwierig und beruht auf der *Ellipsoidmethode* von Khachiyan.

# Vertex Cover ist **NP**-vollständig

Eine Knotenüberdeckung (vertex cover) für einen ungerichteten Graphen $G = (V, E)$ ist eine Teilmenge $C \subseteq V$, so das für jede Kante $\{u, v\} \in E$ gilt: $\{u, v\} \cap C \neq \emptyset$

Vertex Cover (VC) ist das folgende Problem:

**Eingabe:** Ein ungerichteter Graph $G = (V, E)$ und ein $k \geq 0$.

**Frage:** Hat $G$ Knotenüberdeckung $C$ mit $|C| \leq k$?

### Satz 28

$\mathrm{VC}$ ist **NP**-vollständig.

**Beweis:**

(1) $\mathrm{VC} \in$ **NP**: Rate eine Teilmenge $C$ der Knoten mit $|C| \leq k$ und überprüfe danach in Polynomialzeit, ob $C$ eine Knotenüberdeckung ist.

(1) $\mathrm{VC}$ ist **NP**-schwierig:

Wir zeigen $3\text{-}\mathrm{SAT} \leq_m^{\log} \mathrm{VC}$.

Sei

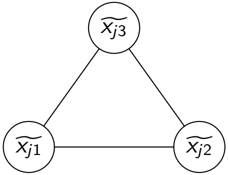
$$F = c_1 \wedge \cdots \wedge c_q$$

eine Formel in 3-$\mathrm{KNF}$, wobei

$$c_j = (\widetilde{x_{j1}} \vee \widetilde{x_{j2}} \vee \widetilde{x_{j3}}).$$

Wir konstruieren einen Graphen $G(F)$ wie folgt:

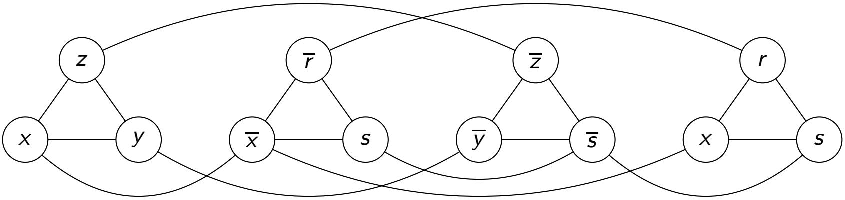
Zunächst bilden wir zu jeder Klausel $c_j = (\widetilde{x_{j1}} \vee \widetilde{x_{j2}} \vee \widetilde{x_{j3}})$ den folgenden Graphen $G(c_j)$:



Der Graph $G(F)$ entsteht aus der disjunkten Vereinigung $\bigcup_{j=1}^{q} G(c_j)$ aller dieser Teilgraphen $G(c_j)$ durch Einfügen aller Kanten $(x, \overline{x})$ ($x$ ist eine Variable aus $F$).

**Beispiel:**

Für die Formel $F = (x \vee y \vee z) \wedge (\overline{x} \vee s \vee \overline{r}) \wedge (\overline{y} \vee \overline{s} \vee \overline{z}) \wedge (x \vee s \vee r)$ führt diese Konstruktion zu folgendem Graphen $G(F)$:



# Vertex Cover ist **NP**-schwierig

**Beachte:** In $G(F)$ kann es kein Vertex Cover $U$ mit weniger als $2q$ Knoten geben, da in jedem der $q$ Dreiecke mindestens 2 Knoten zu $U$ gehören müssen.

**Behauptung**: $F \in 3\text{-SAT}$ genau dann, wenn $G(F)$ ein Vertex Cover $U$ mit $|U| = 2q$ hat.

(A) Sei $\sigma$ eine erfüllende Belegung für $F$.

Dann wird in jeder Klausel $c_j$ mindestens ein Literal $\widetilde{x_{ji}}$ wahr.

Sei $U$ eine Knotenmenge, die für jeden Teilgraphen $G(c_j)$ genau zwei Knoten enthält, so dass nicht-erfüllte Literale zu $U$ gehören.

Dann gilt $|U| = 2q$ und $U$ ist ein Vertex-Cover.

# Vertex Cover ist **NP**-schwierig

(B) Sei $U$ ein Vertex-Cover mit $|U| = 2q$.

Dann enthält $U$ aus jedem Teilgraphen $G(c_j)$ genau zwei Knoten.

Definiere Belegung $\sigma$ durch

$$\sigma(x) = \begin{cases} 1 & \text{falls eine Kopie von } x \text{ nicht zu } U \text{ gehört.} \\ 0 & \text{falls eine Kopie von } \overline{x} \text{ nicht zu } U \text{ gehört.} \\ 0 & \text{falls alle Kopien von } x \text{ und } \overline{x} \text{ zu } U \text{ gehören.} \end{cases}$$

Beachte: Da $U$ ein Vertex Cover ist, und alle Kanten $(x, \overline{x})$ in $G(F)$ vorhanden sind, wird keine Variable gleichzeitig auf 0 und 1 gesetzt.

Offensichtlich gilt $\sigma(F) = 1$. □

# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

Ein Hamilton-Pfad in einem gerichteten Graphen $G = (V, E)$ ist eine Folge von Knoten $v_1, v_2, \dots, v_n$ mit

- $(v_i, v_{i+1}) \in E$ für alle $1 \le i \le n-1$ und
- für jeden Knoten $v \in V$ existiert genau ein $1 \le i \le n$ mit $v = v_i$.

Ein Hamilton-Kreis ist ein Hamilton-Pfad $v_1, v_2, \dots, v_n$ mit $(v_n, v_1) \in E$.

Es sei

$$\begin{aligned} \mathrm{HP} &= \{G \mid G \text{ ist ein Graph mit einem Hamilton-Pfad}\} \\ \mathrm{HC} &= \{G \mid G \text{ ist ein Graph mit einem Hamilton-Kreis}\} \end{aligned}$$

**Satz 29**

$\mathrm{HP}$ und $\mathrm{HC}$ sind **NP**-vollständig (sogar für ungerichtete Graphen).

# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beweis:** Wir zeigen nur die NP-Vollständigkeit von $\mathrm{HC}$.

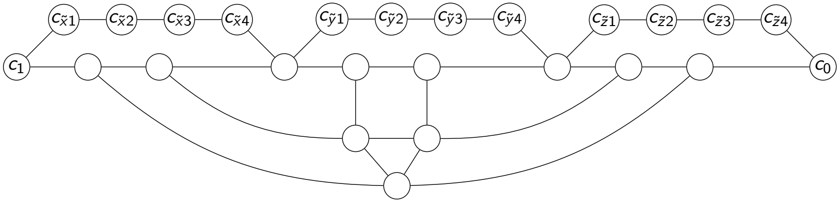
(A) $\mathrm{HC} \in$ **NP**: trivial.

(B) $3\text{-}\mathrm{SAT} \leq_m^{\log} \mathrm{HC}$:

Sei $F = \bigwedge_{c \in C} c$ eine Formel in $3\text{-}\mathrm{KNF}$. Jede Klausel $c \in C$ besteht aus 3 Literalen, und wir betrachten $c$ als 3-elementige Menge.

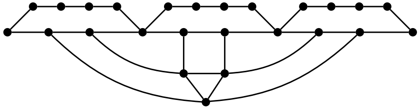
Wir konstruieren einen Graphen $G(F)$, der genau dann einen Hamilton-Kreis hat, falls $F \in \mathrm{SAT}$ gilt.

Wir definieren zu jeder Klausel $c = (\tilde{x} \vee \tilde{y} \vee \tilde{z}) \in C$ den Graphen $G(c)$:



**Beachte:**

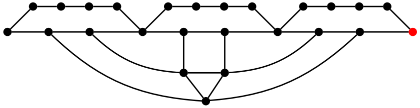
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

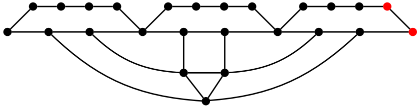
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

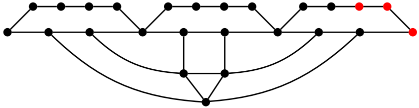
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

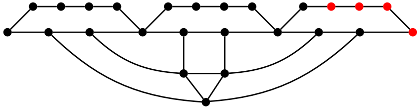
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

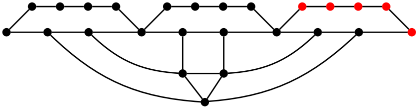
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



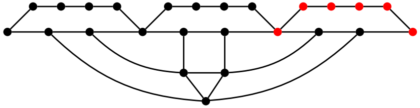
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

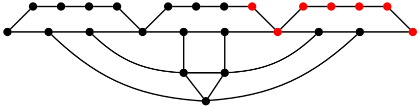
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

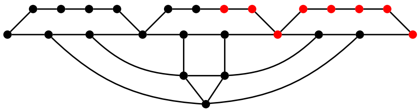
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

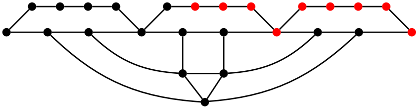
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



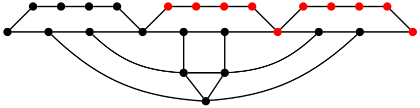
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



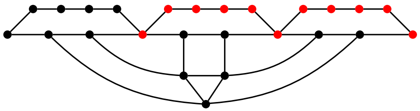
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

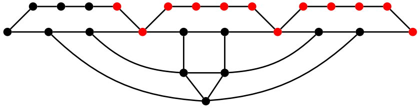
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

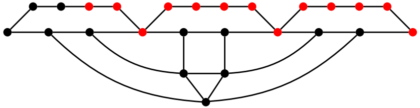
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

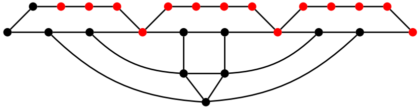
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



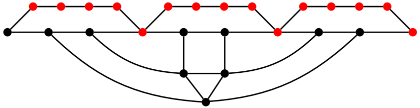
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



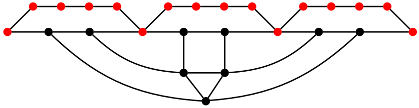
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

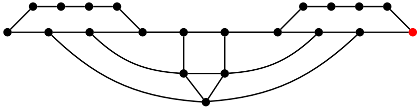
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

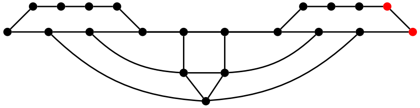
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

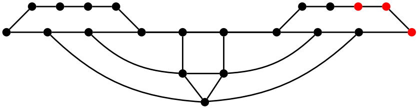
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



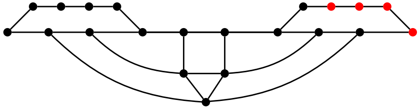
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



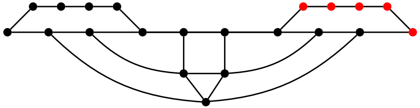
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



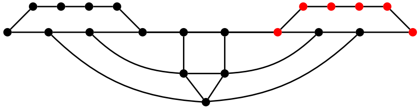
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

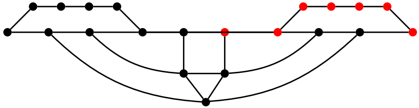
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

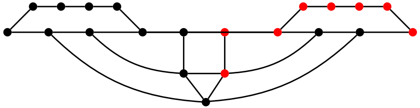
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



## Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

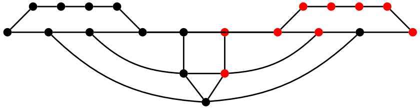
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



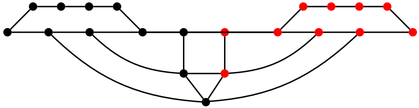
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



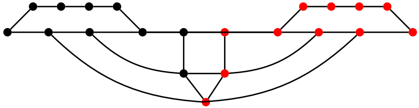
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



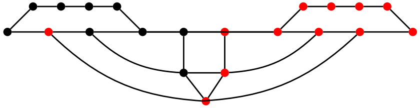
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

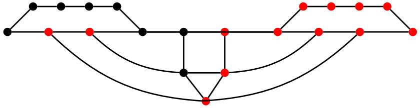
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

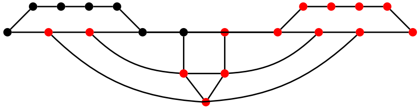
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



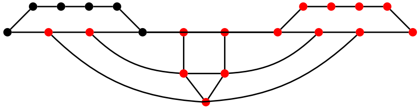
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



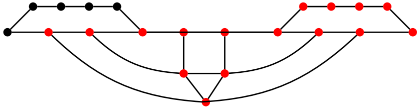
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



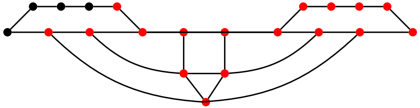
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

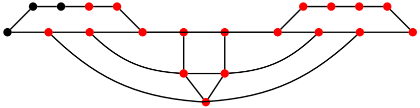
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

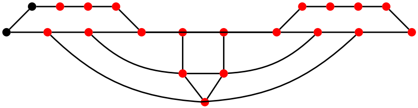
**Beachte:**

- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



**Beachte:**

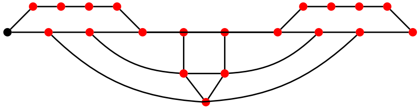
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

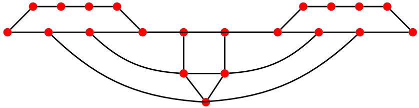
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beachte:**

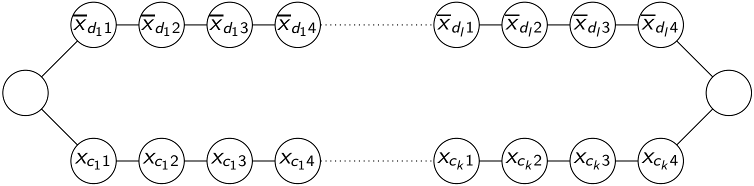
- In $G(c)$ gibt es keinen Hamilton-Pfad von $c_0$ nach $c_1$.
- Lässt man jedoch in $G(c)$ mindestens einen der Wege $c_{l1} - c_{l2} - c_{l3} - c_{l4}$, $l \in c$, weg, so gibt es einen Hamilton-Pfad von $c_0$ nach $c_1$.



## Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

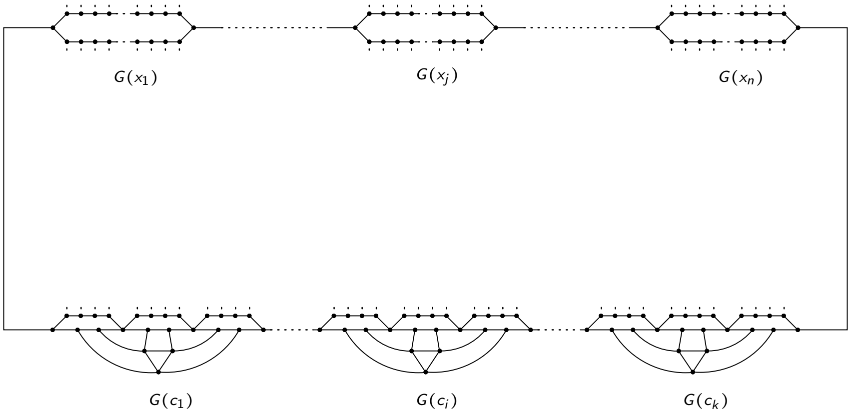
Für eine Variable $x$ sei $\{c_1, \ldots, c_k\}$ die Menge der Klauseln mit $x \in c_i$ und $\{d_1, \ldots, d_l\}$ die Menge der Klauseln mit $\overline{x} \in d_j$.

Zu jeder Variablen $x$ definieren wir nun einen Graphen $G(x)$:

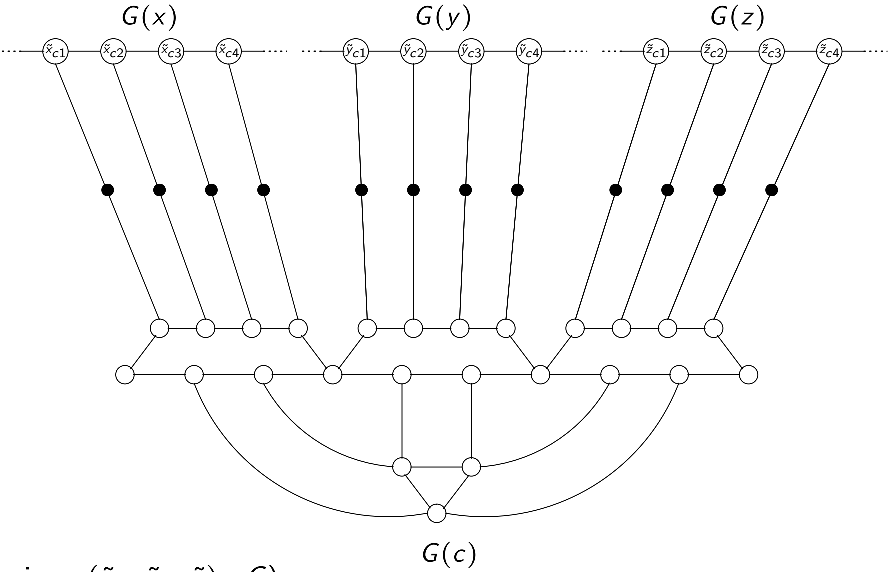


# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

Den Graphen $G(F)$ bilden wir durch Zusammenfügen der Graphen $G(c_i)$ und $G(x_j)$, wobei $C = \{c_1, \ldots, c_k\}$ und $x_1, \ldots, x_n$ are Variablen in $F$ sind.



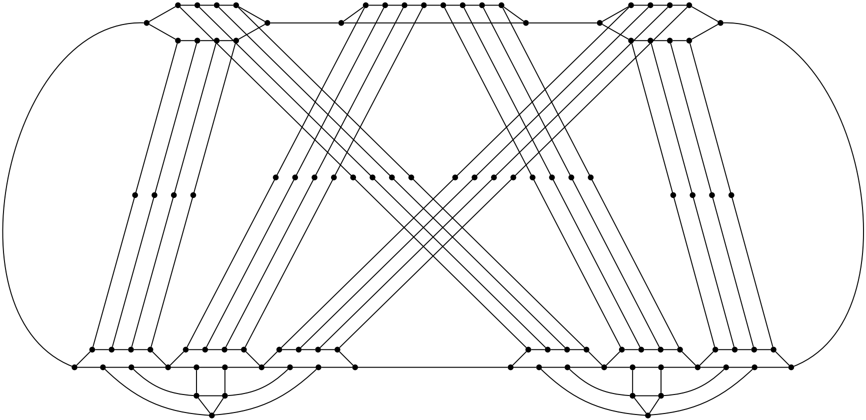

Für jede Klausel $c$, jedes Literal $\tilde{x} \in c$, und alle $1 \leq i \leq 4$ verbinden wir noch $c_{\tilde{x},i}$ (ein Knoten aus $G(c)$) und $\tilde{x}_{c,i}$ über einen Zwischenknoten:




(sei $c = (\tilde{x} \vee \tilde{y} \vee \tilde{z}) \in C$)

# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Beispiel:** Der Graph $G(F)$ für $F = \overbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}^{c_1} \wedge \overbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}^{c_2}$.



Der Hamiltonkreis, der zur Belegung $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ gehört, findet sich unter `https://www.eti.uni-siegen.de/ti/lehre/ws2021/komplexitaetstheorie/example-hamilton.pdf`.

# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Behauptung:** $F \in \mathrm{SAT} \iff G(F)$ hat einen Hamilton-Kreis.

Angenommen $\sigma$ ist eine erfüllende Belegung von $F$.

Wir erhalten einen Hamilton-Kreis für $G(F)$ wie folgt:

Der Weg führt für jede Variable $x$ über den $x$-Zweig (bzw. $\overline{x}$-Zweig), falls $\sigma(x) = 1$ (bzw. $\sigma(x) = 0$). Dabei besuchen wir über die zuletzt eingefügten Zwischenknoten in jedem Graphen $G(c)$ mindestens einen der Pfade $c_{\tilde{x}1} - c_{\tilde{x}2} - c_{\tilde{x}3} - c_{\tilde{x}4}$, wobei $\tilde{x} \in c$ ein erfülltes Literal ist.

Dies ist möglich, da in jeder Klausel mindestens ein Literal erfüllt ist.

Nachdem alle Graphen $G(x)$ durchlaufen wurden, werden die noch unbesuchten Knoten in den Teilgraphen $G(c)$ und $G(x)$ besucht. Dies ist nach Behauptung 1 möglich.

Anschließend endet der Weg im Startknoten.

# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

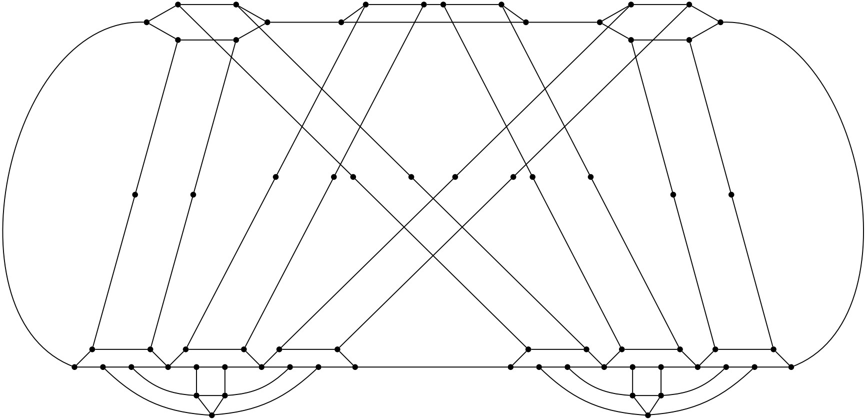
Sei $C$ ein Hamilton-Kreis für $G(F)$.

Dieser durchläuft in jedem der Graphen $G(x)$ einen der beiden Zweige.

Dies definiert eine erfüllende Belegung $\sigma$ von $F$. □

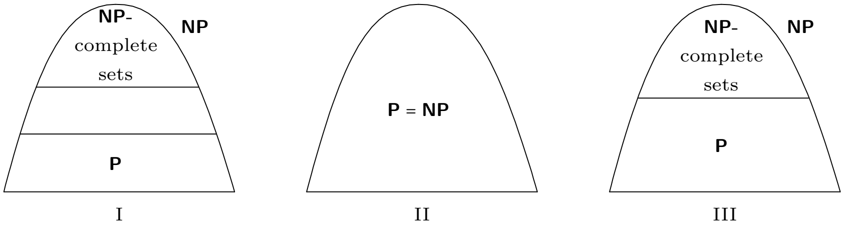
# Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

**Übung:** Würde auch folgende Konstruktion funktionieren?



# Nicht **NP**-vollständige Mengen innerhalb von **NP** ∖ **P**

Nach unserem bisherigen Wissen wären folgende 3 Situationen möglich:




Im folgenden werden wir zeigen, dass die 3. Möglichkeit prinzipiell nicht möglich ist.

**Satz 30 (Ladner)**

Ist **P** $\neq$ **NP**, dann existiert (effektiv) eine Sprache $L \in$ **NP** ∖ **P**, die nicht **NP**-vollständig (unter Polynomialzeitreduktionen) ist.

# Nicht **NP**-vollständige Mengen innerhalb von $\mathbf{NP} \setminus \mathbf{P}$

**Beweis:** Für eine schwach monoton wachsende Funktion $f : \mathbb{N} \to \mathbb{N}$ $(x_2 > x_1 \Rightarrow f(x_2) \geq f(x_1))$ sei

$$L_f = \{x \in \Sigma^* \mid x \in \mathrm{SAT} \wedge f(|x|) \text{ ist gerade}\}.$$

Im folgenden geben wir ein det. Programm an, welches eine Funktion $f : \mathbb{N} \to \mathbb{N}$ rekursiv in Zeit $\mathcal{O}(n)$ berechnet. $\leadsto$ $L_f \in \mathbf{NP}$.

Sei $M_1, M_2, \ldots$ eine Aufzählung aller deterministischen Turingmaschinen mit einer zusätzlichen polynomiellen Zeitschranke.

Eigentlich zählen wir alle Paare $(M_i, p_j)$ von deterministischen Turingmaschinen und Polynomen auf.

Wir erhalten daraus obige Aufzählung, indem $M_i$ mit einem zusätzlichen Schrittzähler, welcher die polynomiale Zeitschranke $p_j$ sicherstellt, versehen wird.

Beachte: Jedes Polynom ist zeitkonstruierbar.

# Nicht **NP**-vollständige Mengen innerhalb von **NP** $\setminus$ **P**

Sei analog $R_1, R_2, \ldots$ eine Aufzählung aller Polynomialzeitreduktionen.

Wir verwenden ferner im folgenden einen beliebigen (exponentiellen) deterministischen Algorithmus zur Erkennung von $\mathrm{SAT}$.

Für Mengen $L$ und $K$ ist

$$L \,\Delta\, K = L \setminus K \,\cup\, K \setminus L$$

die symmetrische Differez.

**Beachte:** Ist $L \in \mathcal{C}$ für eine der in der Vorlesung betrachteten Komplexitätsklassen $\mathcal{C}$ und ist $|L \,\Delta\, K| < \infty$, so gilt auch $K \in \mathcal{C}$ (bis auf endliche viele Ausnahmen kann eine Maschine für $L$ auch für $K$ verwendet werden).

# Nicht **NP**-vollständige Mengen innerhalb von **NP** $\smallsetminus$ **P**

**FUNCTION** $f(n)$
**if** $n = 0$ **then return** 0
**else** **(*** $n > 0$ ***)**
  $i := 0$; $k := 0$
  **loop** für genau $n$ Befehlsschritte
    $k := f(i)$; $i := i + 1$
  **endloop**
  **(*** Beachte: alle rekursiven Aufrufe $f(i)$ geschehen nur mit $i < n$. ***)**
  **(*** Der Wert von $k$ ist nach Durchlaufen dieser Schleife sehr klein. ***)**
  **if** $k = 2j$ **(*** $k$ gerade ***)** **then**
    suche für max. $n$ Schritte in längenlexikograph. Reihenfolge ein $x \in L(M_j) \Delta L_f$
  **endif**
  **if** $k = 2j + 1$ **(*** $k$ ungerade ***)** **then**
    suche für max. $n$ Schritte in längenlexikograph. Reihenfolge ein $x \in \Sigma^*$ mit
    $(x \in \mathrm{SAT} \wedge R_j(x) \notin L_f) \vee (x \notin \mathrm{SAT} \wedge R_j(x) \in L_f)$
  **endif**
  **if** ein solcher Zeuge $x \in \Sigma^*$ wurde gefunden **then return** $k + 1$
  **else return** $k$
  **endif**
**endif**

# Nicht **NP**-vollständige Mengen innerhalb von $\mathbf{NP} \setminus \mathbf{P}$

**Behauptung 1:** $f(n)$ ist wohldefiniert und wird in $\mathcal{O}(n)$ Schritten berechnet. Es gilt weiter $f(n) \le f(n+1) \le f(n)+1$ für alle $n \in \mathbb{N}$.

Wir zeigen nur $f(n) \le f(n+1) \le f(n)+1$ für alle $n \in \mathbb{N}$, der Rest ist klar.

Zunächst zeigen wir mit Induktion über $n$, dass $f(n) \le f(n+1)$ gilt.

Induktionsanfang: Da $f(0) = 0$ gilt, folgt $f(0) \le f(1)$.

Induktionsschritt: Sei $k_0$ (bzw. $k_1$) der im Algorithmus bei Eingabe $n$ (bzw. $n+1$) in der **loop**-Schleife berechnete $k$-Wert.

Also gibt es $i_0 < n$, $i_1 < n+1$ mit $i_0 \le i_1$, $f(i_0) = k_0$ und $f(i_1) = k_1$.

Aus der Induktionshypothese folgt $k_0 \le k_1$.

Falls $k_0 = k_1$, so folgt $f(n) \le f(n+1)$ direkt aus dem Algorithmus für $f$.

Falls $k_0 + 1 \le k_1$, so gilt $f(n) \le k_0 + 1 \le k_1 \le f(n+1)$.

