Ausgewählte Kapitel der Komplexitätstheorie

Rupert Hölzl
Universität der Bundeswehr München

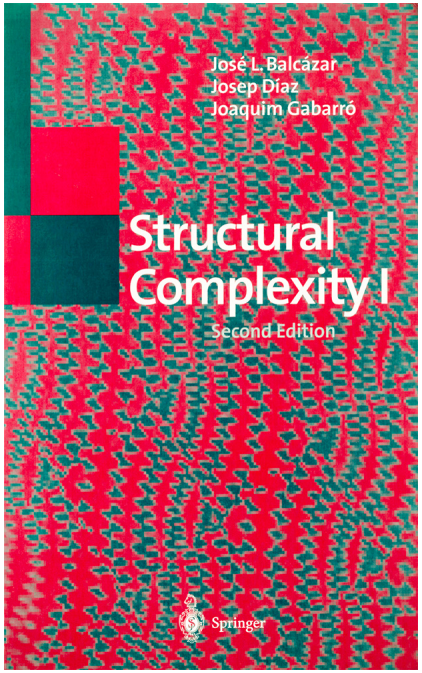
Version vom 9. Januar 2024

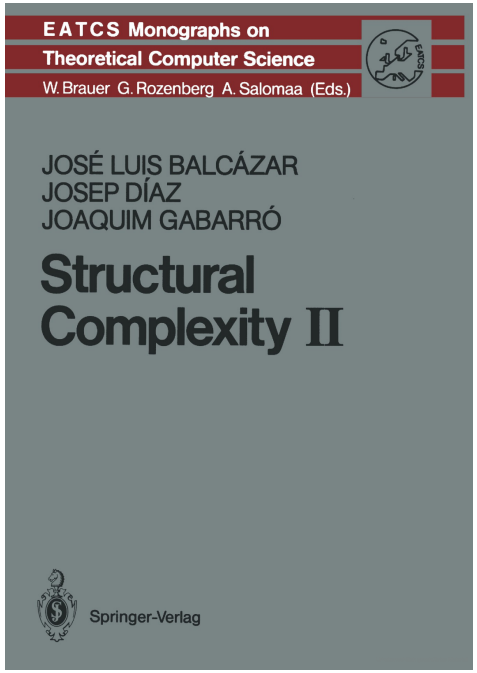
1 Vorbemerkungen

Themenübersicht

1. $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ relativ zu Orakeln und 2SAT ist in $\mathbf{P}$
   "Sanity checks" für angebliche Anworten auf die Frage $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$.
2. Die Polynomialzeithierarchie
   Eine aufsteigende Komplexitätshierarchie, die vielleicht kollabiert.
3. Kolmogorovkomplexität
   Bestmögliche Kompression, Tetris, und beinahe berechenbare Mengen
4. Interaktive Beweise
   Merlin überzeugt Artus von der Wahrheit einer Behauptung.
5. Das PCP-Theorem
   Eine wichtige Verschärfung des Satzes von Cook-Levin.
6. Nichtuniforme Komplexität
   Schaltkreise statt größenunabhängiger Algorithmen.

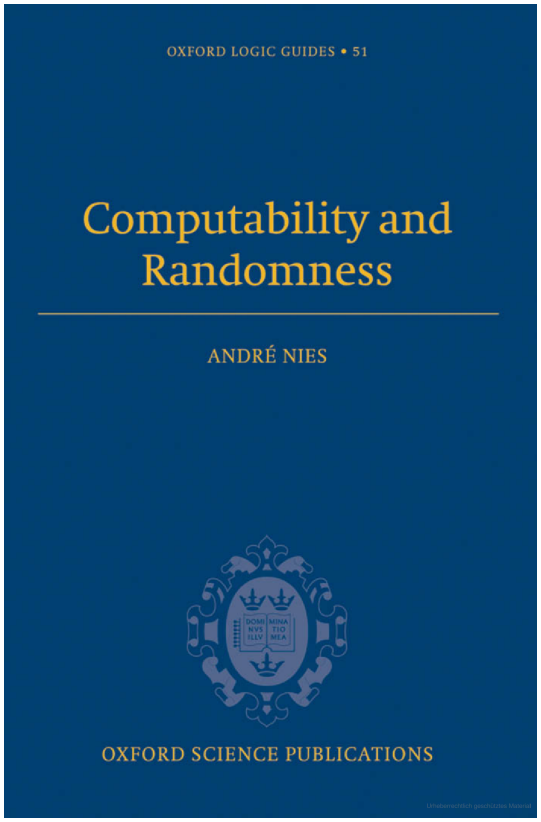
Literatur





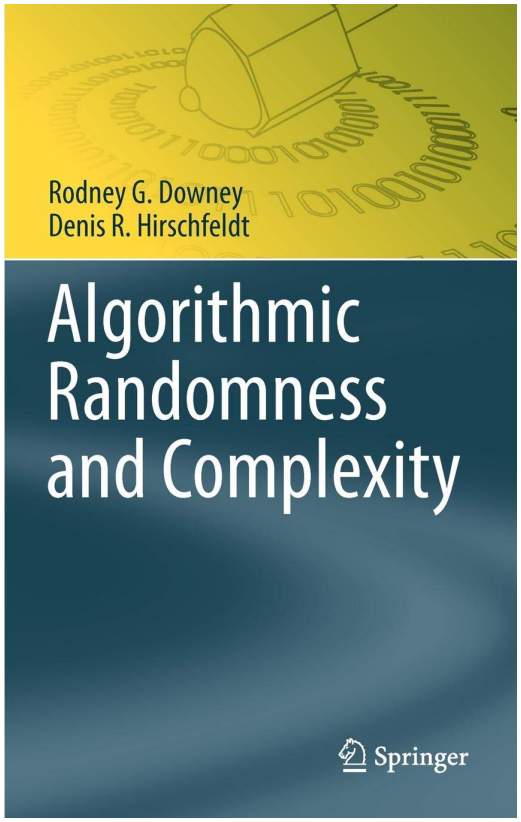
1. Sehr verständlich, aber nicht mehr zu allen Themen aktuell.
2. **Themen:** Relativierung (II:7.1–7.2), Polynomialzeit-hierarchie (I:8.1–8.4), Interaktive Beweise (II:11.1–11.5), Schaltkreise (I:5.1–5.5)
3. **Bibliothek:** *opac.unibw.de/search?bvnr=BV007955465*
   opac.unibw.de/search?bvnr=BV007955466

Literatur



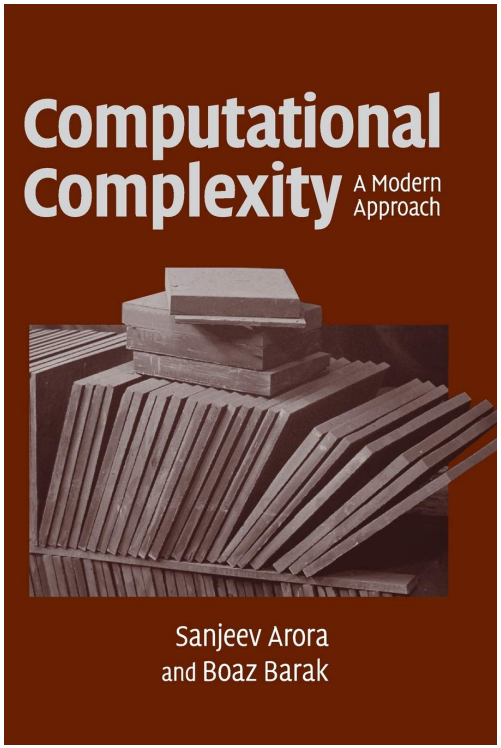
1. Recht elegant; manchmal leidet darunter die Verständlichkeit.
2. **Themen:** Satz von Kraft & Chaitin (Theorem 2.2.17)
3. **Bibliothek:** *opac.unibw.de/search?bvnr=BV046259824*

Literatur



1. Das Standardwerk über Algorithmische Zufälligkeit.
2. **Themen:** Konstruktion K-trivialer Mengen (11.1.1)
3. **Bibliothek:** *opac.unibw.de/search?bvnr=BV046259862*

Literatur



1. Moderne Themenauswahl; manche Kapitel sind voller Fehler.
2. **Themen:** Interaktive Beweise (8.1–8.3), PCP-Theorem (11.1–11.4), Schaltkreise (6.1–6.3), Satz von Karp & Lipton (6.4)
3. **Bibliothek:** *opac.unibw.de/search?bvnr=BV036054102*

2 Turing-Reduktion

Orakel-Turingmaschinen

1. **Definition.** Eine Orakel-Turingmaschine M ist eine normale Turingmaschine, die zusätzlich
   - ein besonderes *Orakelband* mit Vollzugriff,
   - einen besonderen *Fragezustand*, und
   - zwei besondere Antwortzustände „ja“ und „nein“ hat.
2. Sei eine Menge $A \subseteq \{0,1\}^{<\mathbb{N}}$ gegeben. M darf ein Wort auf das Orakelband schreiben.
3. Falls M irgendwann in den Fragezustand wechselt, und x das Wort ist, was in diesem Moment auf dem Orakelband steht, so wechselt M im nächsten Schritt in den Antwortzustand „ja“, falls $x \in A$, sonst in den Antwortzustand „nein“.
4. Danach geht die Rechnung normal weiter.
5. **Definition.** Wir sagen, dass *M die Sprache A als Orakel benutzt*.

Turingreduktion

1. **Definition.** Wir schreiben $L(M,A)$ für die Sprache, die von der Orakelturingmaschine M akzeptiert wird, wenn M A benutzt.
2. **Beachte:** Dies impliziert, dass für das gegebene A und *jedes* $x \in \{0,1\}^{<\mathbb{N}}$ die Rechnung von M mit Eingabe x unter Benutzung von Orakel A terminiert und ein Ergebnis $M^A(x)$ ausgibt.
3. **Testfrage.** Ist $L(M,A)$ für alle M und A definiert?
4. **Testfrage.** Falls $L(M,A)$ für ein festes M und A definiert ist, ist $L(M,A')$ dann auch für alle anderen A' definiert?
5. **Definition.** Sei $L \subseteq \{0,1\}^{<\mathbb{N}}$ und existiere M mit $L = L(M,A)$. Wir sagen, L ist *auf A Turing-reduzierbar* und schreiben $L \leq_{\mathrm{T}} A$.

Nichtdeterministische Turingreduktion

1. Man kann auch nicht-deterministische Orakel-Turing-maschinen N definieren. Das Orakelband und die speziellen Zustände funktionieren genau wie auf der vorherigen Folie.
2. Die Berechnungen sind nun nicht-deterministisch.
3. Das Akzeptanzkriterium für ein Wort x ist das übliche Kriterium für nicht-deterministische Turingmaschinen, nämlich die Existenz mindestens eines akzeptierenden Rechenpfades von N bei Eingabe x.
4. Damit ist auch klar, wie $L(N,A)$ definiert ist.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?
2. **Eine Turing-Reduktion muss nicht positiv sein.**

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

   Bei der many-one-Reduktion wird für x immer genau ein Wort $f(x)$ abgefragt.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

   Bei der many-one-Reduktion wird für x immer genau ein Wort $f(x)$ abgefragt.

   Hier könnte zum Beispiel $x \in L$ sein genau dann, wenn $y_1 \in A$ *und* $y_2 \in A$.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

   Bei der many-one-Reduktion wird für x immer genau ein Wort $f(x)$ abgefragt.

   Hier könnte zum Beispiel $x \in L$ sein genau dann, wenn $y_1 \in A$ *und* $y_2 \in A$.

4. **Die Reduktion kann *adaptiv* sein.**

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

   Bei der many-one-Reduktion wird für x immer genau ein Wort $f(x)$ abgefragt.

   Hier könnte zum Beispiel $x \in L$ sein genau dann, wenn $y_1 \in A$ *und* $y_2 \in A$.

4. **Die Reduktion kann *adaptiv* sein.**

   Welche Frage als zweites gestellt wird, kann von der Antwort auf die erste Frage abhängen.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Testfrage.** Sie kennen die many-one-Reduktion aus der Vorlesung „Berechenbarkeitstheorie“. Was ist der Unterschied?

2. **Eine Turing-Reduktion muss nicht positiv sein.**

   Bei der many-one-Reduktion gilt $x \in L$ genau dann, wenn $f(x) \in A$.

   Hier dagegen könnte zum Beispiel $x \in L$ sein genau dann, wenn $y \notin A$ ist.

   Da die „Reduktionsfunktion“ eine Turingmaschine ist, kann mit der Antwort des Orakels alles mögliche angestellt werden, z.B. eine Negierung.

3. **Die Antwort kann von *mehreren* Fragen abhängen.**

   Bei der many-one-Reduktion wird für x immer genau ein Wort $f(x)$ abgefragt.

   Hier könnte zum Beispiel $x \in L$ sein genau dann, wenn $y_1 \in A$ *und* $y_2 \in A$.

4. **Die Reduktion kann *adaptiv* sein.**

   Welche Frage als zweites gestellt wird, kann von der Antwort auf die erste Frage abhängen. Sogar *ob überhaupt* noch eine weitere Frage gestellt werden soll, kann von den bisherigen Antworten abhängen.

Unterschiede zwischen Turing- und many-one-Reduktion

1. **Intuition.** Turingreduktion entspricht dem Nachschlagen von Informationen in einer Datenbank A innerhalb des Kontrollflusses eines Programms M zur Berechnung von L.
2. **Intuition.** Many-one-Reduktion entspricht eher dem Einbetten einer Datenbank L in einer anderen Datenbank A.
3. **Testfrage.** Bestehen Implikationen zwischen den Aussagen

$$„L \leq_{\mathrm{T}} A“ \quad \text{und} \quad „L \leq_{\mathrm{m}} A“?$$

Turingreduktion in Polynomialzeit

1. **Definition.** Sei $L = L(M, A)$ und existiere zusätzlich ein Polynom p, so dass $L(M, B)$ für alle $B \subseteq \{0, 1\}^{<\mathbb{N}}$ definiert ist und M für alle B bei Eingabe x nur $p(|x|)$ viele Rechenschritte lang arbeitet. Dann sagen wir dass L *in Polynomialzeit Turing-reduzierbar auf* A ist und schreiben $L \leq_{\mathrm{T}}^{\mathrm{p}} A$.
2. **Testfrage.** Ist es eine echte Einschränkung, zu fordern, dass sich M für *alle* Orakel B (statt nur für A) an die Zeitschranke p hält?

