

Orakel Berechnungen

Berechnungswelten

Warum ist die $P \stackrel{?}{=} NP$ Frage immer noch unbeantwortet?

- ? Vielleicht, weil $P \neq NP$ zwar wahr, aber **nicht beweisbar** ist?
- ? Vielleicht, weil $P = NP$ in einigen „**Berechnungswelten**“ sogar wahr ist?

Sei $A \subseteq \Sigma^*$ eine Sprache.

(a) Eine Turingmaschine M mit Orakel A besitzt ein zusätzliches **Orakelband**.

- ▶ Wenn das Orakelband mit der Eingabe $w \#$ beschrieben ist, dann wird in einem einzigen Berechnungsschritt mitgeteilt, ob w zur Sprache A gehört.
- ▶ Die Beschriftung des Orakelbands benötigt andererseits eine Laufzeit proportional zur Länge der Anfrage.

(b) Sei K eine durch die Beschränkung einer Ressource
wie Laufzeit oder Speicherplatz

definierte Komplexitätsklasse. Dann ist K^A entsprechend zu definieren, wobei jetzt Fragen an das Orakel A zugelassen sind.

Gibt es Berechnungswelten A, B mit $P^A = NP^A$ und $P^B \neq NP^B$?

- (a) Wenn $A \in P$, dann ist $P^A = P$.
- (b) Wenn $A \in PSPACE$, dann ist $PSPACE^A = PSPACE$.
- (c) Wenn A PSPACE-vollständig ist, dann ist $P^A = NP^A = PSPACE$.

- (a) $P^A \subseteq P$, denn eine Turingmaschine mit Orakel A kann Orakelanfragen mit nur polynomiellem Mehraufwand auch selbst beantworten.
- (b) analog zu (b).
- (c)
 - ▶ $NP^A \subseteq PSPACE$: ✓.
 - ▶ Die Beziehung $PSPACE \subseteq P^A$ folgt aus der PSPACE-Vollständigkeit von A :
 - ★ Für jede Sprache $L \in PSPACE$ gibt es eine effiziente deterministische TM M mit

$$w \in L \Leftrightarrow M(w) \in A.$$

- (a) $P^A = NP^A$, wenn A eine PSPACE-vollständige Sprache ist.
- (b) Es gibt ein Orakel A mit $P^A \neq NP^A$.

Wir konstruieren ein Orakel A , so dass die Sprache

$$L_A = \{w \mid \exists x \in A (|x| = |w|)\}$$

zu NP^A , nicht aber zu P^A gehört.

- Offensichtlich gilt $L_A \in NP^A$ für jedes Orakel A .
 - ▶ Für Eingabe w rate einen String x gleicher Länge und frage, ob $x \in A$.
 - ▶ Akzeptiere genau dann, wenn die Antwort positiv ist.
- Sei M_k eine beliebige Aufzählung aller Orakel-Turingmaschinen, wobei M_k in Zeit $O(n^k)$ rechnet.
 - ▶ Um $L_A \notin P^A$ zu garantieren, stellen wir sicher, dass sich M_k und L_A auf einer Eingabe $w = 1^{n_k}$ unterscheiden.
 - ▶ Wir nehmen an, dass wir dieses Ziel bereits für M_1, \dots, M_{k-1} erreicht haben:
 - ★ $\{w_1, \dots, w_m\}$ sei die Menge der während der Berechnung irgendeiner Maschine M_i auf Eingabe 1^{n_i} ($1 \leq i \leq k-1$) an das Orakel A gestellten Anfragen.
 - ★ Wie ist n_k zu definieren und wie soll das Orakel die von M_k auf Eingabe 1^{n_k} gestellten Fragen beantworten?

$$L_A = \{ \mathbf{w} \mid \exists \mathbf{x} \in A (|\mathbf{x}| = |\mathbf{w}|) \}$$