Da $f(n+1) \le f(i)+1$ für ein $i \le n$ gilt, erhalten wir somit auch $f(n+1) \le f(i)+1 \le f(n)+1$.

# Nicht **NP**-vollständige Mengen innerhalb von $\mathbf{NP} \setminus \mathbf{P}$

Wir zeigen jetzt mit Widerspruch: wenn $\mathbf{P} \neq \mathbf{NP}$ gilt, so liegt $L_f$ weder in **P** noch ist $L_f$ **NP**-vollständig (beachte: $L_f \in \mathbf{NP}$).

Gelte also $\mathbf{P} \neq \mathbf{NP}$ und ($L_f \in \mathbf{P}$ oder $L_f$ ist **NP**-vollständig).

**Behauptung 2:** $f$ ist beschränkt, d.h. $\exists n_0, m \ \forall n \geq n_0 : f(n) = m$.

1. Fall: $L_f \in \mathbf{P}$

Dann existiert ein $j$ mit $L_f = L(M_j)$.

Angenommen es existiert ein $n \in \mathbb{N}$ mit $f(n) > 2j$.

Wegen Behauptung 1 existiert ein $n \in \mathbb{N}$ mit $f(n) = 2j + 1$.

Sei $n$ minimal, so dass $f(n) = 2j + 1$.

Falls bei der Berechnung von $f(n)$ kein Zeuge $x$ gefunden wird, gibt es ein $i < n$ mit $f(i) = f(n) = 2j + 1$.

Widerspruch zur Minimalität von $n$!

Falls bei der Berechnung von $f(n)$ ein Zeuge $x$ gefunden wird, muss $k = 2j$ bei der Berechnung von $f(n)$ gelten.

Also gehört der gefundene Zeuge $x$ zu $L(M_j) \,\Delta\, L_f$.

Es gilt aber $L(M_j) \,\Delta\, L_f = \emptyset$, Widerspruch!

Also gilt $f(n) \le 2j$ für alle $n$.

2. Fall: $L_f$ ist **NP**-vollständig, d.h. $\mathrm{SAT} \le_m^p L_f$.

Also gibt es ein $j \ge 0$ mit $\forall x \in \Sigma^* : x \in \mathrm{SAT} \Leftrightarrow R_j(x) \in L_f$.

$\leadsto \neg\exists x \in \Sigma^* : (x \in \mathrm{SAT} \wedge R_j(x) \notin L_f) \;\vee\; (x \notin \mathrm{SAT} \wedge R_j(x) \in L_f)$.

Wie in Fall 1 zeigt man, dass $f(n) \le 2j + 1$ für alle $n$.

# Nicht **NP**-vollständige Mengen innerhalb von $\mathbf{NP} \setminus \mathbf{P}$

Sei also $f(n) = m$ für alle $n \geq n_0$.

Aus der Definition von $L_f$ (Folie 131) folgt:

1. Ist $m$ gerade, so gilt $|L_f \,\Delta\, \mathrm{SAT}| < \infty$ und $L_f$ ist **NP**-vollständig.
2. Ist $m$ ungerade, so ist $L_f$ endlich und damit $L_f \in \mathbf{P}$.

Aus der Vorraussetzung $\mathbf{P} \neq \mathbf{NP}$ folgt nun:

- Wenn $m = 2j$, dann ist $L(M_j) = L_f$, und diese Sprache ist nach (1) **NP**-vollständig.

  Widerspruch zu $L(M_j) \in \mathbf{P}$.

- Wenn $m = 2j + 1$, dann ist $R_j$ eine Reduktion von $\mathrm{SAT}$ auf $L_f$.

  Widerspruch zu (2), da $L_f$ endlich ist und wegen $\mathbf{P} \neq \mathbf{NP}$ die Sprache $\mathrm{SAT}$ nicht auf eine Sprache in **P** reduziert werden kann. □

# Dünne Mengen und der Satz von Mahaney

Eine Menge $L \subseteq \Sigma^*$ ist dünn, wenn ein Polynom $p(n)$ mit

$$\forall n \geq 0 : \left|\left\{w \in L \mid |w| = n\right\}\right| \leq p(n).$$

existiert.

Z. B. ist jede Sprache über einem unären Alphabet $\{a\}$ dünn.

**Satz von Mahaney, 1982**

Gilt $\mathbf{P} \neq \mathbf{NP}$, so gibt es keine dünne Sprache, die $\mathbf{NP}$-schwierig (unter Polynomialzeitreduktionen) ist.

# Beweis des Satzes von Mahaney

**Beweis:** Es wird zunächst eine Menge SAT′ definiert:

$$\mathrm{SAT}' = \left\{ \langle F, x \rangle \,\middle|\, \begin{array}{l} F \text{ ist boolesche Formel in den Variablen } x_1, \ldots, x_n, \\ x \in \{0,1\}^n \text{ und } \exists y \in \{0,1\}^n : (y \geq x \wedge F(y) = 1) \end{array} \right\}$$

Hierbei ist

- $\geq$ die lexikographische Ordnung auf $\{0,1\}^*$ und
- $F(y) = 1$ bedeutet, dass der Bitstring $y$ als Belegung der Variablen $x_1, \ldots, x_n$ interpretiert wird und $F$ sich unter dieser Belegung zu wahr auswertet.

**Behauptung:** SAT′ ist **NP**-vollständig.

1. SAT′ $\in$ **NP**: Rate Belegung $y \geq x$ und überprüfe, ob $F(y) = 1$.
2. SAT′ ist **NP**-schwierig:
   O.B.d.A. enthalte $F$ die Variablen $x_1, \ldots, x_n$. Wir reduzieren SAT auf SAT′ mit der Abbildung $F \mapsto \langle F, 0^n \rangle$. Es gilt offensichtlich:

$$F \in \mathrm{SAT} \iff \langle F, 0^n \rangle \in \mathrm{SAT}'$$

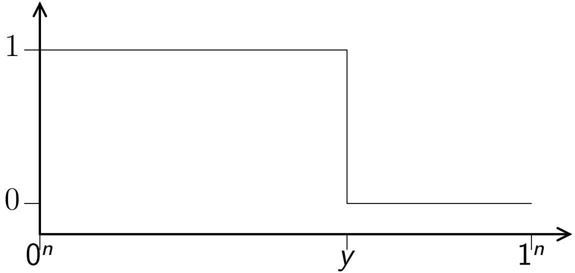
# Beweis des Satzes von Mahaney

Für eine Formel $F$ in den Variablen $x_1, \ldots, x_n$ definieren wir eine Funktion $f_F : [0, \ldots, 2^n - 1] \to \{0,1\}$ durch

$$f_F(x) = \begin{cases} 1, \text{ falls } \langle F, x \rangle \in \text{SAT}' \\ 0, \text{ sonst} \end{cases}$$

Der Definitionsbereich von $f_F$ wird hier mit $\{0,1\}^n$ identifiziert.

Der Graph von $f_F$ ist eine Stufenfunktion:




Der Wert $y$ ist hier die (lexikographisch) größte erfüllende Belegung für $F$.

# Beweis des Satzes von Mahaney

**Annahme:** Sei $L \subseteq \Gamma^*$ eine dünne **NP**-schwierige Sprache.

Da $\text{SAT}' \in \mathbf{NP}$, gibt es eine Polynomialzeitreduktion $H : \Sigma^* \to \Gamma^*$ von $\text{SAT}'$ auf $L$:

$$\langle F, x\rangle \in \text{SAT}' \Leftrightarrow H(\langle F, x\rangle) \in L$$

Die Länge einer Eingabe $\langle F, x\rangle$ definieren wir im folgenden zu $n = |F|$ und wir können o.B.d.A. annehmen, dass die Formel $F$ nur Variablen aus der Menge $\{x_1, \dots, x_n\}$ enthält.

Da $L$ nach Annahme eine dünne Sprache ist, gibt es ein Polynom $p(n)$ mit

$$\left|\left\{H(\langle F, x\rangle) \mid \langle F, x\rangle \in \text{SAT}' \wedge |\langle F, x\rangle| \leq n\right\}\right| \leq p(n). \tag{6}$$

Wir zeigen, dass unter obigen Annahmen $\text{SAT} \in \mathbf{P}$, d. h. $\mathbf{P} = \mathbf{NP}$ gilt.

Sei $F$ eine boolesche Formel mit $n = |F|$, in der nur Variablen aus der Menge $\{x_1, \dots, x_n\}$ vorkommen.

Sei $I = [0, \ldots, 2^n - 1]$.

Wir werden den Verlauf der Stufenfunktion $f_F : I \to \{0,1\}$ berechnen.

Aus (6) folgt

$$\left|\left\{H(\langle F, x\rangle) \mid \langle F, x\rangle \in \mathrm{SAT}' \ \wedge \ x \in I\right\}\right| \leq p(n). \tag{7}$$

Wir unterteilen nun $I$ in $2 \cdot p(n)$ (in etwa) gleichgroße Teilintervalle.

Jedes Intervall wird durch die linke Grenze und die Länge repräsentiert.

Die linken Grenzen seien $y_1 < y_2 < \cdots < y_{2p(n)}$.

Für die linken Grenzen $y_i$ wird $H(\langle F, y_i\rangle)$ berechnet und in einer Tabelle abgelegt.

# Beweis des Satzes von Mahaney

Angenommen es gilt $H(\langle F, y_i \rangle) = H(\langle F, y_j \rangle)$ für $i < j$.

Dann gilt $\langle F, y_i \rangle \in \mathrm{SAT}' \Leftrightarrow \langle F, y_j \rangle \in \mathrm{SAT}'$ und damit $f_F(y_i) = f_F(y_j)$, d.h.

$$f_F(y_i) = f_F(y_{i+1}) = \cdots = f_F(y_{j-1}) = f_F(y_j).$$

Sollte $F$ erfüllbar sein, kann die größte erfüllende Belegung $y$ von $F$ in keinem der Intervalle mit den linken Grenzen $y_i, y_{i+1}, \ldots, y_{j-1}$ liegen.

Ansonsten: $y_i \leq y < y_j \;\Rightarrow\; f_F(y_i) \neq f_F(y_j) \;\Rightarrow\; H(\langle F, y_i \rangle) \neq H(\langle F, y_j \rangle)$

**Kollisionsauflösung**

Lösche alle Teilintervalle mit linken Grenzen $y_i, y_{i+1}, \ldots, y_{j-1}$.

Dabei wird das Intervall, das die größte erfüllende Belegung $y$ enthält (sofern überhaupt eine erfüllende Belegung existiert), nicht gelöscht.

# Beweis des Satzes von Mahaney

Seien $z_1 < z_2 < \ldots < z_l$ die verbleibenden linken Intervallgrenzen.

Am Ende dieser Prozedur sind die Werte $H(\langle F, z_i \rangle)$ für alle linken Grenzen der verbleibenden Intervalle verschieden.

Angenommen $F$ ist erfüllbar und für die größte erfüllende Belegung $y$ von $F$ gilt $z_{p(n)+1} \leq y$.

Dann gilt $z_i \leq y$ und somit $\langle F, z_i \rangle \in \mathrm{SAT}'$ für alle $1 \leq i \leq p(n) + 1$.

Also gilt $H(\langle F, z_i \rangle) \in L$ für alle $1 \leq i \leq p(n) + 1$.

Dies widerspricht (7)!

Also gehört $y$ zu keinem der Intervalle mit den linken Grenzen $z_{p(n)+1}, \ldots, z_l$.

Wir löschen daher auch diese Intervalle.

Danach bleiben nur noch höchsten $p(n)$ Intervalle übrig.

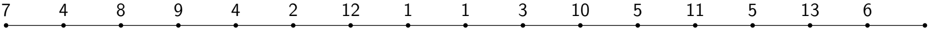
# Beweis des Satzes von Mahaney

Jetzt wird eine Intervallhalbierung bei den verbleibenden Intervallen durchgeführt.

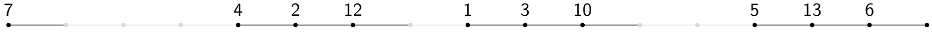
Dies erzeugt maximal $2 \cdot p(n)$ Intervalle, die nach dem gleichen Verfahren wiederum auf maximal $p(n)$ Intervalle verringert werden. Dann erfolgt eine erneute Intervallhalbierung usw.

Nach höchstens $n$ Halbierungen gibt es nur noch 1-Punkt-Intervalle (das Intervall $I$ enthält $2^n$ Elemente) und das Verfahren der Intervallhalbierungen wird abgebrochen.

**Example**: $p(n) = 8$

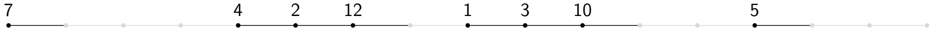


**Example**: $p(n) = 8$

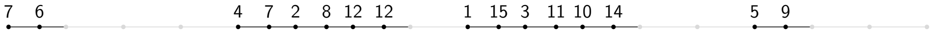


# Beweis des Satzes von Mahaney

**Example**: $p(n) = 8$

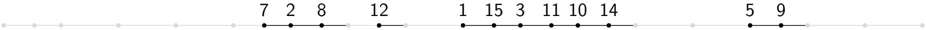


**Example**: $p(n) = 8$



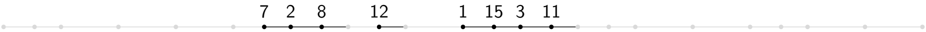
# Beweis des Satzes von Mahaney

**Example**: $p(n) = 8$



# Beweis des Satzes von Mahaney

**Example**: $p(n) = 8$



# Beweis des Satzes von Mahaney

Die bis zu diesem Zeitpunkt verbrauchte Zeit ist polynomial beschränkt.

Seien $z_1, \dots, z_l$ ($l \le p(n)$) die linken Grenzen der verbleibenden 1-Punkt-Intervalle.

Jetzt kann in polynomieller Zeit für jedes $i$, $1 \le i \le l$, der Wert $F(z_i)$ berechnet werden.

Ist $F(z_i) = 0$ für alle $i$, so ist $F$ unerfüllbar. Sonst ist $F$ erfüllbar (das größte $z_i$ mit $F(z_i) = 1$ ergibt die größte erfüllende Belegung).

Damit können wir in Polynomialzeit entscheiden, ob $F \in \mathrm{SAT}$. $\square$

# Vollständige Probleme für **P**

Sei $L_{cfe} = \{\langle G \rangle \mid G$ ist eine kontextfreie Grammatik mit $L(G) \neq \emptyset\}$.
Dabei stellen die spitzen Klammern $\langle\rangle$ eine geeignete Kodierung von Grammatiken dar, *cfe* steht für context–free–empty.

**Satz 31**

$L_{cfe}$ ist **P**-vollständig.

**Beweis:**

**(A)** $L_{cfe} \in \mathbf{P}$

Teste für eine gegebene Grammatik $G$, ob das Startsymbol $S$ produktiv ist.

Sei $P$ die Menge der Produktionen von $G$, $\Sigma$ die Menge der Terminale und $N$ die Menge der Nichtterminale.

Ein Nichtterminal $A$ ist produktiv, falls ein Wort $w \in \Sigma^*$ mit $A \Rightarrow_G^* w$ existiert.

# Vollständige Probleme für **P**

Folgender Algorithmus berechnet die Menge der produktiven Nichtterminale in polynomieller Zeit:

$M := \{A \in N \mid \text{ es existiert } (A \to w) \in P \text{ mit } w \in \Sigma^*\};$
$M' = \varnothing;$
**while** $M \neq M'$ **do**
  $M' := M;$
  $M := M' \cup \{A \in N \mid \text{ es existiert } (A \to w) \in P \text{ mit } w \in (M' \cup \Sigma)^*\};$
**endwhile**

**(B)** $L_{cfe}$ ist **P**-schwierig.

Sei $L \in \mathbf{P}$ und $L(M) = L$ für eine $p(n)$-zeitbeschränkte deterministische Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$, $p(n) > n$ ein Polynom.

Sei $w = w_1 \cdots w_n \in \Sigma^*$ eine Eingabe für $M$ mit $|w| = n \geq 1$.

# Leerheit kontextfreier Sprachen ist **P**-vollständig

Wir setzen für $M$ o.B.d.A. ähnliche Konventionen wie im Beweis des Satzes von Cook voraus, wobei $\Omega = (Q \times \Gamma) \cup \Gamma$ (Folie 88 und 90):

1. Konfigurationen von $M$ werden durch Wörter aus der Sprache $\mathrm{Conf} = \{\Box u(q,a)v\Box \mid (q,a) \in Q \times \Gamma,\ uv \in \Gamma^{2p(n)}\}$ beschrieben.
2. Die Startkonfiguration ist $\alpha_0 := \Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$.
3. $w \in L(M)$ genau dann, wenn $M$ nach höchstens $p(n)$ Schritten den akzeptierenden Zustand $q_J$ erreicht.

Da $M$ deterministisch ist, wird die Relation $\Delta \subseteq \Omega^4$ aus dem Satz von Cook (Folie 92) zu einer Funktion $\Delta : \Omega^3 \to \Omega$, so dass für alle Wörter $\alpha, \alpha' \in \Box\Omega^*\Box$ mit $|\alpha| = |\alpha'|$ gilt:

$$\alpha, \alpha' \in \mathrm{Conf} \text{ und } \alpha \vdash_M \alpha'$$

$$\Longleftrightarrow$$

$$\alpha \in \mathrm{Conf} \text{ und } \forall i \in \{-p(n), \ldots, p(n)\} : \Delta(\alpha[i-1], \alpha[i], \alpha[i+1]) = \alpha'[i].$$

# Leerheit kontextfreier Sprachen ist **P**-vollständig

Wir definieren jetzt die Grammatik $G(w) = (V, \varnothing, P, S)$ mit der Variablenmenge

$$V = \{S\} \cup \{V(a,t,j) \mid a \in \Omega,\ 0 \le t \le p(n),\ |j| \le p(n)+1\},$$

leerem Terminalalphabet, Startsymbol $S$ und der Produktionsmenge $P$, bestehend aus ($\lambda$ = leeres Wort):

- $S \to V((q_J, a), t, j)$ für $0 \le t \le p(n), |j| \le p(n)+1, a \in \Gamma$
- $V(a, t+1, j) \to V(b, t, j-1) V(c, t, j) V(d, t, j+1)$
  falls $\Delta(b,c,d) = a, 0 \le t \le p(n)-1, |j| \le p(n)$
- $V(\Box, t, j) \to \lambda$ für $0 \le t \le p(n), |j| = p(n)+1$,
- $V((q_0, w_1), 0, 0) \to \lambda$,
- $V(w_{j+1}, 0, j) \to \lambda$ für $1 \le j \le n-1$,
- $V(\Box, 0, j) \to \lambda$ für $j \in \{-p(n), \dots, -1\} \cup \{n, \dots, p(n)\}$

# Leerheit kontextfreier Sprachen ist **P**-vollständig

**Behauptung:** $L(G(w)) \neq \emptyset \iff w \in L$.

Sei $\alpha_0 \vdash_M \alpha_1 \vdash_M \cdots \vdash_M \alpha_{p(n)}$ ($\alpha_i \in \mathrm{Conf}$) die eindeutige bei der Startkonfiguration $\alpha_0$ beginnende Berechnung.

Für $-p(n) - 1 \leq j \leq p(n) + 1$ und $0 \leq t \leq p(n)$ sei $\alpha(t,j) = \alpha_t[j]$.

Wir zeigen die Behauptung durch den Beweis der folgenden allgemeineren Aussage, wobei $L(V(a,t,j)) \subseteq \{\lambda\}$ die Menge aller Terminalwörter ist, die von $V(a,t,j)$ abgeleitet werden können:

$$L(V(a,t,j)) \neq \emptyset \iff \alpha(t,j) = a$$

$\Longleftarrow$: Sei $\alpha(t,j) = a$.

Die Fälle $t = 0$ und $j \in \{-p(n) - 1, p(n) + 1\}$ folgen sofort aus der Definition von $G(w)$.

# Leerheit kontextfreier Sprachen ist **P**-vollständig

Falls $t \geq 1$ und $-p(n) \leq j \leq p(n)$, so gibt es $b, c, d \in \Omega$ mit $\Delta(b, c, d) = a$ und

- $\alpha(t-1, j-1) = b$,
- $\alpha(t-1, j) = c$,
- $\alpha(t-1, j+1) = d$.

Mit Induktion über $t$ folgt

- $L(V(b, t-1, j-1)) \neq \varnothing$,
- $L(V(c, t-1, j)) \neq \varnothing$,
- $L(V(d, t-1, j+1)) \neq \varnothing$.

Da $G(w)$ die Produktion

$$V(a, t, j) \to V(b, t-1, j-1)V(c, t-1, j)V(d, t-1, j+1)$$

enthält, folgt $L(V(a, t, j)) \neq \varnothing$.

# Leerheit kontextfreier Sprachen ist $\mathbf{P}$-vollständig

$\Longrightarrow$: Sei $L(V(a,t,j)) \neq \emptyset$.

Die Fälle $t = 0$ und $j \in \{-p(n)-1, p(n)+1\}$ folgen sofort aus der Definition von $G(w)$.

Falls $t \geq 1$ und $-p(n) \leq j \leq p(n)$, so muss es eine Produktion

$$V(a,t,j) \to V(b,t-1,j-1)V(c,t-1,j)V(d,t-1,j+1)$$

geben (inbesondere $\Delta(b,c,d) = a$), so dass

- $L(V(b,t-1,j-1)) \neq \emptyset$,
- $L(V(c,t-1,j)) \neq \emptyset$,
- $L(V(d,t-1,j+1)) \neq \emptyset$.

Induktion $\Rightarrow$ $\alpha(t-1,j-1) = b$, $\alpha(t-1,j) = c$, $\alpha(t-1,j+1) = d$.

Wegen $\Delta(b,c,d) = a$ folgt $\alpha(t,j) = a$. $\square$

## Definition boolescher Schaltkreis

Ein boolescher Schaltkreis $C$ ist ein gerichteter markierter Graph $C = (\{1, \ldots, o\}, E, s)$ für ein $o \in \mathbb{N}$ mit folgenden Bedingungen.

- $\forall (i,j) \in E : i < j$, d.h. $C$ ist azyklisch.
- $s : \{1, \ldots, o\} \to \{\neg, \wedge, \vee, 0, 1\}$ wobei gilt:

$$\begin{aligned} s(i) \in \{\wedge, \vee\} &\quad\Rightarrow\quad \text{Eingangsgrad}(i) = 2 \\ s(i) = \neg &\quad\Rightarrow\quad \text{Eingangsgrad}(i) = 1 \\ s(i) \in \{0, 1\} &\quad\Rightarrow\quad \text{Eingangsgrad}(i) = 0 \end{aligned}$$

  $s(i)$ ist die Sorte von Knoten $i$.

Die Knoten werden als Gatter bezeichnet.

Das Gatter $o$ (output) ist das Ausgangsgatter von $C$.

# Boolesche Schaltkreise

Durch Auswerten (im intuitiven Sinne) des Schaltkreises $C$ kann jedem Gatter $i$ ein Wert $v(i) \in \{0,1\}$ zugeordnet werden.

Ein Schaltkreis ist monoton, falls er keine $\neg$-Gatter enthält.

Circuit Value $(\mathrm{CV})$ ist das folgende Problem:

**INPUT:** Ein boolescher Schaltkreis $C$

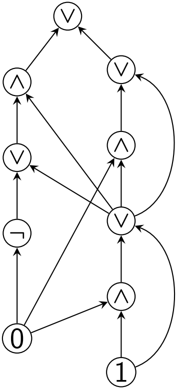
**FRAGE:** Wertet sich das Ausgangsgatter von $C$ zu 1 aus?

Monotone Circuit Value $(\mathrm{MCV})$ ist das folgende Problem:

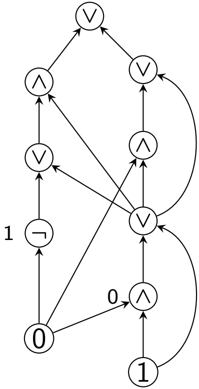
**INPUT:** Ein monotoner boolescher Schaltkreis $C$

**FRAGE:** Wertet sich das Ausgangsgatter von $C$ zu 1 aus?

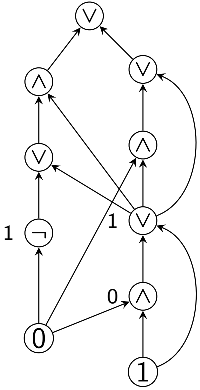
**Example:**



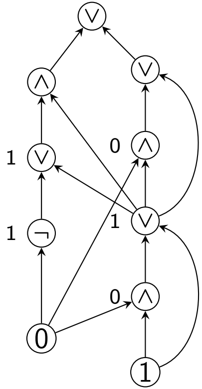
**Example:**



**Example:**

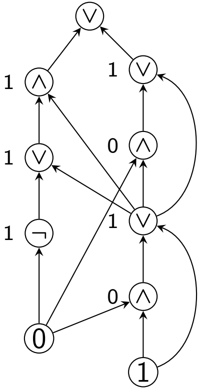


**Example:**

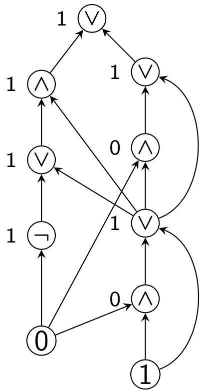


# Boolesche Schaltkreise

**Example:**



**Example:**



# Circuit Value ist **P**-vollständig

**Satz 32**

$\mathrm{CV}$ und $\mathrm{MCV}$ sind **P**-vollständig.

**Beweis:**

**(i)** $\mathrm{CV} \in \mathbf{P}$: Klar, werte alle Gatter in der Reihnfolge $1, 2, \ldots, o$ aus.

**(ii)** $\mathrm{MCV}$ ist **P**-schwierig:

Betrachte den Beweis zur **P**-Vollständigkeit von $L_{cfe}$.

Zu einer Sprache $L \in \mathbf{P}$ und einer Eingabe $w \in \Sigma^*$ wurde eine kontextfreie Grammatik $G(w)$ konstruiert mit: $w \in L$ genau dann, wenn $\lambda \in L(G(w))$.

Alle Produktionen von $G(w)$ sind von der Form $A \to \alpha$ wobei $\alpha$ eine (evtl. leere) Folge von Nichtterminalen ist.

Außerdem ist $G(w)$ azyklisch (keine Ableitungen der Form $A \Rightarrow^+ uAv$ möglich).

# Circuit Value ist **P**-vollständig

**1. Schritt:** Ersetze jede Produktion $A \to A_1 A_2 \cdots A_n$ mit $n \geq 3$ durch

$$A \to A_1 A_2',\ A_i' \to A_i A_{i+1}'\ (2 \leq i \leq n-2),\ A_{n-1}' \to A_{n-1} A_n$$

für neue Nichtterminale $A_2', \ldots, A_{n-1}'$.

**2. Schritt:** Ersetze jede Produktion $A \to B$ durch $A \to BB$.

Alle Produktionen sind nun vom Typ $A \to \lambda$ oder $A \to BC$.

**3. Schritt:** Für jedes Nichtterminal $A$, das in mindestens 2 Produktionen links steht, d. h. $A \to \alpha_1 | \alpha_2 | \cdots | \alpha_n$ ($n \geq 2$), ersetzen wir diese $n$ Produktionen durch

$$A \to A_1 | A_2,\ A_1 \to \alpha_1,\ A_2 \to \alpha_2$$

falls $n = 2$ ($A_1, A_2$ sind neue Nichtterminale) bzw.

$$A \to A_1 | A_2',\ A_i' \to A_i | A_{i+1}'\ (2 \leq i \leq n-2),$$
$$A_{n-1}' \to A_{n-1} | A_n,\ A_i \to \alpha_i\ (1 \leq i \leq n).$$

falls $n \geq 3$ ($A_1, \ldots, A_n, A_2', \ldots, A_{n-1}'$ sind neue Nichtterminale).

