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 $\mathbf{A} \subseteq \Sigma^*$ 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 $\mathbf{K}^{\mathbf{A}}$ entsprechend zu definieren, wobei jetzt Fragen an das Orakel *A* zugelassen sind.

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

P^A und NP^A

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

(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)
- $NP^A \subseteq \text{PSPACE}$: ✓.
- Die Beziehung $\text{PSPACE} \subseteq P^A$ folgt aus der PSPACE-Vollständigkeit von A:
  - Für jede Sprache $L \in \text{PSPACE}$ gibt es eine effiziente deterministische TM M mit

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

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

Wir konstruieren ein Orakel $\mathbf{A}$, so dass die Sprache

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

zu $\mathrm{NP}^{\mathbf{A}}$, nicht aber zu $\mathrm{P}^{\mathbf{A}}$ gehört.

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

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

- Simuliere M_k auf der Eingabe 1^{n_k}. (Es gelte $n_k > \max\{|w_1|, \ldots |w_m|\}$ und $2^{n_k} > {n_k}^k$.)
  - Wenn M_k eine Anfrage y aus $\{w_1, \ldots, w_m\}$ 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.

Konsequenzen

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

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 $\mathbf{H} : \mathbb{N} \to \mathbb{N}$ muss noch konstruiert werden.

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

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

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?

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

- Wenn $\mathrm{SAT}_H \in \mathrm{P}$, dann gibt es eine TM $\mathbf{M}^*$, die SAT_H berechnet.
- Wir machen uns auf die Suche nach $\mathbf{M}^*$.
  - $\mathbf{H(n)}$ muss effizient berechenbar sein: Für die Definition von $H(n)$ betrachten wir nur **„kurze“ Eingaben** $\mathbf{x} = \phi\,\mathbf{0}\,\mathbf{1}^{\mathbf{m}^{\mathbf{H(m)}}}$ mit $|\phi| = \mathbf{m}$ und $|x| \leqslant \log_2 n$.
  - Setze $\mathbf{H(n)} = \langle \mathbf{M} \rangle$ ($\langle M \rangle$ ist die „Gödelnummer“ von M) genau dann, wenn
    - $\langle \mathbf{M} \rangle$ minimal mit $\langle \mathbf{M} \rangle < \log_2 \log_2 \mathbf{n}$ ist und
    - $\mathbf{M}$ auf allen kurzen Eingaben korrekt in höchstens $\langle \mathbf{M} \rangle n^{\langle \mathbf{M} \rangle}$ Schritten rechnet.

    Gibt es eine solche Maschine $\mathbf{M}$ nicht, setze $\mathbf{H(n)} = \log_2 \log_2 \mathbf{n}$.
    - 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 $\mathrm{SAT}_H \in \mathrm{P}$, 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 $\mathrm{P} \neq \mathrm{NP}$ an!

SAT_H liegt nicht in P

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

- $\mathrm{SAT}_H \in P \Leftrightarrow \mathbf{H(n)} = \mathbf{O(1)}$.
  - $\Rightarrow$ Wenn die TM **M** die Sprache SAT_H akzeptiert, dann ist $H(n) \leqslant \langle M \rangle$ für genügend große n.
  - $\Leftarrow$ Wenn $\mathbf{H(n)} = \mathbf{O(1)}$, 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 $\mathrm{SAT}_H \notin$ P, dann ist $\mathbf{H(n)} = \omega\mathbf{(1)}$.
  - 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 $\mathbf{H(n)} = \mathbf{O(1)}$.
  - 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

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

Angenommen, $\mathrm{SAT}_{\mathbf{H}}$ 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

$$\psi \in \mathrm{SAT} \;\Leftrightarrow\; T(\psi) \in \mathrm{SAT}_H.$$

- SAT_H liegt nicht in P und $H(n) = \omega(1)$ folgt.
- Wenn $T(\psi) = \phi\,0\,1^{H(|\phi|)}$, 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

Nicht-uniforme Berechnungsmodelle

Ist die $\mathrm{P} \stackrel{?}{=} \mathrm{NP}$ 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 $\mathrm{P}_{/\mathrm{poly}}$ aller Sprachen mit Schaltkreisen polynomieller Größe.
- Es genügt somit der Nachweis, dass (irgend)eine NP-vollständige Sprache nicht zu $\mathrm{P}_{/\mathrm{poly}}$ 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?