- Simuliere M_k auf der Eingabe 1^{n_k} . (Es gelte $n_k > \max\{|w_1|, \dots, |w_m|\}$ und $2^{n_k} > n_k^k$.)
 - ▶ Wenn M_k eine Anfrage y aus $\{w_1, \dots, w_m\}$ stellt, dann antwortet A konsistent.
 - ▶ Ist die Anfrage y hingegen neu, dann antwortet A mit **nein**.
- Wenn M_k die Eingabe 1^{n_k} akzeptiert:
 - ▶ Erzwinge $1^n \notin L_A$ durch Ausschluss **aller** Worte der Länge n für A .
- Wenn M_k die Eingabe 1^n verwirft:
 - ▶ M_k rechnet in Zeit $n^k < 2^n$. Es gibt also ein Wort u der Länge n , das von M_k (für Eingabe 1^{n_k}) nicht nachgefragt wurde.
 - ▶ Definiere A so, dass u das einzige akzeptierte Wort der Länge n ist.

Kann $P \neq NP$ mit einem Diagonalisierargument bewiesen werden?

- Unser Beweis der Zeit-Hierarchie „**relativiert**“, gilt also in jeder Berechnungswelt:

Im Beweis der Zeithierarchie fragen wir nicht nach, wie denn die simulierte Turingmaschine rechnet.

- Aber die Aussage $P = NP$ **relativiert nicht!**

Ein Beweis von $P \neq NP$ muss weitere Methoden nutzen, um die Arbeitsweise der Turingmaschine genau „unter die Lupe“ nehmen zu können.

Schwierigkeitsgrade in NP

- Für nur wenige, aber anscheinend sehr schwierige Probleme
wie die **Graph-Isomorphie** und das **Faktorisierungsproblem**
kann man bisher **kein** NP-Vollständigkeitsergebnis nachweisen,
Es gibt sogar starke Indizien gegen eine NP-Vollständigkeit.
- Wir zeigen jetzt: Wenn $P \neq NP$, dann gibt es sogar zwangsläufig
Probleme in **NP**, die weder **NP-vollständig** noch **effizient lösbar** sind.

Ein Problem mit mittlerem Schwierigkeitsgrad

Die Funktion $\mathbf{H} : \mathbb{N} \rightarrow \mathbb{N}$ muss noch konstruiert werden.

Es gelte $P \neq NP$. Wir zeigen, dass

$$SAT_H = \{ \phi \mathbf{0}^n \mathbf{1}^{H(n)} \mid |\phi| = n \text{ und die aussagenlogische Formel } \phi \text{ ist erfüllbar} \}$$

weder NP-vollständig ist noch in P liegt.

- Im Vergleich zum Erfüllbarkeitsproblem SAT ist die Eingabelänge durch das Hinzufügen vieler Einsen stark angestiegen.

Zur Lösung des Erfüllbarkeitsproblems in SAT_H steht viel mehr Laufzeit zur Verfügung als in SAT.

- Wie ist $\mathbf{H}(n)$ zu definieren?

Wie ist H zu definieren?

$SAT_H = \{ \phi 0 1^{H(n)} \mid |\phi| = n \text{ und die aussagenlogische Formel } \phi \text{ ist erfüllbar} \}$.

- Wenn $SAT_H \in P$, dann gibt es eine TM M^* , die SAT_H berechnet.
 - Wir machen uns auf die Suche nach M^* .
 - ▶ $H(n)$ muss effizient berechenbar sein: Für die Definition von $H(n)$ betrachten wir nur „kurze“ Eingaben $x = \phi 0 1^{H(m)}$ mit $|\phi| = m$ und $|x| \leq \log_2 n$.
 - ▶ Setze $H(n) = \langle M \rangle$ ($\langle M \rangle$ ist die „Gödelnummer“ von M) genau dann, wenn
 - ★ $\langle M \rangle$ minimal mit $\langle M \rangle < \log_2 \log_2 n$ ist und
 - ★ M auf allen kurzen Eingaben korrekt in höchstens $\langle M \rangle n^{\langle M \rangle}$ Schritten rechnet.
- Gibt es eine solche Maschine M nicht, setze $H(n) = \log_2 \log_2 n$.
- ★ Ist $H(n)$ wohldefiniert? Ja, denn nur das Verhalten auf SAT_H -Eingaben von höchstens logarithmischer Länge ist relevant.
 - ★ Korrektheit auf kurzen Eingaben kann in polynomieller Zeit in n überprüft werden
 $\Rightarrow H$ ist effizient berechenbar.