# Circuit Value ist **P**-vollständig

Nun gilt für jedes Nichtterminal $A$ genau einer der 4 folgenden Fälle:

1. Es gibt keine Produktion für $A$.
2. $A$ steht in genau einer Produktion auf der linken Seite, und diese Produktion ist vom Typ $A \to \lambda$.
3. $A$ steht in genau einer Produktion auf der linken Seite, und diese Produktion ist vom Typ $A \to BC$.
4. $A$ steht in genau zwei Produktion auf der linken Seite und diese Produktionen sind beide vom Typ $A \to B$: $A \to B|C$

Die neue Grammatik erzeugt $\lambda$ genau dann, wenn die alte Grammatik dies tut.

Wir bezeichnen die neue Grammatik wieder mit $G(w)$.

# Circuit Value ist **P**-vollständig

Wir definieren nun einen Schaltkreis $C(w)$ wie folgt:

Jedem Nichtterminal von $G(w)$ wird nun ein Gatter in $C(w)$ zugeordnet.

Das Startsymbol von $G(w)$ ist das Ausgangsgatter von $C(w)$.

1. Ein Nichtterminal $A$ vom Typ 1 wird ein 0-Inputgatter.
2. Ein Nichtterminal $A$ vom Typ 2 wird ein 1-Inputgatter
3. Ein Nichtterminal $A$ vom Typ 3 wird ein $\wedge$-Gatter mit Eingängen $B$ und $C$.
4. Ein Nichtterminal $A$ vom Typ 4 wird ein $\vee$-Gatter mit Eingängen $B$ und $C$.

# Circuit Value ist **P**-vollständig

Der so erzeugte Schaltkreis $C(v)$ ist in der Tat azyklisch, denn $G(w)$ ist azyklisch.

Es gilt: $L(A) \neq \emptyset \iff$ Gatter $A$ wertet sich in $C(w)$ zu 1 aus.

Also: $L(G(w)) \neq \emptyset \iff$ Ausgangsgatter von $C(w)$ wertet sich zu 1 aus. $\square$

**Bemerkung:** In einem Schaltkreis kann ein Gatter Ausgangsgrad $> 1$ haben. Dies scheint für die **P**-Schwierigkeit wichtig zu sein:

Die Menge der (variablen-freien) booleschen Ausdrücke kann durch folgende Grammatik definiert werden:

$$A ::= 0 \mid 1 \mid (\neg A) \mid (A \wedge A') \mid (A \vee A')$$

Boolesche Ausdrücke werden somit zu Bäumen, wenn man sie in Schaltkreise umwandelt.

**Buss 1987:** Die Menge der booleschen Ausdrücke, die sich zu 1 auswerten ist vollständig für die Komplexitätsklasse $\mathbf{NC}^1 \subseteq \mathbf{L}$.

# Vollständige Probleme für **PSPACE**: Quantifizierte Boolesche Formeln

## Quantifizierte boolesche Formeln

Die Menge $M$ der quantifizierten booleschen Formeln ist die kleinste Menge mit:

- $x_i \in M$ für alle $i \geq 1$
- $0, 1 \in M$
- $E, F \in M, i \geq 1 \implies (\neg E), (E \wedge F), (E \vee F), \forall x_i E, \exists x_i E \in M$

Alternativ: $M$ lässt sich durch eine eindeutige kontextfreie Grammatik über dem Terminalalphabet $\Sigma = \{x, 0, 1, (, ), \neg, \wedge, \vee, \forall, \exists\}$ erzeugen.

Dabei werden Variablen durch Wörter aus $x1\{0,1\}^*$ dargestellt.

**Beispiel:** $\forall x_1 \exists x_2 \exists x_3 ((x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3))$

# Erfüllbarkeit von quantifizierten booleschen Formeln

Die Erfüllbarkeit von quantifizierten booleschen Formeln wird durch die Existenz einer erfüllenden Belegung definiert.

Eine Belegung ist eine Funktion $b : \{x_1, x_2, \ldots\} \to \{0,1\}$.

Diese kann für eine gegebene Formel $F$ auf die in $F$ vorkommenden Variablen eingeschränkt werden.

Für $z \in \{0,1\}$ und eine Belegung $b$ sei $b[x_j \mapsto z]$ die Belegung mit

- $b[x_j \mapsto z](x_i) = b(x_i)$ für $i \neq j$ und
- $b[x_j \mapsto z](x_j) = z$.

# Erfüllbarkeit von quantifizierten booleschen Formeln

Induktive Definition der Erfüllbarkeit einer Formel $F$ bezüglich einer Belegung $b$:

Die Belegung $b$ erfüllt die Formel $F$ genau dann, wenn eine der folgenden Bedingungen zutrifft:

$$\begin{array}{lll} F = 1, & & \\ F = x_j & \text{und} & b(x_j) = 1, \\ F = (\neg E) & \text{und} & b \text{ erfüllt } E \text{ nicht}, \\ F = (F_1 \wedge F_2) & \text{und} & b \text{ erfüllt } F_1 \text{ und } F_2, \\ F = (F_1 \vee F_2) & \text{und} & b \text{ erfüllt } F_1 \text{ oder } F_2, \\ F = \exists x_j E & \text{und} & b[x_j \mapsto 0] \text{ oder } b[x_j \mapsto 1] \text{ erfüllt } E, \\ F = \forall x_j E & \text{und} & b[x_j \mapsto 0] \text{ und } b[x_j \mapsto 1] \text{ erfüllen } E. \end{array}$$

Wird $F$ von jeder Belegung erfüllt, so heißt $F$ gültig.

# Erfüllbarkeit von quantifizierten booleschen Formeln

Die Menge $\mathrm{Frei}(F)$ der freien Variablen der Formel $F$ ist wie folgt definiert:

- $\mathrm{Frei}(0) = \mathrm{Frei}(1) = \emptyset$
- $\mathrm{Frei}(x_i) = \{x_i\}$
- $\mathrm{Frei}(\neg F) = \mathrm{Frei}(F)$
- $\mathrm{Frei}((F \wedge G)) = \mathrm{Frei}((F \vee G)) = \mathrm{Frei}(F) \cup \mathrm{Frei}(G)$
- $\mathrm{Frei}(\exists x_j F) = \mathrm{Frei}(\forall x_j F) = \mathrm{Frei}(F) \setminus \{x_j\}$

Eine Formel $F$ mit $\mathrm{Frei}(F) = \emptyset$ nennt man geschlossen.

**Beachte:** Die Erfüllbarkeit einer geschlossenen Formel ist nicht von der Belegung abhängig, d. h. existiert eine erfüllende Belegung, so ist die Formel bereits gültig.

**Bezeichnung:** $\mathrm{QBF}$ ist die Menge der geschlossenen quantifizierten booleschen Formeln, die gültig sind.

### Satz 33

QBF ist **PSPACE**-vollständig.

**Beweis:**

**(i)** QBF $\in$ **PSPACE**:

Sei $E$ eine geschlossene quantifizierte boolesche Formel, in der die Variablen $x_1, \dots, x_n$ vorkommen.

O.b.d.A. ist $E$ nur aus $1, x_1, \dots, x_n, \neg, \wedge, \exists$ aufgebaut, und keine Variable $x_i$ wird in $E$ zweimal quantifiziert (der Algorithmus auf der nächsten Folie, erweitert um 0 und $\vee$, würde z.B. für $\exists x((\exists x\, 0) \vee x)$ ein falsches Ergebnis liefern).

Der folgende rekursive deterministische Algorithmus, in dem $x_1, \dots, x_n$ globale Variablen sind, überprüft in polynomiellen Platz, ob $E$ gültig ist.

# QBF ist **PSPACE**-vollständig

```
FUNCTION check(F)
  if F = 1 then return(1)
  elseif F = x_i then return(x_i)
  elseif F = (¬G) then return(not check(G))
  elseif F = (F_1 ∧ F_2) then return(check(F_1) and  check(F_2))
  elseif F = ∃x_i G then
    x_i := 1
    if check(G) = 1 then
      return(1)
    else
      x_i := 0
      return(check(G))
    endif
  endif
ENDFUNC
```

**(ii)** QBF ist **PSPACE**-schwierig:

Sei $L \in$ **PSPACE** und $L(M) = L$ für eine $p(n)$-platzbeschränkte deterministische Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$, $p(n) > n$ ist ein Polynom.

Sei $w = w_1 \cdots w_n \in \Sigma^*$ eine Eingabe für $M$ mit $|w| = n \geq 1$.

Wir setzen für $M$ o.B.d.A. ähnliche Konventionen wie im Beweis des Satzes von Cook voraus, wobei $\Omega = (Q \times \Gamma) \cup \Gamma$:

1. Konfigurationen von $M$ werden durch Wörter aus der Sprache $\mathrm{Conf} = \{\Box u(q,a)v\Box \mid (q,a) \in Q \times \Gamma,\ uv \in \Gamma^{2p(n)}\}$ beschrieben.
2. $\mathrm{Start}(w) = \Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$.
3. $\alpha_f = \Box^{p(n)+1}(q_J, \Box)\Box^{p(n)+1}$ ist o.B.d.A. die einzige akzeptierende Konfiguration, die von $\mathrm{Start}(w)$ aus möglicherweise erreichbar ist.

# QBF ist **PSPACE**-vollständig

Es existiert eine Funktion $\Delta : \Omega^3 \to \Omega$, so dass für alle $\alpha, \alpha' \in \Box\Omega^*\Box$ mit $|\alpha| = |\alpha'|$ gilt:

$$\alpha, \alpha' \in \mathrm{Conf} \text{ und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \mathrm{Conf} \text{ und } \forall i \in \{-p(n), \dots, p(n)\} : \Delta(\alpha[i-1], \alpha[i], \alpha[i+1]) = \alpha'[i].$$

Ausserdem gibt eine Konstante $c$, so dass die Menge der von $\mathrm{Start}(w)$ aus erreichbaren Konfigurationen höchstens $2^{c \cdot p(n)}$ Elemente hat (Lemma 3).

Betrachte den Savitch-Ansatz:

$$\mathrm{Reach}(\mathrm{Start}(w), \alpha_f, c \cdot p(n)) \Longleftrightarrow w \in L$$

$$\mathrm{Reach}(\alpha, \beta, i) = \exists \gamma \Big( \mathrm{Reach}(\alpha, \gamma, i-1) \wedge \mathrm{Reach}(\gamma, \beta, i-1) \Big) \quad \text{für } i > 0$$

$$\mathrm{Reach}(\alpha, \beta, 0) = \alpha \vdash_M^{\leq 1} \beta$$

# QBF ist **PSPACE**-vollständig

Die direkte Einsetzung würde auf eine Formel exponentieller Länge führen.

**Lösung:** Wir führen für die Konfigurationen Konfigurationsvariablen $X, Y, U, V, \ldots$ ein, die über die Menge Conf laufen, und definieren für $i > 0$:

$$\begin{aligned} &\mathrm{Reach}(X, Y, i) := \\ &\qquad \exists U \, \forall V \, \forall W \left( \begin{array}{l} ((V = X \wedge W = U) \vee (V = U \wedge W = Y)) \\ \to \mathrm{Reach}(V, W, i-1) \end{array} \right) \end{aligned}$$

**1. Schritt:** Berechne für Eingabe $w$ durch iterierte Anwendung obiger Rekursion, beginnend mit $\mathrm{Reach}(\mathrm{Start}(w), \alpha_f, c \cdot p(n))$, eine Formel $F$ der Größe $\mathcal{O}(c \cdot p(n))$ in den Konfigurationsvariablen $X, Y, \ldots$.

In $F$ kommen atomare Formeln der Gestalt $\mathrm{Reach}(X, Y, 0)$ und $X = Y$ sowie die Konstanten $\mathrm{Start}(w)$ und $\alpha_f$ vor.

**2. Schritt:** Wir wandeln $F$ in eine geschlossene quantifizierte boolesche Formel um:

- Wir kodieren eine Konfiguration $X$ durch eine Belegung für die Booleschen Variablen $x_{a,i}$ für $a \in \Omega$ und $|i| \leq p(n) + 1$.

  Intuition: $x_{a,i} = 1$ g.d.w. in der Konfiguration $X$ an Position $i$ das Symbol $a$ steht.

- Es gibt dann eine boolesche Formel $\gamma((x_{a,i})_{a \in \Omega, |i| \leq p(n)+1})$ der Größe $\mathcal{O}(p(n))$, die sich für ein Belegung der $x_{a,i}$ genau dann zu 1 auswertet, wenn die Belegung eine gültige Konfiguration beschreibt.

- Die Konstanten $\mathrm{Start}(w)$ und $\alpha_f$ werden durch konkrete Wahrheitswerte ersetzt.

# $\mathrm{QBF}$ ist **PSPACE**-vollständig

- $\forall X \cdots$ bzw. $\exists X \cdots$ wird somit zum Quantoren-Block

  $\forall x_{a,i}\ (a \in \Omega, |i| \leq p(n)+1) : \gamma((x_{a,i})_{a\in\Omega,|i|\leq p(n)+1}) \to \cdots$ bzw.

  $\exists x_{a,i}\ (a \in \Omega, |i| \leq p(n)+1) : \gamma((x_{a,i})_{a\in\Omega,|i|\leq p(n)+1}) \wedge \cdots$

- $X = Y$ ersetzen wir durch die Formel $\bigwedge_{a\in\Omega,|i|\leq p(n)+1}(x_{a,i} \leftrightarrow y_{a,i})$.

- Aus einer atomaren Formel $\mathrm{Reach}(X, Y, 0)$ wird $X = Y \vee X \vdash_M Y$, wobei $X \vdash_M Y$ ersetzt wird durch

$$\bigwedge_{|i|\leq p(n)} \bigvee_{(a,b,c)\in\Delta} (x_{a,i-1} \wedge x_{b,i} \wedge x_{c,i+1} \wedge y_{\Delta(a,b,c),i})$$

Wir erhalten so eine geschlossene quantifizierte boolesche Formel, die genau dann erfüllbar ist, wenn $w \in L$. $\square$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

**Erinnerung:** Für ein endliches Alphabet $\Sigma$ ist die Menge $\mathrm{Reg}(\Sigma)$ der regulären Ausdrücke über $\Sigma$ induktiv wie folgt definiert:

- $\emptyset, \varepsilon \in \mathrm{Reg}(\Sigma)$
- $\Sigma \subseteq \mathrm{Reg}(\Sigma)$
- Wenn $\alpha, \beta \in \mathrm{Reg}(\Sigma)$, dann auch $(\alpha \cup \beta), (\alpha \cdot \beta), \alpha^* \in \mathrm{Reg}(\Sigma)$

Die durch einen regulären Ausdruck $\alpha$ definierte Sprache $L \subseteq \Sigma^*$ ist wie folgt induktiv definiert:

- $L(\emptyset) = \emptyset,\ L(\varepsilon) = \{\lambda\}$
- $L(a) = \{a\}$ für $a \in \Sigma$
- $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta),\ L(\alpha \cdot \beta) = L(\alpha)L(\beta),\ L(\alpha^*) = L(\alpha)^*$

Seien

$$\begin{aligned}
\mathrm{Reg\ddot{A}quiv}(\Sigma) &= \{(\alpha, \beta) \mid \alpha, \beta \in \mathrm{Reg}(\Sigma), L(\alpha) = L(\beta)\} \\
\mathrm{RegUniv}(\Sigma) &= \{\alpha \mid \alpha \in \mathrm{Reg}(\Sigma), L(\alpha) = \Sigma^*\}
\end{aligned}$$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

**Satz 34**

RegÄquiv(Σ) und RegUniv(Σ) sind **PSPACE**-vollständig für jedes endliche Alphabet Σ mit $|\Sigma| \geq 2$.

**Beweis:**

(1) RegÄquiv(Σ) ∈ **PSPACE**.

Seien $\alpha, \beta \in \mathsf{Reg}(\Sigma)$.

Zunächst wandeln wir $\alpha, \beta$ in nichtdeterministische endliche Automaten $A, B$ um mit $L(A) = L(\alpha)$, $L(B) = L(\beta)$.

Dies ist in Polynomialzeit möglich (siehe Konstruktion aus der GTI).

Wir testen in polynomiellen Platz, ob $L(A) \subseteq L(B)$ und $L(B) \subseteq L(A)$.

Wir überprüfen nur $L(A) \subseteq L(B)$, $L(B) \subseteq L(A)$ geht analog.

Es gilt: $L(A) \subseteq L(B) \quad \Leftrightarrow \quad L(A) \cap (\Sigma^* \setminus L(B)) = \emptyset$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Sei $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ und $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$.

Die Potenzmengenkonstruktion liefert für $\Sigma^* \setminus L(B)$ den Automaten

$$B' = (2^{Q_B}, \Sigma, \delta'_B, \{q_{0,B}\}, \{P \subseteq Q_B \mid P \cap F_B = \emptyset\})$$

wobei für alle $a \in \Sigma$, $P, R \subseteq Q_B$ gilt:

$$(P, a, R) \in \delta'_B \;\Leftrightarrow\; R = \{q \in Q_B \mid \exists p \in P : (p, a, q) \in \delta_B\}.$$

Ein Automat für $L(A) \cap (\Sigma^* \setminus L(B)) = L(A) \cap L(B')$ ist dann

$$C = (Q_A \times 2^{Q_B}, \Sigma, \delta_C, (q_{0,A}, \{q_{0,B}\}), F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\})$$

wobei für alle $a \in \Sigma$, $p, r \in Q_A$, $P, R \subseteq Q_B$ gilt:

$$((p, P), a, (r, R)) \in \delta_C \;\Leftrightarrow\; (p, a, r) \in \delta_A \wedge (P, a, R) \in \delta'_B$$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Wir müssen in polynomiellen Platz überprüfen, ob $L(C) \neq \emptyset$ gilt.

**Vorsicht:** Den Automaten $C$ (oder $B'$) können wir nicht konstruieren, er passt nicht in polynomiellen Platz!

Definiere den gerichteten Graphen

$$G = (Q_A \times 2^{Q_B}, \{((p,P),(r,R)) \mid \exists a \in \Sigma : ((p,P),a,(r,R)) \in \delta_C\}).$$

Es gilt: $L(C) \neq \emptyset$ g.d.w. in dem Graphen $G$ ein Pfad von $(q_{0,A}, \{q_{0,B}\})$ zu einem Zustand aus $F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\}$ existiert.

Letzteres kann nichtdeterministisch in poly. Platz überprüft werden:

- Rate einen Zustand $(p,P) \in F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\}$ (passt in polynomiellen Platz)
- Rate einen Pfad von $(q_{0,A}, \{q_{0,B}\})$ nach $(p,P)$. Dabei müssen wir uns immer nur den aktuellen Knoten aus $G$ merken, und dieser passt in polynomiellen Platz.

(2) RegUniv($\Sigma$) ist **PSPACE**-schwierig.

Sei $L \in$ **PSPACE** und $L(M) = L$ für eine $p(n)$-platzbeschränkte deterministische Turingmaschine $M = (Q, \Sigma', \Gamma, \delta, q_0, q_J, q_N, \Box)$, $p(n) > n$ ein Polynom.

Sei $\Omega = (Q \times \Gamma) \cup \Gamma$.

Sei $w = w_1 \cdots w_n \in \Sigma^*$ eine Eingabe für $M$ mit $|w| = n \geq 1$.

Konfigurationen von $M$ werden wieder durch Wörter aus der Sprache $\text{Conf} = \{\Box u(q,a)v\Box \mid (q,a) \in Q \times \Gamma,\ uv \in \Gamma^{2p(n)}\} \subseteq \Omega^{2p(n)+3}$ beschrieben.

Es existiert eine Funktion $\Delta : \Omega^3 \to \Omega$ existiert, so dass für alle $\alpha, \alpha' \in \Box\Omega^*\Box$ mit $|\alpha| = |\alpha'|$ gilt:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \text{Conf und } \forall i \in \{-p(n), \ldots, p(n)\} : \Delta(\alpha[i-1], \alpha[i], \alpha[i+1]) = \alpha'[i].$$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Die Startkonfiguration ist $\alpha_0 := \Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$.

Eine akzeptierende Berechnung (sofern sie existiert)

$$\alpha_0 \vdash_M \alpha_1 \vdash_M \alpha_2 \vdash_M \cdots \vdash_M \alpha_l \in \mathrm{Accept}_M$$

von $M$ bei Eingabe $w$ kodieren wir durch das Wort $\alpha_0 \alpha_1 \alpha_2 \cdots \alpha_l \in \Omega^*$.

Wir konstruieren aus $w$ einen regulären Ausdruck $\beta(w)$ (mittels eines logspace transducers), so dass $L(\beta(w))$ die Menge aller Wörter über dem Alphabet $\Omega$, die keine akzeptierende Berechnung von $M$ bei Eingabe $w$ beschreiben, ist.

Also gilt: $w \notin L(M)$ genau dann, wenn $L(\beta(w)) = \Omega^*$.

Für $C \subseteq \Omega$ identifizieren wir $C$ mit dem regulären Ausdruck $\bigcup_{a \in C} a$.

$\Omega^k$ bezeichnet den regulären Ausdruck $\underbrace{\Omega \cdot \Omega \cdots \Omega}_{k \text{ viele}}$.

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Es ist $\beta(w) = \beta_1 \cup \beta_2 \cup \beta_3 \cup \beta_4 \cup \beta_5$, wobei die Ausdrücke $\beta_i$ $(1 \le i \le 5)$ wie folgt definiert sind.

(a) Alle Wörter, die nicht die richtige Länge haben:

$$\beta_1 = \varepsilon \cup \bigcup_{i=1}^{2p(n)+2} \left(\Omega^{2p(n)+3}\right)^* \Omega^i$$

(b) Alle Wörter, die nicht mit der Startkonfiguration $\alpha_0 = \Box^{p(n)+1}(q_0, w_1)w_2 \cdots w_n \Box^{p(n)-n+2}$ beginnen:

$$\beta_2 = \bigcup_{i=-p(n)-1}^{p(n)+1} \Omega^{i+p(n)+1} \cdot (\Omega \setminus \{\alpha_0[i]\}) \cdot \Omega^*$$

(c) Alle Wörter, wo ein Block der Länge $2p(n)+3$ nicht mit $\Box$ beginnt oder endet:

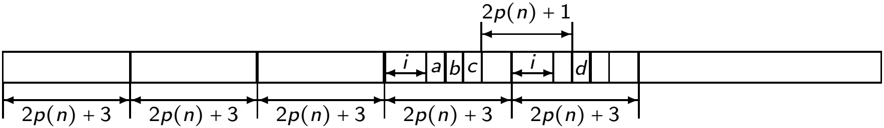
$$\beta_3 = \left(\Omega^{2p(n)+3}\right)^* (\Omega \setminus \{\Box\}) \cup \left(\Omega^{2p(n)+3}\right)^* \Omega^{2p(n)+2} (\Omega \setminus \{\Box\})$$

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

(d) Alle Wörter, die "irgendwo $\Delta$ nicht respektieren":

$$\beta_4 = \bigcup_{i=0}^{2p(n)} (\Omega^{2p(n)+3})^* \Omega^i \left( \bigcup_{u \in \Omega^3} u \cdot \Omega^{2p(n)+1} \cdot (\Omega \setminus \{\Delta(u)\}) \right) \Omega^*$$

Im Bild (mit $u = abc$ und $d \notin \Omega \setminus \{\Delta(u)\}$):




(e) Alle Wörter, die $q_J$ nicht enthalten:

$$\beta_5 = (\Omega \setminus (\{q_J\} \times \Gamma))^*$$

**Behauptung:** $\Omega^* \setminus L(\beta(w)) = \bigcap_{i=1}^{5} (\Omega^* \setminus L(\beta_i))$ ist die Menge aller Wörter $\alpha_0\alpha_1\alpha_2 \cdots \alpha_l$, die eine akzeptierende Berechnung beschreiben.

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Das $x$ zu $\bigcap_{i=1}^{5}(\Omega^* \setminus L(\beta_i))$ gehört, bedeutet:

- $x$ ist von der Form $\alpha_0\alpha_1\cdots\alpha_l$ mit $|\alpha_i| = 2p(n)+3$ für alle $1 \leq i \leq l$ und $\alpha_0$ ist die Startkonfiguration (wegen $\beta_1$ und $\beta_2$).
- Für alle $1 \leq i \leq l$ gilt $\alpha_i \in \Box\Omega^*\Box$ (wegen $\beta_3$).
- Für alle $1 \leq t \leq l-1$ und alle Positionen $i$ mit $|i| \leq p(n)$ gilt: $\alpha_{t+1}[i] = \Delta(\alpha_t[i-1], \alpha_t[i], \alpha_t[i+1])$ (wegen $\beta_4$).

  Wegen der Äquivalenz von Folie 178 (unten) und den obigen Punkten ist dies äquivalent dazu, dass $\alpha_0 \vdash_M \alpha_1 \vdash_M \alpha_2 \vdash_M \cdots \vdash_M \alpha_l$ gilt.
- Eine der Konfigurationen $\alpha_i$ muss akzeptierend sein (wegen $\beta_5$).

  Dies muss zwangsweise $\alpha_l$ sein (da unsere Turing-Maschinen keine Übergänge aus dem Zustand $q_J$ machen können).

Diese Eigenschaften zusammen genommen sind äquivalent dazu, dass $x$ eine akzeptierende Berechnung bei Eingabe $w$ ist.

# Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Abschließend muss das Alphabet $\Omega = (Q \times \Gamma) \cup \Gamma$ noch binär kodiert werden, um die **PSPACE**-Vollständigkeit für jedes Alphabet mit mindestens zwei Symbolen zu bekommen:

Wenn z.B. $\Omega = \{a_1, a_2, \dots, a_k\}$, dann ersetzen wir in dem regulären Ausdruck $\beta(w)$ jedes Vorkommen des Symbols $a_i$ durch $a^i b^{k-i}$; sei $\beta'(w)$ der resultierende reguläre Ausdruck über $\{a, b\}$.

Sei weiter $\beta''$ ein regulärer Ausdruck über $\{a, b\}$ mit

$$L(\beta'') = \{a, b\}^* \setminus \{a^i b^{k-i} \mid 1 \leq i \leq k\}^*.$$

Dann gilt:

$$L(\beta'(w) \cup \beta'') = \{a, b\}^* \iff L(\beta(w)) = \Omega^* \iff w \notin L.$$

$\square$

# Orakel-Turingmaschinen & relative Schwierigkeit von $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Eine nichtdeterministische Orakel-Turingmaschine (kurz OTM) $M^?$ ist eine nichtdeterministische Turingmaschine mit folgenden Besonderheiten:

- $M^?$ hat drei ausgezeichneten Zuständen $o_J, o_N, o_?$ sowie
- ein ausgezeichnetes Arbeitsband — das Orakelband — auf welches nur geschrieben wird.
- Das Orakelband enthält zu jedem Zeitpunkt einen String $w(o)$ über einem Alphabet $\Gamma$.
- Befindet sich $M^?$ im Zustand $o_?$, so kann $M^?$ unabhängig von den gelesenen Bandsymbolen nichtdeterministisch in den Zustand $o_N$ oder den Zustand $o_J$ gehen. Der Inhalt des Orakelbands wird dabei gelöscht.

Eine OTM $M^?$ ist deterministisch, falls sich $M^?$ auf allen Zuständen außer $o_?$ deterministisch verhält.

# Orakel-Turingmaschinen

Zeit- und Platzschranken werden für OTMs wie für normale Turingmaschinen definiert.

Wir betrachten dafür eine OTM als eine nichtdeterministische Turingmaschine (d. h. auf allen Berechnungspfaden muss die Zeitschranke bzw. Platzschranke eingehalten werden).

Sei $A \subseteq \Gamma^*$ eine beliebige (nicht notwendigerweise entscheidbare) Menge.

Wir definieren Berechnungen einer Maschine $M^A$ auf eine Eingabe $v \in \Sigma^*$ wie folgt:

- Auf Zuständen in $Q \setminus \{o_?\}$ verhält sich $M^A$ wie $M^?$.
- Befindet sich $M^?$ im Zustand $o_?$, so ist der Folgezustand (unabhängig von gelesenen Bandsymbolen) $o_J$ (bzw. $o_N$), falls aktuell $w(o) \in A$ (bzw. $w(o) \notin A$) gilt. Der Inhalt des Orakelbands wird wieder gelöscht.