Natürliche Beweise

Besitzen Funktionen $f_n : \{0,1\}^n \to \{0,1\}$ 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 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 $|C_n| \geqslant 2^{-n} \cdot |B_n|$ für alle $n \in \mathbb{N}$.

(3) $\mathcal{C}$ **nützlich** gegen 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 SIZE(n^c).

Natürliche Beweise und AC^0

$\mathrm{SIZE}_{\mathbf{d}}(\mathbf{n}^{\mathbf{c}})$ ist die Klasse aller Funktionen mit Schaltkreisen der Tiefe höchstens d und der Größe höchstens n^c.

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

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 $\mathrm{SIZE}_{\mathbf{d}}(\mathbf{n}^{\mathbf{c}})$ für jede Konstante c ist.

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

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 $f : \{0,1\}^* \to \{0,1\}^*$ 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

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

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 mod p.
  - Im Umkehrproblem sind p, g und g^i mod 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 $y \equiv x^e \mod N$.
  - 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 $(m, x^2 \mod m)$.
  - 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 $\varepsilon > 0$:

  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 $\mathbf{G_k} : \{\mathbf{0},\mathbf{1}\}^{\mathbf{k}} \to \{\mathbf{0},\mathbf{1}\}^{\mathbf{2k}}$ sei vorgegeben, wobei $\mathbf{G_k(x)}$ für jedes 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

$$| \operatorname{pr}[\mathbf{S_k}(\mathbf{G_k}(\mathbf{x})) = \mathbf{1}] - \operatorname{pr}[\mathbf{S_k}(\mathbf{y}) = \mathbf{1}] | < \frac{\mathbf{1}}{\mathbf{s_k}}$$

gilt. ($x \in \{0,1\}^k$ und $y \in \{0,1\}^{2k}$ 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

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

  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

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

  Die Ausgabe des Generators für die Saat s_0 ist die Bitfolge $s_i \bmod 2$.
- Der **Blum-Blum-Shub Generator** berechnet für eine Saat s_0 die Folge

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

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

Von Pseudo-Random Generatoren zu Pseudo-Zufallsfunktionen

Pseudo-Zufallsfunktionen

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

(a) $\mathcal{F}_{\mathbf{k}}$ **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

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

$t \in \{0,1\}^k$ 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

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

- Für jedes binäre Wort $\mathbf{y = y_1 \cdots y_m \in \{0,1\}^*}$ definieren wir die Funktion $F_y : \{0,1\}^k \to \{0,1\}^k$ durch

$$\mathbf{F_y = F_{y_m} \circ \cdots \circ F_{y_1}}.$$

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

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

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

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

Natürliche Beweise knacken Pseudo-Zufallsfunktionen

Natürliche Beweise und One-way Funktionen (1/4)

$\varepsilon > 0$ sei beliebig

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

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

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

$$|\ \mathrm{pr}[\mathbf{C_m(f) = 1}] - \mathrm{pr}[\mathbf{C_m(f_t) = 1}]\ | = \mathrm{pr}[\mathbf{C_m(f) = 1}] \geqslant \frac{\mathbf{1}}{\mathbf{2^{O(m)}}}.$$

Natürliche Beweise und One-way Funktionen (2/4)

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

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

arbeiten?

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

Natürliche Beweise und One-way Funktionen (3/4)

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

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

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

$$\begin{aligned} | \operatorname{pr}[C_n(f(*0^{m-n})) = 1] - \operatorname{pr}[C_n(f_t(*0^{m-n})) = 1] | &= \\ \operatorname{pr}[C_n(f(*0^{m-n})) = 1] &\geqslant \frac{1}{2^{O(n)}}. \end{aligned}$$

Natürliche Beweise und One-way Funktionen (4/4)

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

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

$$\mid \mathrm{pr}[\mathbf{C'_m(f) = 1}] - \mathrm{pr}[\mathbf{C'_m(f_t) = 1}], \mid = \mathrm{pr}[\mathbf{C'_m(f) = 1}] \geqslant \frac{\mathbf{1}}{\mathbf{2^{O(m^{\delta/2})}}}.$$

- 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!