Relativierte Komplexitätsklassen

1. Wir schreiben $\mathbf{P}^A$ für

$$\{L : (\exists M \text{ deterministisch in Polynomialzeit})(L = L(M, A))\}$$

   und nennen dies die *Komplexitätsklasse* **P** *relativ zu* A.
2. Analog kann man $\mathbf{NP}^A$ definieren, also die *Komplexitätsklasse* **NP** *relativ zu* A, indem man nicht-deterministische Turingmaschinen N betrachtet.
3. **Testfrage.** Wie ist dann sinnvollerweise $\mathbf{coNP}^A$ definiert?

Kapitelzusammenfassung

1. Wir haben Orakel-Turingmaschinen definiert, und gesehen, wie ihr Verhalten von der als Orakel benutzen Menge abhängt.
2. Wir haben die Turing-Reduktion $\leq_{\mathrm{T}}$ definiert, und gesehen, dass dies eine Verallgemeinerung der bereits bekannten many-one-Reduktion $\leq_{\mathrm{m}}$ ist.
3. Ressourcenbeschränkung der Orakel-Turingmaschinen führt
   - zur Polynomialzeit-Turing-Reduktion $\leq_{\mathrm{T}}^{\mathrm{p}}$,
   - zur Polynomialzeit-many-one-Reduktion $\leq_{\mathrm{m}}^{\mathrm{p}}$,
   - zu relativierten Komplexitätsklassen wie $\mathbf{P}^A$, $\mathbf{NP}^A$, $\mathbf{coNP}^A$.

3

$\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ relativ zu Orakeln

Erinnerung: Das $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$-Problem

1. Das $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$-Problem kennen wir aus der Komplexitätstheorie.
2. **Intuition.** Ist das Finden von Lösungen bestimmter Probleme aufwändiger, als das reine Überprüfen einer angeblichen Lösung?
3. Dies ist die wichtigste offene Frage der Informatik.
4. Viele Beweisansätze für *beide* möglichen Lösungen wurden erfolglos ausprobiert. Ein naheliegender nächster Schritt ist, zu versuchen zu verstehen, woher die Schwierigkeit stammt und welche Beweisansätze überhaupt eine Erfolgsaussicht haben.
5. Eine der Erkenntnisse ist, dass Beweistechniken hoffnungslos sind, die genauso auch relativ zu Orakeln funktionieren würden.
6. Dies ist so, weil es, wie wir sehen werden, A und B gibt mit
$$\mathbf{P}^A = \mathbf{NP}^A \quad \text{und} \quad \mathbf{P}^B \neq \mathbf{NP}^B.$$

Ein Orakel, relativ zu dem $\mathbf{P} = \mathbf{NP}$ gilt

1. **Testfrage.** Sei $\mathscr{C} \in \{\mathbf{NP}, \mathbf{PSPACE}\}$. Bestehen Implikationen zwischen den Aussagen

   „A ist $\mathscr{C}$-schwer für $\leq_{\mathrm{T}}^{\mathrm{p}}$“ und „$A$ ist $\mathscr{C}$-schwer für $\leq_{\mathrm{m}}^{\mathrm{p}}$“?

2. **Testfrage.** Was ist eine sinnvolle Definition von $\mathbf{PSPACE}^A$?
3. **Testfrage.** Warum gilt für alle A, dass $\mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A$?

Ein Orakel, relativ zu dem $\mathbf{P} = \mathbf{NP}$ gilt

1. **Testfrage.** Sei $\mathscr{C} \in \{\mathbf{NP}, \mathbf{PSPACE}\}$. Bestehen Implikationen zwischen den Aussagen

   „A ist $\mathscr{C}$-schwer für $\leq_{\mathrm{T}}^{\mathrm{p}}$“ und „$A$ ist $\mathscr{C}$-schwer für $\leq_{\mathrm{m}}^{\mathrm{p}}$“?
2. **Testfrage.** Was ist eine sinnvolle Definition von $\mathbf{PSPACE}^A$?
3. **Testfrage.** Warum gilt für alle A, dass $\mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A$?
4. Wir erinnern an die für $\leq_{\mathrm{m}}^{\mathrm{p}}$ **PSPACE**-vollständige Menge QBF aus der Vorlesung „Komplexitätstheorie“.
5. **Testfrage.** Wie lässt sich $\mathbf{P}^{\mathrm{QBF}}$ einfacher ausdrücken?
6. **Testfrage.** Wie lässt sich $\mathbf{PSPACE}^{\mathrm{QBF}}$ einfacher ausdrücken?

Ein Orakel, relativ zu dem $\mathbf{P} = \mathbf{NP}$ gilt

1. **Testfrage.** Sei $\mathscr{C} \in \{\mathbf{NP}, \mathbf{PSPACE}\}$. Bestehen Implikationen zwischen den Aussagen

   „A ist $\mathscr{C}$-schwer für $\leq_{\mathrm{T}}^{\mathrm{p}}$“ und „$A$ ist $\mathscr{C}$-schwer für $\leq_{\mathrm{m}}^{\mathrm{p}}$“?
2. **Testfrage.** Was ist eine sinnvolle Definition von $\mathbf{PSPACE}^A$?
3. **Testfrage.** Warum gilt für alle A, dass $\mathbf{P}^A \subseteq \mathbf{NP}^A \subseteq \mathbf{PSPACE}^A$?
4. Wir erinnern an die für $\leq_{\mathrm{m}}^{\mathrm{p}}$ **PSPACE**-vollständige Menge QBF aus der Vorlesung „Komplexitätstheorie“.
5. **Testfrage.** Wie lässt sich $\mathbf{P}^{\mathrm{QBF}}$ einfacher ausdrücken?
6. **Testfrage.** Wie lässt sich $\mathbf{PSPACE}^{\mathrm{QBF}}$ einfacher ausdrücken?
7. **Testfrage.** Man sieht nun leicht $\mathbf{P}^{\mathrm{QBF}} = \mathbf{NP}^{\mathrm{QBF}}$. Wie?

Ein Orakel B, relativ zu dem $\mathbf{P} \neq \mathbf{NP}$ gilt

1. **Satz.** Es gibt ein B mit $\mathbf{P}^B \neq \mathbf{NP}^B$.
2. Das gesucht Orakel B werden wir von Hand bauen.
3. Dann werden wir eine Hilfsmenge H definieren via
$$H = \{0^n : (\exists x \in B)(|x| = n)\}$$
und zeigen dass $H \in \mathbf{NP}^B \setminus \mathbf{P}^B$, also $\mathbf{NP}^B \not\subseteq \mathbf{P}^B$.
4. **Testfrage.** Warum ist sowieso klar, dass $H \in \mathbf{NP}^B$?
5. **Unsere Aufgabe:** B so bauen, dass $H \notin \mathbf{P}^B$.

Ein Orakel B, relativ zu dem $\mathbf{P} \neq \mathbf{NP}$ gilt

1. **Erinnerung Aufgabe:** B so bauen, dass $H \notin \mathbf{P}^B$.
2. **Ansatz:** Konstruiere B so, dass es keine deterministische Polynomialzeit-Turingmaschine M gibt, die $H \leq_{\mathrm{T}}^{\mathrm{p}} B$ bezeugt.
3. Dies erreichen wir, indem wir sicherstellen, dass es für jedes solche M ein Wort 0^ℓ gibt, wo M versagt, also nicht H berechnet.
4. Genauer gesagt:
   - Wenn $0^\ell \in L(M, B)$, so darf kein x der Länge ℓ in B sein.
   - Wenn $0^\ell \notin L(M, B)$, so *muss* ein Wort der Länge ℓ in B sein.

   Dann ist klar, dass M nicht aus B das dazugehörige H erzeugt.
5. Schaffen wir das für alle in Frage kommenden M, sind wir fertig.
6. Diese Beweistechnik nennt man **„Diagonalisierung“**.

   (Der Name stammt vom klassischen Beweis der Überabzählbarkeit der reellen Zahlen.)

Ein Orakel B, relativ zu dem $\mathbf{P} \neq \mathbf{NP}$ gilt

1. **Erinnerung Ziel:**
   - Wenn $0^\ell \in L(M,B)$, so darf kein x der Länge ℓ in B sein.
   - Wenn $0^\ell \notin L(M,B)$, so *muss* ein Wort der Länge ℓ in B sein.
2. Die Idee ist, dass M als Polynomialzeitmaschine nur polynomiell viele Orakelfragen stellen kann.
3. Es gibt aber *exponentiell* viele Wörter der Länge ℓ.
4. Es gibt also viele Wörter der Länge ℓ, von denen M nie wissen kann, ob sie in B sind, oder nicht.
5. Diese Wörter x können wir also in B hineinstecken oder nicht, je nachdem, was hilft, das Ziel zu erreichen.
6. **Aber:** Wir müssen die Adaptivität der Turing-Reduktion im Hinterkopf behalten. Wenn wir B ändern, so könnte dies beeinflussen, welche Wörter M überhaupt abfragt, und vielleicht sieht M dann plötzlich unser obiges x doch.

Die Konstruktion von B

1. Seien $P_1, P_2, \ldots$ alle deterministischen Polyzeit-Turingmaschinen.
2. Für jedes P_i sei p_i eine nichtfallende, polynomielle Zeitschranke.
3. Ohne Einschränkung sei $p_i(n) < p_{i+1}(n)$ für alle i und n.
4. **Testfrage.** Warum dürfen wir diese Annahmen machen?

Die Konstruktion von B

1. **Konstruktion:** *(Komplexität egal!)*

$B(0) \leftarrow \emptyset;$
$k(0) \leftarrow 0;$
Für $n = 1, 2, 3, \ldots$ **tu**
  $k(n) \leftarrow \min\{i > k(n-1) \colon\ 2^i > p_n(i) \wedge i > p_{n-1}(k(n-1))\};$
  Wenn $0^{k(n)} \in L(P_n, B(n-1))$ **dann**
    $B(n) \leftarrow B(n-1);$
  sonst
    $w(n) \leftarrow \min_{\text{lex}} \left\{ w \colon \begin{array}{l} |w| = k(n) \text{ und } P_n \text{ fragt bei Eingabe } 0^{k(n)} \\ \text{das Orakel } B(n-1) \text{ nicht nach } w. \end{array} \right\};$
    $B(n) \leftarrow B(n-1) \cup \{w(n)\};$
  Ende
Ende
$B \leftarrow \lim_n B(n);$

Die Konstruktion von B

1. **Konstruktion:** *(Komplexität egal!)*

$B(0) \leftarrow \emptyset;$
$k(0) \leftarrow 0;$
Für $n = 1, 2, 3, \ldots$ **tu**
  $k(n) \leftarrow \min\{i > k(n-1) \colon\ 2^i > p_n(i) \wedge i > p_{n-1}(k(n-1))\};$
  Wenn $0^{k(n)} \in L(P_n, B(n-1))$ **dann**
    $B(n) \leftarrow B(n-1);$
  sonst
    $w(n) \leftarrow \min_{\text{lex}} \left\{ w \colon \begin{array}{l} |w| = k(n) \text{ und } P_n \text{ fragt bei Eingabe } 0^{k(n)} \\ \text{das Orakel } B(n-1) \text{ nicht nach } w. \end{array} \right\};$
    $B(n) \leftarrow B(n-1) \cup \{w(n)\};$
  Ende
Ende
$B \leftarrow \lim_n B(n);$

2. **Testfrage.** Haben Sie eine Ahnung, wozu k dienen könnte?

Verifikation

1. $k(n)$ existiert immer, denn $i \mapsto 2^i$ wächst schneller als p_n.
2. Bei Eingabe $0^{k(n)}$ kann P_n höchstens $p_n(k(n))$ viele Fragen an das Orakel stellen. Es gibt aber $2^{k(n)} > p_n(k(n))$ viele Kandidaten für $w(n)$. Also existiert $w(n)$ immer, wenn wir es brauchen.
3. **Behauptung.** Wenn man P_n mit Eingabe $0^{k(n)}$ ausführt, ist es egal ob das Orakel $B(n-1)$ oder B ist.