Wir definieren die Komplexitätslassen

$$\begin{aligned}
\mathbf{P}^A &= \{L(M^A) \mid M^? \text{ ist eine det. polynomial zeitbeschränkte OTM}\} \\
\mathbf{NP}^A &= \{L(M^A) \mid M^? \text{ ist eine nichtdet. polynomial zeitbeschränkte OTM}\}
\end{aligned}$$

Für eine Komplexitätsklasse $\mathcal{C}$ sein $\mathbf{P}^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} \mathbf{P}^A$ und analog für $\mathbf{NP}^{\mathcal{C}}$.

**Bemerkung:**

1. $\mathbf{NP} \cup \mathbf{CoNP} \subseteq \mathbf{P}^{\mathrm{SAT}} \subseteq \mathbf{PSPACE}$. Es ist offen, ob „=“ gilt.
2. $A \in \mathbf{P}^A$ für jede Sprache $A$, damit kann insbesondere $\mathbf{P}^A$ unentscheidbare Sprachen enthalten.
3. Ist $A$ entscheidbar, so ist $\mathbf{NP}^A$ eine Familie entscheidbarer Sprachen.
4. Trivialerweise gilt $\mathbf{P}^A \subseteq \mathbf{NP}^A$.

# Polynomiale Turing-Reduzierbarkeit

Seien $A$ und $B$ Sprachen. Dann ist $A$ polynomial Turing-reduzierbar auf $B$ (kurz $A \leq_T^P B$), falls $A \in \mathbf{P}^B$ gilt.

Mit dieser Definition gelten folgende Aussagen:

**Lemma 35**

Wenn $A \leq_m^P B$, dann $A \leq_T^P B$

**Beweis:** Sei $f$ eine Polynomialzeitreduktion $f$ von $A$ auf $B$.

Berechne bei Eingabe $x$ das Wort $f(x)$ und frage das Orakel für $B$, ob $f(x) \in B$ ($\Leftrightarrow x \in A$) gilt. $\square$

# Polynomiale Turing-Reduzierbarkeit

**Lemma 36**

Wenn $A \leq_T^P B$ und $B \in \mathbf{P}$, dann auch $A \in \mathbf{P}$ (d.h. $\mathbf{P}^{\mathbf{P}} = \mathbf{P}$).

**Beweis:** Es sei

- $M_1^?$ eine deterministische polynomial zeitbeschränkte OTM mit $A = L(M_1^B)$ (Zeitschranke $p_1(n)$) und
- $M_2$ eine deterministische polynomial zeitbeschränkte TM mit $B = L(M_2)$ (Zeitschranke $p_2(n)$).

Eine polynomial zeitbeschränkte deterministische TM für $A$ arbeitet bei Eingabe $x$ ($|x| = n$) wie folgt:

- Lasse die Maschine $M_1^?$ auf der Eingabe $x$ laufen.
- Wird im Laufe der Berechnung das Orakel gefragt, ob $w(o) \in B$ gilt, so muss $|w(o)| \leq p_1(n)$ gelten.
- Indem wir die Maschine $M_2$ mit Eingabe $w(o)$ starten, können wir in Zeit $p_2(p_1(n))$ überprüfen, ob $w(o) \in B$ gilt, und danach im Zustand $o_J$ oder $o_N$ die Berechnung von $M_1^?$ fortführen. $\square$

Der gleiche Beweis (wobei $M_1^?$ eine nichtdeterministische polynomial zeitbeschränkte OTM ist und $M_2$ eine deterministische polynomial platzbeschränkte TM ist) zeigt:

**Lemma 37**

$\mathbf{NP}^{\mathbf{PSPACE}} = \mathbf{PSPACE}$

**Satz 38**

Es gibt ein Orakel $A \subseteq \Sigma^*$ in **PSPACE** mit $\mathbf{P}^A = \mathbf{NP}^A$.

**Beweis:**

Betrachte eine **PSPACE**-vollständige Sprache, etwa $A = \mathrm{QBF}$.

Dann gilt $\mathbf{PSPACE} \subseteq \mathbf{P}^{\mathrm{QBF}} \subseteq \mathbf{NP}^{\mathrm{QBF}} \overset{\text{(L. 37)}}{\subseteq} \mathbf{PSPACE}$.

Also gilt $\mathbf{PSPACE} = \mathbf{P}^{\mathrm{QBF}} = \mathbf{NP}^{\mathrm{QBF}}$. $\square$

# Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

**Satz 39 (Baker, Gill, Solovay, 1975)**

Es gibt ein entscheidbares Orakel $B \subseteq \{0,1\}^*$ mit $\mathbf{P}^B \neq \mathbf{NP}^B$.

**Beweis:**

Das Orakel $B \subseteq \{0,1\}^*$ wird so definiert, dass gilt:

$$\forall n \geq 0 : |B \cap \{w \in \{0,1\}^* \mid |w| = n\}| \leq 1$$

Für eine Sprache $B \subseteq \{0,1\}^*$ sei

$$L_B = \left\{1^n \;\middle|\; \exists w \in B \text{ mit } |w| = n\right\}.$$

Dann gilt offensichtlich: $L_B \in \mathbf{NP}^B$.

Noch zu zeigen: Es gibt eine entscheidbare Sprache $B$ mit $L_B \notin \mathbf{P}^B$.

# Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Sei $M_1^?, M_2^?, \dots$ eine effektive Aufzählung aller deterministischen OTMs (mit Eingabealphabet $\{1\}$) mit zusätzlicher polynomieller Zeitschranke.

Wir nehmen an, dass jedes $M_i^?$ in der Aufzählung unendlich oft vorkommt.

Dies kann z.B. dadurch erreicht werden, dass $M_{ij}^? := M_i^?$ aus einer ursprünglichen Aufzählung definiert wird und dann die OTMs aus $\{M_{ij}^? \mid i,j \in \mathbb{N}\}$ aufgezählt werden.

Wir definieren rekursiv das Orakel $B = \bigcup_{i \geq 0} B_i$ durch Mengen $B_0 \subseteq B_1 \subseteq B_2 \subseteq \cdots$ mit $B_i \subseteq \{w \in \{0,1\}^* \mid |w| \leq i\}$ und eine Ausnahmemengen $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$:

## Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

- Für $i = 0$ setze $B_0 = X_0 = \emptyset$.
- Für $i > 0$ simulieren wir die Rechnung von $M_i^?$ auf Input $1^i$ für $i^{\lceil \log i \rceil}$ Schritte.

  Wir nehmen an, dass $B_{i-1}$ und $X_{i-1}$ mit $B_{i-1} \cap X_{i-1} = \emptyset$ schon definiert sind.
  - Setze $X_i = X_{i-1}$.
  - Befragt $M_i^?$ im Laufe der Rechnung ein Wort $w$ mit $|w| < i$, so wird die Rechnung entsprechend dem Orakel $B_{i-1}$ fortgesetzt.
  - Befragt $M_i^?$ im Laufe der Rechnung ein Wort $w$ mit $|w| \geq i$, so setze die Rechnung im nein-Zustand $o_N$ fort und setze $X_i := X_i \cup \{w\}$.
  - Akzeptiert $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ die Eingabe $1^i$ oder kommt $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ zu keiner Entscheidung, so setze $B_i := B_{i-1}$.
  - Verwirft $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ die Eingabe $1^i$, so betrachte das lexikographisch erste Wort $b_i$ in $\{0,1\}^i \setminus X_i$. Setze dann

$$B_i = \begin{cases} B_{i-1} \cup \{b_i\} & \text{falls } b_i \text{ existiert} \\ B_{i-1} & \text{sonst} \end{cases}$$

# Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Es gilt mit $B = \bigcup_{i \geq 0} B_i$ und $X = \bigcup_{i \geq 0} X_i$:

- $B_i \subseteq \{w \in \{0,1\}^* \mid |w| \leq i\}$
- $B$ ist entscheidbar, denn für $|w| = i$ gilt: $w \in B \Leftrightarrow w \in B_i$.
- $B_i \cap X_i = \varnothing$ für alle $i \geq 0$ und damit $B \cap X = \varnothing$.

Wir zeigen jetzt $L_B \notin \mathbf{P}^B$.

Angenommen, es wäre $L_B = L(M^B)$ für eine deterministische, polynomial zeitbeschränkte OTM $M^?$.

Dann gibt es ein Polynom $p(n)$ so, dass $M^?$ $p(n)$-zeitbeschränkt ist.

Wähle jetzt $i$ genügend groß so, dass $\forall n \geq i : p(n) \leq n^{\lceil \log i \rceil}$ und $M^? = M_i^?$ gilt.

Dies ist möglich, da $M^?$ in der Aufzählung beliebig oft vorkommt.

**1. Fall:** $1^i \in L_B = L(M_i^B)$, d.h. $M_i^B$ akzeptiert $1^i$ innerhalb der Zeitschranke $p(i) \leq i^{\lceil \log i \rceil}$.

Dann gilt $B_{i-1} = B_i$ nach Konstruktion von $B$ und damit $1^i \notin L_B$ nach Definition von $L_B$. Widerspruch!

**2. Fall:** $1^i \notin L_B = L(M_i^B)$, d.h. $M_i^B$ verwirft $1^i$ innerhalb der Zeitschranke $p(i) \leq i^{\lceil \log i \rceil}$.

Ist $\{0,1\}^i \setminus X_i \neq \emptyset$, so existiert ein $b_i \in B$ mit $|b_i| = i$ und damit $1^i \in L_B$. Widerspruch!

Es ist daher noch zu zeigen, dass für alle zuvor genügend groß gewählten $i$ gilt: $\{0,1\}^i \setminus X_i \neq \emptyset$

Bei der Berechnung von $X_0, X_1, \ldots, X_i$ werden maximal

$$\sum_{j=1}^{i} j^{\lceil \log j \rceil} \leq i \cdot i^{\lceil \log i \rceil} \leq 2^{\log i + \lceil \log i \rceil^2}$$

Orakelanfragen gestellt.

# Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

$X_i$ enthält also maximal $2^{\log i + \lceil \log i \rceil^2}$ Wörter.

Für alle zuvor genügend groß gewählten $i$ gilt nun

$$2^{\log i + \lceil \log i \rceil^2} < 2^i,$$

weshalb $\{0,1\}^i \setminus X_i$ nicht leer ist. $\square$

Man kann sogar zeigen, dass die Menge aller Orakel $B \subseteq \{0,1\}^*$ mit $\mathbf{P}^B = \mathbf{NP}^B$ Maß 0 innerhalb der Menge aller Sprachen über $\{0,1\}$ hat (Bennett, Gill, 1981).

# Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Was sagen Satz 38 und 39 über das **P**-versus-**NP**-Problem aus?

Viele Beweistechniken in der Komplexitätstheorie sind relativierbar.

Das bedeutet: Zeigt man für Komplexitätsklassen $\mathcal{C}_1$ und $\mathcal{C}_2$ mittels einer relativierbaren Beweistechnik $\mathcal{C}_1 = \mathcal{C}_2$ (bzw. $\mathcal{C}_1 = \mathcal{C}_2$), so kann der Beweis zu einem Beweis von $\mathcal{C}_1^A = \mathcal{C}_2^A$ (bzw. $\mathcal{C}_1^A = \mathcal{C}_2^A$) für ein beliebiges Orakel $A$ verallgemeinert werden.

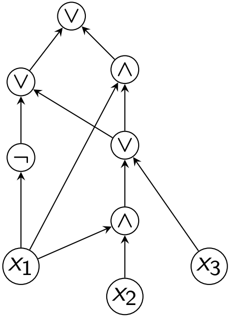
Wegen Satz 38 und 39 kann das **P**-versus-**NP**-Problem nicht mittels relativierbarer Beweistechniken gelöst werden.

# Monotone Schaltkreise und der Satz von Razborov

Sei $C = (C_n)_{n \geq 0}$ eine Familie von booleschen Schaltkreisen, wobei $C_n$ genau $n$ Eingabegatter $x_1, \ldots, x_n$ hat.

Dann berechnet $C_n$ eine boolesche Funktion $f_n : \{0,1\}^n \to \{0,1\}$ und $C$ definiert die Sprache $L(C) = \bigcup_{n \geq 0} \{w \in \{0,1\}^n \mid f_n(w) = 1\}$.

**Beispiel:** Sei $C_3$ der folgende Schaltkreis:




Es gilt $f_3(1,0,0) = 0$ und $f_3(a,b,c) = 1$ für alle $(a,b,c) \in \{0,1\}^3 \setminus \{(1,0,0)\}$.

Also: $L(C) \cap \{0,1\}^3 = \{0,1\}^3 \setminus \{100\}$.

# Monotone Schaltkreise und der Satz von Razborov

Die Familie $C = (C_n)_{n \geq 0}$ ist polynomiell, falls es ein Polynom $p(n)$ gibt, so dass $C_n$ nur höchstens $p(n)$ viele Gatter hat.

Aus der Konstruktion im Satz von Cook (bzw. der **P**-Vollständigkeit von Circuit Value) folgt sehr einfach:

**Lemma 40**

Zu jeder Sprache $L \in \mathbf{P}$ eine polynomielle Familie $C = (C_n)_{n \geq 0}$ von Schaltkreisen gibt mit $L = L(C)$.

Eine Strategie zum Beweis von $\mathbf{P} \neq \mathbf{NP}$ könnte also sein:

Zeige für eine **NP**-vollständige Sprache $L$, dass es keine polynomielle Familie $C = (C_n)_{n \geq 0}$ von Schaltkreisen gibt mit $L = L(C)$.

**Beachte:** Es gibt nicht-entscheidbare Sprachen mit polynomiellen Schaltkreisen.

# Monotone Schaltkreise und der Satz von Razborov

Es ist jedoch leider extrem schwierig für ein konkretes Problem (Sprache) eine exponentielle Schranke für die Größe von Schaltkreisen zu zeigen.

Razborov gelang dies für das **NP**-vollständige Problem CLIQUE unter der Einschränkung auf monotone Schaltkreise.

Eine boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt monoton, falls $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ gilt, sofern $x_i \leq y_i$ für alle $1 \leq i \leq n$.

D.h., durch einen Wechsel von 0 nach 1 in der Eingabe kann die Ausgabe nicht auf 0 zurückfallen.

Ein Schaltkreis heißt monoton, falls er keine NOT-Gatter besitzt.

Ein monotoner Schaltkreis berechnet eine monotone Funktion.

Umgekehrt gibt es zu jeder monotonen Funktion $f : \{0,1\}^n \to \{0,1\}$ (mit $n \geq 1$) einen monotonen Schaltkreis mit höchstens $2^n + 2^{n-1} - 2$ Gattern, der $f$ berechnet.

# Monotone Schaltkreise und der Satz von Razborov

Das folgende Problem CLIQUE ist **NP**-vollständig (Übung):

**Eingabe:** ungerichteter Graph $G = (V, E)$ und ein $k \geq 2$.

**Frage:** Gibt es in $G$ eine Clique der Größe $k$, d.h. gibt es $C \subseteq V$ mit: $|C| \geq k$ und $\{u, v\} \in E$ für alle $u, v \in C$ mit $u \neq v$.

Ein ungerichteter Graph $G = (\{1, \dots, n\}, E)$ wird im folgenden kodiert durch Bits $g_{ij} \in \{0, 1\}$ mit: $g_{ij} = 1 \iff \{i, j\} \in E$.

Sei $\mathrm{cl}_{n,k}$ die folgende monotone boolesche Funktion:

$$\mathrm{cl}_{n,k}((g_{ij})_{1 \leq i < j \leq n}) = 1 \iff \text{der durch } (g_{ij})_{1 \leq i < j \leq n} \text{ definierte Graph hat eine Clique der Größe } k$$

**Satz (Razborov, 1985)**

Es gibt eine Konstante $c_0 > 0$, sodass für jedes genügend große $n$ und $k = \sqrt[4]{n}$ jeder monotone Schaltkreis für $\mathrm{cl}_{n,k}$ mindestens $r = n^{c_0 \cdot \sqrt[8]{n}}$ viele Gatter besitzt.

# Sonnenblumen

Für den Beweis des Satzes von Razborov benötigen wir eine Eigenschaft von Sonnenblumen.

**Definition**

Eine Sonnenblume der Größe $p \geq 2$ mit Kern $Z$ ist eine Menge $\{X_1, \ldots, X_p\}$ von $p$ Mengen, für die gilt: $X_i \cap X_j = Z$ für alle $1 \leq i < j \leq p$. Die Mengen $X_i \setminus Z$ bezeichnen wir als Blütenblätter.

**Lemma (Erdős, Rado)**

Sei $p \geq 2$, $\ell \geq 1$ und $\mathcal{F} = \{X_1, \ldots, X_m\}$ eine Menge von $m$ verschiedenen Mengen mit $m > (p-1)^\ell \cdot \ell!$ und $|X_i| \leq \ell$ für alle $1 \leq i \leq m$. Dann enthält $\mathcal{F}$ eine Sonnenblume der Größe $p$.

# Sonnenblumen

**Beweis:** Induktion nach $\ell$.

**Induktionsanfang:** $\ell = 1$.

Dann enthält $\mathcal{F}$ mindestens $p \leq m$ paarweise disjunkte Mengen der Kardinalität $\leq 1$.

Diese bilden eine Sonnenblume der Größe $p$ mit Kern $Z = \emptyset$.

**Induktionsschritt:** $\ell > 1$.

Sei $\mathcal{D} \subseteq \mathcal{F}$ eine maximale Teilmenge paarweise disjunkter Mengen.

**Fall 1.** $|\mathcal{D}| \geq p$.

Wähle analog zu $\ell = 1$ den Kern $Z = \emptyset$.

**Fall 2.** $|\mathcal{D}| \leq p - 1$.

Dann gilt $|D| \leq (p-1) \cdot \ell$ für $D = \bigcup\{X \mid X \in \mathcal{D}\}$.

# Sonnenblumen

Da $\mathcal{D}$ maximal ist, gilt $\forall X \in \mathcal{F} \setminus \mathcal{D} : X \cap D \neq \emptyset$.

Aus $|\mathcal{F} \setminus \mathcal{D}| \geq m - (p-1)$ und $|D| \leq (p-1) \cdot \ell$ folgt, dass es ein $d \in D$ und mindestens

$$\frac{m-(p-1)}{(p-1)\cdot \ell} > \frac{(p-1)^{\ell} \cdot \ell! - (p-1)}{(p-1)\cdot \ell} = (p-1)^{\ell-1} \cdot (\ell-1)! - \frac{1}{\ell}$$

viele Mengen $X \in \mathcal{F} \setminus \mathcal{D}$ mit $d \in X$ gibt.

Für eine weitere Menge $D_0 \in \mathcal{D}$ gilt $d \in D_0$.

Also gibt es mehr als $(p-1)^{\ell-1} \cdot (\ell-1)!$ Mengen $X \in \mathcal{F}$ mit $d \in X$.

Sei $\mathcal{F}' = \{X \setminus \{d\} \mid d \in X \in \mathcal{F}\}$.

Nach Induktionshypothese enthält $\mathcal{F}'$ eine Sonnenblume $\{Y_1 \setminus \{d\}, \ldots, Y_p \setminus \{d\}\}$ der Größe $p$ mit Kern $Z'$.

Damit ist $\{Y_1, \ldots, Y_p\}$ eine Sonnenblume der Größe $p$ in $\mathcal{F}$ mit Kern $Z = Z' \cup \{d\}$. $\square$

# Beweis des Satzes von Razborov

**Annahme**

Für jede Konstante $c_0 > 0$ gibt es für unendlich viele $n$ einen monotonen Schaltkreis $C_{n,k}$ für die Funktion $\mathrm{cl}_{n,k}$ (mit $k = \sqrt[4]{n}$), welcher weniger als $r = n^{c_0 \cdot \sqrt[8]{n}}$ viele Gatter besitzt.

Sei $c_0 = 1/5$ im folgenden.

Wir werden für große $n$ die obige Annahme zum Widerspruch führen.

Sei $C_{n,k}$ ein Schaltkreis für $\mathrm{cl}_{n,k}$, so wie in der Annahme gefordert.

Wir legen zunächst einige Parameter fest:

$$\begin{aligned} k &:= \sqrt[4]{n} & \qquad p &:= \ell \cdot \log n \\ \ell &:= \sqrt[8]{n} & \qquad M &:= (p-1)^{\ell} \cdot \ell!. \end{aligned}$$

Der Logarithmus wird dabei zur Basis 2 gebildet und die Werte auf- oder abgerundet (die Rechnung ist genügend robust).

# Beweis des Satzes von Razborov

Wir beschriften nun die Gatter von $C_{n,k}$ mit Mengen $\{X_1, \dots, X_m\}$, wobei $X_i \subseteq \{1, \dots, n\}$.

Sei $\mathcal{B}_g$ die Beschriftung für ein Gatter $g$.

Dann definieren wir für einen Graphen $G = (\{1, \dots, n\}, E)$ den Wert $\mathcal{B}_g(G) \in \{0, 1\}$ durch:

$$\mathcal{B}_g(G) = 1 \quad \Longleftrightarrow \quad \exists X \in \mathcal{B}_g : \binom{X}{2} \subseteq E$$

Hierbei ist $\binom{X}{2} = \{\{u, v\} \mid u, v \in X, u \neq v\}$.

D.h.: $\mathcal{B}_g(G) = 1 \iff \mathcal{B}_g$ enthält eine Clique von $G$.

# Beweis des Satzes von Razborov

Eine Beschriftung $\mathcal{B}$ ist zulässig, falls $|\mathcal{B}| \leq M$ und $|X| \leq \ell$ für alle $X \in \mathcal{B}$.

Z. B. ist $\mathcal{B} = \emptyset$ zulässig.

Die folgende reduce-Prozedur wandelt eine beliebige Beschriftung $\mathcal{B}$ in eine zulässige Beschriftung um:

1. Ersetze $\mathcal{B}$ durch $\mathcal{B}' = \{X \in \mathcal{B} \mid |X| \leq \ell\}$.
2. Solange noch $|\mathcal{B}'| > M$ gilt, tue folgendes:

   Suche eine beliebige Sonnenblume $\{X_1, \ldots, X_p\} \subseteq \mathcal{B}'$ mit Kern $Z$ (existiert auf Grund des Lemmas von Erdős und Rado) und ersetze $\{X_1, \ldots, X_p\}$ durch $Z$.

   Diesen Vorgang bezeichnen wir als Pflücken der Sonnenblume.

Die zulässige Beschriftung, die wir so erhalten, nennen wir reduce($\mathcal{B}$).

# Beweis des Satzes von Razborov

Nun definieren wir eine zulässige Beschriftung der Gatter von $C_{n,k}$ induktiv:

1. Für ein Eingabegatter $g_{ij}$ setzen wir $\mathcal{B}_{g_{ij}} = \{\{i,j\}\}$.

   Dies ist eine zulässige Beschriftung, falls $n$ genügend groß ist.

2. Sei $g$ ein Gatter mit den Eingangsgattern $f$ und $h$.

   Induktiv seien $f$ mit $\mathcal{B}_f$ und $h$ mit $\mathcal{B}_h$ jeweils zulässig beschriftet.

   Die Beschriftung $\mathcal{B}_g$ von $g$ definieren wir als

$$\mathcal{B}_g = \begin{cases} \text{reduce}(\mathcal{B}_f \cup \mathcal{B}_h), & \text{falls } g = f \vee h \\ \text{reduce}(\mathcal{B}_f \cdot \mathcal{B}_h), & \text{falls } g = f \wedge h \end{cases}$$

   Dabei ist $\mathcal{B}_f \cdot \mathcal{B}_h = \{X \cup Y \mid X \in \mathcal{B}_f, Y \in \mathcal{B}_h\}$.

Am Ausgangsgatter *out* von $C_{n,k}$ erhalten wir so eine zulässige Beschriftung $\mathcal{B}_{out}$.

Mit $C_{n,k}(g, G_P)$ bezeichnen wir den Wert des Gatters $g$ in $C_{n,k}$ bei Eingabe $G_P$.

Insbesondere: $C_{n,k}(out, G_P) = C_{n,k}(G_P) = 1$.

**Behauptung 1:** Für jeden Graphen $G = (\{1, \ldots, n\}, E)$ und jedes Eingabegatter $g_{ij}$ gilt $\mathcal{B}_{g_{ij}}(G) = C_{n,k}(g_{ij}, G)$.

**Beweis von Behauptung 1:** Es gilt

$$\begin{aligned} \mathcal{B}_{g_{ij}}(G) = 1 \quad & \iff \quad \{i,j\} \text{ ist Clique von } G \\ & \iff \quad \{i,j\} \in E \\ & \iff \quad C_{n,k}(g_{ij}, G) = 1. \end{aligned}$$

# Beweis des Satzes von Razborov

**Behauptung 2:** $\mathcal{B}_{out} \neq \emptyset$, falls $n$ genügend groß ist.

**Beweis von Behauptung 2:**

Für $P \subseteq \{1, \ldots, n\}$ mit $|P| = k$ definieren wir den positiven Testgraphen

$$G_P = (\{1, \ldots, n\}, \binom{P}{2}).$$

Also gilt $C_{n,k}(G_P) = 1$, da $P$ eine Clique der Größe $k$ ist.

Es gibt $\binom{n}{k}$ viele positive Testgraphen $G_P$.

Wir zeigen, dass mindestens ein $P \subseteq \{1, \ldots, n\}$ mit $|P| = k$ und $\mathcal{B}_{out}(G_P) = 1$ existiert.

Dies impliziert dann $\mathcal{B}_{out} \neq \emptyset$.

Um einen Widerspruch abzuleiten, nehmen wir an:

$$\forall P \subseteq \{1, \ldots, n\} \text{ mit } |P| = k : \mathcal{B}_{out}(G_P) = 0 \neq 1 = C_{n,k}(G_P).$$

# Beweis des Satzes von Razborov

Wir stellen uns die Gatter von $C_{n,k}$ nun so aufgelistet vor, dass ein Gatter $g$ stets nach seinen Eingangsgattern $h$ und $f$ kommt (topologische Sortierung).

Für jede Teilmenge $P \subseteq \{1, \dots, n\}$ mit $|P| = k$ gibt es ein kleinstes Gatter $g$ (bzgl. unserer Auflistung) mit

$$\mathcal{B}_g(G_P) = 0 \quad \text{und} \quad C_{n,k}(g, G_P) = 1.$$

Wegen Behauptung 1 kann $g$ kein Eingabegatter $g_{ij}$ sein.

Seien $f$ und $h$ die Eingangsgatter von $g$.

Also gilt: $\mathcal{B}_f(G_P) \geq C_{n,k}(f, G_P)$ und $\mathcal{B}_h(G_P) \geq C_{n,k}(h, G_P)$

# Beweis des Satzes von Razborov

Angenommen, $g$ ist ein OR-Gatter.

Dann gilt $1 = C_{n,k}(g, G_P) = C_{n,k}(f, G_P) \vee C_{n,k}(h, G_P)$.

Gelte o.B.d.A. $C_{n,k}(f, G_P) = 1$.

Mit $\mathcal{B}_f(G_P) \geq C_{n,k}(f, G_P) = 1$ folgt $\mathcal{B}_f(G_P) = 1$.

Sei etwa $X \in \mathcal{B}_f$ und $X \subseteq P$.

$\leadsto$ $X \in \mathcal{B}_f \cup \mathcal{B}_h$

$\leadsto$ $\exists X' \in \mathcal{B}_g = \text{reduce}(\mathcal{B}_f \cup \mathcal{B}_h) : X' \subseteq X \subseteq P$.

$\leadsto$ $\mathcal{B}_g(G_P) = 1$ — ein Widerspruch.

Also muss $g$ ein AND-Gatter sein, d.h. $\mathcal{B}_g = \text{reduce}(\mathcal{B}_f \cdot \mathcal{B}_h)$.

Aus $C_{n,k}(g, G_P) = 1$ folgt $C_{n,k}(f, G_P) = C_{n,k}(h, G_P) = 1$.

$\leadsto$ $\mathcal{B}_f(G_P) = \mathcal{B}_h(G_P) = 1$.

Sei etwa $X \in \mathcal{B}_f$, $Y \in \mathcal{B}_h$, $X \subseteq P$ und $Y \subseteq P$.

$\leadsto$ $X \cup Y \subseteq P$ und $X \cup Y \in \mathcal{B}_f \cdot \mathcal{B}_h$.