- Wenn $SAT_H \in P$, dann ist $H(n) = O(1)$. **Warum?**
- Aber wir haben dann doch nur polynomiell viele, also relativ wenige Einsen zur Eingabe ϕ hinzugefügt, und wir nehmen $P \neq NP$ an!

SAT_H liegt nicht in P

$SAT_H = \{ \phi \mathbf{0} \mathbf{1}^{n^{H(n)}} \mid |\phi| = \mathbf{n} \text{ und die aussagenlogische Formel } \phi \text{ ist erfüllbar} \}$.

- $SAT_H \in P \Leftrightarrow H(n) = O(1)$.
 - ⇒ Wenn die TM M die Sprache SAT_H akzeptiert, dann ist $H(n) \leq \langle M \rangle$ für genügend große n .
 - ⇐ Wenn $H(n) = O(1)$, dann gibt es M mit $H(n) = \langle M \rangle$ für unendliche viele n . Aber dann wird SAT_H in polynomieller Zeit von M akzeptiert.
- Wenn $SAT_H \notin P$, dann ist $H(n) = \omega(1)$.
 - ▶ Ansonsten gibt es eine TM M , so dass $H(n) = \langle M \rangle$ für unendlich viele n .
 - ▶ M löst SAT_H in polynomieller Zeit.
- SAT_H liegt nicht in P, wenn $P \neq NP$.
 - ▶ Wenn SAT_H tatsächlich in P liegt, folgt $H(n) = O(1)$.
 - ▶ Dann ist SAT_H das Erfüllbarkeitsproblem SAT, dessen Eingabe mit nur polynomiell vielen Einsen aufgefüllt ist.
 - ▶ Da H in polynomieller Zeit berechnet werden kann, ist SAT im Widerspruch zur Annahme $P \neq NP$ effizient lösbar.

SAT_H ist nicht NP-vollständig

$SAT_H = \{ \phi \mathbf{01}^{n^{H(n)}} \mid |\phi| = n \text{ und die aussagenlogische Formel } \phi \text{ ist erfüllbar} \}$.

Angenommen, SAT_H ist NP-vollständig

- Es gibt eine polynomielle Reduktion von SAT auf SAT_H .
 - ▶ Also gibt es eine in Zeit n^i berechenbare Transformation T mit

$$\psi \in SAT \Leftrightarrow T(\psi) \in SAT_H.$$

- SAT_H liegt nicht in P und $H(n) = \omega(1)$ folgt.
- Wenn $T(\psi) = \phi \mathbf{01}^{H(|\phi|)}$, dann wird das Erfüllbarkeitsproblem von ψ auf das Erfüllbarkeitsproblem der wesentlich kleineren Formel ϕ reduziert:

Wir erhalten einen effizienten rekursiven Algorithmus für SAT.

Natürliche Beweise

Ist die $P \stackrel{?}{=} NP$ Frage vielleicht deshalb so schwer, weil wir sie mit „**gängigen**“ Methoden gar nicht beantworten können?

- Jede Sprache in P kann von einer Schaltkreisfamilie polynomieller Größe erkannt werden, gehört also zur Klasse $P_{/poly}$ aller Sprachen mit Schaltkreisen polynomieller Größe.
- Es genügt somit der Nachweis, dass (irgend)eine NP-vollständige Sprache nicht zu $P_{/poly}$ gehört.
- Natürliche Beweise stellen einen „natürlichen Ansatz“ in dieser Richtung dar. (Man nutzt in diesem Ansatz nicht aus, dass die Sprachen in P sogar durch uniforme Schaltkreisfamilien berechnet werden können.)

Was sind natürliche Beweise?

Besitzen Funktionen $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$ Schaltkreise der Größe $O(n^c)$?

B_n bezeichnet die Menge der Booleschen Funktionen mit n Eingabebits.

(a) $\mathcal{C} = (C_n \mid n \in \mathbb{N})$ heißt **kombinatorische Eigenschaft**, falls $C_n \subseteq B_n$.