Verifikation

1. $k(n)$ existiert immer, denn $i \mapsto 2^i$ wächst schneller als p_n.
2. Bei Eingabe $0^{k(n)}$ kann P_n höchstens $p_n(k(n))$ viele Fragen an das Orakel stellen. Es gibt aber $2^{k(n)} > p_n(k(n))$ viele Kandidaten für $w(n)$. Also existiert $w(n)$ immer, wenn wir es brauchen.
3. **Behauptung.** Wenn man P_n mit Eingabe $0^{k(n)}$ ausführt, ist es egal ob das Orakel $B(n-1)$ oder B ist.
   - Falls das Wort $w(n)$ zu B hinzugefügt wurde, ändert dies nichts, da es von P_n nicht abgefragt wird.
   - Später hinzugefügte Wörter $w(j)$ für $j > n$ können auch keinen Unterschied machen, da für alle diese Wörter $|w(j)| > p_n(k(n))$ gilt, und P_n deshalb keines dieser Wörter abfragen kann.
   - **Testfrage.** Warum sind auch $w(j)$ mit $j < n$ kein Problem? □

Verifikation

1. **Behauptung.** $H \notin \mathbf{P}^B$.
   - Wähle n beliebig.
   - $0^{k(n)} \in H$ genau dann, falls $w(n)$ zu B hinzugefügt wurde.
   - Das wurde getan genau dann, wenn $0^{k(n)} \notin L(P_n, B(n-1))$.
   - Wegen der letzten Behauptung gilt das genau dann, wenn
   $$0^{k(n)} \notin L(P_n, B).$$
   - Also gilt $0^{k(n)} \in H$ genau dann, wenn $0^{k(n)} \notin L(P_n, B)$.
   - Somit $H \neq L(P_n, B)$ und, weil n beliebig war, $H \notin \mathbf{P}^B$. □
2. Wegen $H \in \mathbf{NP}^B$ folgt $\mathbf{P}^B \neq \mathbf{NP}^B$. □

Zusammenfassung zur Diagonalisierung

1. Wir wollten eine Menge B bauen, so dass jede mögliche Reduktion von H auf B gestört wird.
2. In diesem Fall war das einfach, weil eine Polynomialzeit-reduktion nur wenige Fragen an ein Orakel stellen kann.
3. Sobald eine Reduktion erfolgreich gestört wurde, mussten wir sicherstellen, dass diese Eigenschaft dauerhaft erhalten bleibt. Man sagt, die *bestehenden Berechnungen wurden geschützt*.
4. In diesem Fall war das einfach, weil nichts dagegen sprach, beliebig viel Platz zwischen den einzelnen Störstellen zu lassen. Der notwendige Abstand wurde durch die Funktion k bestimmt.
5. Es gibt auch wesentlich komplexere Diagonalargumente, sowohl in der Komplexitäts- als auch in der Berechenbarkeitstheorie.

Folgerungen

1. **Korollar.** Es gibt ein B mit $\mathbf{P}^B \neq \mathbf{PSPACE}^B$.
2. **Korollar.** Es gibt ein B mit $\mathbf{P}^B \neq \mathbf{coNP}^B$.
3. **Testfrage.** Wie folgen diese Korollare aus dem Satz?

Komplementierung von NP

1. Wir betrachten ein zweites Beispiel für Diagonalisierung.
2. **Testfrage.** Existiert ein A mit $\mathbf{NP}^A = \mathbf{coNP}^A$?
3. **Satz.** Es gibt eine Menge C mit $\mathbf{NP}^C \neq \mathbf{coNP}^C$.
4. Wir bauen C und betrachten wieder die Hilfsmenge
$$H = \{0^n : (\exists x \in C)(|x| = n)\} \in \mathbf{NP}^C.$$
5. Wir müssen erreichen, dass $H \notin \mathbf{coNP}^C$, also dass $\overline{H} \notin \mathbf{NP}^C$.

6. Seien $N_1, N_2, \ldots$ alle nicht-deterministischen Turingmaschinen, die in Polynomialzeit arbeiten.
7. Für jedes N_i sei p_i eine nichtfallende, polynomielle Zeitschranke.
8. Ohne Einschränkung sei $p_i(n) < p_{i+1}(n)$ für alle i und n.

Komplementierung von **NP**

1. **Erinnerung Ziel:** $\overline{H} \notin \mathbf{NP}^C$
2. Dafür genügt, dass es für alle n ein Wort 0^ℓ gibt mit

$$0^\ell \in \overline{H} \triangle L(N_n, C) = (H \cap L(N_n, C)) \cup (\overline{H} \cap \overline{L(N_n, C)}).$$

Testfrage. Warum?

Komplementierung von **NP**

1. **Erinnerung Ziel:** $\overline{H} \notin \mathbf{NP}^C$
2. Dafür genügt, dass es für alle n ein Wort 0^ℓ gibt mit
$$0^\ell \in \overline{H} \triangle L(N_n, C) = (H \cap L(N_n, C)) \cup (\overline{H} \cap \overline{L(N_n, C)}).$$
Testfrage. Warum?
3. Wir werden wieder für $C(n)$, $n \in \mathbb{N}$, eine Konstruktionsvorschrift angeben und dann $C = \lim_n C(n)$ definieren.

Komplementierung von NP

1. **Konstruktion:** *(Komplexität egal!)*

$C(0) \leftarrow \emptyset;$
$k(0) \leftarrow 0;$
Für $n = 1, 2, 3, \ldots$ **tu**
  $k(n) \leftarrow \min\{i > k(n-1) \colon\ 2^i > p_n(i) \wedge i > p_{n-1}(k(n-1))\};$
  Wenn $0^{k(n)} \in L(N_n, C(n-1))$ **dann**
    $r(n) \leftarrow \min_{\text{lex}} \left\{ r \colon \begin{array}{c} r \text{ ist akzeptierender Rechenpfad von } N_n \\ \text{mit Orakel } C(n-1) \text{ bei Eingabe } 0^{k(n)}. \end{array} \right\};$
    $w(n) \leftarrow \min_{\text{lex}} \left\{ w \colon \begin{array}{c} |w| = k(n) \text{ und } N_n \text{ mit Orakel} \\ C(n-1) \text{ fragt auf Rechenpfad } r(n) \\ \text{das Orakel nicht nach } w. \end{array} \right\};$
    $C(n) \leftarrow C(n-1) \cup \{w(n)\};$
  sonst
    $C(n) \leftarrow C(n-1);$
  Ende
Ende
$C \leftarrow \lim_n C(n);$

Verifikation

1. $k(n)$ und $w(n)$ existieren aus den gleichen Gründen wie zuvor.
2. **Behauptung.** Wenn man N_n mit Eingabe $0^{k(n)}$ ausführt, ist es *für den Rechenpfad* $r(n)$ egal ob das Orakel $C(n-1)$ oder C ist.
   - Der Grund ist der gleiche wie zuvor.
3. **Achtung:**
   - Die Wahl von $w(n)$ hängt von $r(n)$ ab. Nur $r(n)$ wird *geschützt*.
   - Andere Pfade fragen $w(n)$ möglicherweise ab, und für diese kann es einen Unterschied machen, ob das Orakel $C(n-1)$ oder C ist.
   - Das müssen wir bei den folgenden Überlegungen beachten!

Verifikation

1. $k(n)$ und $w(n)$ existieren aus den gleichen Gründen wie zuvor.
2. **Behauptung.** Wenn man N_n mit Eingabe $0^{k(n)}$ ausführt, ist es *für den Rechenpfad* $r(n)$ egal ob das Orakel $C(n-1)$ oder C ist.
   - Der Grund ist der gleiche wie zuvor.
3. **Achtung:**
   - Die Wahl von $w(n)$ hängt von $r(n)$ ab. Nur $r(n)$ wird *geschützt*.
   - Andere Pfade fragen $w(n)$ möglicherweise ab, und für diese kann es einen Unterschied machen, ob das Orakel $C(n-1)$ oder C ist.
   - Das müssen wir bei den folgenden Überlegungen beachten!
4. **Es folgt:** $0^{k(n)} \in L(N_n, C(n-1)) \iff 0^{k(n)} \in L(N_n, C)$.

Testfrage. Folgt „$\Longrightarrow$“ oder „$\Longleftarrow$“ aus obigen Überlegungen?

Verifikation

1. $k(n)$ und $w(n)$ existieren aus den gleichen Gründen wie zuvor.
2. **Behauptung.** Wenn man N_n mit Eingabe $0^{k(n)}$ ausführt, ist es *für den Rechenpfad* $r(n)$ egal ob das Orakel $C(n-1)$ oder C ist.
   - Der Grund ist der gleiche wie zuvor.
3. **Achtung:**
   - Die Wahl von $w(n)$ hängt von $r(n)$ ab. Nur $r(n)$ wird *geschützt*.
   - Andere Pfade fragen $w(n)$ möglicherweise ab, und für diese kann es einen Unterschied machen, ob das Orakel $C(n-1)$ oder C ist.
   - Das müssen wir bei den folgenden Überlegungen beachten!
4. **Es folgt:** $0^{k(n)} \in L(N_n, C(n-1)) \iff 0^{k(n)} \in L(N_n, C)$.

   Die Richtung „$\Longrightarrow$“ folgt aus obigen Überlegungen.

Verifikation

1. $k(n)$ und $w(n)$ existieren aus den gleichen Gründen wie zuvor.
2. **Behauptung.** Wenn man N_n mit Eingabe $0^{k(n)}$ ausführt, ist es *für den Rechenpfad* $r(n)$ egal ob das Orakel $C(n-1)$ oder C ist.
   - Der Grund ist der gleiche wie zuvor.
3. **Achtung:**
   - Die Wahl von $w(n)$ hängt von $r(n)$ ab. Nur $r(n)$ wird *geschützt*.
   - Andere Pfade fragen $w(n)$ möglicherweise ab, und für diese kann es einen Unterschied machen, ob das Orakel $C(n-1)$ oder C ist.
   - Das müssen wir bei den folgenden Überlegungen beachten!
4. **Es folgt:** $0^{k(n)} \in L(N_n, C(n-1)) \iff 0^{k(n)} \in L(N_n, C)$.

   Die Richtung „$\Longrightarrow$“ folgt aus obigen Überlegungen.

   Testfrage. Woraus folgt die Umkehrung „$\Longleftarrow$“, also äquivalent

$$0^{k(n)} \notin L(N_n, C(n-1)) \Longrightarrow 0^{k(n)} \notin L(N_n, C)?$$

Verifikation

1. **Behauptung.** $\overline{H} \notin \mathbf{NP}^C$.
   - Wähle n beliebig.
   - $0^{k(n)} \in H$ genau dann, falls $w(n)$ zu C hinzugefügt wurde.
   - Das wurde getan genau dann, wenn $0^{k(n)} \in L(N_n, C(n-1))$.
   - Wegen der letzten Folie gilt das genau dann, wenn
   $$0^{k(n)} \in L(N_n, C).$$
   - Also gilt $0^{k(n)} \in H$ genau dann, wenn $0^{k(n)} \in L(N_n, C)$.
   - Somit $\overline{H} \neq L(N_n, C)$ und, weil n beliebig war, $\overline{H} \notin \mathbf{NP}^C$. □
2. Also $H \notin \mathbf{coNP}^C$ und mit $H \in \mathbf{NP}^C$ folgt $\mathbf{NP}^C \neq \mathbf{coNP}^C$. □

Folgerung

1. **Korollar.** Es gibt ein C mit
$$\mathbf{NP}^C \neq \mathbf{PSPACE}^C \quad \text{und} \quad \mathbf{coNP}^C \neq \mathbf{PSPACE}^C.$$
2. **Testfrage.** Wie folgt dies aus dem Satz?

Kapitelzusammenfassung

1. Wir haben gezeigt, dass wenn man die Frage $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ relativ zu Orakeln betracht, die Antwort vom jeweiligen Orakel abhängt.
2. Das sieht man daran, dass einerseits $\mathbf{P}^{\mathrm{QBF}} = \mathbf{NP}^{\mathrm{QBF}}$ gilt, während es andererseits Mengen B gibt mit $\mathbf{P}^B \neq \mathbf{NP}^B$.
3. B haben wir mit *Diagonalisierung* gegen Berechnungen in Polynomialzeit konstruiert. Wir haben uns zunutze gemacht, dass eine solche Berechnung nur wenige Orakelanfragen stellen kann.
4. Wir haben gesehen, wie wichtig es ist, Berechnungen zu *schützen*.
5. Als zweites Beispiel haben wir gezeigt, dass es eine Menge C mit $\mathbf{NP}^C \neq \mathbf{coNP}^C$ gibt. Hierbei mussten wir gegen *nichtdeterministische Berechnungen* diagonalisieren. Dabei konnte nur je *ein* Rechenpfad geschützt werden, was aber ausreichend war.

4

2SAT ist in P

„Sanity checks“

1. Im letzten Abschnitt haben wir einen „sanity check“ gesehen, also eine simples Kriterium, um offensichtlich falsche Antworten auf die Frage $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ schnell zu erkennen.
2. **Nämlich:** Eine Mindestvoraussetzung für einen erfolgreichen Beweis für $\mathbf{P} = \mathbf{NP}$ oder $\mathbf{P} \neq \mathbf{NP}$ ist, dass er *nicht* auch relativiert zu beliebigen Orakeln funktioniert.
3. In diesem Abschnitt betrachten wir einen zweiten wichtigen „sanity check“.

2SAT als sanity check

1. In der Vorlesung „Komplexitätstheorie“ wurde das Problem 2SAT erwähnt, eine Variante des 3SAT-Problems, bei dem in jeder Klausel statt 3 genau 2 Literale auftauchen.
   Erinnerung. Ein „Literal“ ist eine Variable A oder ihre Negation $\neg A$.
2. **Beispiel.** $((A \vee \neg B) \wedge (\neg A \vee \neg C) \wedge (B \vee \neg C)) \in \mathrm{2SAT}$.
3. In diesem Abschnitt werden wir zeigen, dass 2SAT in **P** liegt.
4. Falls also beispielweise jemand einen (angeblichen) Beweis für $\mathbf{P} \neq \mathbf{NP}$ vorschlägt, bei dem versucht wird, zu zeigen, dass $\mathrm{3SAT} \notin \mathbf{P}$ gilt, so sollte dieser Beweis *nicht* auch für 2SAT funktionieren. Sonst ergäbe sich ein Widerspruch.
5. Scott Aaronson schreibt zu Tests dieser Art:

   Historically, this has probably been the single most important “sanity check” for claimed proofs of $\mathbf{P} \neq \mathbf{NP}$*: in fact, I’m pretty sure that every attempt I’ve ever seen has been refuted by it.*

   (Quelle und weitere Informationen: https://www.scottaaronson.com/blog/?p=458)

2SAT ist in Polynomialzeit lösbar

1. **Satz.** $2\mathrm{SAT} \in \mathbf{P}$.
2. **Notation.**
   - Im Beweis wird es an einigen Stellen notwendig sein, allgemein über Literale L (statt über Variablen A) zu sprechen.
   - Sei A eine Variable und $L = \neg A$. Dann meinen wir mit $\neg L$ einfach die Variable A (und nicht die Formel $\neg\neg A$).
   - Wir nehmen also an, dass doppelte Negationen stillschweigend sofort gelöscht werden. Das vereinfacht die Notation.
3. **Beweis.** Wir zeigen, dass es ein Verfahren gibt, dass nur Polynomialzeit benötigt und das korrekt für jede Formel $F \in 2\mathrm{SAT}$ entscheidet, ob F erfüllbar ist.