# Beweis des Satzes von Razborov

Wegen $\mathcal{B}_g(G_P) = 0$ kann keine Teilmenge von $X \cup Y$ in $\mathcal{B}_g$ sein.

Also gilt $|X \cup Y| \geq \ell + 1$ (sonst wäre nach jedem Pflücken einer Sonnenblume noch eine Teilmenge von $X \cup Y$ in der aktuellen Beschriftung).

Wir haben somit gezeigt: Wenn $g$ das kleinste Gatter mit $\mathcal{B}_g(G_P) = 0$ und $C_{n,k}(g, G_P) = 1$ ist, dann

- ist $g$ ein AND-Gatter und
- es existiert $Z \in \mathcal{B}_f \cdot \mathcal{B}_h$ mit $Z \subseteq P$ und $|Z| \geq \ell + 1$.

Definiere für ein AND-Gatter $g$:

$$\mathcal{P}_g = \{P \subseteq \{1, \ldots, n\} \mid |P| = k, g = \text{kleinste Gatter mit } \mathcal{B}_g(G_P) = 0 \text{ und } C_{n,k}(g, G_P) = 1\}$$

Beachte: Für verschiedene AND-Gatter $g$ und $h$ gilt $\mathcal{P}_g \cap \mathcal{P}_h = \emptyset$.

# Beweis des Satzes von Razborov

Außerdem gilt für jedes AND-Gatter $g$: $|\mathcal{P}_g| \le M^2 \cdot \binom{n-\ell-1}{k-\ell-1}$.

Begründung: Jede Menge $P \in \mathcal{P}_g$ lässt sich schreiben als $P = Z \uplus P'$ mit

- $Z \in \mathcal{B}_f \cdot \mathcal{B}_h$, $|Z| \ge \ell + 1$ ($M^2$ Möglichkeiten) und
- $P' \subseteq \{1, \dots, n\} \setminus Z$, $|P'| = k - |Z|$.

  Hierfür gibt es $\binom{n-|Z|}{k-|Z|} \le \binom{n-\ell-1}{k-\ell-1}$ viele Möglichkeiten.

Andererseits gibt es für jede $k$-elementige Teilmenge $P$ ein AND-Gatter $g$ mit $P \in \mathcal{P}_g$. Da $C_{n,k}$ weniger als $r$ viele AND-Gatter hat, folgt:

$$\begin{aligned} r \cdot M^2 \cdot \binom{n-\ell-1}{k-\ell-1} &> (\text{Anzahl AND-Gatter von } C_{n,k}) \cdot M^2 \cdot \binom{n-\ell-1}{k-\ell-1} \\ &\ge \sum_{g \text{ ist AND-Gatter von } C_{n,k}} |\mathcal{P}_g| = \binom{n}{k} \end{aligned}$$

# Beweis des Satzes von Razborov

Also gilt:

$$
\begin{aligned}
r \quad &> \quad \frac{1}{M^2} \cdot \frac{\binom{n}{k}}{\binom{n-\ell-1}{k-\ell-1}} \;=\; \frac{1}{M^2} \cdot \frac{n! \cdot (k-\ell-1)!}{k! \cdot (n-\ell-1)!} \\
&= \quad \frac{1}{M^2} \cdot \frac{n(n-1)\cdots(n-\ell)}{k(k-1)\cdots(k-\ell)} \;\geq\; \frac{1}{M^2} \cdot \left(\frac{n-\ell}{k}\right)^{\ell+1} \\
&= \quad \frac{1}{(p-1)^{2\ell} \cdot \ell!^2} \cdot \left(\frac{n-\ell}{k}\right)^{\ell+1} \geq \left(\frac{n-\ell}{p^2 \cdot \ell^2 \cdot k}\right)^{\ell+1} \\
&= \quad \left(\frac{n - n^{1/8}}{n^{1/4} \cdot (\log n)^2 \cdot n^{1/4} \cdot n^{1/4}}\right)^{\ell+1} \;=\; \left(\frac{n - n^{1/8}}{n^{3/4} \cdot (\log n)^2}\right)^{\ell+1} \\
&= \quad \left(\frac{n^{1/4}}{(\log n)^2} - \frac{1}{n^{5/8} \cdot (\log n)^2}\right)^{\ell+1} \;\color{red}{\geq}\; n^{1/5 \cdot (\ell+1)} \geq n^{c_0 \sqrt[8]{n}}
\end{aligned}
$$

wobei $\color{red}{\geq}$ gilt, falls $n$ genügend groß ist.

Dies ist ein Widerspruch zu $r = n^{c_0 \sqrt[8]{n}}$, was Behauptung 2 beweist.

# Beweis des Satzes von Razborov

Für die weitere Argumentation benötigen wir negative Testgraphen.

Eine Färbung $c : \{1, \dots, n\} \to \{1, \dots, k-1\}$ der Knotenmenge definiert den Graphen $G_c = (\{1, \dots, n\}, E)$ mit $E = \{\{i, j\} \mid c(i) \neq c(j)\}$.

An den Eingabegattern gilt: $C_{n,k}(g_{ij}, G_c) = 1 \iff c(i) \neq c(j)$.

Der Graph $G_c$ ist $(k-1)$-färbbar, also gilt $C_{n,k}(G_c) = 0$.

Es gibt $(k-1)^n$ Färbungen, aber sehr viel weniger negative Testgraphen.

Sei $c : \{1, \dots, n\} \to \{1, \dots, k-1\}$ im folgenden eine zufällig und gleichverteilt gewählte Färbung (jede Färbung hat Wahrscheinlichkeit $(k-1)^{-n}$

Für $Z \subseteq \{1, \dots, n\}$ sei $R(Z)$ das Ereignis, dass zwei Knoten aus $Z$ gleich gefärbt werden (repeated color):

$$R(Z) \quad \iff \quad \exists i, j \in Z : i \neq j \wedge c(i) = c(j)$$

# Beweis des Satzes von Razborov

Für eine Beschriftung $\mathcal{B}$ gilt damit:

$$\mathcal{B}(G_c) = 1 \iff \mathcal{B} \text{ enthält eine Clique von } G_c \iff \exists Z \in \mathcal{B} : \neg R(Z).$$

**Behauptung 3:** Für jede zulässige Beschriftung $\mathcal{B} \neq \emptyset$ gilt $\Pr(\mathcal{B}(G_c) = 1) \geq 1/2$.

**Beweis von Behauptung 3:**

Für $i, j \in \{1, \ldots, n\}$ mit $i \neq j$ gilt $\Pr(c(i) = c(j)) = \frac{(k-1)^{n-2} \cdot (k-1)}{(k-1)^n} = \frac{1}{k-1}$.

Sei $Z \in \mathcal{B}$ mit $|Z| \leq \ell$. Dann gilt

$$\Pr(R(Z)) \leq \binom{|Z|}{2} \cdot \frac{1}{k-1} \leq \frac{\ell(\ell-1)}{2(k-1)} = \frac{\ell(\ell-1)}{2(\ell^2-1)} = \frac{\ell}{2(\ell+1)} \leq \frac{1}{2}.$$

Also gilt $\Pr(\neg R(Z)) \geq 1/2$, was Behauptung 3 zeigt.

**Beachte:**

- Aus Behauptung 2 ($\mathcal{B}_{out} \neq \emptyset$) und 3 folgt $\Pr(\mathcal{B}_{out}(G_c) = 1) \geq 1/2$.
- Andererseits gilt $C_{n,k}(G_c) = 0$ für alle Färbungen $c$.

# Beweis des Satzes von Razborov

**Behauptung 4:** Sei $\{Z_1, \ldots, Z_p\}$ eine Sonnenblume mit Kern $Z$ und $|Z_i| \leq \ell$ für alle $1 \leq i \leq p$. Dann gilt

$$\Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)) \leq (1/2)^p.$$

**Beweis von Behauptung 4:** Sei $\mathrm{inj}_Z$ die Menge aller injektiven Färbungen $f : Z \to \{1, \ldots, k-1\}$.

Dann gilt ($c{\restriction}_Z$ ist die Einschränkung der zufälligen Färbung $c$ auf $Z$):

$$\begin{aligned}
\Pr\left(\bigwedge_{i=1}^{p} R(Z_i) \wedge \neg R(Z)\right) &= \Pr\left(\bigvee_{f \in \mathrm{inj}_Z} \left(\bigwedge_{i=1}^{p} R(Z_i) \wedge c{\restriction}_Z = f\right)\right) \\
&= \sum_{f \in \mathrm{inj}_Z} \Pr\left(\bigwedge_{i=1}^{p} R(Z_i) \wedge c{\restriction}_Z = f\right) \\
&= \sum_{f \in \mathrm{inj}_Z} \Pr\left(\bigwedge_{i=1}^{p} R(Z_i) \;\middle|\; c{\restriction}_Z = f\right) \cdot \Pr(c{\restriction}_Z = f)
\end{aligned}$$

# Beweis des Satzes von Razborov

**Beachte:** Unter der Voraussetzung $c\restriction_Z = f$ gilt

$$R(Z_i) \iff R(Z_i \setminus Z) \;\vee\; c(Z_i \setminus Z) \cap f(Z) \neq \varnothing.$$

Die Mengen $Z_1 \setminus Z, \dots, Z_p \setminus Z$ sind paarweise disjunkt. Daher sind die Ereignisse $R(Z_1), \dots, R(Z_p)$ unter der Voraussetzung $c\restriction_Z = f$ unabhängig.

Damit erhalten wir:

$$\begin{aligned}
&\sum_{f \in \mathrm{inj}_Z} \Pr\left( \bigwedge_{i=1}^{p} R(Z_i) \;\middle|\; c\restriction_Z = f \right) \cdot \Pr(c\restriction_Z = f) \\
= \quad &\sum_{f \in \mathrm{inj}_Z} \Pr(c\restriction_Z = f) \cdot \prod_{i=1}^{p} \Pr(R(Z_i) \mid c\restriction_Z = f) \\
\leq \quad &\sum_{f \in \mathrm{inj}_Z} \Pr(c\restriction_Z = f) \cdot \prod_{i=1}^{p} \Pr(R(Z_i)) \\
= \quad &\prod_{i=1}^{p} \Pr(R(Z_i)) \cdot \sum_{f \in \mathrm{inj}_Z} \Pr(c\restriction_Z = f) \;\leq\; (1/2)^p,
\end{aligned}$$

# Beweis des Satzes von Razborov

Beachte hierbei:

- Für $|Z_i| = y$ und $|Z| = z \le y$ zeigen elementare Umformungen

$$\Pr(\neg R(Z_i)) = \frac{\binom{k-1}{y} \cdot y!}{(k-1)^y} \le \frac{\binom{k-1-z}{y-z} \cdot (y-z)!}{(k-1)^{y-z}} = \Pr(\neg R(Z_i) \mid c{\restriction}_Z = f).$$

  Also gilt

$$\begin{aligned} \Pr(R(Z_i)) = 1 - \Pr(\neg R(Z_i)) \quad & \ge \quad 1 - \Pr(\neg R(Z_i) \mid c{\restriction}_Z = f) \\ & = \quad \Pr(R(Z_i) \mid c{\restriction}_Z = f) \end{aligned}$$

- $\sum_{f \in \mathrm{inj}_Z} \Pr(c{\restriction}_Z = f) \le 1$

- $\Pr(R(Z_i)) \le 1/2$, falls $|Z_i| \le \ell$ (siehe Beweis von Behauptung 3)

Dies beweist Behauptung 4.

# Randomisierte Turingmaschinen

## Definition

Eine randomisierte Turingmaschine (RTM) ist eine deterministische Turingmaschine $M$ mit einem zusätzlichen Eingabeband (das Zufallsband), welches mit einem nach rechts unendlichen String über dem Alphabet $\{0, 1\}$ gefüllt ist.

$M$ ist $t(n)$-zeitbeschränkt, falls $M$ bei Eingabe $x$ und jeder Belegung des Zufallsbandes nach höchstens $t(|x|)$ Schritten terminiert.

Wir können dann das Zufallsband auf die Länge $t(|x|)$ einschränken.

Dann akzepiert $M$ eine Eingabe $x$ mit der Wahrscheinlichkeit

$$\operatorname*{Prob}_{r \in \{0,1\}^{t(|x|)}} [M \text{ akzeptiert bei Eingabe } x \text{ und Zufallsbandbelegung } r].$$

### Definition IPS (Goldwasser, Micali, Rackoff 1985)

Ein interaktives Beweissystem (IPS) ist ein Paar $(A, B)$ mit:

- $A$ (Alice) ist eine beliebige Funktion

$$A : \bigcup_{i \geq 0} (\{0,1\}^*)^{1+2i} \to \{0,1\}^*.$$

- $B$ (Bob) ist eine randomisierte Turingmaschine mit Ausgabe, dessen Eingabe von der Form $(x, \ldots)$ ($\ldots$ steht für eine Folge von Bitstrings) ist. Bobs Rechenzeit ist durch $q(|x|)$ für ein Polynom $q(n)$ beschränkt.
- Alice und Bob erhalten die gleiche Eingabe $x$ und kommunizieren über ein gemeinsames Arbeitsband (Kommunikationsband) in $p(|x|)$ vielen Runden für ein Polynom $p(n)$.

## Fortsetzung Definition IPS (Goldwasser, Micali, Rackoff 1985)

- Runde $i$ beginnt mit der Botschaft

$$a_i = A(x, a_1, b_1, \dots, a_{i-1}, b_{i-1})$$

  von Alice an Bob, die auf das Kommunikationsband geschrieben wird.

- Danach sendet Bob die Nachricht

$$b_i = B(x, a_1, b_1, \dots, a_{i-1}, b_{i-1}, a_i, r_1, \dots, r_i)$$

  als Antwort an Alice. Hierbei ist $r_i$ der Zufallsstring für Runde $i$.

- In der letzten Runde $p(|x|)$ entscheidet Bob, ob die Eingabe $x$ akzeptiert oder abgelehnt wird.

**Bemerkungen:**

- Da Bob in Polynomialzeit arbeitet, sind Bobs Nachrichten $b_i$ an Alice automatisch polynomiell in der Länge von $x$ beschränkt. Für Alice muss dies nicht unbedingt gelten.

  Aber: Ist Bob $q(n)$-zeitbeschränkt (für ein Polynom $q(n)$), so kann Bob nur die ersten $q(|x|)$ vielen Bits jeder Nachricht $a_i$ von Alice an Bob lesen.

  Also können wir o.B.d.A. verlangen, dass $|a_i| \leq q(|x|)$ gilt. Wir setzen dies im folgenden stets voraus.

- Bob teilt seine Zufallsstrings $r_i$ nicht Alice mit (private coins).

  Später werden wir aber sehen, dass man auch verlangen kann, dass Bob seine Zufallsstrings $r_i$ Alice mitteilt (public coins).

- Die letzte Nachricht $b_{p(n)}$ von Bob an Alice ist eigentlich redundant.

  Später wird es bequem sein, wenn Bob das letzte Bit von $b_{p(n)}$ auf 1 (bzw. 0) setzt, falls er $x$ am Ende akzeptiert (bzw. ablehnt).

- Der Wahrscheinlichkeitsraum wird durch die Zufallsstrings $r_1, r_2, \ldots, r_{p(|x|)}$ erzeugt. Diese befinden sich zu Beginn auf dem Zufallsband. In Runde $i$ hat Bob Zugriff auf das Anfangsstück $r_1, r_2, \ldots, r_i$.

### Definition

Ein IPS $(A, B)$ entscheidet eine Sprache $L \subseteq \Sigma^*$, falls für alle Eingaben $x \in \Sigma^*$ gilt:

- Ist $x \in L$, so akzeptiert Bob die Eingabe mit einer Wahrscheinlichkeit $\geq (1 - 2^{-|x|})$.
- Ist $x \notin L$, so gibt es kein IPS $(A', B)$ in dem Bob die Eingabe mit einer höheren Wahrscheinlichkeit als $2^{-|x|}$ akzeptiert.

**IP** ist die Menge der Sprachen, die durch ein IPS entschieden werden können:

$$\mathbf{IP} \quad = \quad \{L \subseteq \Sigma^* \mid \exists \text{ IPS } (A, B) : (A, B) \text{ entscheidet } L\}$$

# Intuition zu interaktiven Beweissystemen

- Bob wird häufig auch als der Verifizierer bezeichnet, Alice als der Beweiser.
- Mittels eines Kommunikationsprotokolls soll herausgefunden werden, ob $x \in L$ oder nicht.
- Vollständigkeit: Wenn $x \in L$, dann überzeugt ein Beweiser, der sich an das Protokoll hält, den Verifizierer mit hoher Wahrscheinlichkeit $(1 - 2^{-|x|})$.
- Korrektheit: Wenn $x \notin L$, dann überzeugt jeder Beweiser (auch solche, die sich nicht an das Protokoll halten), den Verifizierer nur mit geringer Wahrscheinlichkeit $(2^{-|x|})$.

# Die Klasse **IP**

**Bemerkung: NP** $\subseteq$ **IP**: Alice kann die erfolgreiche Rechnung einer **NP**-Maschine Bob zur Überprüfung vorlegen.

Für ein konkretes Problem aus **NP** kann Alice auch eine Lösung angeben: Für $\mathrm{SAT}$ übermittelt Alice beispielsweise eine erfüllende Belegung an Bob. Bob wertet die Formel unter dieser Belegung aus und akzeptiert, wenn sich als Wert der Formel „wahr“ ergibt.

**Lemma 41**

**IP** ist unter polynomialer Zeitreduktion abgeschlossen:

$$L \leq_m^p L' \text{ und } L' \in \mathbf{IP} \implies L \in \mathbf{IP}$$

**Beweis:** Gelte $L \leq_m^p L' \in \mathbf{IP}$ und sei $f$ die Reduktion von $L$ auf $L'$.

Dann berechnen sowohl Bob und Alice bei Eingabe $x$ zunächst $f(x)$ und lassen dann das Protokoll für $L'$ laufen. $\square$

# Beispiel: Nicht-Isomorphie von Graphen

Das Problem, ob zwei ungerichtete Graphen isomorph sind, liegt in **NP**.

Es ist aber weder bekannt, ob es in **P** liegt, noch, ob es **NP**-vollständig ist.

Es gibt allerdings Hinweise, die darauf hindeuten, dass dieses Problem nicht **NP**-schwierig ist.

Es ist nicht bekannt, ob das Problem der Nicht-Isomorphie von Graphen in **NP** liegt.

**Satz 42 (Goldreich, Micali, Wigderson 1987)**

Nicht-Isomorphie von Graphen gehört zu **IP**.

**Beweis:**

Die Eingabe besteht aus 2 Graphen $G_0 = (V_0, E_0)$ und $G_1 = (V_1, E_1)$. o.B.d.A. $V_0 = V_1 = \{1, \ldots, n\}$ (wenn $|V_0| \neq |V_1|$, dann kann Bob sofort akzeptieren).

Das Protokoll zwischen Alice und Bob arbeitet wie folgt:

Bob wählt zunächst $m$ Permutationen $\pi_i \in \mathrm{Perm}(\{1, \ldots, n\})$ und $m$ Bits $b_i \in \{0, 1\}$ zufällig ($1 \leq i \leq m$).

Dann übermittelt Bob an Alice in einer einzigen Runde die Liste $(\pi_1(G_{b_1}), \ldots, \pi_m(G_{b_m}))$.

Bob verlangt von Alice jetzt einen Bitstring $b_1' b_2' \cdots b_m'$ und akzeptiert, falls $b_i = b_i'$ für alle $i \in \{1, \ldots, m\}$.

# Beispiel: Nicht-Isomorphie von Graphen

**Fall 1:** $G_0 \ncong G_1$.

Dann kann Alice den Bitstring $b_1 b_2 \cdots b_m$ aus $(\pi_1(G_{b_1}), \ldots, \pi_m(G_{b_m}))$ rekonstruieren.

Also wird Bob mit Wahrscheinlichkeit 1 akzeptieren.

**Fall 2:** $G_0 \cong G_1$.

In diesem Fall sieht Alice lediglich eine Liste von $m$ vielen Zufallspermutation von $G_0$.

Alice muss also einen von Bob zufällig gewählten und ihr nicht bekannten String der Länge $m$ "erraten".

Die Wahrscheinlichkeit, dass ihr das gelingt, ist nur $2^{-m}$, d. h. Bobs Antwort lautet (für jede Alice) "Isomorph" mit Wahrscheinlichket $1 - 2^{-m}$. $\square$

# Zero-Knowledge-Beweise

Das obige Protokoll für Nicht-Isomorphie von Graphen hat eine interessante zusätzliche Eigenschaft:

Falls $G_0 \cong G_1$, so teilt Alice Bob keinen Isomorphismus zwischen $G_0$ und $G_1$ mit. Mehr noch, Bob erhält keinerlei Information (zero-knowledge) über solch einen Isomorphismus.

Solche Beweissysteme nennt man auch Zero-Knowledge-Beweissysteme, sie spielen in der Kryptographie eine wichtige Rolle.

### Satz 43

**IP ⊆ PSPACE**

**Beweis:** Sei $L \in \mathbf{IP}$, o.B.d.A. $L \subseteq \{0,1\}^*$.

Fixiere eine fest gewählte randomisierte polynomiell zeitbeschränkte Turingmaschine $B$ (Bob) und eine Eingabe $x \in \Sigma^*$ mit $|x| = n$.

Sei $p(n)$ das Polynom, das die Anzahl der Runden in dem Protokoll bestimmt.

O.B.d.A sei $q(n)$ ein Polynom, so dass alle Nachrichten von Alice sowie Bob $(a_1, b_1 \ldots, a_{p(n)}, b_{p(n)})$ sowie alle Zufallsstrings $r_1, \ldots, r_{p(n)}$ Länge genau $q(n)$ haben.

Eine mögliche Alice ist dann gegeben durch eine Funktion

$$A : \bigcup_{i=0}^{p(n)-1} \{0,1\}^{n+i\cdot 2q(n)} \to \{0,1\}^{q(n)}.$$

# IP $\subseteq$ PSPACE

Wir konstruieren nun eine optimale Alice in polynomiellen Platz.

Legt man eine mögliche Alice $A$ sowie den Zufallsstring

$$\bar{r} = r_1 \cdots r_{p(n)} \in \{0,1\}^{p(n)q(n)}$$

fest, so ist dadurch die Folge der ausgetauschten Nachrichten

$$P(A, \bar{r}) := a_1 b_1 \cdots a_{p(n)} b_{p(n)}$$

(das zu $A$ und $r$ gehörende Protokoll) eindeutig bestimmt.

Falls Bob am Ende des Protokolls $P(A, \bar{r})$ die Eingabe $x$ akzeptiert, so sagen wir, dass $(A, \bar{r})$ akzeptierend ist.

Ein Protokollpräfix ist ein Tupel $P \in \{0,1\}^{i \cdot q(n)}$ mit $0 \leq i \leq 2p(n)$.

$(A, \bar{r})$ passt zu diesem Protokollpräfix $P$, falls $P$ ein Anfangsstück (Präfix) von $P(A, \bar{r})$ ist.

# IP $\subseteq$ PSPACE

Für eine mögliche Alice $A$ und das Protokollpräfix $P$ definieren wir

$$f(A,P) = |\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A,\bar{r}) \text{ ist akzeptierend und passt zu } P\}|.$$

Eine mögliche Alice ist optimal bzgl. $P$, falls $f(A,P) = f(P)$ für das folgende Maximum:

$$f(P) = \max\{f(A,P) \mid A \text{ ist eine mögliche Alice}\}.$$

Für den leeren Protokollpräfix $\lambda$ gilt

$$f(A,\lambda) = |\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A,\bar{r}) \text{ ist akzeptierend}\}|.$$

und damit:

$$\frac{f(A,\lambda)}{2^{p(n)q(n)}} = \mathrm{Prob}[\text{das IPS } (A,B) \text{ akzeptiert } x]$$

$$\frac{f(\lambda)}{2^{p(n)q(n)}} = \max\{\mathrm{Prob}[(A,B) \text{ akzeptiert } x] \mid A \text{ ist eine mögliche Alice}\}$$

# IP ⊆ PSPACE

Also gilt:

$$
\begin{aligned}
x \in L &\implies f(\lambda) \geq (1-2^{-n}) \cdot 2^{p(n)q(n)} \\
x \notin L &\implies f(\lambda) < 2^{-n} \cdot 2^{p(n)q(n)}.
\end{aligned}
$$

**Ziel:** Berechnung von $f(\lambda)$ in polynomiellen Platz.

Wir berechnen die Werte $f(P)$ rekursiv für jedes Protokollpräfix $P$.

**Fall 1:** $P = a_1 b_1 \cdots a_{p(n)} b_{p(n)}$ ein vollständiges Protokoll.

**Fall 1.1:** Bob lehnt am Ende des Protokolls $P$ die Eingabe $x$ ab (d.h. das letzte Bit von $b_{p(n)}$ ist 0).

Dann gilt $f(A, P) = 0$ für jede mögliche Alice und damit $f(P) = 0$.

# IP ⊆ PSPACE

**Fall 1.2:** Bob akzeptiert am Ende des Protokolls $P$ die Eingabe $x$.
(d.h. das letzte Bit von $b_{p(n)}$ ist 1).

Dann gilt für jede mögliche Alice

$$\begin{aligned} f(A,P) &= |\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A,\bar{r}) \text{ ist akzeptierend und passt zu } P\}| \\ &= |\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid P(A,\bar{r}) = P = a_1 b_1 \cdots a_{p(n)} b_{p(n)}\}| \end{aligned}$$

Dieser Wert wird maximal für jede mögliche Alice $A$ mit

$$A(x, a_1, b_1, \ldots, a_{j-1}, b_{j-1}) = a_j$$

für alle $1 \le j \le p(n)$. Also gilt:

$$f(P) = |\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid \bar{r} = r_1 \cdots r_{p(n)} \text{ und für alle } 1 \le j \le p(n) \text{ gilt } B(x, a_1, b_1, \ldots, a_{j-1}, b_{j-1}, a_j, r_1, \ldots, r_j) = b_j\}|.$$

# IP $\subseteq$ PSPACE

**Fall 2:** Das Protokollpräfix ist von der Form $P = a_1 b_1 \cdots a_{i-1} b_{i-1} a_i$ mit $1 \leq i \leq p(n)$.

Induktionshyp.: Alle Werte $f(Pb_i)$ ($b_i \in \{0,1\}^{q(n)}$) sind bereits bekannt.

**Behauptung 1:** $f(P) = \sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i)$.

Wir zeigen zunächst $f(P) \leq \sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i)$

**Beobachtung:** Seien

- $A$ und $A'$ zwei mögliche Alices,
- $\bar{r} \in \{0,1\}^{p(n)q(n)}$ ein Zufallsstring und
- $b_i, b_i' \in \{0,1\}^{q(n)}$ Nachrichten (von Bob an Alice) mit $b_i \neq b_i'$.

Dann kann nicht $(A, \bar{r})$ passend zu $Pb_i$ sein und gleichzeitig $(A', \bar{r})$ passend zu $Pb_i'$ sein.

# IP ⊆ PSPACE

Daher gilt für jede mögliche Alice $A$:

$$
\begin{aligned}
f(A,P) &= |\{\bar r \in \{0,1\}^{p(n)q(n)} \mid (A,\bar r) \text{ ist akzept. und passt zu } P\}| \\
&= \left| \biguplus_{b_i \in \{0,1\}^{q(n)}} \{\bar r \in \{0,1\}^{p(n)q(n)} \mid (A,\bar r) \text{ ist akzept.und passt zu } Pb_i\} \right| \\
&= \sum_{b_i \in \{0,1\}^{q(n)}} |\{\bar r \in \{0,1\}^{p(n)q(n)} \mid (A,\bar r) \text{ ist akzept. und passt zu } Pb_i\}| \\
&= \sum_{b_i \in \{0,1\}^{q(n)}} f(A,Pb_i).
\end{aligned}
$$

Sei nun $A$ eine optimale Alice bzgl. $P$, d.h. $f(A,P) = f(P)$:

$$
f(P) = f(A,P) = \sum_{b_i \in \{0,1\}^{q(n)}} f(A,Pb_i) \le \sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i)
$$

## IP $\subseteq$ PSPACE

Nun zeigen wir $\sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i) \leq f(P)$.