(b) $\mathcal{C} = (C_n \mid n \in \mathbb{N})$ ist **natürlich** (gegen $\text{SIZE}(n^c)$), falls

- (1) \mathcal{C} **konstruktiv** ist: Wenn $f \in B_n$ durch ihre Funktionstabelle spezifiziert ist, dann kann in polynomieller Zeit (d.h. in Zeit polynomiell in 2^n) entschieden werden, ob $f \in C_n$, d.h. ob f die Eigenschaft \mathcal{C} besitzt.
- (2) \mathcal{C} **hinreichend groß** ist: Es ist $|C_n| \geq 2^{-n} \cdot |B_n|$ für alle $n \in \mathbb{N}$.
- (3) \mathcal{C} **nützlich** gegen $\text{SIZE}(n^c)$ ist: Wenn $f = (f_n \mid n \in \mathbb{N})$ die Eigenschaft \mathcal{C} für unendlich viele Eingabelängen n hat, gehört f **nicht** zu $\text{SIZE}(n^c)$.

$SIZE_d(n^c)$ ist die Klasse aller Funktionen mit Schaltkreisen der Tiefe höchstens d und der Größe höchstens n^c .

Die Eigenschaft C_n treffe auf $f \in B_n$ genau dann zu, wenn die **Empfindlichkeit von f** mindestens $n/4$ ist. Es gelte $\mathcal{C} = (C_n \mid n \in \mathbb{N})$.

- 1 \mathcal{C} ist **konstruktiv**, denn die Empfindlichkeit ist in Zeit $2^{O(n)}$ berechenbar.
- 2 \mathcal{C} ist **hinreichend groß**, denn Funktionen haben hochwahrscheinlich eine Empfindlichkeit von mindestens $n/4$.
- 3 Wir wissen, dass \mathcal{C} **nützlich** gegen $SIZE_d(n^c)$ für jede Konstante c ist.

\mathcal{C} ist ein **natürlicher Beweis** gegen $SIZE_d(n^c)$, leider aber nicht gegen $SIZE(n^c)$.

Haben wir die richtigen Eigenschaften gefordert?

● Hinreichende Größe.

- ▶ Fast alle Funktionen in B_n sind Zufallssfunktionen und die Berechnung von Zufallsfunktionen ist schwierig.
- ▶ Eine kombinatorische Eigenschaft \mathcal{C} , die von einer kleinen Minderheit angenommen wird, deckt nur exotische Schwierigkeitseigenschaften auf?!
- ▶ Ausgeschlossen ist die Existenz einer solchen Eigenschaft aber nicht.

● Konstruktivität.

- ▶ Natürliche Beweise erfassen nicht-konstruktive Eigenschaften nicht, aber
- ▶ exponentielle Zeit $2^{O(n)}$, oder polynomielle Zeit in der Länge der Funktionstabelle erlaubt die Überprüfung vieler vernünftiger Eigenschaften?!
- ▶ Achtung: Die **Methode der größten monochromatischen Teilmatrix** aus der Kommunikationskomplexität ist wahrscheinlich nicht konstruktiv, die **Rang-Methode** aber sehr wohl.

One-Way Funktionen

werden von natürlichen Beweisen geknackt

One-Way Funktionen können effizient ausgewertet werden, aber das Invertieren ist teuer.

Eine Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ heißt eine **one-way Funktion** mit **Komplexität s** , wenn f durch einen effizienten Algorithmus berechenbar ist und wenn für jeden probabilistischen Algorithmus P mit Laufzeit s gilt

$$\text{prob}[P \text{ berechnet auf Eingabe } f(x) \text{ ein } z \text{ mit } f(z) = f(x) \mid \mathbf{x} \in \{0, 1\}^n] < \frac{1}{s(n)}.$$

Für jedes n wird die Gleichverteilung auf den Argumenten $x \in \{0, 1\}^n$ von f und auf den Münzwürfen von P zugrunde gelegt.

Gesucht sind one-way Funktionen mit Komplexität $n^{\omega(1)}$.