Umwandlung in einen Graphen

1. F ist von der Form
$$(L_{i_1} \vee L_{i_2}) \wedge (L_{i_3} \vee L_{i_4}) \wedge \cdots \wedge (L_{i_{\ell-1}} \vee L_{i_\ell}).$$
2. Wir generieren aus F den Graphen G_F mit der Knotenmenge
$$\{L_{i_1}, \neg L_{i_1}, L_{i_2}, \neg L_{i_2}, \ldots, L_{i_\ell}, \neg L_{i_\ell}\}$$
und der Kantenmenge
$$\left\{ \begin{array}{ccc} \neg L_{i_1} \longrightarrow L_{i_2}, & & \neg L_{i_2} \longrightarrow L_{i_1}, \\ \vdots & & \vdots \\ \neg L_{i_{\ell-1}} \longrightarrow L_{i_\ell}, & & \neg L_{i_\ell} \longrightarrow L_{i_{\ell-1}} \end{array} \right\}$$
3. **Intuition.** Eine Kante „$\neg L_i \to L_j$“ bedeutet, dass eine Belegung σ, die $\neg L_i$ wahr macht, F *höchstens* dann erfüllen kann, wenn sie auch L_j wahr macht. Denn sonst gäbe es in F die Klausel $(L_i \vee L_j)$, die von σ nicht erfüllt wird.

Konsistenz

1. **Definition.** Wir nennen G_F konsistent, falls für *jede* Variable A aus F gilt, dass G_F nicht sowohl einen Pfad $A \to \cdots \to \neg A$ als auch einen Pfad $\neg A \to \cdots \to A$ enthält.
2. **Behauptung.** F erfüllbar $\iff G_F$ konsistent.

   (Beweis siehe unten.)
3. Dann prüft der folgende Algorithmus, ob F erfüllbar ist:

   Berechne G_F aus F;
   Für jedes A *aus* F **tu**
   | Prüfe, ob ein Pfad $A \to \cdots \to \neg A$ existiert;
   | Prüfe, ob ein Pfad $\neg A \to \cdots \to A$ existiert;
   Ende
   Wenn *Inkonsistenz entdeckt* **dann** verwirf **sonst** akzeptiere;
4. **Testfrage.** Warum läuft der Algorithmus in Polynomialzeit? □

Äquivalenz von Konsistenz und Erfüllbarkeit

1. Es bleibt noch die Behauptung zu beweisen.
2. **Behauptung.** F erfüllbar $\iff G_F$ konsistent.
3. **Beweis.** Für „$\Longrightarrow$“ nimm an, G_F sei inkonsistent.
   - Dann gibt es A in F, so dass G_F sowohl einen Pfad der Form $A \to \cdots \to \neg A$ als auch der Form $\neg A \to \cdots \to A$ enthält.
   - Wegen $A \to \cdots \to \neg A$ kann es keine Belegung σ mit $\sigma(A) = 1$ geben, die F erfüllt.
   - Wegen $\neg A \to \cdots \to A$ kann es keine Belegung σ mit $\sigma(A) = 0$ geben, die F erfüllt.
   - Also gibt es gar keine Belegung, die F erfüllt.
   - Also ist F nicht erfüllbar.

Äquivalenz von Konsistenz und Erfüllbarkeit

1. „$\Longleftarrow$“ ist aufwändiger. Wir müssen unter der Annahme, G_F sei konsistent, eine Belegung σ konstruieren, die F erfüllt.
2. **Grobe Vorgehensweise:**
   - Wir erweitern G_F um zusätzliche Kanten zu einem Graph $G_{F'}$, der einer Formel $F' \in 2\text{SAT}$ der Form $F' = F \wedge C_1 \wedge \cdots \wedge C_k$ entspricht, wobei die C_i neue Klauseln mit je zwei Literalen sind.
   - Wir stellen sicher, dass die Konsistenz des Graphs bei jedem Hinzufügen einer Kante erhalten bleibt.
   - Wir definieren eine Belegung σ und zeigen, dass sie F' erfüllt.
   - Trivialerweise erfüllt dann σ auch F.

Äquivalenz von Konsistenz und Erfüllbarkeit

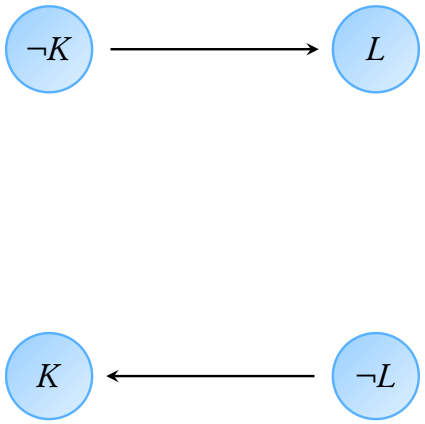
1. **Unterbehauptung.** Sei $H \in 2\mathrm{SAT}$, A eine Variable in H, und G_H konsistent. Dann ist mindestens einer konsistent von
   - $G^+ := (V(G_H), E(G_H) \cup \{A \to \neg A\})$ und
   - $G^- := (V(G_H), E(G_H) \cup \{\neg A \to A\})$.

   (Beweis siehe unten.)
2. Aufgrund der Unterbehauptung können wir für Variablen A aus F den Graph G_F so lange auf konsistente Weise um je eine der Kanten $A \to \neg A$ oder $\neg A \to A$ erweitern, bis für *jede* Variable A aus F eine der beiden Kanten vorhanden ist.
3. Der neue Graph $G_{F'}$ entspricht einer Formel der Form
$$F' = F \wedge (L'_1 \vee L'_1) \wedge \cdots \wedge (L'_k \vee L'_k).$$
4. Dann definiere die Belegung σ via
$$\sigma(A) = \begin{cases} 0 & \text{falls } A \to \neg A \in E(G_{F'}), \\ 1 & \text{falls } \neg A \to A \in E(G_{F'}). \end{cases}$$

Äquivalenz von Konsistenz und Erfüllbarkeit

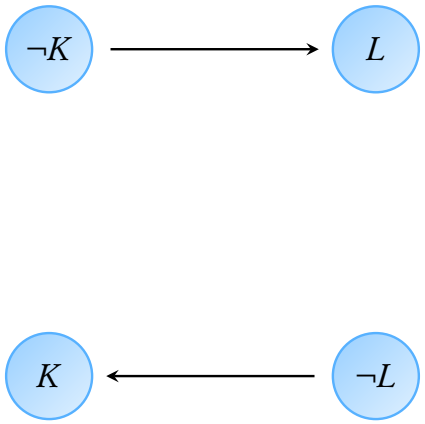
1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.

Äquivalenz von Konsistenz und Erfüllbarkeit



1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F',

Äquivalenz von Konsistenz und Erfüllbarkeit



1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird.

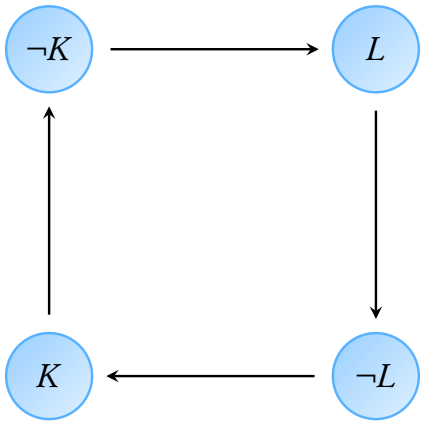
Äquivalenz von Konsistenz und Erfüllbarkeit

$$\neg K \longrightarrow L$$

$$K \longleftarrow \neg L$$

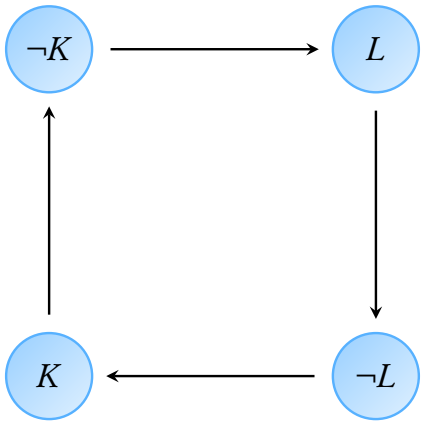
1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird. Also weist σ K und L jeweils den Wahrheitswert 0 zu.

Äquivalenz von Konsistenz und Erfüllbarkeit



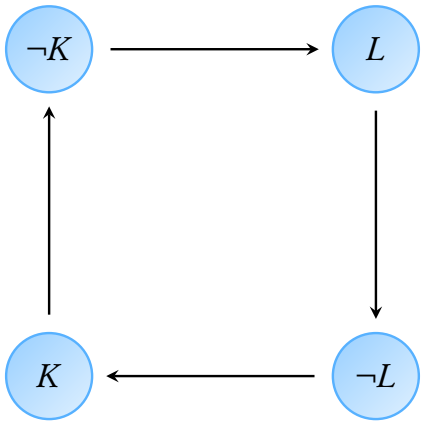
1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird. Also weist σ K und L jeweils den Wahrheitswert 0 zu.
3. Laut der Definitionsvorschrift für σ bedeutet das, dass die Kanten $L \to \neg L$ und $K \to \neg K$ in $G_{F'}$ existieren.

Äquivalenz von Konsistenz und Erfüllbarkeit



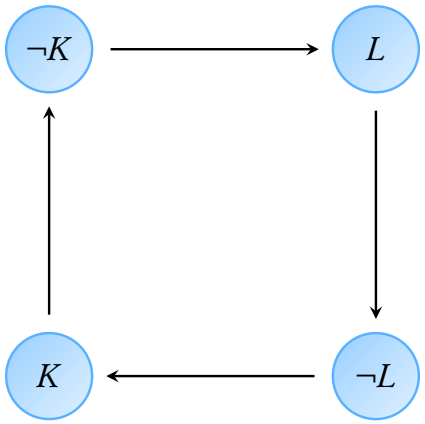
1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird. Also weist σ K und L jeweils den Wahrheitswert 0 zu.
3. Laut der Definitionsvorschrift für σ bedeutet das, dass die Kanten $L \to \neg L$ und $K \to \neg K$ in $G_{F'}$ existieren.
4. Man sieht sofort, dass $G_{F'}$ inkonsistent ist. **Testfrage.** Woran?

Äquivalenz von Konsistenz und Erfüllbarkeit



1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird. Also weist σ K und L jeweils den Wahrheitswert 0 zu.
3. Laut der Definitionsvorschrift für σ bedeutet das, dass die Kanten $L \to \neg L$ und $K \to \neg K$ in $G_{F'}$ existieren.
4. Man sieht sofort, dass $G_{F'}$ inkonsistent ist. **Testfrage.** Woran?
5. Das steht im Widerspruch zu unserer Konstruktion, die die Konsistenz durchgehend erhalten hat.

Äquivalenz von Konsistenz und Erfüllbarkeit



1. Wir zeigen, das F' von σ erfüllt wird. Nimm das Gegenteil an.
2. Dann existiert eine Klausel $(K \vee L)$ in F', die von σ nicht erfüllt wird. Also weist σ K und L jeweils den Wahrheitswert 0 zu.
3. Laut der Definitionsvorschrift für σ bedeutet das, dass die Kanten $L \to \neg L$ und $K \to \neg K$ in $G_{F'}$ existieren.
4. Man sieht sofort, dass $G_{F'}$ inkonsistent ist. **Testfrage.** Woran?
5. Das steht im Widerspruch zu unserer Konstruktion, die die Konsistenz durchgehend erhalten hat.
6. Also war die Widerspruchsannahme falsch, und σ erfüllt F'. $\square$

Konsistenzerhaltende Erweiterung des Graphen

1. Es bleibt zuletzt noch die Unterbehauptung zu zeigen.
2. **Unterbehauptung.** Sei $H \in 2\text{SAT}$, A eine Variable in H, und G_H konsistent. Dann ist mindestens einer konsistent von
   - $G^+ := (V(G_H), E(G_H) \cup \{A \to \neg A\})$ und
   - $G^- := (V(G_H), E(G_H) \cup \{\neg A \to A\})$.
3. **Beweis.** Wir untersuchen drei verschiedene Fälle.
   - **Fall 1:** Es gab in G_H bereits einen Pfad $A \to \cdots \to \neg A$. Dann ist G^+ konsistent, denn gemeinsam mit $A \to \neg A$ entstehen keine neuen durch Pfade verbundenen Paare von Knoten, also auch keine neuen Widersprüche.

     (Höchstens könnte es zwischen manchen Paaren von Knoten neue *kürzere* Pfade geben, da jetzt eventuell $A \to \neg A$ als „Abkürzung“ genutzt werden kann.)
   - **Fall 2:** Es gab in G_H bereits einen Pfad $\neg A \to \cdots \to A$. Dann ist G^- konsistent aufgrund eines symmetrischen Arguments.

Konsistenzerhaltende Erweiterung des Graphen

1. **Fall 3:** Es gab in G_H weder einen Pfad $A \to \cdots \to \neg A$ noch einen Pfad $\neg A \to \cdots \to A$. Dann sind G^+ und G^- konsistent.

   (Wir geben das Argument für G^+, das für G^- ist symmetrisch.)
2. Nimm an, G^+ sei inkonsistent. Dann muss ein Literal $L \neq A, \neg A$ diese Inkonsistenz bezeugen.
3. **Testfrage.** Warum nicht A oder $\neg A$?