Für jedes $b_i \in \{0,1\}^{q(n)}$ sei $A_{b_i}$ eine Alice mit $f(Pb_i) = f(A_{b_i}, Pb_i)$.

Dann definieren wir eine Alice $A'$ wie folgt:

- In den ersten $i$ Runden schickt $A'$ die durch den Protokollpräfix $P$ festegelegten Nachrichten an Bob.
- Ist $b_i$ Bobs Nachricht an Alice in Runde $i$, so verhält sich $A'$ danach wie $A_{b_i}$.

# IP ⊆ PSPACE

Damit gilt:

$$
\begin{aligned}
& \sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i) \\
= & \sum_{b_i \in \{0,1\}^{q(n)}} f(A_{b_i}, Pb_i) \\
= & \sum_{b_i \in \{0,1\}^{q(n)}} \left|\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A_{b_i}, \bar{r}) \text{ ist akzept. und passt zu } Pb_i\}\right| \\
= & \left| \biguplus_{b_i \in \{0,1\}^{q(n)}} \{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A_{b_i}, \bar{r}) \text{ ist akzept. und passt zu } Pb_i\} \right| \\
\leq & \left|\{\bar{r} \in \{0,1\}^{p(n)q(n)} \mid (A', \bar{r}) \text{ ist akzept. und passt zu } P\}\right| \\
= & f(A', P) \\
\leq & f(P)
\end{aligned}
$$

# IP ⊆ PSPACE

**Fall 3:** Das Protokollpräfix ist von der Form $P = a_1 b_1 \cdots a_{i-1}, b_{i-1}$ mit $1 \leq i \leq p(n)$.

Alice kann nun eine Antwort $a_i \in \{0,1\}^{q(n)}$ wählen.

Induktionshypothese: alle Werte $f(Pa_i)$ sind bereits bekannt.

**Behauptung 2:** $f(P) = \max\{f(Pa_i) \mid a_i \in \{0,1\}^{q(n)}\}$.

$$
\begin{aligned}
f(P) &= \max\{f(A,P) \mid \ A \text{ ist eine mögliche Alice}\} \\
&= \max\{f(A,Pa_i) \mid a_i \in \{0,1\}^{q(n)},\ A \text{ ist eine mögliche Alice}\} \\
&= \max_{a_i \in \{0,1\}^{q(n)}} \max\{f(A,Pa_i) \mid \ A \text{ ist eine mögliche Alice}\} \\
&= \max_{a_i \in \{0,1\}^{q(n)}} f(Pa_i)
\end{aligned}
$$

Behauptung 1 und 2 liefern einen rekursiven Algorithmus zur Berechnung von $f(\lambda)$.

# IP ⊆ PSPACE

**Platzbedarf des rekursiven Algorithmus:**

- Die Rekursionstiefe ist $2p(n)$ (polynomiell).
- Für jeden lokalen Aufruf müssen folgende lokalen Variablen gespeichert werden: ein Protokollpräfix, ein String aus $\{0,1\}^{q(n)}$ (ein $a_i$ oder $b_i$) sowie eine Zahl $\leq 2^{p(n)q(n)}$ (aktueller $f$-Wert) .

  Hierfür reicht polynomieller Platz.
- Für einen terminalen Aufruf mit $P = a_1 b_1 \cdots a_{p(n)} b_{p(n)}$ muss für alle Zufallsstrings $r = (r_1, \dots, r_{p(n)})$ überprüft werden, ob $r$ zu $P$ passt.

  Hierfür muss für alle $1 \leq i \leq p(n)$ die Bedingung

  $$b_i = B(x, a_1, b_1, \dots, a_i, r_i)$$

  überprüft werden. Dies ist in Polynomialzeit möglich (uns reicht die Aussage, dass dies in polynomiellen Platz möglich ist). $\square$

**Satz 44 (Shamir, 1990)**

**IP** = **PSPACE**.

**Beweis:** Noch zu zeigen: **PSPACE** $\subseteq$ **IP**.

Da **IP** unter Polynomialzeitreduktionen abgeschlossen ist und QBF **PSPACE**-vollständig ist, genügt es zu zeigen: QBF $\in$ **IP**.

Wir werden sogar ein IPS konstruieren, in dem Bob seine Zufallszahlen Alice mitteilt und am Ende (mit Wahrscheinlichkeit 1) „ja“ sagt, falls $x \in$ QBF gilt.

Sei also die Eingabe eine geschlossene quantifizierte Formel $F$ über $0, 1, \wedge, \vee, \forall, \exists$ und den Literalen $\tilde{x}_1, \tilde{x}_2, \ldots$ mit $\tilde{x}_i \in \{x_i, \overline{x_i}\}$ gegeben.

O.B.d.A. wird Negation nur auf Variablen und nicht auf größere Teilformeln angewendet.

$F$ kann wie folgt in einen arithmetischen Ausdruck $a(F)$ umgeformt werden: Wir ersetzen

| | | |
|---|---|---|
| $\overline{x}$ | durch | $(1-x)$ |
| $\vee$ | durch | $+$ |
| $\wedge$ | durch | $\cdot$ (Multiplikation) |
| $\forall x$ | durch | $\prod_{x\in\{0,1\}}$ |
| $\exists x$ | durch | $\sum_{x\in\{0,1\}}$ |

Mit dieser Übersetzung gilt:

$$F \in \text{QBF} \iff a(F) > 0$$
$$F \notin \text{QBF} \iff a(F) = 0$$

# Beweis des Satzes von Shamir (Arithmetisierung)

**Beispiel:**

Sei $F = \forall x \exists y \Big( (x \vee \overline{y}) \wedge \exists z (\overline{x} \wedge z) \Big)$. Dann ist

$$\begin{aligned}
a(F) &= \prod_{x=0,1} \Big( \sum_{y=0,1} \Big( (x + (1-y)) \cdot \sum_{z=0,1} ((1-x) \cdot z) \Big) \Big) \\
&= \prod_{x=0,1} \Big( \sum_{y=0,1} \Big( (x + (1-y)) \cdot (1-x) \Big) \Big) \\
&= \prod_{x=0,1} \Big( (1-x^2) + (x - x^2) \Big) \\
&= 0
\end{aligned}$$

Das bedeutet, dass die Auswertung der Formel $F$ „falsch“ ergibt.

# Beweis des Satzes von Shamir (Arithmetisierung)

Für die Negation $\neg F = \exists x \forall y \Big( (\overline{x} \wedge y) \vee \forall z (x \vee \overline{z}) \Big)$ gilt:

$$\begin{aligned}
a(\neg F) &= \sum_{x=0,1} \Big( \prod_{y=0,1} \Big( (1-x) \cdot y + \prod_{z=0,1} (x + 1 - z) \Big) \Big) \\
&= \sum_{x=0,1} \prod_{y=0,1} \Big( (1-x) \cdot y + x^2 + x \Big) \\
&= \sum_{x=0,1} (x^2 + x) \cdot (1 + x^2) \\
&= 4
\end{aligned}$$

Die Auswertung der Formel $\neg F$ ergibt also „wahr“.

# Beweis des Satzes von Shamir

Es stellt sich die Frage, wie groß die Zahl $a(F)$ werden kann.

Die Länge $|F|$ einer Formel definieren wir induktiv wie folgt:

$$\begin{aligned} |0| = |1| = |x| = |\overline{x}| &= 1 \\ |F \wedge G| = |F \vee G| &= |F| + |G| \\ |\forall x F| = |\exists x F| &= 1 + |F| \end{aligned}$$

**Lemma 45**

Für eine geschlossene quantifizierte Formel $F$ gilt: $a(F) \le 2^{2^{|F|}}$.

**Beweis:**

Zunächst eliminieren wir Quantoren, indem wir jedes Vorkommen von $\forall x G$ ($\exists x G$) in $F$ durch $G_{x=0} \wedge G_{x=1}$ ($G_{x=0} \vee G_{x=1}$) ersetzen.

# Beweis des Satzes von Shamir

Sei $F'$ die resultierende Formel.

Durch Induktion über den Aufbau von $F$ zeigt wir $|F'| \leq 2^{|F|}$:

Für $F$ von der Form 0, 1, $x$ oder $\overline{x}$ gilt $F = F'$ und damit

$$|F'| = 1 \leq 2^1 = 2^{|F|}.$$

Für $F = G \circ H$ mit $\circ \in \{\wedge, \vee\}$ gilt

$$|F'| = |G' \circ H'| = |G'| + |H'| \leq 2^{|G|} + 2^{|H|} \leq 2^{|G|} \cdot 2^{|H|} = 2^{|G|+|H|} = 2^{|F|}.$$

Für $F = \forall x G$ gilt

$$|F'| = |G'_{x=0} \wedge G'_{x=1}| = |G'_{x=0}| + |G'_{x=1}| = 2|G'| \leq 2 \cdot 2^{|G|} = 2^{1+|G|} = 2^{|F|}.$$

Der Fall $F = \exists x G$ ist analog.

# Beweis des Satzes von Shamir

Nun zeigen wir $a(F') \le 2^{|F'|}$ durch Induktion über den Aufbau einer Quantoren-freien Formel $F'$:

Für $F' = 0$ oder $F' = 1$ gilt $a(F') \le 1 \le 2^1 = 2^{|F'|}$.

Für $F' = G' \vee H'$ gilt

$$a(G' \vee H') = a(G') + a(H') \le 2^{|G'|} + 2^{|H'|} \le 2^{|G'|} \cdot 2^{|H'|} = 2^{|G'|+|H'|} = 2^{|F'|}.$$

Für $F' = G' \wedge H'$ gilt

$$a(G' \wedge H') = a(G') \cdot a(H') \le 2^{|G'|} \cdot 2^{|H'|} = 2^{|G'|+|H'|} = 2^{|F'|}.$$

Insgesamt erhalten wir: $a(F) = a(F') \le 2^{|F'|} \le 2^{2^{|F|}}$. $\square$

# Beweis des Satzes von Shamir

**Beispiel:** Sei $F = \forall x_1 \ldots \forall x_k \exists y \exists z (y \vee z)$. Dann ist

$$
\begin{aligned}
a(F) &= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} \sum_{y=0,1} \sum_{z=0,1} (y+z) \\
&= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} \sum_{y=0,1} (2y+1) \\
&= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} (4) \\
&= 4^{2^k} \\
&\leq 2^{2^{k+4}} \\
&= 2^{2^{|F|}}
\end{aligned}
$$

# Beweis des Satzes von Shamir

**Problem:** Die Zahl $2^{2^{|F|}}$ benötigt selbst bei binärer Kodierung noch $2^{|F|}$ Bits und kann deshalb nicht in einem IPS übermittelt werden.

**Lösung:** Rechnen modulo einer Primzahl.

**Lemma 46**

Für $n \geq 4$ gibt es im Intervall $[2^n, 2^{2n}]$ mindestens $2^n$ Primzahlen.

**Beweis:** Sei $\pi(n)$ die Anzahl der Primzahlen im Intervall $[2, n]$.

Mit elementaren Methoden kann man zeigen: $\pi(n) \geq \frac{n}{\log_2 n} - 2$, siehe z. B. Theorem 3.6.3 in *Primality Testing in Polynomial Time* von M. Dietzfelbinger, Springer 2004.

Also gibt es im Intervall $[2^n, 2^{2n}]$ mindestens

$$\left(\frac{2^{2n}}{2n} - 2\right) - (2^n - 2) = 2^n \cdot \left(\frac{2^n}{2n} - 1\right) \geq 2^n$$

viele Primzahlen, falls $n \geq 4$. □

# Beweis des Satzes von Shamir

Sei im Folgenden $n = |F|$ und seien $p_1, \ldots, p_k$ die Primzahlen zwischen $2^n$ und $2^{2n}$ (d.h. $k \geq 2^n$).

Dann gilt:

$$m = p_1 \cdot p_2 \cdots p_k > (2^n)^{(2^n)} = 2^{n \cdot 2^n} \geq 2^{2^n}$$

und damit $m > a(F)$, da $a(F) \leq 2^{2^n}$.

Damit gilt:

$$\begin{aligned} F \notin \text{ QBF} &\iff a(F) \equiv 0 \bmod m \quad (\text{da } a(F) = 0) \\ F \in \text{ QBF} &\iff \exists \text{ Primzahl } p \in [2^n, 2^{2n}] : a(F) \not\equiv 0 \bmod p \end{aligned}$$

Die zweite Zeile folgt aus der Tatsache, dass $m = p_1 \cdot p_2 \cdots p_k > a(F)$ gilt und $a(F)$ damit nicht alle $p_i$ als Teiler haben kann, falls $a(F) > 0$.

# Beweis des Satzes von Shamir

Falls nun $F \in \mathrm{QBF}$ gilt, dann berechnet Alice die kleinste Primzahl $p \geq 2^n$ mit der Eigenschaft $a(F) \not\equiv 0 \bmod p$.

Sie sendet dann $p$ an Bob, der überprüfen kann, ob $p$ auch wirklich eine Primzahl ist ($\mathrm{PRIM} \in \mathbf{P}$).

**Beachte:** $p$ hat höchstens $2n$ viele Bits.

Alle weiteren Rechnungen werden von nun an modulo $p$ durchgeführt.

O.B.d.A. beginne $F$ mit einem Quantor: $F = QxF'(x)$ mit $Q \in \{\forall, \exists\}$.

Jetzt wird aus dem arithmetischen Ausdruck $a(F)$ das Polynom $a(F'(x))$ berechnet, indem in $a(F)$ das erste Zeichen $\prod$ bzw. $\sum$ gestrichen wird.

**Problem:** Der Grad dieses Polynoms kann exponentiell in $n$ sein.

**Lösung:** einfache Formeln.

## Definition (einfache Formel)

Eine Formel ist einfach, wenn für jede Teilformel $Qx\, G$ ($Q \in \{\forall, \exists\}$) jedes Vorkommen von $x$ in $G$ nur innerhalb eines $\forall$-Quantors liegt.

## Lemma 47

Jede Formel lässt sich in polynomieller Zeit in eine äquivalente einfache Formel umwandeln.

**Beweis:**

Jede Teilformel $\forall y : G(x_1, \ldots, x_k, y)$ (wobei $x_1, \ldots, x_k, y$ alle freien Variablen in $G$ sind) wird ersetzt durch:

$$\forall y\ \exists y_1 \cdots \exists y_k : \bigwedge_{i=1}^{k} x_i \leftrightarrow y_i \wedge G(y_1, \ldots, y_k, y).$$

$\square$

# Beweis des Satzes von Shamir (einfache Formeln)

**Beispiel:**

$F = \forall x_1 \forall x_2 \forall x_3 \exists x_4 \, G(x_1, x_2, x_3, x_4)$ ($G$ Quantoren-frei) wird zu

$$\forall x_1 \forall x_2 \exists y_1 \big(x_1 \leftrightarrow y_1 \wedge \forall x_3 \exists z_1 \exists z_2 \big(y_1 \leftrightarrow z_1 \wedge x_2 \leftrightarrow z_2 \wedge \exists x_4 \, G(z_1, z_2, x_3, x_4)\big)\big)$$

Wenn die Ausgangsformel in Pränexnormalform ist ($Q_1 x_1 \, Q_2 x_2 \cdots Q_k x_k \, G$ mit $G$ Quantoren-frei), so erhalten wir eine Formel $F$ mit:

- $F$ beginnt mit einem Quantor (falls $k \geq 1$).
- $F$ ist einfach.
- Wenn $F_i = Q_i x_i G_i$ eine Teilformel von $F$ ist, dann ist $G_i$ entweder Quantoren-frei oder lässt sich schreiben als $G_i = H_i \wedge F_{i+1}$ mit $H_i$ Quantoren-frei und $F_{i+1} = Q_{i+1} x_{i+1} G_{i+1}$.

# Beweis des Satzes von Shamir (einfache Formeln)

**Lemma 48**

Sei $G = Qx : G'$ ($Q \in \{\exists, \forall\}$) eine einfache Formel der Länge $n$. Dann ist $a(G')$ ein Polynom welches Grad $\leq 2n$ in $x$ hat.

**Beweis:**

Wir ersetzen zunächst in $G'$ jede Teilformel der Gestalt $\forall y : H$, in der $x$ frei vorkommt, durch $H_{y=0} \wedge H_{y=1}$.

Dies verdoppelt die Länge der Formel nur, da alle Teilformeln der Gestalt $\forall y : H$, in denen $x$ frei vorkommt, disjunkt zueinander liegen.

Unsere neue (äquivalente) Formel hat also die Länge $2n$.

Es genügt somit zu zeigen: Sei $G'$ eine Formel, so dass keine Teilformel der Gestalt $\forall y : H$ mit $x$ frei in $H$ existiert. Dann ist $a(G')$ ein Polynom in den freien Variablen von $G'$ mit $\mathrm{Grad}_x(a(G')) \leq |G'|$.

# Beweis des Satzes von Shamir (einfache Formeln)

Diese Aussage zeigen wir durch Induktion über den Aufbau von $G'$:

**Fall 1:** $G'$ ist eine Konstante oder eine (evtl. negierte) Variable: klar.

**Fall 2:** $G' = G_1 \vee G_2$. Dann gilt:

$$\mathrm{Grad}_x(a(G')) \le \max\{\mathrm{Grad}_x(a(G_1)), \mathrm{Grad}_x(a(G_2))\} \le \max\{|G_1|, |G_2|\} \le |G'|$$

**Fall 3:** $G' = G_1 \wedge G_2$. Dann gilt:

$$\mathrm{Grad}_x(a(G')) = \mathrm{Grad}_x(a(G_1)) + \mathrm{Grad}_x(a(G_2)) \le |G_1| + |G_2| = |G'|$$

**Fall 4:** $G' = \exists y : G_1$, o.B.d.A $y \neq x$. Dann gilt:

$$\begin{aligned} \mathrm{Grad}_x(a(G')) &\le \max\{\mathrm{Grad}_x(a(G_1)_{y=0}), \mathrm{Grad}_x(a(G_1)_{y=1})\} \\ &\le \mathrm{Grad}_x(a(G_1)) \\ &\le |G_1| \le |G'|. \end{aligned}$$

**Fall 5:** $G' = \forall y : G_1$. Dann kann $x$ nicht frei in $G_1$ vorkommen.

Also gilt $\mathrm{Grad}_x(a(G')) = 0 \le |G'|$. $\square$

# Beweis des Satzes von Shamir: das Protokoll

Erinnerung : Alice hat Bob bereits eine Primzahl $p \in [2^n, 2^{2n}]$ geschickt.
Es gilt $a(F) \not\equiv 0 \bmod p$, falls $F \in \mathrm{QBF}$.

Nun werden insgesamt $m$ Runden gespielt, falls $m \leq n$ die Anzahl der Quantoren in $F$ ist.

Sei $F = F_1 = Q_1 x_1 : G_1(x_1)$ und $p_1(x_1) = a(G_1(x_1)) \bmod p$
(ein Polynom in $x_1$ vom Grad höchstens $2n$).

**Runde 1:**

- Alice sendet Bob den Wert $a_1 = a(F_1) \bmod p$.
- Bob gibt "$F \notin \mathrm{QBF}$" aus, falls $a_1 = 0$, sonst verlangt er einen Beweis für die Korrektheit von $a_1$.
- Diesen gibt Alice, indem sie Bob das Polynom $p_1(x_1)$ mitteilt.
- Falls $Q_1 = \forall$: Bob überprüft, ob $a_1 \equiv p_1(0) \cdot p_1(1) \bmod p$.
- Falls $Q_1 = \exists$: Bob überprüft, ob $a_1 \equiv p_1(0) + p_1(1) \bmod p$.

# Beweis des Satzes von Shamir: das Protokoll

- Nun wählt Bob zufällig eine Zahl $r_1 \in \mathbb{F}_p$, teilt diese Zahl Alice mit (public coins) und berechnet $p_1(r_1) \pmod{p}$.
- Der arithmetische Ausdruck $a(G_1(x_1))_{x_1=r_1}$ lässt sich schreiben als

$$a(G_1(x_1))_{x_1=r_1} = b_1 \cdot a(F_2(x_1))_{x_1=r_1},$$

  wobei $F_2$ der Teilausdruck von $G_1$ ist, der mit dem ersten Quantor von $G_1$ beginnt. Der Wert $b_1 \in \mathbb{F}_p$ kann von Bob aus $G_1$ und $r_1$ berechnet werden.
- Falls $b_1 = 0$: Bob akzeptiert, falls $p_1(r_1) = 0$ gilt, sonst lehnt er ab.
- Falls $b_1 \neq 0$: Bob berechnet $a_2 = p_1(r_1) \cdot b_1^{-1}$.

**Beachte:** Wenn Alice in Runde 1 das korrekte Polynom $p_1(x_1) = a(G_1(x_1))$ angibt, gilt $a(F_2(x_1))_{x_1=r_1} \equiv a_2 \pmod{p}$.

# Beweis des Satzes von Shamir: das Protokoll

Sei $F_2(x_1) = Q_2 x_2 : G_2(x_1, x_2)$ und $p_2(x_2) = a(G_2(x_1, x_2))_{x_1 = r_1} \pmod{p}$.

**Runde 2:**

- Bob will von Alice einen Beweis für $a(F_2(x_1))_{x_1=r_1} \equiv a_2 \pmod{p}$.
- Alice sendet hierfür Bob das Polynom $p_2(x_2)$.
- Bob überprüft wieder $a_2 \equiv p_2(0) \cdot p_2(1) \bmod p$ (falls $Q_2 = \forall$) bzw. $a_2 \equiv p_2(0) + p_2(1) \bmod p$ (falls $Q_2 = \exists$).
- Bob wählt die Zufallszahl $r_2$, teilt $r_2$ Alice mit und berechnet $p_2(r_2)$.
- Sei $a(G_2(x_1, x_2))_{x_1=r_1, x_2=r_2} = b_2 \cdot a(F_3(x_1, x_2))_{x_1=r_1, x_2=r_2}$, wobei $F_3$ der Teilausdruck von $G_2$ ist, der mit dem ersten Quantor von $G_2$ beginnt.
- Falls $b_2 = 0$: Bob akzeptiert, falls $p_2(r_2) = 0$, sonst lehnt er ab.
- Falls $b_2 \neq 0$: Bob berechnet $a_3 = p_2(r_2) \cdot b_2^{-1}$.
- In Runde 3 muss Alice $a(F_3(x_1, x_2))_{x_1=r_1, x_2=r_2} \equiv a_3 \bmod p$ beweisen.

# Beweis des Satzes von Shamir: das Protokoll

Diese Protokoll wird für höchstens $m =$ (Anzahl der Quantoren in $F$) viele Runden durchgeführt.

Sei $F_m(x_1, \dots, x_{m-1}) = Q_m x_m : G_m(x_1, \dots, x_m)$ und

$$p_m(x_m) = a(G_m(x_1, \dots, x_{m-1}, x_m))_{x_1=r_1,\dots,x_{m-1}=r_{m-1}} \bmod p$$

mit $G_m$ Quantoren-frei.

**Runde** $m$**:**

- Bob will von Alice einen Beweis für $a(F_m(x_1, \dots, x_{m-1}))_{x_1=r_1,\dots,x_{m-1}=r_{m-1}} \equiv a_m \pmod{p}$.
- Alice sendet hierfür Bob das Polynom $p_m(x_m)$.
- Bob überprüft wieder $a_m \equiv p_m(0) \cdot p_m(1) \bmod p$ (falls $Q_m = \forall$) bzw. $a_m \equiv p_m(0) + p_m(1) \bmod p$ (falls $Q_m = \exists$).
- Bob wählt eine weitere Zufallszahl $r_m$ und akzeptiert genau dann, wenn $p_m(r_m) = a(G_m(x_1, \dots, x_m))_{x_1=r_1,\dots,x_m=r_m} \pmod{p}$.

Offensichtlich: Wenn $F \in \mathrm{QBF}$, dann akzeptiert Bob mit Wahrscheinlichkeit 1.

Gelte nun $F \notin \mathrm{QBF}$, d. h. $a(F) \equiv 0 \bmod p$.

Mit welcher Wahrscheinlickkeit kann eine Alice Bob vom Gegenteil $F \in \mathrm{QBF}$ überzeugen?

Angenommen, Bob gibt schließlich "$F \in \mathrm{QBF}$" aus.

Alice muss in Runde 1 einen Wert $a_1 \not\equiv a(F_1) \equiv 0 \bmod p$ an Bob schicken, sonst würde Bob "$F \notin \mathrm{QBF}$" ausgeben.

Da Bob $a_1 \equiv p_1(0) \cdot p_1(1) \bmod p$ (falls $Q_1 = \forall$) bzw $a_1 \equiv p_1(0) + p_1(1) \bmod p$ (falls $Q_1 = \exists$) prüft, muss auch das von Alice an Bob gesendete Polynom $p_1(x_1)$ falsch sein, d. h. $p_1(x_1) \neq a(G_1(x_1))$.

# Beweis des Satzes von Shamir: Analyse des Protokolls

Dann ist $p_1(x_1) - a(G_1(x_1))$ ein Polynom vom Grad $\leq 2n$ und hat damit höchstens $2n$ Nullstellen in dem Körper $\mathbb{F}_p$.

Es gilt also $p_1(r_1) = a(G_1(x_1))_{x_1=r_1}$ an höchstens $2n$ Stellen $r_1 \in \mathbb{F}_p$.

Wegen $p \geq 2^n$ folgt $\underset{r_1 \in \mathbb{F}_p}{\text{Prob}}(p_1(r_1) \neq a(G_1(x_1))_{x_1=r_1}) \geq 1 - \dfrac{2n}{2^n}$.

Falls $p_1(r_1) \neq a(G_1(x_1))_{x_1=r_1}$ und $b_1 = 0$ in der ersten Runde, so wird Bob auf jeden Fall ablehnen.

Falls $p_1(r_1) \neq a(G_1(x_1))_{x_1=r_1}$ und $b_1 \neq 0$ in der ersten Runde (und das Protokoll in Runde 2 geht), so gilt $a_2 \neq a(F_2(x_1))_{x_1=r_1}$.

In diesem Fall muss Alice Bob mit Runde 2 beginnend von der falschen Identität $a_2 = a(F_2(x_1))_{x_1=r_1}$ überzeugen.

Diese Situation wiederholt sich ständig (höchstens $m \leq n$ mal).

In Runde $i$ muss Alice Bob von einer Identität

$$a_i = a(F_i(x_1, \ldots, x_{i-1}))_{x_1=r_1,\ldots,x_{i-1}=r_{i-1}} \qquad (8)$$

überzeugen.

Hierzu schickt sie ihm ein Polynom $p_i(x_i)$ (von Grad höchstens $2n$).

Bob stellt sicher, dass wenn (8) nicht gilt (Alice schummelt), dann auch

$$p_i(x_i) = a(G_i(x_1, \ldots, x_{i-1}, x_i))_{x_1=r_1,\ldots,x_{i-1}=r_{i-1}}$$

nicht gilt.

Dann wird Bob wieder mit Wahrscheinlichkeit mindestens $1 - \frac{2n}{2^n}$ ein $r_i \in \mathbb{F}_p$ mit $p_i(r_i) \neq a(G_i(x_1, \ldots, x_i))_{x_1=r_1,\ldots,x_i=r_i}$ wählen.

In diesem Fall geht es in Runde $i+1$ mit $a_{i+1} \neq a(F_{i+1}(x_1, \ldots, x_i))_{x_1=r_1,\ldots,x_{i-1}=r_i}$.

# Beweis des Satzes von Shamir: Analyse des Protokolls

Andererseits: Wenn Bob einmal (etwa in Runde $i$) eine Zufallszahl $r_i$ mit

$$p_i(r_i) = a(G_i(x_1, \ldots, x_{i-1}, x_i))_{x_1=r_1,\ldots,x_{i-1}=r_{i-1},x_i=r_i}$$

trifft, dann gilt $a_{i+1} = a(F_{i+1}(x_1, \ldots, x_i))_{x_1=r_1,\ldots,x_{i-1}=r_i}$. Alice kann dann ohne erneute Täuschungsmanöver weiter spielen und Bob wird am Ende fälschlicherweise "$F \in$ QBF" ausgeben.