Potentielle One-Way Funktionen

- **Faktorisierung.**

- ▶ Die Eingabe besteht aus zwei Primzahlen p und q .
- ▶ Die Ausgabe ist das Produkt $N = p \cdot q$.
- ▶ Im Umkehrproblem ist also die Zahl N zu faktorisieren.

- **Das Problem des diskreten Logarithmus.**

- ▶ Die Eingabe ist eine Primzahl p , ein erzeugendes Element g modulo p und eine natürliche Zahl i .
- ▶ Die Ausgabe ist die Potenz $g^i \bmod p$.
- ▶ Im Umkehrproblem sind p , g und $g^i \bmod p$ gegeben. Der „Logarithmus“ i ist zu berechnen.

- **Das RSA-Problem**

- ▶ Die Eingabe besteht aus den Zahlen N , e und x (e und $\phi(N)$, die Anzahl der primen Restklassen modulo N , sind teilerfremd. N ist ein Produkt von zwei (nicht bekannten) Primzahlen.)
- ▶ Die Ausgabe ist $y \equiv x^e \pmod{N}$.
- ▶ Im Umkehrproblem ist x zu bestimmen, wobei y , N und e gegeben sind.

- **Das Problem der diskreten Quadratwurzelberechnung**

- ▶ Als Eingabe sind natürliche Zahlen m und $x < m$ gegeben.
- ▶ Die Ausgabe ist $(m, x^2 \bmod m)$.
- ▶ Das Umkehrproblem ist die Bestimmung der Wurzel x modulo m . (Eine effiziente Lösung gelingt mit probabilistischen Algorithmen, solange m eine Primzahl ist.)

Wie sicher sind die obigen Funktionen „vermutlich“?

- Selbst nach Jahrzehnten intensiver Forschung hat man keine ernstzunehmenden Attacken gegen eine der obigen Funktionen konstruieren können.
- Die Vermutung ist nicht abwegig, dass hier tatsächlich one-way Funktionen vorliegen und zwar mit

Komplexität 2^{n^ϵ}

für irgendein $\epsilon > 0$:

Stimmt die Vermutung werden uniforme Schaltkreise der Größe mindestens 2^{n^ϵ} für die Invertierung benötigt.

Von One-Way Funktionen zu Pseudo-Random Generatoren

Pseudo-Random Generatoren

Pseudo-Random Generatoren blähen einen Zufallsstring der Länge k , die „Saat“, zu einem „zufällig wirkenden“ String der Länge $2k$ auf.

Die Funktion $\mathbf{G}_k : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$ sei vorgegeben, wobei $\mathbf{G}_k(\mathbf{x})$ für jedes \mathbf{x} effizient berechenbar ist. Dann ist \mathbf{G}_k ein Pseudo-Random Generator mit Komplexität \mathbf{s}_k ,

wenn kein Schaltkreis S_k der Größe höchstens \mathbf{s}_k den Generator \mathbf{G}_k von einer Zufallsquelle unterscheiden kann,

wenn also stets

$$| \text{pr}[\mathbf{S}_k(\mathbf{G}_k(\mathbf{x})) = \mathbf{1}] - \text{pr}[\mathbf{S}_k(\mathbf{y}) = \mathbf{1}] | < \frac{1}{\mathbf{s}_k}$$

gilt. ($\mathbf{x} \in \{0, 1\}^k$ und $\mathbf{y} \in \{0, 1\}^{2k}$ werden jeweils gemäß der Gleichverteilung gewählt.)

Gesucht sind Pseudo-Random Generatoren mit Komplexität $n^{\omega(1)}$.

Potentielle Pseudo-Random Generatoren

Die Existenz von one-way Funktionen bedingt die Existenz von Pseudo-Random Generatoren und umgekehrt.

(Ohne Beweis)

- Der **Blum-Micali Generator** berechnet für eine Saat s_0 , eine Primzahl p und eine erzeugende Restklasse g modulo p die Iteration

$$s_{i+1} = g^{s_i} \text{ mod } p.$$

Die Ausgabe des Generators ist die Bitfolge $b_i = \begin{cases} 1 & \text{wenn } s_i < p/2 \\ 0 & \text{sonst.} \end{cases}$

- Der **RSA Generator** berechnet für eine Saat s_0 die Folge

$$s_{i+1} = s_i^e \text{ mod } N.$$

Die Ausgabe des Generators für die Saat s_0 ist die Bitfolge $s_i \text{ mod } 2$.