Konsistenzerhaltende Erweiterung des Graphen

1. **Fall 3:** Es gab in G_H weder einen Pfad $A \to \cdots \to \neg A$ noch einen Pfad $\neg A \to \cdots \to A$. Dann sind G^+ und G^- konsistent.

   (Wir geben das Argument für G^+, das für G^- ist symmetrisch.)

2. Nimm an, G^+ sei inkonsistent. Dann muss ein Literal $L \neq A, \neg A$ diese Inkonsistenz bezeugen.

3. **Testfrage.** Warum nicht A oder $\neg A$?

   Antwort. Da wir im Fall 3 sind gibt es weder einen Pfad $A \to \cdots \to \neg A$ noch einen Pfad $\neg A \to \cdots \to A$. Durch Hinzufügen von $A \to \neg A$ entsteht nur einer dieser beiden Pfade.

Konsistenzerhaltende Erweiterung des Graphen

1. **Fall 3:** Es gab in G_H weder einen Pfad $A \to \cdots \to \neg A$ noch einen Pfad $\neg A \to \cdots \to A$. Dann sind G^+ und G^- konsistent.

   (Wir geben das Argument für G^+, das für G^- ist symmetrisch.)

2. Nimm an, G^+ sei inkonsistent. Dann muss ein Literal $L \neq A, \neg A$ diese Inkonsistenz bezeugen.

3. **Testfrage.** Warum nicht A oder $\neg A$?

   Antwort. Da wir im Fall 3 sind gibt es weder einen Pfad $A \to \cdots \to \neg A$ noch einen Pfad $\neg A \to \cdots \to A$. Durch Hinzufügen von $A \to \neg A$ entsteht nur einer dieser beiden Pfade.

4. G_H war also bezüglich einem $L \neq A, \neg A$ noch konsistent, G^+ ist es aber (laut Widerspruchsannahme) bezüglich L nicht mehr.

Konsistenzerhaltende Erweiterung des Graphen

1. G_H war also bezüglich einem $L \neq A, \neg A$ noch konsistent, G^+ ist es aber (laut Widerspruchsannahme) bezüglich L nicht mehr.
2. Somit muss einer der folgenden beiden Fälle gegeben sein:
   - **Unterfall 1:** In G_H gab es bereits die Pfade
     - $L \to \cdots \to \neg L$
     - $\neg L \to \cdots \to A$
     - $\neg A \to \cdots \to L$

     (Durch Hinzufügen von $A \to \neg A$ entstünde dann tatsächlich eine Inkonsistenz.)

     Doch dann gäbe es in G_H bereits den zusammengefügten Pfad

     $$\neg A \to \cdots \to L \to \cdots \to \neg L \to \ldots A,$$

     was der Annahme widerspricht, wir seien im Fall 3.

Konsistenzerhaltende Erweiterung des Graphen

1. G_H war also bezüglich einem $L \neq A, \neg A$ noch konsistent, G^+ ist es aber (laut Widerspruchsannahme) bezüglich L nicht mehr.
2. Somit muss einer der folgenden beiden Fälle gegeben sein:
   - **Unterfall 1:** In G_H gab es bereits die Pfade
     - $L \to \cdots \to \neg L$
     - $\neg L \to \cdots \to A$
     - $\neg A \to \cdots \to L$

     (Durch Hinzufügen von $A \to \neg A$ entstünde dann tatsächlich eine Inkonsistenz.)

     Doch dann gäbe es in G_H bereits den zusammengefügten Pfad
     $$\neg A \to \cdots \to L \to \cdots \to \neg L \to \ldots A,$$
     was der Annahme widerspricht, wir seien im Fall 3.
   - **Unterfall 2:** In G_H gab es bereits die Pfade
     - $L \to \cdots \to A$
     - $\neg L \to \cdots \to A$
     - $\neg A \to \cdots \to \neg L$
     - $\neg A \to \cdots \to L$

     (Durch Hinzufügen von $A \to \neg A$ entstünde dann tatsächlich eine Inkonsistenz.)

     Testfrage. Zeigen Sie, dass auch dieser Fall unmöglich ist! □

Konsistenzerhaltende Erweiterung des Graphen

1. G_H war also bezüglich einem $L \neq A, \neg A$ noch konsistent, G^+ ist es aber (laut Widerspruchsannahme) bezüglich L nicht mehr.
2. Somit muss einer der folgenden beiden Fälle gegeben sein:
   - **Unterfall 1:** In G_H gab es bereits die Pfade
     - $L \to \cdots \to \neg L$
     - $\neg L \to \cdots \to A$
     - $\neg A \to \cdots \to L$

     (Durch Hinzufügen von $A \to \neg A$ entstünde dann tatsächlich eine Inkonsistenz.)

     Doch dann gäbe es in G_H bereits den zusammengefügten Pfad
     $$\neg A \to \cdots \to L \to \cdots \to \neg L \to \ldots A,$$
     was der Annahme widerspricht, wir seien im Fall 3.
   - **Unterfall 2:** In G_H gab es bereits die Pfade
     - $L \to \cdots \to A$
     - $\neg L \to \cdots \to A$
     - $\neg A \to \cdots \to \neg L$
     - $\neg A \to \cdots \to L$

     (Durch Hinzufügen von $A \to \neg A$ entstünde dann tatsächlich eine Inkonsistenz.)

     Testfrage. Zeigen Sie, dass auch dieser Fall unmöglich ist! □
3. Damit ist der Beweis, dass 2SAT $\in$ **P**, vollständig. □

Ein “sanity check” für unser eigenes Argument

1. **Testfrage.** Woran scheitert der Versuch, den Beweis für $2\mathrm{SAT} \in \mathbf{P}$ so zu verändern, dass er $3\mathrm{SAT} \in \mathbf{P}$ zeigen würde?

2. **Bemerkung.** Tatsächlich ist 2SAT sogar **NL**-vollständig.
   - **Testfrage.** Haben Sie eine Idee, wie man $2\mathrm{SAT} \in \mathbf{NL}$ zeigt?
     (Benutzen Sie zusätzlich den Satz von Immerman und Szelepcsényi.)
   - **Testfrage.** Und eine grobe Idee, wie man **NL**-Schwere zeigt?
     (Benutzen Sie zusätzlich das Unerreichbarkeitsproblem.)

5 Die Polynomialzeithierarchie

Komplexitätsklassen relativ zu Komplexitätsklassen

1. Im letzten Abschnitt haben wir relativierte Komplexitätsklassen kennen gelernt. Dabei wurde auf ein bestimmtes Orakel relativiert, also auf eine fest gewählte Menge A. Beispiel: $\mathbf{NP}^A$.
2. Man kann Komplexitätsklassen aber auch auf andere Komplexitäts*klassen* relativieren.
3. **Definition.** Sei $\mathbf{R} \in \{\mathbf{P}, \mathbf{NP}, \mathbf{coNP}\}$ und $\mathbf{Q}$ eine beliebige Komplexitätsklasse. Dann sei $\mathbf{R}^{\mathbf{Q}} = \bigcup_{A \in \mathbf{Q}} \mathbf{R}^A$.
4. **Beispiel.** $\mathbf{P}^{\mathbf{NP}} = \bigcup_{A \in \mathbf{NP}} \mathbf{P}^A = \{L : (\exists A \in \mathbf{NP})(L \in \mathbf{P}^A)\}$.
5. **Testfrage.** Ein deterministischer und ein nicht-deterministischer Rechenpfad können kombiniert werden zu einem nicht-deterministischen. Warum könnte trotzdem $\mathbf{P}^{\mathbf{NP}} \neq \mathbf{NP}$ gelten?

Die Hierarchie

1. **Definition.** Die *Polynomialzeithierarchie* ist die Struktur aus den Mengen $\mathbf{\Delta}_k$, $\mathbf{\Sigma}_k$ und $\mathbf{\Pi}_k$, für alle $k \in \mathbb{N}$, definiert via
   - $\mathbf{\Sigma}_0 = \mathbf{\Pi}_0 = \mathbf{\Delta}_0 = \mathbf{P}$,
   - $\mathbf{\Delta}_{k+1} = \mathbf{P}^{\mathbf{\Sigma}_k}$.
   - $\mathbf{\Sigma}_{k+1} = \mathbf{NP}^{\mathbf{\Sigma}_k}$,
   - $\mathbf{\Pi}_{k+1} = \mathbf{coNP}^{\mathbf{\Sigma}_k}$,

   (Normalerweise markiert man die Klassen mit einem „^{p}“, um sie von den Stufen der Arithmetischen Hierarchie aus der Berechenbarkeitstheorie abzugrenzen. Da wir uns mit letzterer in dieser Vorlesung aber nicht beschäftigen, ersparen wir uns das.)
2. **Definition.** Schreibe **PH** für $\bigcup_{k \geq 0} \mathbf{\Delta}_k = \bigcup_{k \geq 0} \mathbf{\Sigma}_k = \bigcup_{k \geq 0} \mathbf{\Pi}_k$.

   (Die Gleichungen folgen aus den Eigenschaften auf den folgenden Folien.)

Einfache Eigenschaften

1. **Proposition.** $\mathbf{\Delta}_1 = \mathbf{P}$.
2. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Pi}_k = \mathbf{co\Sigma}_k$.
3. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Sigma}_{k+1} = \mathbf{NP}^{\mathbf{\Pi}_k}$.
4. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Delta}_{k+1} = \mathbf{P}^{\mathbf{\Pi}_k}$.
5. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Pi}_{k+1} = \mathbf{coNP}^{\mathbf{\Pi}_k}$.
6. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Sigma}_{k+1} = \mathbf{NP}^{\mathbf{\Delta}_{k+1}}$.
7. **Proposition.** Für alle $k \geq 0$, $\mathbf{\Pi}_{k+1} = \mathbf{coNP}^{\mathbf{\Delta}_{k+1}}$.

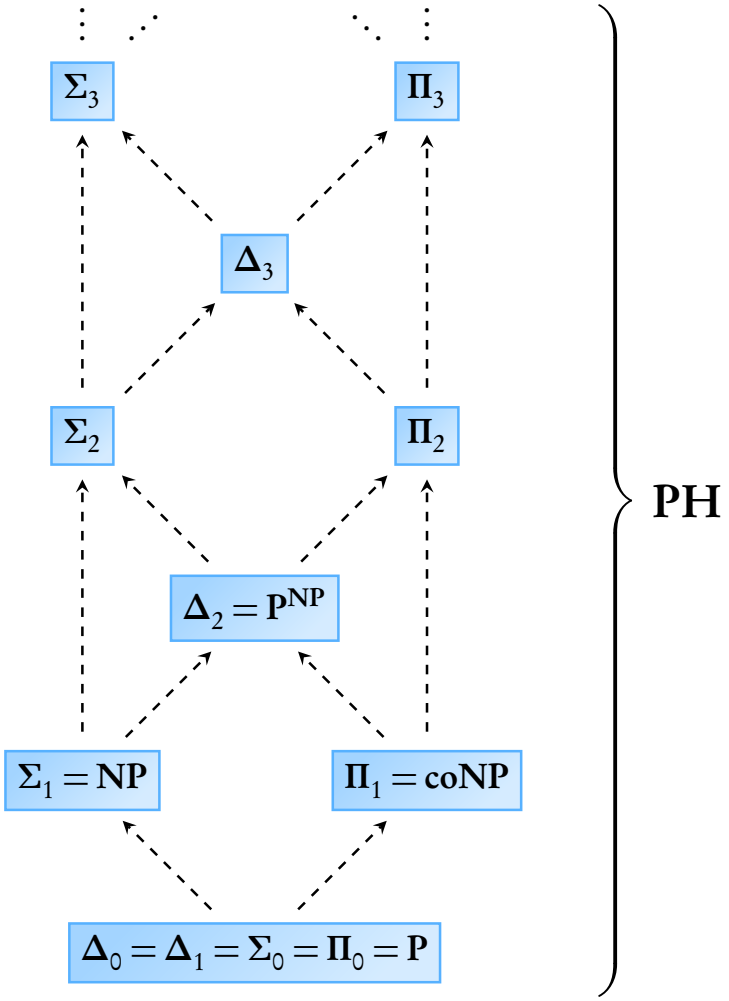
(Beweise → Übungszettel)

Weitere einfache Eigenschaften für alle $k \geq 0$

1. **Proposition.** $\mathbf{\Delta}_k$ ist unter Komplement abgeschlossen.
2. **Proposition.** $\mathbf{P}^{\mathbf{\Delta}_k} = \mathbf{\Delta}_k$. Also ist $\mathbf{\Delta}_k$ unter $\leq_{\mathrm{T}}^{\mathrm{p}}$ abgeschlossen.
3. **Proposition.** $\mathbf{\Sigma}_k$, $\mathbf{\Pi}_k$ sind unter $\leq_{\mathrm{m}}^{\mathrm{p}}$ abgeschlossen.
4. **Proposition.** $\mathbf{\Sigma}_k \cup \mathbf{\Pi}_k \subseteq \mathbf{\Delta}_{k+1}$.
5. **Proposition.** $\mathbf{\Delta}_k \subseteq \mathbf{\Sigma}_k \cap \mathbf{\Pi}_k$
6. **Proposition.** $\mathbf{\Delta}_k$, $\mathbf{\Sigma}_k$, $\mathbf{\Pi}_k$ sind unter $\oplus$, $\cup$, $\cap$ abgeschlossen.
7. **Proposition.** $\mathbf{\Sigma}_k \subseteq \mathbf{\Pi}_k$ oder $\mathbf{\Pi}_k \subseteq \mathbf{\Sigma}_k$ impliziert $\mathbf{\Sigma}_k = \mathbf{\Pi}_k$.

(Beweise → Übungszettel)

Die Polynomialzeithierarchie



1. **Satz. PH $\subseteq$ PSPACE** *(Beweis → Übungszettel)*
2. **Bemerkung.** Von keiner Inklusion ist bekannt, ob sie echt ist.