Es werden jedoch maximal $m \leq n$ viele Runden durchgeführt.

Wegen $(1-x)^n \geq 1 - nx$ für $0 \leq x \leq 1$ gilt:

$$\text{Prob}[\text{Bob gibt "}F \notin \text{QBF" aus}] \geq \left(1 - \frac{2n}{2^n}\right)^m \geq \left(1 - \frac{2n}{2^n}\right)^n \geq 1 - n \cdot \frac{2n}{2^n},$$

Also: $\text{Prob}[\text{Bob gibt "}F \in \text{QBF" aus}] \leq \frac{2n^2}{2^n}$.

Wird das Protokoll zweimal durchlaufen, so folgt für genügend große $n$:

$$\text{Prob}[\text{Bob gibt "}F \in \text{QBF" aus}] \leq \left(\frac{2n^2}{2^n}\right)^2 = \frac{4n^4}{2^{2n}} < 2^{-n} \qquad \square$$

# Bemerkungen zu **IP** = **PSPACE**

- Die Inklusion **PSPACE** $\subseteq$ **IP** und damit **IP** = **PSPACE** gilt auch dann, wenn die Zufallsbits von Bob öffentlich sind, also Alice bekannt sind.
- Die Inklusion **PSPACE** $\subseteq$ **IP** gilt auch dann, wenn man Alice auf eine polynomiell platzbeschränkte Turingmaschine einschränkt.
- Schränkt man die Anzahl der Runden in interaktiven Beweissystemen auf eine Konstante ein, so erhält man die Komplexitätsklasse **AM** (Arthur-Merlin-Spiele). Es wird **AM** $\subsetneq$ **PSPACE** vermutet.
- **IP** = **PSPACE** ist nicht relativierbar: Es gibt ein Orakel $A$ mit $\mathbf{IP}^A \neq \mathbf{PSPACE}^A$.

  In der Tat gilt dies sogar für fast alle Orakel $A$.

# Erweiterungen von **IP**

- Ersetzt man Alice durch zwei Beweiser, mit denen Bob kommunizieren kann, so erhält man die Komplexitätsklasse **MIP**.

  Die beiden Beweiser können hierbei nicht miteinander kommunizieren.

  Analogie: Die beiden Beweiser sind zwei Verdächtige, die Bob (ein Polizeibeamter) von ihrer Unschuld zu überzeugen versuchen. Bob verhört die beiden Verdächtigen in getrennten Räumen.

  Babai, Fortnow, Lund 1991: **MIP** = **NEXPTIME**, wobei **NEXPTIME** $= \bigcup_{k \geq 1} \mathbf{NTIME}(2^{n^k})$.

- Werden diese beiden Beweiser zu Beginn mit einer beliebigen Zahl von verschränkten qubits ausgestattet, so erhält man die Klasse $\mathbf{MIP}^*$.

  Ji, Natarajan, Vidick, Wright, Yuen 2020: $\mathbf{MIP}^* = \mathbf{RE}$ (die Klasse aller rekursiv aufzählbaren Sprachen!).

Der folgende Text basiert auf Diekert, Kufleitner, Rosenberger, Diskrete algebraische Methoden, de Gryuter 2013.

Zur Erinnerung:

$$\text{PRIMES} = \{w \mid w \in 1\{0,1\}^*, w \text{ ist Binärkodierung einer Primzahl}\}$$

2002 konnten Agrawal, Kayal und Saxena zeigen, dass PRIMES $\in$ **P** gilt.

Ihr Algorithmus ist heute als der AKS-Test bekannt.

**Beachte:** Für eine Eingabezahl $n$ arbeitet der AKS-Test in Zeit $O(\log^k n)$ für eine Konstante $k$.

Die Grundidee des AKS-Test basiert auf folgendem Lemma:

**Lemma 49**

Sei $n$ eine Primzahl und $a \in \mathbb{Z}$. Dann gilt in dem Polynomring $\mathbb{Z}[x]$ die Identität

$$(x+a)^n \equiv x^n + a \bmod n.$$

**Bemerkung:** $(x+a)^n \equiv x^n + a \bmod n$ bedeutet, dass die Polynome $(x+a)^n$ und $x^n + a$ identisch werden, wenn wir Koeffizienten modulo $n$ betrachten.

Anders gesagt: $p(x) \equiv q(x) \bmod n$ genau dann, wenn ein Polynom $a(x) \in \mathbb{Z}[x]$ existiert mit $p(x) = q(x) + n \cdot a(x)$.

Beispiel: $(x+2)^3 = x^3 + 6x^2 + 12x + 8 \equiv x^3 + 2 \bmod 3$

**Beweis von Lemma 49:** Es gilt:

$$(x+a)^n = x^n + \sum_{k=1}^{n-1} \binom{n}{k} x^k a^{n-k} + a^n.$$

Da $n$ prim ist, gilt für alle $k \in \{1, 2, \dots, n-1\}$:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \equiv 0 \mod n.$$

Nach dem kleinen Satz von Fermat gilt außerdem $a^n \equiv a \mod n$.

Also: $(x+a)^n \equiv x^n + a \mod n$. $\square$

Es gilt auch folgende Umkehrung von Lemma 49 (die wir im Folgenden nicht benötigen):

**Lemma 50**

Seien $n$ und $a$ teilerfremd. Wenn $(x+a)^n \equiv x^n + a \bmod n$ gilt, dann ist $n$ eine Primzahl.

**Beweis:** Seien $n$ und $a$ teilerfremd und sei $n$ keine Primzahl.

Wir zeigen, dass $(x+a)^n \not\equiv x^n + a \bmod n$ gilt.

Sei $p$ ein Primteiler von $n$ und sei $p^k$ die größte $p$-Potenz, die $n$ teilt.

Da $p$ teilerfremd zu den Zahlen $n-1, n-2, \ldots, n-p+1$ ist, ist $p^k$ auch die größte $p$-Potenz, die $n(n-1)(n-2)\cdots(n-p+1)$ teilt.

Außerdem ist $p^2$ kein Teiler von $p!$.

Damit ist $p^{k-1}$ die größte $p$-Potenz, die $\binom{n}{p} = \frac{n(n-1)\cdots(n-p+1)}{p!}$ teilt.

Insbesondere gilt $\binom{n}{p} \cdot a^{n-p} \not\equiv 0 \bmod n$.

Also verschwindet der Summand $\binom{n}{p} \cdot x^p \cdot a^{n-p}$ in der Summe

$$(x+a)^n = \sum_{k=0}^{n} \binom{n}{k} \cdot x^k \cdot a^{n-k}$$

nicht modulo $n$. □

Aus Lemma 49 und 50 folgt, dass $n$ eine Primzahl ist, genau dann wenn

$$(x+1)^n \equiv x^n + 1 \bmod n.$$

**Problem:** Dies ergibt keinen Polynomialzeitalgorithmus, da die Berechnung von $(x+1)^n$ modulo $n$ exponentielle Zeit gemessen in der Eingabelänge $\log n$ benötigt.

Anstatt einer Kongruenz $(x+1)^n \equiv x^n + 1 \bmod n$ testet der AKS-Test Kongruenzen

$$(x+a)^n \equiv x^n + a \bmod (x^r - 1, n)$$

für eine natürliche Zahl $1 \le r \le (\log n)^5$ und alle $1 \le a \le \ell$, wobei $\ell \le \sqrt{r} \cdot \log n$.

Hierbei bedeutet $p(x) \equiv q(x) \bmod (x^r - 1, n)$, dass Polynome $a(x), b(x) \in \mathbb{Z}[x]$ existieren mit

$$p(x) = q(x) + (x^r - 1) \cdot a(x) + b(x) n.$$

Anders gesagt: Ein Ideal $I$ ist eine Teilmenge $I \subseteq \mathbb{Z}[x]$ mit folgenden beiden Eigenschaften:

- $p(x), q(x) \in I \Rightarrow p(x) + q(x) \in I$,
- $p(x) \in I, q(x) \in \mathbb{Z}[x] \Rightarrow p(x) \cdot q(x) \in I$

# Anhang: PRIMES $\in$ **P**

Die Menge

$$I = (x^r - 1, n) := \{(x^r - 1) \cdot a(x) + b(x)n \mid a(x), b(x) \in \mathbb{Z}[x]\}$$

ist das kleineste Ideal mit $n \in I$ und $x^r - 1 \in I$.

Dann bedeutet $p(x) \equiv q(x) \mod I$, dass $p(x) - q(x) \in I$ gilt.

Bevor wir den AKS-Test beschreiben, zeigen wir, wie eine Kongruenz $(x+a)^n \equiv x^n + a \mod (x^r - 1, n)$ in Zeit polynomiell in $\log n$ überprüft werden kann.

Die binäre Relation $\{(p(x), q(x)) \mid p(x) \equiv q(x) \mod I\}$ auf $\mathbb{Z}[x]$ ist eine Äquivalenzrelation und Kongruenzrelation:

Wenn $p_1(x) \equiv q_1(x) \mod I$ und $p_2(x) \equiv q_2(x) \mod I$, dann auch

- $p_1(x) + p_2(x) \equiv q_1(x) + q_2(x) \mod I$ und
- $p_1(x) \cdot p_2(x) \equiv q_1(x) \cdot q_2(x) \mod I$.

Sei wieder $I = (x^r - 1, n)$.

### Lemma 51

Für jedes Polynom $p(x)$ gibt es ein eindeutig bestimmtes Polynom $\tilde{p}(x)$ mit:

- $\tilde{p}(x) \equiv p(x) \mod I$
- Der Grad von $\tilde{p}(x)$ ist kleiner als $r$.
- Alle Koeffizienten von $\tilde{p}(x)$ liegen in $\{0, 1, \ldots, n-1\}$.

**Beweis:** Für $m \geq r$ gilt

$$x^m = x^{m-r} \cdot (x^r - 1) + x^{m-r} \equiv x^{m-r} \mod I$$

und damit $x^m \equiv x^{m \bmod r} \mod I$.

Damit erhalten wir ein Polynom $\tilde{p}(x)$ mit den Eigenschaften aus dem Lemma, indem wir in $p(x)$

- jede Potenz $x^m$ mit $m \geq r$ durch $x^{m \bmod r}$ ersetzen, und danach
- jeden Summanden $a \cdot x^k$ $(0 \leq k < r)$ durch $(a \bmod n) \cdot x^k$ ersetzen.

Es bleibt die Eindeutigkeit von $\tilde{p}(x)$ zu zeigen.

Hierzu zeigen wir Folgendes: Sind $p_1(x)$ und $p_2(x)$ Polynome mit:

- $p_1(x) \equiv p_2(x) \mod I$
- Die Grade von $p_1(x)$ und $p_2(x)$ sind kleiner als $r$.
- Alle Koeffizienten in $p_1(x)$ und $p_2(x)$ liegen in $\{0, 1, \ldots, n-1\}$.

Dann gilt $p_1(x) = p_2(x)$.

Angenommen, $p_1(x) \neq p_2(x)$.

Wegen $p_1(x) \equiv p_2(x) \mod I$ existieren Polynome $a(x), b(x)$ mit

$$0 \neq p_1(x) - p_2(x) = (x^r - 1) \cdot a(x) + n \cdot b(x).$$

Wähle das Polynom $a(x)$ so, dass es minimalen Grad $d$ hat:

Also gilt $a(x) = c \cdot x^d + c(x)$ mit $c \in \mathbb{Z} \setminus \{0\}$ und $\mathrm{Grad}(c(x)) < d$.

Angenommen $d > 0$.

Dann gilt:

$$\begin{aligned} p_1(x) - p_2(x) &= (x^r - 1) \cdot (c \cdot x^d + c(x)) + n \cdot b(x) \\ &= c \cdot x^{r+d} + x^r \cdot c(x) - c \cdot x^d - c(x) + n \cdot b(x) \end{aligned}$$

Da der Summand $c \cdot x^{r+d}$ wegfallen muss $(\mathrm{Grad}(p_1(x) - p_2(x)) < r)$, muss $c$ ein Vielfaches von $n$ sein, d.h. $c = \alpha \cdot n$ mit $\alpha \in \mathbb{Z}$.

Also gilt $a(x) = \alpha \cdot n \cdot x^d + c(x)$ und somit

$$\begin{aligned} p_1(x) - p_2(x) &= (x^r - 1) \cdot (\alpha \cdot n \cdot x^d + c(x)) + n \cdot b(x) \\ &= (x^r - 1) \cdot c(x) + n \cdot (b(x) + \alpha \cdot x^d \cdot (x^r - 1)) \end{aligned}$$

Dies widerspricht der Wahl von $a(x)$ (minimaler Grad).

Also muss der Grad von $a(x)$ null sein, d.h. $a(x) = a \in \mathbb{Z}$.

Damit gilt

$$p_1(x) - p_2(x) = a \cdot (x^r - 1) + n \cdot b(x) = a \cdot x^r - a + n \cdot b(x).$$

Wie oben folgt, dass $a$ ein Vielfaches von $n$ sein muss (damit sich der Term $a \cdot x^r$ wegkürzt).

Sei also $a = a' \cdot n$.

# Anhang: PRIMES $\in$ **P**

Es folgt

$$p_1(x) - p_2(x) = a' \cdot n \cdot (x^r - 1) + n \cdot b(x) = n \cdot (a' \cdot (x^r - 1) + b(x)).$$

Da wir aber $p_1(x) \neq p_2(x)$ annehmen und alle Koeffizienten von $p_1(x), p_2(x)$ im Bereich $\{0, \ldots, n-1\}$ liegen, enthält das Polynom $p_1(x) - p_2(x)$ einen Summanden $\alpha \cdot x^k$ mit $\alpha \not\equiv 0 \mod n$.

Dies ist ein Widerspruch! $\square$

Das Polynom $\tilde{p}(x)$ aus Lemma 51 bezeichnen wir mit $p(x) \bmod l$.

Da $\equiv \bmod l$ eine Kongruenzrelation ist (Folie 276) gilt

- $(p_1(x) + p_2(x)) \bmod l = ((p_1(x) \bmod l) + (p_2(x) \bmod l)) \bmod l$ und
- $(p_1(x) \cdot p_2(x)) \bmod l = ((p_1(x) \bmod l) \cdot (p_2(x) \bmod l)) \bmod l$

**Lemma 52**

Die Polynome $(x+a)^n \mod l$ und $x^n + a \mod l$ können in Zeit polynomiell in $\log n$ berechnet werden.

**Beweis:** schnelle Exponentation

Wir zeigen das Lemma nur für $(x+a)^n \bmod l$.

Wir berechnen nacheinander die Polynome

$$(x+a) \bmod l,\ (x+a)^2 \bmod l,\ (x+a)^4 \bmod l, \ldots, (x+a)^{2^k} \bmod l,$$

wobei $k$ maximal mit $2^k \leq n$ ist.

Durch Multiplikation einer Teilmenge dieser Polynome erhalten wir $(x+a)^n \bmod l$. $\square$

Wir benötigen im Folgenden eine zahltentheoretische Abschätzung.

Für $n \in \mathbb{N}$ sei $\text{kgV}(n)$ das kleinste gemeinsame Vielfache der Zahlen $2, 3, 4, \ldots, n$.

**Satz 53**

Für alle $n \geq 7$ gilt $\text{kgV}(n) > 2^n$.

Für den Beweis von Satz 53 benötigen wir das folgende Lemma:

**Lemma 54**

Für alle $m, n \in \mathbb{N}$ mit $1 \leq m \leq n$ teilt $m \cdot \binom{n}{m}$ die Zahl $\text{kgV}(n)$.

**Beweis:** Für $1 \leq m \leq n$ betrachten wir das Integral

$$\mathcal{I}_{n,m} = \int_0^1 x^{m-1}(1-x)^{n-m}dx$$

**Behauptung 1:** $\mathcal{I}_{n,m} \cdot \mathrm{kgV}(n) \in \mathbb{N}$.

Aus dem Binomialsatz angewendet auf $(1-x)^{n-m}$ folgt

$$x^{m-1} \cdot (1-x)^{n-m} = \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} x^{m-1+k}.$$

Also gilt

$$\mathcal{I}_{n,m} = \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} \cdot \int_0^1 x^{m-1+k} dx = \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} \cdot \frac{1}{m+k}$$

und damit

$$\mathcal{I}_{n,m} \cdot \mathrm{kgV}(n) = \sum_{k=0}^{n-m} (-1)^k \binom{n-m}{k} \cdot \underbrace{\frac{\mathrm{kgV}(n)}{m+k}}_{\in \mathbb{N}} \in \mathbb{Z}$$

Außerdem gilt offensichtlich $\mathcal{I}_{n,m} \cdot \mathrm{kgV}(n) \geq 0$.

**Behauptung 2:** $\frac{1}{\mathcal{I}_{n,m}} = m \cdot \binom{n}{m}$

Wir zeigen Behauptung 2 durch Induktion über $n - m$.

**Induktionsanfang:** $n - m = 0$, d.h. $n = m$.

Dann gilt

$$\mathcal{I}_{n,m} = \int_0^1 x^{m-1}(1-x)^{n-m} dx = \int_0^1 x^{m-1}\, dx = \left[\frac{1}{m}x^m\right]_0^1 = \frac{1}{m} = \frac{1}{m\binom{m}{m}}.$$

**Induktionsschritt:** Sei nun $1 \leq m < n$.

Partielle Integration: Sei $u(x) = \frac{1}{m}x^m$ und $v(x) = (1-x)^{n-m}$.

Dann gilt

$$\int_0^1 u'(x)v(x)dx = [u(x)v(x)]_0^1 - \int_0^1 u(x)v'(x)dx,$$

d.h.

$$
\begin{aligned}
\mathcal{I}_{n,m} &= \int_0^1 x^{m-1}(1-x)^{n-m}dx \\
&= \int_0^1 \frac{1}{m}x^m(n-m)(1-x)^{n-m-1}dx \\
&= \frac{n-m}{m} \cdot \int_0^1 x^{(m+1)-1}(1-x)^{n-(m+1)}dx \\
&= \frac{n-m}{m} \cdot \mathcal{I}_{n,m+1} \\
&= \frac{n-m}{m} \cdot \frac{1}{(m+1)\cdot\binom{n}{m+1}} \\
&= \frac{n-m}{m} \cdot \frac{(m+1)!\cdot(n-m-1)!}{(m+1)\cdot n!} \\
&= \frac{1}{m} \cdot \frac{m!\cdot(n-m)!}{n!} = \frac{1}{m\cdot\binom{n}{m}}
\end{aligned}
$$

# Anhang: PRIMES $\in$ **P**

Aus Behauptung 1 und 2 folgt

$$\mathrm{kgV}(n) = \underbrace{\mathcal{I}_{n,m} \cdot \mathrm{kgV}(n)}_{\in \mathbb{N}} \cdot m \cdot \binom{n}{m}.$$

Also teilt $m \cdot \binom{n}{m}$ die Zahl $\mathrm{kgV}(n)$. $\square$

Wir können nun Satz 53 beweisen: Zunächst gilt

$$4^n = 2^{2n} = (1+1)^{2n} = \sum_{k=0}^{2n} \binom{2n}{k}.$$

Das Maximum der Binomialkoeffizienten $\binom{2n}{0}, \binom{2n}{1}, \ldots, \binom{2n}{2n}$ ist $\binom{2n}{n}$.

Dieses Maximum muss (da die Binomialkoeffizienten nicht alle gleich sind) größer als der Mittelwert sein, d.h.

$$\frac{1}{2n+1} \sum_{k=0}^{2n} \binom{2n}{k} = \frac{4^n}{2n+1}.$$

Also gilt für $n \geq 1$:

$$\binom{2n}{n} > \frac{4^n}{2n+1}. \tag{9}$$

Weiter erhalten wir mit Lemma 54 zwei Teiler von $\text{kgV}(2n+1)$:

- $(2n+1) \cdot \binom{2n}{n} = \frac{(2n+1)!}{n! \cdot n!} = (n+1) \cdot \binom{2n+1}{n+1}$ teilt $\text{kgV}(2n+1)$.
- $n \cdot \binom{2n}{n}$ teilt $\text{kgV}(2n)$, was wiederum $\text{kgV}(2n+1)$ teilt.

Anders gesagt:

- $2n+1$ teilt $\text{kgV}(2n+1)/\binom{2n}{n}$.
- $n$ teilt $\text{kgV}(2n+1)/\binom{2n}{n}$.

Da $n$ und $2n+1$ teilerfremd sind, teilt auch $n \cdot (2n+1)$ die natürliche Zahl $\text{kgV}(2n+1)/\binom{2n}{n}$, d.h. $n \cdot (2n+1) \cdot \binom{2n}{n}$ teilt $\text{kgV}(2n+1)$.

## Anhang: PRIMES $\in$ **P**

Also gilt:

$$n \cdot 4^n < n \cdot (2n+1) \cdot \binom{2n}{n} \leq \mathrm{kgV}(2n+1)$$

Sei nun $n \geq 4$. Es gilt dann

$$2^{2n+1} < 2^{2n+2} = 4 \cdot 4^n \leq n \cdot 4^n < \mathrm{kgV}(2n+1) \leq \mathrm{kgV}(2n+2).$$

Also gilt $\mathrm{kgV}(2n+1) > 2^{2n+1}$ und $\mathrm{kgV}(2n+2) > 2^{2n+2}$ für alle $n \geq 4$.

Damit gilt $\mathrm{kgV}(n) > 2^n$ für alle $n \geq 9$.

Für $n = 7$ gilt $2^7 = 128 < 420 = \mathrm{kgV}(7)$.

Für $n = 8$ gilt $2^8 = 256 < 840 = \mathrm{kgV}(8)$. $\square$

# Anhang: PRIMES $\in$ **P**

## Lemma 55

Für $k, t \geq 1$ gilt $\left|\{(e_1, \dots, e_k) \in \mathbb{N}^k \mid \sum_{i=1}^{k} e_i < t\}\right| = \binom{t+k-1}{t-1}$.

**Beweis:** Es gilt

$$\begin{aligned} & \left|\{(e_1, \dots, e_k) \in \mathbb{N}^k \mid \sum_{i=1}^{k} e_i \leq t\}\right| \\ = \quad & \left|\{(e_1, \dots, e_k, e_{k+1}) \in \mathbb{N}^{k+1} \mid \sum_{i=1}^{k+1} e_i = t\}\right| = \binom{t+k}{k} \end{aligned}$$

und damit

$$\left|\{(e_1, \dots, e_k) \in \mathbb{N}^k \mid \sum_{i=1}^{\ell} e_i < t\}\right| = \binom{t-1+k}{k} = \binom{t-1+k}{t-1}.$$

$\square$

Für $r \geq 1$ ist

$$(\mathbb{Z}/r\mathbb{Z})^* = \{n \mid 1 \leq n < r,\ \mathrm{ggT}(n,r) = 1\}$$

die multiplikative Gruppe modulo $r$.

Die Eulersche $\varphi$-Funktion ist definiert als

$$\varphi(r) = |(\mathbb{Z}/r\mathbb{Z})^*| = |\{n \mid 1 \leq n < r, \mathrm{ggT}(n,r) = 1\}|$$

Für $n$ mit $\mathrm{ggT}(n,r) = 1$ ist

$$\mathrm{ord}_r(n) = \min\{i \geq 1 \mid n^i \equiv 1 \bmod r\} = \min\{i \geq 1 \mid r \text{ teilt } (n^i - 1)\}$$

die Ordnung von $n$ in der Gruppe $(\mathbb{Z}/r\mathbb{Z})^*$.

**Lemma 56**

Sei $n$ eine Primzahl und sei $m \geq 2$ mit $7 \leq m^2 \log n < n$. Dann gibt es eine Zahl $r \in \mathbb{N}$ mit $1 \leq r \leq m^2 \log n$ und $\mathrm{ord}_r(n) > m$.

**Beweis:** Sei $s = \lfloor m^2 \log n \rfloor < n$ (beachte: $s \geq 7$).

Da $n$ eine Primzahl ist, gilt $\mathrm{ggT}(n, r) = 1$ für alle $1 \leq r \leq s$.

Angenommen es gilt $\mathrm{ord}_r(n) \leq m$ für alle $1 \leq r \leq s$.

Dann gibt es für jedes $1 \leq r \leq s$ ein $1 \leq i \leq m$ mit $r \mid (n^i - 1)$.

Also gilt $r \mid \prod_{i=1}^m (n^i - 1)$ und damit $\mathrm{kgV}(s) \mid \prod_{i=1}^m (n^i - 1)$.

Satz 53 und $m \geq 2 \;\Rightarrow\; 2^s < \mathrm{kgV}(s) \leq \prod_{i=1}^m (n^i - 1) < n^{\frac{m \cdot (m+1)}{2}} \leq n^{\frac{3}{4} m^2}$.

Als gilt $s < \frac{3}{4} m^2 \log n$. Widerspruch zu $s = \lfloor m^2 \log n \rfloor \geq 7$. $\square$

**Satz 57**

Sei $n > 2^{25}$ eine Primzahl. Dann gibt es ein $r \in \mathbb{N}$ mit $(\log n)^2 < r \leq (\log n)^5$ und $\mathrm{ord}_r(n) > (\log n)^2$.

**Beweis:** Wegen $n > 2^{25}$ gilt $n > (\log n)^5$.

Sei $m = \lfloor (\log n)^2 \rfloor \geq 25^2$.

Dann gilt $7 \leq m^2 \log n \leq (\log n)^5 < n$.

Wegen Lemma 56 gibt es $1 \leq r \leq m^2 \log n \leq (\log n)^5$ mit $\mathrm{ord}_r(n) > m = \lfloor (\log n)^2 \rfloor$.

Wegen $\mathrm{ord}_r(n) \in \mathbb{N}$ muss $\mathrm{ord}_r(n) > (\log n)^2$ gelten.

Weiter gilt $\mathrm{ord}_r(n) \leq |(\mathbb{Z}/r\mathbb{Z})^*| \leq r$, d.h. $r > (\log n)^2$. $\square$

# Anhang: PRIMES $\in$ **P**

Der AKS-Test: Sei $n$ die binär kodierte Eingabezahl.

1. Falls $n \leq 2^{25}$, teste direkt, ob $n$ eine Primzahl ist. Sei jetzt $n > 2^{25}$ und damit $n > (\log n)^5$.
2. Teste, ob natürliche Zahlen $m, k \geq 2$ mit $n = m^k$ existieren. Wenn ja, ist $n$ keine Primzahl. Gelte ab jetzt $n \neq m^k$ für alle $m, k \geq 2$.
3. Suche eine Zahl $r$ mit $(\log n)^2 < r \leq (\log n)^5$, $\mathrm{ggT}(n, r) = 1$ und $\mathrm{ord}_r(n) > (\log n)^2$.

   Wird eine solche Zahl $r$ nicht gefunden, oder wird ein $r$ mit $\mathrm{ggT}(n, r) \geq 2$ gefunden, dann ist $n$ keine Primzahl.

   Sei jetzt $r$ wie oben.
4. Für alle $2 \leq a \leq r - 1$ teste, ob $\mathrm{ggT}(a, n) = 1$.

   Nehmen wir an, dies gelte (wenn nicht, ist $n$ keine Primzahl).
5. Sei $\ell = \lfloor \sqrt{\varphi(r)} \log n \rfloor$ und teste für alle $1 \leq a \leq \ell$, ob $(x + a)^n \equiv x^n + a \mod (x^r - 1, n)$ gilt.

   Wenn ja, ist $n$ eine Primzahl, sonst nicht.

# Anhang: PRIMES $\in$ **P**

Wir müssen jetzt 3 Punkte zeigen:

(a) Der AKS-Test kann so implementiert werden, dass er in Zeit polynomiell in $\log n$ läuft.

(b) Wenn $n$ eine Primzahl ist, dann akzeptiert der AKS-Test $n$.

(c) Wenn $n$ keine Primzahl ist, dann akzeptiert der AKS-Test $n$ nicht.

Zu Punkt (a): Wir gehen die 5 Punkte aus dem AKS-Test einzeln durch. Wenn wir im Folgenden von polynomieller Zeit sprechen, meinen wir stets "polynomiell in $\log n$".

1. klar
2. Beachte: Wenn $n = m^k$ mit $m, n \geq 2$, dann gilt $k \leq \log n$.