- Der **Blum-Blum-Shub Generator** berechnet für eine Saat s_0 die Folge

$$s_{i+1} = s_i^2 \text{ mod } N,$$

wobei $N = p \cdot q$ mit Primzahlen $p \equiv q \equiv 3 \pmod{4}$ gelte. Die Ausgabe des Generators ist die Bitfolge $s_i \text{ mod } 2$.

Von Pseudo-Random Generatoren zu Pseudo-Zufallsfunktionen

Pseudo-Zufallsfunktionen

Eine Familie $\mathcal{F}_k = (f_t \mid t \in \{0, 1\}^k)$ mit Funktionen $f_t : \{0, 1\}^k \rightarrow \{0, 1\}$ heißt eine Familie von Pseudo-Zufallsfunktionen mit **Komplexität s** \Leftrightarrow

- (a) \mathcal{F}_k **ist effizient auswertbar**: Es gibt einen Algorithmus, der in Zeit polynomiell in k für Eingabe $t, x \in \{0, 1\}^k$ den Wert $f_t(x)$ bestimmt.
- (b) \mathcal{F}_k **ist nicht effizient von Zufallsfunktionen unterscheidbar**:
Für jeden Algorithmus A mit Laufzeit s gilt

$$| \text{pr}[\mathbf{A}(f_t) = \mathbf{1}] - \text{pr}[\mathbf{A}(f) = \mathbf{1}] | < \frac{1}{s}.$$

$t \in \{0, 1\}^k$ und $f \in B_k$ werden gemäß der Gleichverteilung gewählt.

A darf während seiner Berechnung Funktionswerte für bis zu s Argumente anfordern.

Konstruktion von Pseudo-Zufallsfunktionen

$G_k : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$ sei ein Pseudo-Random Generator mit $G_k = (F_0, F_1)$ für Funktionen $F_0, F_1 : \{0, 1\}^k \rightarrow \{0, 1\}^k$.

- Für jedes binäre Wort $\mathbf{y} = \mathbf{y}_1 \cdots \mathbf{y}_m \in \{0, 1\}^*$ definieren wir die Funktion $F_{\mathbf{y}} : \{0, 1\}^k \rightarrow \{0, 1\}^k$ durch

$$F_{\mathbf{y}} = F_{\mathbf{y}_m} \circ \cdots \circ F_{\mathbf{y}_1}.$$

- Die Funktionen $H_x : \{0, 1\}^m \rightarrow \{0, 1\}$ mit

$$H_x(\mathbf{y}) = \text{das erste Bit von } F_{\mathbf{y}}(\mathbf{x})$$

sind die (von $G_k : \{0, 1\}^k \rightarrow \{0, 1\}^{2k}$ produzierten) Pseudo-Zufallsfunktionen.

- ▶ Wähle $m = k$.

Für jede one-way Funktion mit Komplexität 2^{m^ϵ} besitzt die zugehörige Familie der Pseudo-Zufallsfunktionen die Komplexität 2^{m^δ} für ein $\delta > 0$. (Ohne Beweis)

Natürliche Beweise knacken Pseudo-Zufallsfunktionen

$\varepsilon > 0$ sei beliebig

Wenn es one-way Funktionen mit Komplexität 2^{n^ε} gibt, dann gibt es für hinreichend großes c

keine natürlichen Beweise gegen $\text{SIZE}(n^c)$.