Quantifizierte Komplexitätsklassen

1. **Definition.** Sei $\langle\rangle\colon \left(\{0,1\}^{<\mathbb{N}}\right)^{<\mathbb{N}} \to \{0,1\}^{<\mathbb{N}}$ eine bijektive Kodierungsfunktion mit $\langle\rangle, \langle\rangle^{-1} \in \mathbf{FP}$, und so dass es für alle k ein Polynom p_k in k Variablen gebe, so dass für alle $x_1, \ldots, x_k$
$$|\langle x_1, \ldots, x_k\rangle| = p_k(|x_1|, \ldots, |x_k|).$$

2. **Definition.** Für ein Prädikat $R(x)$ über Wörter x schreiben wir
   - $\exists^{p(n)} x\, R(x)$, falls es ein Wort x mit $|x| = p(n)$ gibt so dass $R(x)$ gilt;
   - $\forall^{p(n)} x\, R(x)$, falls $R(x)$ gilt für alle x mit $|x| = p(n)$.

3. **Definition.** Sei $\mathbf{C}$ eine Menge von Sprachen. Dann definiere
$$\exists\mathbf{C} = \{A : \exists B \in \mathbf{C} \text{ und Polynom } p \text{ mit } x \in A \longleftrightarrow \exists^{p(|x|)} y\, \langle x, y\rangle \in B\};$$
$$\forall\mathbf{C} = \{A : \exists B \in \mathbf{C} \text{ und Polynom } p \text{ mit } x \in A \longleftrightarrow \forall^{p(|x|)} y\, \langle x, y\rangle \in B\}.$$

Weitere Definitionen

1. **Definition.** Eine Menge $\mathbf{C}$ von Sprachen heißt *unter Paarbildung abgeschlossen*, falls
$$A \in \mathbf{C} \Longrightarrow \langle A, \{0,1\}^{<\mathbb{N}} \rangle := \{\langle x, y \rangle : x \in A,\, y \in \{0,1\}^{<\mathbb{N}}\} \in \mathbf{C}.$$
2. **Testfrage.** Welche Komplexitätsklassen haben diese Eigenschaft?
3. **Definition.** Sei A eine Sprache. Dann definiere
$$A^* = \{x : \exists n \text{ mit } x = \langle y_1, \ldots, y_n \rangle \text{ und für alle } i \leq n,\, y_i \in A\}.$$

Einfache Eigenschaften

1. **Proposition.** Für jedes $\mathbf{C}$ und A gilt: $A \in \exists\mathbf{C} \iff \overline{A} \in \forall\mathbf{coC}$.
2. **Testfrage.** Wie beweist man das?

Einfache Eigenschaften

1. **Proposition.** Für jedes $\mathbf{C}$ und A gilt: $A \in \exists\mathbf{C} \iff \overline{A} \in \forall\mathbf{coC}$.
2. **Testfrage.** Wie beweist man das?

Antwort.

$$
\begin{aligned}
& \overline{A} \in \forall\mathbf{coC} \\
\Leftrightarrow\ & \exists \overline{B} \in \mathbf{coC} \text{ und Polynom } p \text{ mit } x \in \overline{A} \longleftrightarrow \forall^{p(|x|)} y\ \langle x, y\rangle \in \overline{B} \\
\Leftrightarrow\ & \exists B \in \mathbf{C} \quad \text{ und Polynom } p \text{ mit } x \in A \longleftrightarrow \exists^{p(|x|)} y\ \langle x, y\rangle \in B \\
\Leftrightarrow\ & A \in \exists\mathbf{C}
\end{aligned}
$$

□

Einfache Eigenschaften

1. **Proposition.** Falls $\mathbf{C}$ unter Paarbildung abgeschlossen ist, gilt
$$\mathbf{C} \subseteq \forall\mathbf{C} \text{ und } \mathbf{C} \subseteq \exists\mathbf{C}.$$
2. **Testfrage.** Wie beweist man das?

Einfache Eigenschaften

1. **Proposition.** Falls $\mathbf{C}$ unter Paarbildung abgeschlossen ist, gilt
$$\mathbf{C} \subseteq \forall\mathbf{C} \text{ und } \mathbf{C} \subseteq \exists\mathbf{C}.$$
2. **Testfrage.** Wie beweist man das?

   Antwort.
   - Sei $B \in \mathbf{C}$ und p ein beliebiges Polynom.
   - Da $\mathbf{C}$ unter Paarbildung abgeschlossen ist, gilt auch
   $$\widehat{B} := \langle B, \{0,1\}^{<\mathbb{N}} \rangle \in \mathbf{C}.$$
   - Nach Definition gilt dann für
   $$A := \{x \colon \forall^{p(|x|)} y \, \langle x, y \rangle \in \widehat{B}\} = \{x \colon \exists^{p(|x|)} y \, \langle x, y \rangle \in \widehat{B}\},$$
   dass $B = A$ und $A \in \forall\mathbf{C} \cap \exists\mathbf{C}$. $\square$

Ein technisches Lemma

1. **Lemma.** Sei $\mathbf{C} \in \{\mathbf{\Delta}_k, \mathbf{\Sigma}_k, \mathbf{\Pi}_k : k \in \mathbb{N}\}$. Dann gilt für beliebige A,
$$A \in \mathbf{C} \Longleftrightarrow A^* \in \mathbf{C}.$$
2. **Testfrage.** Die Richtung „$\Longleftarrow$“ ist einfach. Warum gilt sie?
3. **Beweis.** Es bleibt also die Richtung „$\Longrightarrow$“ zu zeigen.

Ein technisches Lemma

1. Wir betrachten zunächst den Fall $\mathbf{C} = \mathbf{\Delta}_k$ für beliebiges $k \geq 0$.
2. Sei M eine Maschine, die bezeugt, dass ein festes A in $\mathbf{C}$ ist.
3. **Testfrage.** Wie sieht M im Fall $k = 0, 1$ aus? Wie im Fall $k \geq 2$?
4. Der folgende Algorithmus zeigt, dass $A^* \in \mathbf{C}$:

   $x = \langle y_1, \ldots, y_n \rangle \leftarrow$ Eingabe;
   Für $i = 1, \ldots, n$ **tu**
   - Simuliere M mit Eingabe y_i;
   - **Wenn** M *verwirft* **dann**
     - Brich den Algorithmus ab und verwirf;
   - **Ende**

   Ende
   Akzeptiere;

5. Der gleiche Algorithmus funktioniert auch im Fall $\mathbf{C} = \mathbf{\Sigma}_k$.

   Testfrage. Warum? Was ist trotzdem anders?

Ein technisches Lemma

1. Es bleibt der Fall $\mathbf{C} = \mathbf{\Pi}_k$.
2. Wähle ein festes $A \in \mathbf{C}$ und ein M, das bezeugt, dass $\overline{A}$ in $\mathbf{\Sigma}_k$ ist.
3. **Testfrage.** Sind $\overline{A^*}$ und $\overline{A}^*$ gleich?

Ein technisches Lemma

1. Es bleibt der Fall $\mathbf{C} = \mathbf{\Pi}_k$.
2. Wähle ein festes $A \in \mathbf{C}$ und ein M, das bezeugt, dass $\overline{A}$ in $\mathbf{\Sigma}_k$ ist.
3. **Testfrage.** Sind $\overline{A^*}$ und $\overline{A}^*$ gleich?
4. Der folgende Algorithmus zeigt, dass $\overline{A^*} \in \mathbf{\Sigma}_k$:

   $x = \langle y_1, \ldots, y_n \rangle \leftarrow$ Eingabe;
   Für $i = 1, \ldots, n$ **tu**
   - Simuliere M mit Eingabe y_i;
   - **Wenn** M *akzeptiert* **dann**
     - Brich den Algorithmus ab und akzeptiere;
   - **Ende**

   Ende
   Verwirf;

5. Also ist A^* in $\mathbf{\Pi}_k = \mathbf{C}$. □

Charakterisierung mit Quantoren

1. **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:
   - $\exists\mathbf{P} = \mathbf{NP}$
   - $\forall\mathbf{P} = \mathbf{coNP}$
   - $\exists\Sigma_k = \Sigma_k$
   - $\forall\Pi_k = \Pi_k$
   - $\exists\Pi_\ell = \Sigma_{\ell+1}$
   - $\forall\Sigma_\ell = \Pi_{\ell+1}$
2. **Beweis $\exists\mathbf{P} = \mathbf{NP}$.** Sei $A \in \exists\mathbf{P}$. Dann gibt es ein $B \in \mathbf{P}$ und ein Polynom p mit $(x \in A \longleftrightarrow \exists^{p(|x|)} y\ \langle x, y\rangle \in B)$.
   Der folgende Algorithmus zeigt $A \in \mathbf{NP}$:

   $x \leftarrow$ Eingabe;
   Rate y mit $|y| = p(|x|)$;
   Wenn $\langle x, y\rangle \in B$ **dann** akzeptiere **sonst** verwirf;

   Sei umgekehrt $A \in \mathbf{NP}$ und N so gewählt, dass es dies bezeugt. Wähle ein Polynom p so groß, dass sich jede Berechnung von N mit Eingabelänge n in $p(n)$ Zeichen darstellen lässt. Dann gilt:

   $$x \in A \longleftrightarrow \exists^{p(|x|)} y\ (y \text{ ist akzeptierende Berechnung von } N \text{ auf } x)$$

   Das Prädikat rechts ist polynomiell entscheidbar, also $A \in \exists\mathbf{P}$.

Charakterisierung mit Quantoren

1. **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:
   - $\exists\mathbf{P} = \mathbf{NP}$ ✓
   - $\forall\mathbf{P} = \mathbf{coNP}$
   - $\exists\Sigma_k = \Sigma_k$
   - $\forall\Pi_k = \Pi_k$
   - $\exists\Pi_\ell = \Sigma_{\ell+1}$
   - $\forall\Sigma_\ell = \Pi_{\ell+1}$

3. **Beweis $\forall\mathbf{P} = \mathbf{coNP}$.** Wir hatten bereits bewiesen, dass

$$\text{für alle } \mathbf{C}, A: \quad A \in \exists\mathbf{C} \iff \overline{A} \in \forall\mathbf{coC}.$$

Also folgt aus $\exists\mathbf{P} = \mathbf{NP}$ dass

$$A \in \forall\mathbf{P} = \forall\mathbf{cocoP} \iff \overline{A} \in \exists\mathbf{coP} = \exists\mathbf{P} = \mathbf{NP} \iff A \in \mathbf{coNP}.$$

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists \mathbf{P} = \mathbf{NP}$ ✓
- $\forall \mathbf{P} = \mathbf{coNP}$ ✓
- $\exists \Sigma_k = \Sigma_k$
- $\forall \Pi_k = \Pi_k$
- $\exists \Pi_\ell = \Sigma_{\ell+1}$
- $\forall \Sigma_\ell = \Pi_{\ell+1}$

4 **Beweis** $\exists \Sigma_k = \Sigma_k$**.** Wir haben bereits $\Sigma_k \subseteq \exists \Sigma_k$ gesehen.

Sei also umgekehrt $A \in \exists \Sigma_k$. Dann gibt es ein $B \in \Sigma_k$ und ein Polynom p mit $(x \in A \longleftrightarrow \exists^{p(|x|)} y \, \langle x, y \rangle \in B)$. Seien N eine nichtdeterministische Maschine und $D \in \Sigma_{k-1}$ ein Orakel, die bezeugen, dass $B \in \mathbf{NP}^{\Sigma_{k-1}} = \Sigma_k$ ist; also $B = L(N, D)$.

$x \leftarrow$ Eingabe;
Rate y mit $|y| = p(|x|)$;
Wenn *N mit Orakel D akzeptiert* $\langle x, y \rangle$ **dann**
  Akzeptiere;
Sonst
  Verwirf;
Ende

Dieser Algorithmus zeigt $A \in \mathbf{NP}^{\Sigma_{k-1}} = \Sigma_k$.

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists \mathbf{P} = \mathbf{NP}$ ✓
- $\forall \mathbf{P} = \mathbf{coNP}$ ✓
- $\exists \mathbf{\Sigma}_k = \mathbf{\Sigma}_k$ ✓
- $\forall \mathbf{\Pi}_k = \mathbf{\Pi}_k$
- $\exists \mathbf{\Pi}_\ell = \mathbf{\Sigma}_{\ell+1}$
- $\forall \mathbf{\Sigma}_\ell = \mathbf{\Pi}_{\ell+1}$

5 **Beweis $\forall \mathbf{\Pi}_k = \mathbf{\Pi}_k$.**
Wir hatten bereits bewiesen, dass

$$\text{für alle } \mathbf{C}, A: \quad A \in \exists \mathbf{C} \iff \overline{A} \in \forall \mathbf{coC}.$$

Also folgt aus $\exists \mathbf{\Sigma}_k = \mathbf{\Sigma}_k$ dass

$$A \in \forall \mathbf{\Pi}_k \iff \overline{A} \in \exists \mathbf{co\Pi}_k = \exists \mathbf{\Sigma}_k = \mathbf{\Sigma}_k \iff A \in \mathbf{co\Sigma}_k = \mathbf{\Pi}_k.$$

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists\mathbf{P} = \mathbf{NP}$ ✓
- $\forall\mathbf{P} = \mathbf{coNP}$ ✓
- $\exists\Sigma_k = \Sigma_k$ ✓
- $\forall\Pi_k = \Pi_k$ ✓
- $\exists\Pi_\ell = \Sigma_{\ell+1}$
- $\forall\Sigma_\ell = \Pi_{\ell+1}$

6 **Beweis** $\exists\Pi_\ell \subseteq \Sigma_{\ell+1}$.