   Also können wir alle $k$ mit $2 \leq k \leq \log n$ einzeln betrachten.

   Für ein festes $k$ mit $2 \leq k \leq \log n$ können wir mittels binärer Suche in $\log n$ vielen Runden überprüfen, ob ein $m$ mit $n = m^k$ existiert.

3. Wir betrachten alle $r$ mit $(\log n)^2 < r \leq (\log n)^5$ nacheinander.

   Für ein festes $r$ können wir mit dem Euklidischen Algorithmus in polynomieller Zeit überprüfen, ob $\text{ggT}(r, n) = 1$.

   Um $\text{ord}_r(n) > (\log n)^2$ zu überprüfen, berechnen wir alle Werte $n^i \bmod r$ für $1 \leq i \leq \lfloor (\log n)^2 \rfloor$.

4. Ähnlich wie der vorherige Punkt.

5. Zunächst berechnen wir $\ell = \lfloor \sqrt{\varphi(r)} \log n \rfloor$ in polynomieller Zeit.

   Die Zahl $\varphi(r) \leq r - 1$ berechnen wir, indem wir $\text{ggT}(i, r)$ für alle $1 \leq i \leq r - 1$ berechnen.

   Dann berechnen wir $\sqrt{\varphi(r)}$ und $\log n$ ausreichend genau, um $\lfloor \sqrt{\varphi(r)} \log n \rfloor$ ermitteln zu können.

   Alternativ kann im AKS-Test auch $\ell = \varphi(r) \cdot (\text{Anzahl Bits von } n)$ gewählen werden.

Dann können wir alle $a$ mit $1 \leq a \leq \ell$ nacheinander durchgehen (beachte: $\ell \leq \sqrt{r} \log n \leq (\log n)^{3,5}$).

Für ein festes $a$ kann die Kongruenz $(x+a)^n \equiv x^n + a \mod (x^r - 1, n)$ wegen Lemma 52 in polynomieller Zeit überprüft werden.

Zu Punkt (b): Sei $n$ eine Primzahl.

Offensichtlich passiert $n$ die Schritte 1 und 2 aus dem AKS-Test.

Wegen Satz 57 gibt es ein $r$ mit den in Schritt 3 gesuchten Eigenschaften.

Offensichtlich passiert $n$ auch Schritt 4.

Wegen Lemma 49 gilt $(x+a)^n \equiv x^n + a \mod n$ und damit auch $(x+a)^n \equiv x^n + a \mod (x^r - 1, n)$ für alle $a$.

Zu Punkt (c): Sei $n$ keine Primzahl, und sei $p$ ein Primteiler von $n$.

Angenommen, $n$ passiert den AKS-Test.

Dann gelten folgende Eigenschaften:

- $n > 2^{25}$
- $n \neq p^k$ für alle $k \geq 1$.
- $(\log n)^2 < r \leq (\log n)^5 < n$
- $\mathrm{ggT}(a, n) = 1$ für alle $1 \leq a \leq r$ (insbesondere $r < p$)
- $(\log n)^2 < \mathrm{ord}_r(n) \leq \varphi(r) \leq r - 1$
- $\ell = \lfloor \sqrt{\varphi(r)} \log n \rfloor \leq \sqrt{\varphi(r)} \log n \leq \varphi(r)$
- $(x + a)^n \equiv x^n + a \mod (x^r - 1, n)$ für alle $0 \leq a \leq \ell$

Wir werden aus diesen Annahmen einen Widerspruch ableiten.

Wir verwenden im Weiteren folgende Sätze aus der Algebra (zum Teil ohne Beweis).

Für einen Körper $\mathbb{F}$ ist die Charakteristik die kleinste Zahl $n \geq 1$, so dass

$$n := \underbrace{1 + 1 + \cdots + 1}_{n \text{ mal}} = 0 \text{ in } \mathbb{F}$$

gilt, bzw. 0 falls ein solches $n$ nicht existiert.

Falls die Charakteristik von $\mathbb{F}$ nicht 0 ist, muss sie eine Primzahl sein:

Angenommen $\mathbb{F}$ hat Charakterisitk $n = a \cdot b$ wobei $1 < a, b < n$.

Dann gilt $ab = n = 0$ in $\mathbb{F}$.

Wegen der Nullteilerfreiheit von $n$ folgt $a = 0$ oder $b = 0$ in $\mathbb{F}$.

Widerspruch zur Minimalität von $n$.

### Satz 58

Sei $f(x) \in \mathbb{F}[x]$ mit $f(x) \neq 0$ ein Polynom von Grad $d$ mit Koeffizienten aus dem Körper $\mathbb{F}$. Dann existiert ein Erweiterungskörper $\mathbb{F}' \supseteq \mathbb{F}$ und Elemente $\alpha_1, \dots, \alpha_d \in \mathbb{F}'$, so dass in $\mathbb{F}'[x]$ gilt:

$$f(x) = \prod_{i=1}^{d} (x - \alpha_i)$$

Der Körper $\mathbb{F}'$ enthält $\mathbb{F}$ und hat damit die gleiche Charakteristik wie $\mathbb{F}$.

Ist $\mathbb{F}$ endlich, dann ist auch $\mathbb{F}'$ endlich.

Man sagt auch, dass $f(x)$ über $\mathbb{F}'$ in Linearfaktoren zerfällt.

Der Körper $\mathbb{F}'$ aus obigen Satz ist der Zerfällungskörper von $f(x)$ über $\mathbb{F}$.

**Beachte:** Sei $\alpha$ Nullstelle des Polynoms $f(x) \in \mathbb{F}[x]$ mit $f(x) \neq 0$: $f(\alpha) = 0$ in $\mathbb{F}$.

Polynomdivision von $f(x)$ durch $(x - \alpha)$ zeigt: Es existiert ein Polynom $g(x) \in \mathbb{F}[x]$ mit $f(x) = (x - \alpha) \cdot g(x)$.

Existiert ein Polynom $g(x) \in \mathbb{F}[x]$ und $n \geq 2$ mit $f(x) = (x - \alpha)^n \cdot g(x)$, so ist $\alpha$ mehrfache Nullstelle von $f(x)$.

Für ein Polynom $f(x) \in \mathbb{F}[x]$ definieren wir die formale Ableitung $f'(x)$ nach den gewohnten Regeln, wobei $a \in \mathbb{F}$:

$$(f(x) + g(x))' = f'(x) + g'(x), \;\; a' = 0, \;\; (a \cdot x^n)' = (n \cdot a) \cdot x^{n-1} \text{ für } n > 0$$

### Satz 59

Sei $f(x) \in \mathbb{F}[x]$, $f(x) \neq 0 \neq f'(x)$, und sei $\alpha \in \mathbb{F}$ Nullstelle von $f(x)$. Dann ist $\alpha$ eine mehrfache Nullstelle von $f(x)$ genau dann, wenn $\alpha$ Nullstelle von $f'(x)$ ist.

**Beweis:**

Da $\alpha$ eine Nullstelle von $f(x) \neq 0$ ist, können wir das Polynom $f(x)$ schreiben als $f(x) = (x-\alpha)\cdot g(x)$.

Mit der Produktregel erhalten wir $f'(x) = g(x) + (x-\alpha)\cdot g'(x)$.

Wegen $f(x) \neq 0$ gilt $g(x) \neq 0$.

Ist nun $\alpha$ eine Nullstelle von $f'(x)$, so folgt $0 = f'(\alpha) = g(\alpha)$.

Also gibt es ein Polynom $h(x)$ mit $g(x) = (x-\alpha)\cdot h(x)$.

Damit gilt $f(x) = (x-\alpha)^2 \cdot h(x)$ und $\alpha$ ist mehrfache Nullstelle von $f(x)$.

Ist andererseits $\alpha$ eine mehrfache Nullstelle von $f(x)$, so gibt es ein Polynom $h(x)$ mit $f(x) = (x-\alpha)^2 \cdot h(x)$.

Mit der Produktregel folgt $f'(x) = 2\cdot(x-\alpha)\cdot h(x) + (x-\alpha)^2\cdot h'(x)$.

Also gilt $f'(\alpha) = 0$. $\square$

**Satz 60**

Sei $\mathbb{F}$ ein endlicher Körper der Charakteristik $p > 0$. Dann ist die Abbildung $\phi : x \mapsto x^p$ ein Automorphismus von $\mathbb{F}$.

**Beweis:**

Für $1 < k < p$ ist der Binomialkoeffizient

$$\binom{p}{k} = \frac{p!}{k! \cdot (p-k)!}$$

durch $p$ teilbar (der Zähler ist durch $p$ teilbar, der Nenner aber nicht).

Da der Körper $\mathbb{F}$ Charakteristik $p$ hat, gilt

$$\binom{p}{k} = 0 \text{ in } \mathbb{F}.$$

Also gilt für alle $a, b \in \mathbb{F}$:

$$(a+b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k b^{p-k} = a^p + b^p \tag{10}$$

$$(a-b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k (-b)^{p-k} = a^p - b^p \tag{11}$$

Aus (11) folgt insbesondere das $\phi$ injektiv ist:

$$\phi(a) = \phi(b) \;\Rightarrow\; 0 = a^p - b^p = (a-b)^p \;\Rightarrow\; a - b = 0 \;\Rightarrow\; a = b$$

Da $\mathbb{F}$ endlich ist, ist $\phi$ bijektiv.

Aus (10), $(ab)^p = a^p b^p$, $0^p = 0$ und $1^p = 1$ folgt weiter, dass $\phi$ ein Automorphismus ist. $\square$

Eine Gruppe $\mathbb{G} = (G, \circ)$ ist zyklisch, falls ein Element $g \in G$ existiert, so dass $G = \{g^n \mid n \in \mathbb{Z}\}$ ($g$ ist dann ein Erzeuger von $\mathbb{G}$).

Jede unendliche zyklische Gruppe ist isomorph zu $(\mathbb{Z}, +)$, und für jede endliche zyklische Gruppe $\mathbb{G}$ existiert ein $n \geq 1$ mit $\mathbb{G} \cong (\mathbb{Z}/n\mathbb{Z}, +)$.

**Satz 61**

Jede Untergruppe einer zyklischen Gruppe $\mathbb{G} = (G, \circ)$ ist wieder zyklisch.

**Beweis:**

Sei $G = \{g^n \mid n \in \mathbb{Z}\}$ und $U$ eine Untergruppe von $\mathbb{G}$, o.B.d.A. $|U| > 1$.

Sei $n > 0$ minimal mit $h := g^n \in U$.

Dann gilt natürlich $h^m \in U$ für alle $m \in \mathbb{Z}$.

Sei umgekehrt $x \in U$.

Es gibt ein $k \in \mathbb{Z}$ mit $x = g^k$.

Division mit Rest: $k = q \cdot n + r$ mit $q \in \mathbb{Z}$ und $0 \leq r < n$.

Also gilt $x = g^{q \cdot n + r} = (g^n)^q g^r = h^q g^r$.

Wegen $x, h \in U$ gilt $g^r \in U$.

Die Minimalität von $n$ impliziert $r = 0$, d.h. $x = h^q$.

Also gilt $U = \{h^m \mid m \in \mathbb{Z}\}$. $\square$

### Satz 62

Sei $\mathbb{G} = (G, \circ)$ eine endliche zyklische Gruppe mit $r \geq 2$ Elementen. Dann hat $\mathbb{G}$ genau $\varphi(r)$ viele Erzeuger.

**Beweis:** Sei $G = \{g^0, g^1, \dots, g^{r-1}\}$.

Beachte: $g^x = g^{x \bmod r}$ für alle $x \in \mathbb{Z}$.

**Behauptung:** Sei $1 \leq a \leq r - 1$. Dann ist $g^a$ ein Erzeuger genau dann, wenn $\mathrm{ggT}(a, r) = 1$.

**Fall 1:** $\mathrm{ggT}(a, r) = 1$.

Dann existieren $x, y \in \mathbb{Z}$ mit $1 = x \cdot a + y \cdot r$.

Also gilt $g = g^{x \cdot a + y \cdot r} = (g^a)^x (g^r)^y = (g^a)^x$.

Also erzeugt $g^a$ die Gruppe $\mathbb{G}$.

**Fall 2:** $\mathrm{ggT}(a, r) = q > 1$.

Angenommen $g = (g^a)^d$ für ein $d \in \mathbb{Z}$.

Also gilt $d \cdot a = 1 + x \cdot r$, d.h. $1 = d \cdot a - x \cdot r$ für ein $x \in \mathbb{Z}$.

Da $q$ Teiler von $a$ und $r$ ist, wäre $q$ auch ein Teiler von 1.

Widerspruch zu $q > 1$.

Also erzeugt $g^a$ nicht die Gruppe. $\square$

**Satz 63**

Sei $\mathbb{F}$ ein endlicher Körper. Dann bildet $\mathbb{F}^* = \mathbb{F} \setminus \{0\}$ zusammen mit der Multiplikation eine zyklische Gruppe (die multiplikative Gruppe von $\mathbb{F}$).

# Anhang: PRIMES $\in$ **P**

Nun zurück zum AKS-Test. Es gelten die Annahmen von Folie 299.

Wir betrachten das Polynom $f(x) = x^r - 1$ über dem Körper $\mathbb{F}_p$ mit $p$ Elementen.

Sei $\mathbb{F}$ der Zerfällungskörper von $f(x)$ über $\mathbb{F}_p$ (Satz 58).

Also ist $\mathbb{F}$ ein endlicher Erweiterungskörper der Charakteristik $p$, in dem Nullstellen $\alpha_1, \ldots, \alpha_r \in \mathbb{F}$ existieren mit

$$f(x) = x^r - 1 = \prod_{i=1}^{r} (x - \alpha_i).$$

Es gilt $f'(x) = r \cdot x^{r-1}$.

Wegen $\alpha_i \neq 0$ und $r \not\equiv 0 \mod p$ gilt $f'(\alpha_i) \neq 0$ in $\mathbb{F}$.

Wegen Satz 59 sind die Nullstellen $\alpha_1, \ldots, \alpha_r$ alle verschieden.

Sei $U = \{\alpha_1, \dots, \alpha_r\} \subseteq \mathbb{F}^*$.

Dann ist $U$ eine Untergruppe von $\mathbb{F}^*$ mit $r$ Elementen.

Wegen Satz 61 und 63 ist $U$ zyklisch.

Nach Satz 62 hat $U$ genau $\varphi(r)$ viele Erzeuger.

Sei $L = \{0, 1, \dots, \ell\} \subseteq \mathbb{F}_p \subseteq \mathbb{F}$ und $-L = \{-a \mid a \in L\} \subseteq \mathbb{F}$.

Da $\ell \le \varphi(r) < r < p$ besteht $L$ aus $\ell + 1$ vielen Elementen.

Weder $0 \in -L$ noch $-1 \in -L$ erzeugen $U$ (da $|U| = r > (\log n)^2 > 25^2$).

Wegen $\ell \le \varphi(r)$ gilt $|(-L) \setminus \{0, -1\}| = \ell - 1 < \varphi(r)$.

Also enthält $U$ einen Erzeuger $\alpha$ mit $\alpha \notin -L$.

Es gilt somit $\alpha + a \neq 0$ in $\mathbb{F}$ für alle $a \in L$.

### Definition (introspektive Zahlen)

Sei $g(x) \in \mathbb{F}_p[x]$. Eine natürliche Zahl $m \in \mathbb{N}$ ist introspektiv für $g(x)$, falls gilt:

$$\forall \beta \in U : g(\beta)^m = g(\beta^m) \text{ in } \mathbb{F}.$$

### Lemma 64

Wenn $m_1$ und $m_2$ introspektiv für $g(x) \in \mathbb{F}_p[x]$ sind, dann ist auch $m_1 m_2$ introspektiv für $g(x)$.

**Beweis:** Sei $\beta \in U$ beliebig. Dann gilt

$$g(\beta)^{m_1 m_2} = (g(\beta)^{m_1})^{m_2} = g(\beta^{m_1})^{m_2} = g((\beta^{m_1})^{m_2}) = g(\beta^{m_1 \cdot m_2})$$

Beachte hierzu: $\beta^{m_1} \in U$. □

### Lemma 65

Sei $a \in L$. Dann sind $p$ und $\frac{n}{p}$ introspektiv für $x + a$.

**Beweis:**

Nach Satz 60 ist $x \mapsto x^p$ ein Automorphismus von $\mathbb{F}$.

Also gilt $g(\alpha^p) = g(\alpha)^p$ für jedes Polynom $g(x) \in \mathbb{F}_p[x]$ und alle $\alpha \in \mathbb{F}$ (beachte: $a^p = a$ für alle $a \in \mathbb{F}_p$ nach dem kleinen Satz von Fermat und damit $a \cdot (\alpha^p)^n = (a \cdot \alpha^n)^p$).

Insbesondere ist $p$ introspektiv für $x + a$.

Es bleibt noch zu zeigen, dass $\frac{n}{p}$ introspektiv für $x + a$ ist, falls $a \in L$.

Sei also $a \in L \subseteq \mathbb{F}_p$ und $\beta \in U$.

Insbesondere gilt $a^p = a$ in $\mathbb{F}_p$ (kleiner Satz von Fermat).

Es folgt:

$$
\begin{aligned}
& (\beta + a)^{\frac{n}{p}} = \beta^{\frac{n}{p}} + a && (12)\\
\Leftrightarrow\quad & (\beta + a)^n = (\beta^{\frac{n}{p}} + a)^p \\
\Leftrightarrow\quad & (\beta + a)^n = \beta^n + a^p \\
\Leftrightarrow\quad & (\beta + a)^n = \beta^n + a && (13)
\end{aligned}
$$

Nun gilt $(x+a)^n \equiv x^n + a \bmod (x^r - 1, n)$, d.h. es existieren Polynome $u(x), v(x) \in \mathbb{Z}[x]$ mit

$$(x+a)^n = x^n + a + u(x) \cdot (x^r - 1) + n \cdot v(x).$$

Durch Einsetzen von $\beta$ erhalten wir $u, v \in \mathbb{F}$ mit:

$$(\beta + a)^n = \beta^n + a + (\beta^r - 1) \cdot u + v \cdot n = \beta^n + a \text{ in } \mathbb{F}$$

(beachte: $p|n$, d.h. $n = 0$ in $\mathbb{F}$ und $\beta^r - 1 = 0$ in $\mathbb{F}$).

Also ist (13) wahr und damit auch (12) wahr. $\square$

### Lemma 66

Wenn $m$ introspektiv für die Polynome $f(x), g(x) \in \mathbb{F}_p[x]$ ist, dann ist $m$ auch introspektiv für das Polynom $(f \cdot g)(x) = f(x) \cdot g(x)$.

**Beweis:** Es gilt

$$(f \cdot g)(\beta)^m = (f(\beta) \cdot g(\beta))^m = f(\beta)^m \cdot g(\beta)^m = f(\beta^m) \cdot g(\beta^m) = (f \cdot g)(\beta^m).$$

$\square$

### Lemma 67

Seien $i, j, e_a \geq 0$ $(a \in L)$ beliebig. Dann ist die Zahl $(\frac{n}{p})^i \cdot p^j$ introspektiv für das Polynom $\prod_{a \in L}(x+a)^{e_a}$.

**Beweis:**

Nach Lemma 65 sind $p$ und $\frac{n}{p}$ introspektiv für $x + a$ $(a \in L)$.

Wegen Lemma 64 ist $(\frac{n}{p})^i \cdot p^j$ introspektiv für $x + a$ $(a \in L)$.

Wegen Lemma 66 ist $(\frac{n}{p})^i \cdot p^j$ introspektiv für $\prod_{a \in L}(x + a)^{e_a}$. $\square$

Wegen $\mathrm{ggT}(n, r) = 1$ gilt auch $\mathrm{ggT}(\frac{n}{p}, r) = 1$ und $\mathrm{ggT}(p, r) = 1$.

Also gilt $(\frac{n}{p} \bmod r), (p \bmod r) \in (\mathbb{Z}/r\mathbb{Z})^*$.

Sei

$$G = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \bmod r \;\middle|\; i, j \geq 0 \right\} \subseteq (\mathbb{Z}/r\mathbb{Z})^*$$

die von $p$ und $\frac{n}{p}$ erzeugte Untergruppe von $(\mathbb{Z}/r\mathbb{Z})^*$.

# Anhang: PRIMES $\in$ **P**

Sei $t = |G|$.

Da $G$ die von $n$ erzeugte Untergruppe von $(\mathbb{Z}/r\mathbb{Z})^*$ enthält, gilt

$$(\log n)^2 < \mathrm{ord}_r(n) \leq t \leq \varphi(r). \tag{14}$$

Sei $P$ die Menge der Polynome

$$P = \{\prod_{a \in L} (x+a)^{e_a} \mid e_a \geq 0, \sum_{a \in L} e_a < t\} \subseteq \mathbb{F}_p[x]$$

Dann gilt

$$|P| \overset{(a)}{=} |\{(e_a)_{a \in L} \mid e_a \geq 0, \sum_{a \in L} e_a < t\}| \overset{(b)}{=} \binom{t+\ell}{t-1}. \tag{15}$$

Gleichung (a) folgt aus der Tatsache, dass sich jedes Polynom aus $\mathbb{F}[x]$ eindeutig in irreduzible Polynome faktorisieren lässt.

Gleichung (b) folgt aus Lemma 55 und $|L| = \ell + 1$.

# Anhang: PRIMES $\in$ **P**

Sei $\alpha$ der auf Folie 311 gewählte Erzeuger von $U$.

Sei $\mathcal{G} = \{g(\alpha) \in \mathbb{F} \mid g \in P\} \subseteq \mathbb{F}$.

Da $\alpha + a \neq 0$ für alle $a \in L$ gilt (Folie 311), folgt $\mathcal{G} \subseteq \mathbb{F}^*$.

**Lemma 68**

Die Abbildung $\Psi : P \to \mathcal{G}$ mit $\Psi(g(x)) = g(\alpha)$ ist bijektiv.

**Beweis:** Offensichtlich ist $\Psi$ surjektiv.

Bleibt zu zeigen, dass $\Psi$ injektiv ist.

Seien $g_1(x), g_2(x) \in P$ mit $g_1(x) \neq g_2(x)$ und $g_1(\alpha) = g_2(\alpha)$.

Wir leiten einen Widerspruch ab.

Sei $m = (\frac{n}{p})^i \cdot p^j$ mit $i, j \geq 0$.

Wegen Lemma 67 ist $m$ introspektiv für jedes Polynom $g(x) \in P$.

Also gilt $g_1(\alpha^m) = g_1(\alpha)^m = g_2(\alpha)^m = g_2(\alpha^m)$.

Also ist $\alpha^m = \alpha^{m \bmod r} \in \mathbb{F}$ Nullstelle des Polynoms $g_1(x) - g_2(x) \neq 0$, welches einen Grad $< t$ hat (und somit weniger als $t$ Nullstellen hat).

Da aber $\alpha^m \neq \alpha^{m'}$ für $m, m' \in G$ mit $m \neq m'$ ($\alpha$ hat Ordnung $r$ und $1 \leq m, m' \leq r - 1$), erhalten wir $|G| = t$ Nullstellen. $\square$

Wegen Lemma 68 gilt

$$|\mathcal{G}| = |P| \stackrel{(15)}{=} \binom{t+\ell}{t-1}. \tag{16}$$

# Anhang: PRIMES $\in$ **P**

Sei

$$\hat{I} = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \;\middle|\; 0 \le i,j \le \lfloor\sqrt{t}\rfloor \right\} \subseteq \mathbb{N}.$$

Wegen Lemma 67 ist jede Zahl $m \in \hat{I}$ introspektiv für jedes Polynom aus $P$.

**Lemma 69**

$|\hat{I}| = (\lfloor\sqrt{t}\rfloor + 1)^2$.

**Beweis:** Eine unsere Annahmen von Folie 299 besagt $n \neq p^k$ für alle $k \geq 1$.

Also hat $\frac{n}{p}$ einen von $p$ verschiedenen Primteiler.

Also gilt $(\frac{n}{p})^{i_1} \cdot p^{j_1} \neq (\frac{n}{p})^{i_2} \cdot p^{j_2}$ falls $(i_1, j_1) \neq (i_2, j_2)$. $\square$

Wegen Lemma 69 gilt $|\hat{I}| > t = |G|$.

Also ist die Abbildung $\theta : \hat{I} \to G$ mit $\theta(m) = m \bmod r$ nicht injektiv.

Seien $m_1, m_2 \in \hat{I}$ mit $m_1 < m_2$ und $m_1 \equiv m_2 \bmod r$.

Aus der Definition von $\hat{I}$ folgt $m_2 \le (\frac{n}{p})^{\sqrt{t}} \cdot p^{\sqrt{t}} = n^{\sqrt{t}}$.

**Lemma 70**

Jedes Element $g(\alpha) \in \mathcal{G}$ $(g(x) \in P)$ ist Nullstelle des Polynoms $y^{m_1} - y^{m_2}$.

**Beweis:**

Wegen $\alpha^r = 1$ in $\mathbb{F}$ und $m_1 \equiv m_2 \bmod r$ gilt $\alpha^{m_1} = \alpha^{m_2}$ in $\mathbb{F}$.

Also gilt für jedes $g(\alpha) \in \mathcal{G}$ $(g(x) \in P)$:

$$g(\alpha)^{m_1} = g(\alpha^{m_1}) = g(\alpha^{m_2}) = g(\alpha)^{m_2}. \qquad \square$$

# Anhang: PRIMES $\in$ **P**

Da das Polynom $y^{m_1} - y^{m_2}$ höchstens $m_2 \leq n^{\sqrt{t}}$ viele Nullstellen hat, folgt aus Lemma 70:

$$|\mathcal{G}| \leq n^{\sqrt{t}}.$$

Andererseits gilt:

$$\begin{aligned} |\mathcal{G}| &\overset{(16)}{=} \binom{t+\ell}{t-1} \overset{(a)}{\geq} \binom{\ell + 1 + \lfloor \sqrt{t} \log n \rfloor}{\lfloor \sqrt{t} \log n \rfloor} \\ &\overset{(b)}{\geq} \binom{2\lfloor \sqrt{t} \log n \rfloor + 1}{\lfloor \sqrt{t} \log n \rfloor} \overset{(c)}{>} 2^{\sqrt{t} \log n} = n^{\sqrt{t}} \end{aligned}$$

Dies liefert dann schließlich den gewünschten Widerspruch zu den Annahmen von Folie 299.

- Zu (a): Es gilt $(\log n)^2 < t$ nach (14) und damit $\lfloor \sqrt{t} \log n \rfloor \leq t - 1$. Außerdem gilt $\binom{a+b}{a+c} \geq \binom{b}{c}$ für $a \geq 0$.

- Zu (b): Es gilt $\ell = \lfloor \sqrt{\varphi(r)} \log n \rfloor \geq \lfloor \sqrt{t} \log n \rfloor$ nach (14).
- Zu (c): Wir zeigen unten $\binom{2x+1}{x} > 2^{x+1}$ für $x \geq 4$ (dies liefert dann (c) mit $x = \lfloor \sqrt{t} \log n \rfloor \geq \lfloor (\log n)^2 \rfloor \geq 25^2 = 625$).

**Lemma 71**

Für $x \geq 4$ gilt $\binom{2x+1}{x} > 2^{x+1}$

**Beweis:** Es gilt

$$\binom{2x+1}{x} = \binom{2x+1}{x+1}$$

und dies sind die maximalen Binomialkoeffizienten der Form $\binom{2x+1}{i}$.

Also gilt

$$\frac{1}{2}\left[\binom{2x+1}{x} + \binom{2x+1}{x+1}\right] \geq \frac{1}{2(x+1)} \sum_{i=0}^{2x+1} \binom{2x+1}{i} = \frac{2^{2x+1}}{2(x+1)}$$

und somit

$$\binom{2x+1}{x} = \frac{1}{2}\left[\binom{2x+1}{x} + \binom{2x+1}{x+1}\right] \geq \frac{2^{2x}}{x+1}.$$

Nun gilt für $x \geq 4$: $x+1 < 2^{x-1}$ und damit

$$\frac{2^{2x}}{x+1} > \frac{2^{2x}}{2^{x-1}} = 2^{x+1}. \qquad \square$$