- Sei $\mathcal{C} = (C_m \mid m \in \mathbb{N})$ ein **natürlicher Beweis** und die Funktionen $f_t : \{0, 1\}^m \rightarrow \{0, 1\}$ seien **Pseudo-Zufallsfunktionen**.
- Die Funktionen f_t sind **effizient auswertbar** und \mathcal{C} ist ein natürlicher Beweis gegen $\text{SIZE}(n^c)$ für hinreichend großes $c \Rightarrow \mathbf{C}_m(f_t) = \mathbf{0}$.
- Andererseits wird die Eigenschaft C_m **von vielen Funktionen erfüllt**, denn es ist $|\mathbf{C}_m| \geq \frac{|B_m|}{2^{O(m)}}$.
 - ▶ Für Zufallsfunktionen $f \in B_m$ und zufällige Wahlen von $t \in \{0, 1\}^m$ folgt

$$|\text{pr}[\mathbf{C}_m(f) = \mathbf{1}] - \text{pr}[\mathbf{C}_m(f_t) = \mathbf{1}]| = \text{pr}[\mathbf{C}_m(f) = \mathbf{1}] \geq \frac{1}{2^{O(m)}}.$$

- Ja, aber!
 - + C_m unterscheidet Pseudo-Zufallsfunktionen von vielen Zufallsfunktionen,
 - aber C_m benötigt die Auswertungszeit $2^{O(m)}$, ist als statistischer Test viel zu langsam!
- Und wenn wir stattdessen mit dem statistischen Test

$$C'_m(g) = C_n(g(*0^{m-n}))$$

arbeiten?

- ▶ C'_m ist schnell –Laufzeit $2^{O(n)}$ ist ausreichend–,
- ▶ Es ist egal, ob eine Zufallsfunktion $f \in B_n$
 - ★ mit der Gleichverteilung aus B_n oder
 - ★ oder mit der Gleichverteilung aus B_m gezogen wird, solange nachfolgend die letzten $m - n$ Bits eines Arguments auf Null gesetzt werden.
- ▶ $f_t(*0^{m-n})$ ist nur in Zeit $\text{poly}(m)$ auswertbar: Nur mit $n = m^\mu$ –für irgendeine Konstante $\mu > 0$ – qualifiziert sich $f_t(*0^{m-n})$ als (mögliche) Pseudo-Zufallsfunktion in B_n .

- $|\text{pr}[\mathbf{C}_m(\mathbf{f}) = \mathbf{1}] - \text{pr}[\mathbf{C}_m(\mathbf{f}_t) = \mathbf{1}]| \geq \frac{1}{2^{O(m)}}.$
- \mathcal{C} ist **konstruktiv** \Rightarrow
 der „statistische Test“ C_m ist in Zeit $2^{O(m)}$ implementierbar.

Setze $n = m^{\delta/2}$.

- Um \mathbf{f} von \mathbf{f}_t zu unterscheiden, übergeben wir dem natürlichen Beweis die Funktionstabellen von $\mathbf{f}_t(*0^{m-n})$ und $\mathbf{f}(*0^{m-n})$.
- Die Funktionen in B_n und die Funktionen $\mathbf{f}(*0^{m-n})$ besitzen die gleiche Verteilung!
- Der natürliche Beweis trennt (mit der gleichen Argumentation wie oben):

$$|\text{pr}[C_n(f(*0^{m-n})) = 1] - \text{pr}[C_n(f_t(*0^{m-n})) = 1]| =$$

$$\text{pr}[C_n(f(*0^{m-n})) = 1] \geq \frac{1}{2^{O(n)}}.$$

Es ist $C'_m(g) = C_n(g(*0^{m-n}))$.

- Der statistische Test C'_m ist in Zeit höchstens $2^{O(n)} = 2^{O(m^{\delta/2})}$ implementierbar.
- Er erreicht die Trennung

$$|\text{pr}[C'_m(f) = \mathbf{1}] - \text{pr}[C'_m(f_t) = \mathbf{1}]| = \text{pr}[C'_m(f) = \mathbf{1}] \geq \frac{1}{2^{O(m^{\delta/2})}}.$$

- Nach Annahme besitzen die One-way Funktionen die Komplexität 2^{n^ϵ} und
- „ihre“ Pseudo-Zufallsfunktionen haben die Komplexität 2^{m^δ} .

Stimmt nicht, denn wir haben die Pseudo-Zufallsfunktionen von Zufallsfunktionen in Zeit $O(2^{m^{\delta/2}})$ unterschieden!