Sei $A \in \exists\Pi_\ell$. Dann gibt es ein $B \in \Pi_\ell$ und ein Polynom p mit

$$(x \in A \longleftrightarrow \exists^{p(|x|)} y \; \langle x, y \rangle \in B).$$

Der Algorithmus

$x \leftarrow$ Eingabe;
Rate y mit $|y| = p(|x|)$;
Wenn $\langle x, y \rangle \in B$ **dann** akzeptiere **sonst** verwirf;

zeigt, dass $A \in \mathbf{NP}^B \subseteq \mathbf{NP}^{\Pi_\ell} = \Sigma_{\ell+1}$. ✓

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists \mathbf{P} = \mathbf{NP}$ ✓
- $\forall \mathbf{P} = \mathbf{coNP}$ ✓
- $\exists \Sigma_k = \Sigma_k$ ✓
- $\forall \Pi_k = \Pi_k$ ✓
- $\exists \Pi_\ell = \Sigma_{\ell+1}$
- $\forall \Sigma_\ell = \Pi_{\ell+1}$

7 **Beweis $\exists \Pi_\ell \supseteq \Sigma_{\ell+1}$.** Wir beweisen dies per Induktion über ℓ.

Der Fall $\ell = 0$ folgt direkt aus $\exists \mathbf{P} \supseteq \mathbf{NP}$. ✓

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists\mathbf{P} = \mathbf{NP}$ ✓
- $\forall\mathbf{P} = \mathbf{coNP}$ ✓
- $\exists\Sigma_k = \Sigma_k$ ✓
- $\forall\Pi_k = \Pi_k$ ✓
- $\exists\Pi_\ell = \Sigma_{\ell+1}$
- $\forall\Sigma_\ell = \Pi_{\ell+1}$

7 **Beweis $\exists\Pi_\ell \supseteq \Sigma_{\ell+1}$.** Wir beweisen dies per Induktion über ℓ.

Nimm also an, $\Sigma_\ell \subseteq \exists\Pi_{\ell-1}$ für ein $\ell \geq 1$ und sei $A \in \Sigma_{\ell+1}$. Dann existiert ein in Polynomialzeit arbeitendes, nicht-deterministisches N und ein Orakel $B \in \Sigma_\ell$ mit $A = L(N, B)$. Wähle ein Polynom p so groß, dass sich jede Berechnung von N mit Eingabelänge n in $p(n)$ Zeichen darstellen lässt. Dann gilt:

$$x \in A \longleftrightarrow \exists^{p(|x|)} y \; \exists\langle z_1, \ldots, z_i\rangle \in B^* \; \exists\langle w_1, \ldots, w_j\rangle \in \overline{B}^*$$

(y ist akzeptierende Berechnung von N auf x, in der die Orakelfragen $z_1, \ldots, z_i$ mit „ja“ und die Orakelfragen $w_1, \ldots, w_j$ mit „nein“ beantwortet werden, und in der keine weiteren Fragen an das Orakel gestellt werden.)

Dieser Ausdruck zeigt schon fast $A \in \exists\Pi_\ell$; wir benötigen nur noch, dass das Prädikat rechts in Π_ℓ ist.

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists\mathbf{P} = \mathbf{NP}$ ✓
- $\forall\mathbf{P} = \mathbf{coNP}$ ✓
- $\exists\Sigma_k = \Sigma_k$ ✓
- $\forall\Pi_k = \Pi_k$ ✓
- $\exists\Pi_\ell = \Sigma_{\ell+1}$
- $\forall\Sigma_\ell = \Pi_{\ell+1}$

7 **Beweis $\exists\Pi_\ell \supseteq \Sigma_{\ell+1}$.** Wir beweisen dies per Induktion über ℓ.

Wegen $B \in \Sigma_\ell$ folgt $\overline{B} \in \Pi_\ell$; und somit $B^* \in \Sigma_\ell$ und $\overline{B}^* \in \Pi_\ell$.

Nach Induktionsannahme $B^* \in \exists\Pi_{\ell-1}$, also für ein $D \in \Pi_{\ell-1}$:

$$u \in B^* \longleftrightarrow \exists^{p'(|u|)} t \, \langle u, t \rangle \in D.$$

Wir können also ein äquivalentes Prädikat in Π_ℓ formulieren:

$$x \in A \longleftrightarrow \exists^{p''(|x|)} \langle y, t, z_1, \ldots, z_i, w_1, \ldots, w_j \rangle$$

$$\left(\begin{array}{c} y \text{ ist akzeptierende Berechnung von } N \text{ auf } x\text{, in der die} \\ \text{Orakelfragen } z_1, \ldots, z_i \text{ mit „ja“ und die Fragen } w_1, \ldots, w_j \\ \text{mit „nein“ beantwortet werden, und in der sonst keine} \\ \text{Fragen gestellt werden, und so dass } \langle \langle z_1, \ldots, z_i \rangle, t \rangle \in D \\ \text{und } |t| = p'(|\langle z_1, \ldots, z_i \rangle|) \text{ und } \langle w_1, \ldots, w_j \rangle \in \overline{B}^*. \end{array} \right)$$

Wegen $D \in \Pi_{\ell-1} \subseteq \Pi_\ell$ und $\overline{B}^* \in \Pi_\ell$ gilt $A \in \exists\Pi_\ell$.

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists \mathbf{P} = \mathbf{NP}$ ✓
- $\forall \mathbf{P} = \mathbf{coNP}$ ✓
- $\exists \Sigma_k = \Sigma_k$ ✓
- $\forall \Pi_k = \Pi_k$ ✓
- $\exists \Pi_\ell = \Sigma_{\ell+1}$ ✓
- $\forall \Sigma_\ell = \Pi_{\ell+1}$

8 **Beweis $\forall \Sigma_\ell = \Pi_{\ell+1}$.** Wendet man in $\exists \Pi_\ell = \Sigma_{\ell+1}$ auf beiden Seiten den Operator **co** an, so erhält man $\mathbf{co}\exists \Pi_\ell = \mathbf{co}\Sigma_{\ell+1}$, also

$$\forall \Sigma_\ell = \forall \mathbf{co}\Pi_\ell = \mathbf{co}\exists \Pi_\ell = \mathbf{co}\Sigma_{\ell+1} = \Pi_{\ell+1}.$$

Charakterisierung mit Quantoren

1 **Satz.** Folgende Aussagen gelten für alle $k \geq 1$ und $\ell \geq 0$:

- $\exists\mathbf{P} = \mathbf{NP}$ ✓
- $\forall\mathbf{P} = \mathbf{coNP}$ ✓
- $\exists\Sigma_k = \Sigma_k$ ✓
- $\forall\Pi_k = \Pi_k$ ✓
- $\exists\Pi_\ell = \Sigma_{\ell+1}$ ✓
- $\forall\Sigma_\ell = \Pi_{\ell+1}$ ✓

9 Damit ist der Satz bewiesen. □

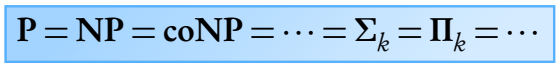
Charakterisierung mit Quantoren

1. **Satz.** Für alle $k \geq 0$ gilt, dass
   - $A \in \Sigma_k$ g.d.w. $B \in \mathbf{P}$ und Polynome $p_1, \ldots, p_k$ existieren mit
     $$x \in A \iff \exists^{p_1(|x|)} y_1 \; \forall^{p_2(|x|)} y_2 \; \ldots \; \exists/\forall^{p_k(|x|)} y_k \; \langle x, y_1, \ldots, y_k \rangle \in B.$$
   - $A \in \mathbf{\Pi}_k$ g.d.w. $B \in \mathbf{P}$ und Polynome $p_1, \ldots, p_k$ existieren mit
     $$x \in A \iff \forall^{p_1(|x|)} y_1 \; \exists^{p_2(|x|)} y_2 \; \ldots \; \forall/\exists^{p_k(|x|)} y_k \; \langle x, y_1, \ldots, y_k \rangle \in B.$$
2. **Beweis.**
   - Der Fall $k = 0$ ist trivial.
   - Wir beweisen die erste Aussage unter der Annahme, sie sei für $k - 1 \geq 0$ wahr. Sei $A \in \Sigma_k$. Laut dem letzten Satz ist $A \in \exists\mathbf{\Pi}_{k-1}$. Also existieren $B \in \mathbf{\Pi}_{k-1}$, p_1 mit $x \in A \longleftrightarrow \exists^{p_1(|x|)} y \; \langle x, y \rangle \in B$.
     Laut Induktionsannahme gibt es $D \in \mathbf{P}$ und $p_2, \ldots, p_k$, so dass
     $$\langle x, y \rangle \in B \longleftrightarrow \forall^{p_2(|\langle x,y \rangle|)} z_1 \; \ldots \; \exists/\forall^{p_k(|\langle x,y \rangle|)} z_{k-1} \; \langle \langle x, y \rangle, z_1, \ldots, z_{k-1} \rangle \in D.$$
     Sei $D' \in \mathbf{P}$ so, dass $\langle \langle x, y \rangle, \ldots \rangle \in D \longleftrightarrow \langle x, y, \ldots \rangle \in D'$. Dann gilt
     $$x \in A \longleftrightarrow \exists^{p_1(|x|)} y \; \forall^{p'_2(|x|)} z_1 \; \ldots \; \exists/\forall^{p'_k(|x|)} z_{k-1} \; \langle x, y, z_1, \ldots, z_{k-1} \rangle \in D'.$$
     Da alle Schritte Äquivalenzen sind, gilt auch die Rückrichtung.
   - **Testfrage.** Wie folgt daraus die zweite Aussage? □

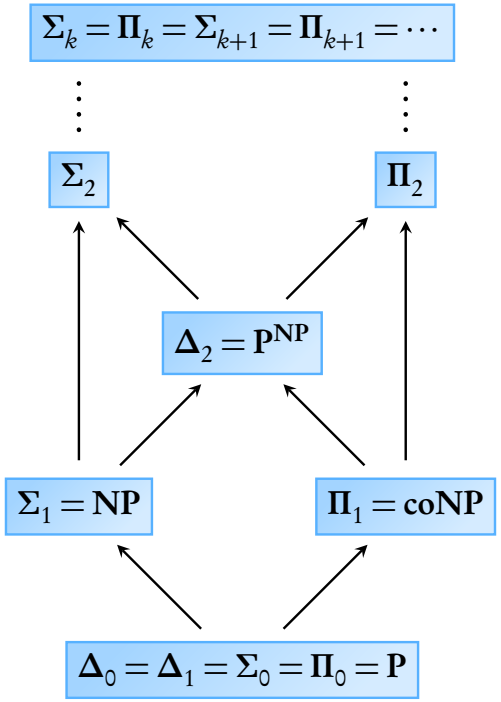
Folgerungen

1. **Korollar.** Falls $\mathbf{\Sigma}_k = \mathbf{\Pi}_k$ für ein $k \geq 1$, dann gilt
$$\mathbf{\Sigma}_k = \mathbf{\Pi}_k = \mathbf{\Sigma}_{k+1} = \mathbf{\Pi}_{k+1} = \mathbf{\Sigma}_{k+2} = \mathbf{\Pi}_{k+2} = \cdots$$
Beweis. Aus den bisherigen Ergebnissen und der Annahme folgt
$$\mathbf{\Sigma}_{k+1} = \exists \mathbf{\Pi}_k = \exists \mathbf{\Sigma}_k = \mathbf{\Sigma}_k.$$
Durch Komplementierung folgt $\mathbf{\Pi}_{k+1} = \mathbf{\Pi}_k = \mathbf{\Sigma}_k = \mathbf{\Sigma}_{k+1}$.
Testfrage. Wie folgt nun die Gleichheit für $_{+2}, _{+3}, _{+4}, \ldots$?
2. **Proposition.** Falls $\mathbf{P} = \mathbf{NP}$, dann $\mathbf{P} = \mathbf{PH}$.
Testfrage. Wie sieht man das?

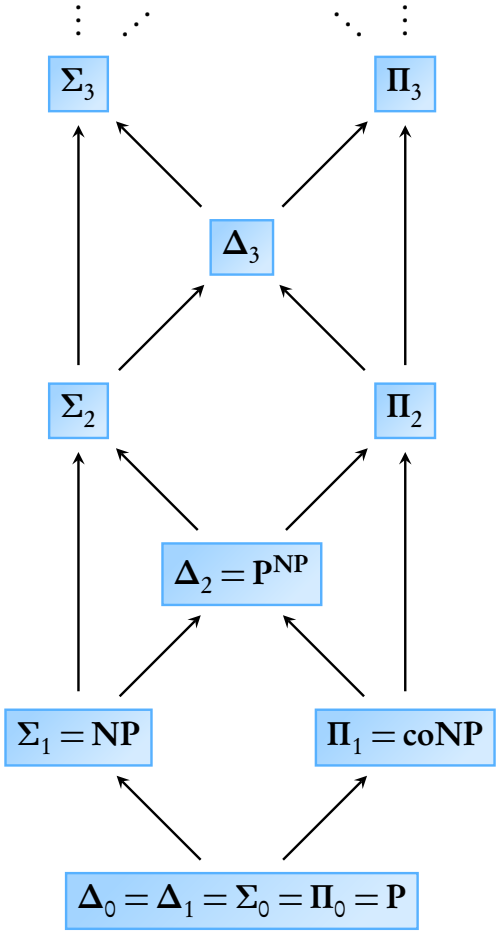
Mögliche Strukturen der Polynomialzeithierarchie



Kollaps auf Ebene 0



Kollaps auf Ebene $k \geq 1$



Kein Kollaps

PH und PSPACE

1. **Satz.** Falls **PH** = **PSPACE**, dann kollabiert **PH**.

(Beweis → Übungszettel)

Eine Variante des Halteproblems

1. **Definition.** Wir bezeichnen die Menge

$$K^A = \left\{ \langle N, x, 1^t \rangle : \begin{array}{c} N \text{ ist eine nichtderministische Turing-} \\ \text{maschine die mit Orakel } A \text{ die Eingabe } x \\ \text{während der ersten } t \text{ Schritte akzeptiert} \end{array} \right\}.$$

als *nichtdeterministisches Halteproblem relativ zu A*.

2. **Definition.** Wir schreiben K für $K^\emptyset$ und K^n für $K^{\left.K^{\cdot^{\cdot^{\cdot^{K}}}}\right\} n \text{ mal}}$.

3. **Proposition.** Für alle $n \geq 1$ und alle für $\leq_{\mathrm{m}}^{\mathrm{p}}$ Σ_n-vollständigen Sprachen A gilt, dass K^A Σ_{n+1}-vollständig für $\leq_{\mathrm{m}}^{\mathrm{p}}$ ist.

   Beweis.
   - **Testfrage.** Warum gilt $K^A \in \mathbf{NP}^A$?
   - **Testfrage.** Warum gilt $K^A \in \Sigma_{n+1}$?
   - **Testfrage.** Warum gilt, dass K^A $\mathbf{NP}^A$-schwer für $\leq_{\mathrm{m}}^{\mathrm{p}}$ ist?
   - **Testfrage.** Warum gilt $\mathbf{NP}^A = \mathbf{NP}^{\Sigma_n} = \Sigma_{n+1}$?
   - **Testfrage.** Beweisen sie nun mit obigen Zutaten die Proposition!

4. **Korollar.** Für alle $n \geq 1$ gilt, dass K^n Σ_n-vollständig für $\leq_{\mathrm{m}}^{\mathrm{p}}$ ist.

Kapitelzusammenfassung

1. Wir haben eine aufsteigende Hierarchie von Polynomialzeit-Komplexitätsklassen definiert.
2. Ein verallgemeinerter Projektionssatz charakterisiert die k-ten Stufen der Hierarchie durch k-fach abwechselnde Quantoren.
3. Es folgt, dass die Hierarchie entweder unendlich aufsteigend ist, oder ab einer bestimmten Stufe kollabiert. Falls $\mathbf{P} = \mathbf{NP}$ gilt, so kollabiert sie bereits auf der untersten Stufe.
4. Wir haben für jede Stufe ein vollständiges Problem identifiziert.

6 Kolmogorovkomplexität

Dyadisches Einzeilen-Tetris

1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.

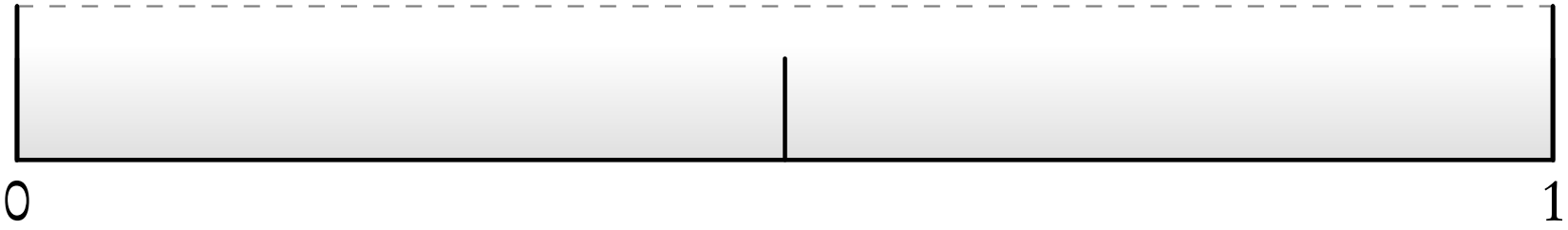
Dyadisches Einzeilen-Tetris

1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.

Dyadisches Einzeilen-Tetris

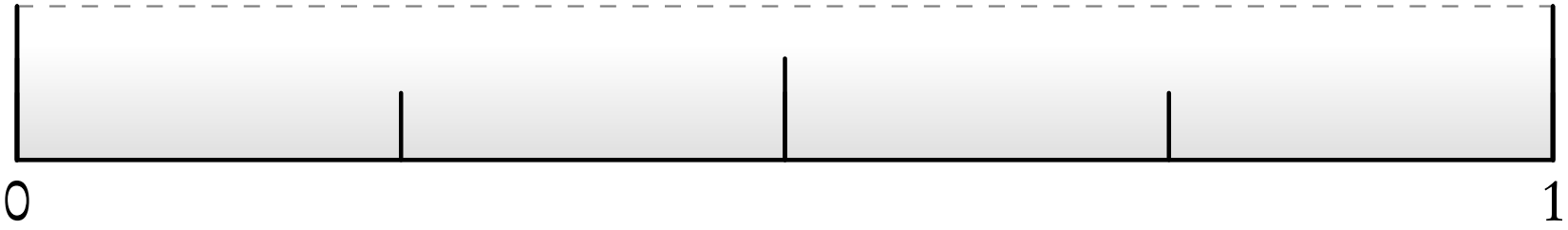
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.

Dyadisches Einzeilen-Tetris



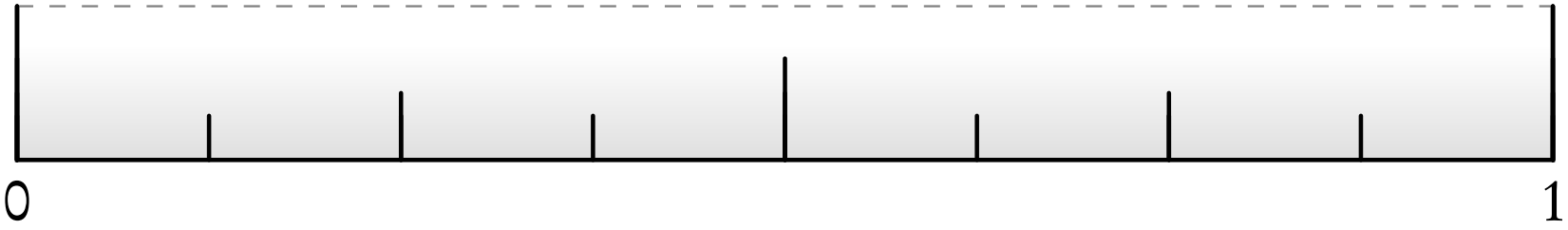
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.

Dyadisches Einzeilen-Tetris



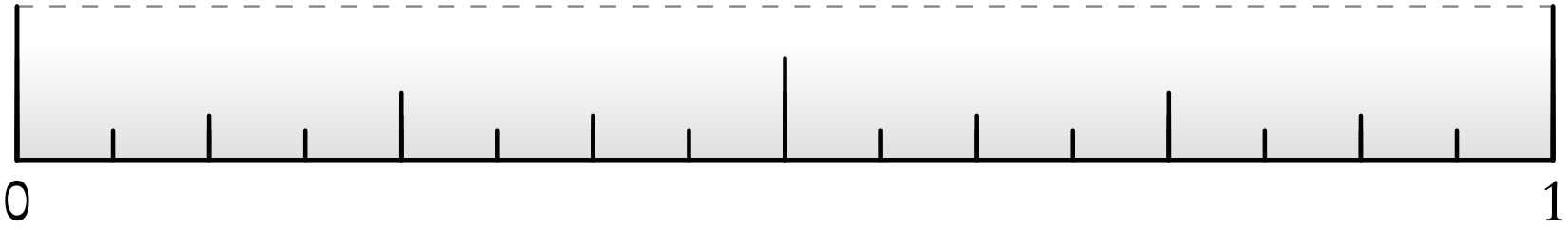
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



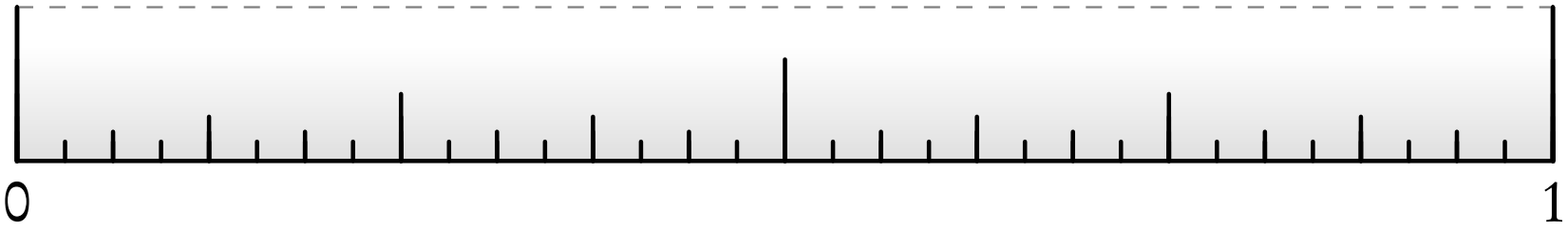
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



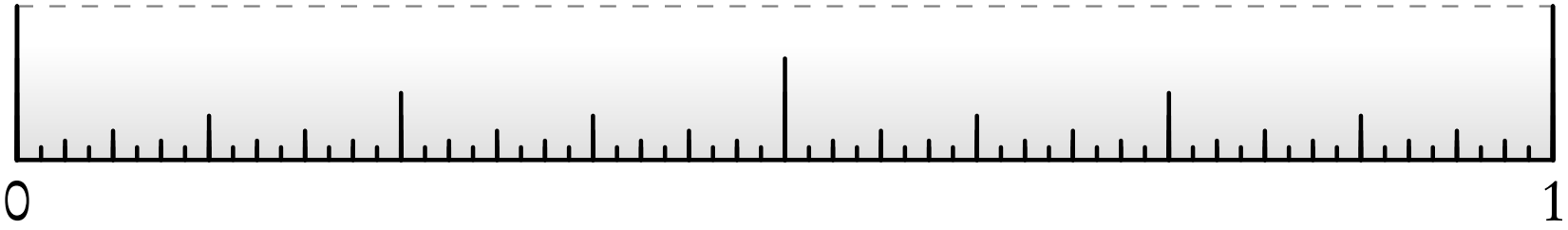
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



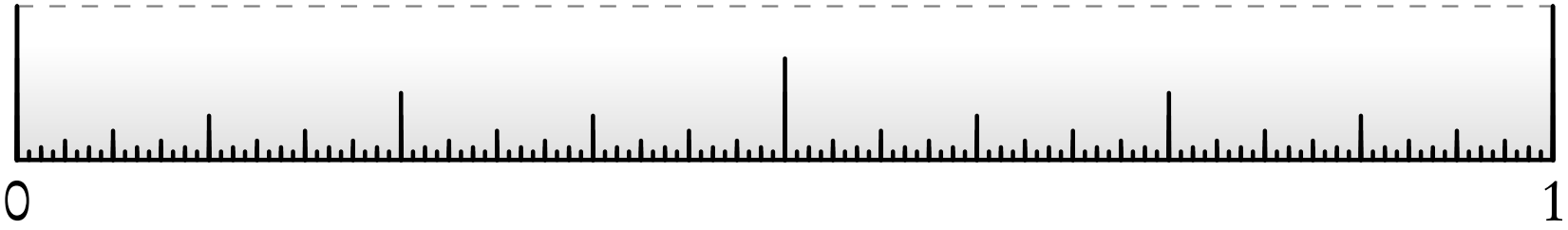
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



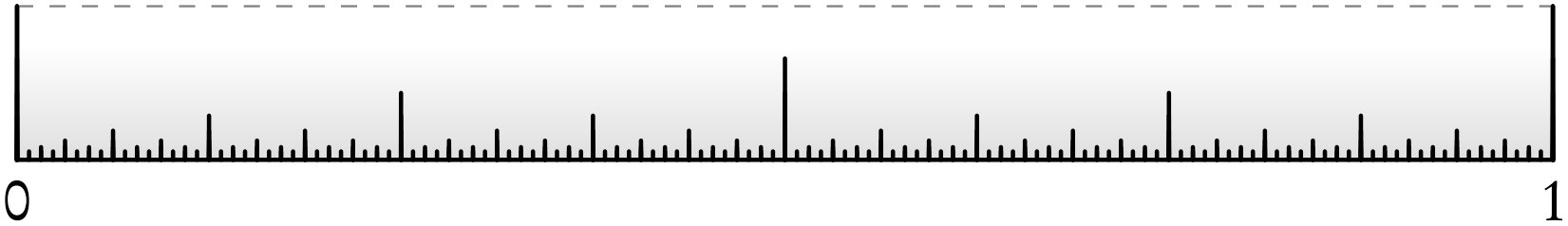
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



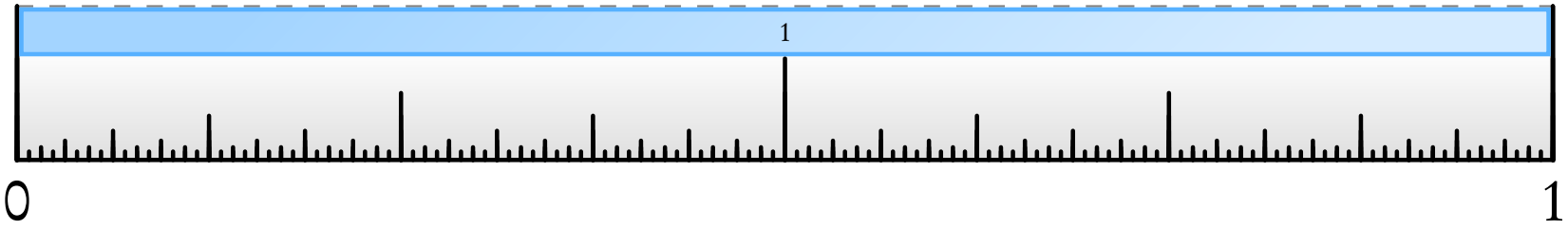
1. Wir untersuchen erst ein Spiel. Erst danach definieren wir die Kolmogorovkomplexität und erklären den Zusammenhang.
2. Das Spielfeld ist eine breite, oben offene Kiste.
3. Die Breite wird mit dem Einheitsintervall identifiziert.
4. Vom Boden der Kiste stehen an den dyadischen Zahlen unterschiedlich lange Spitzen nach oben.
5. Je „feiner“ die dyadische Zahl ist, desto kürzer.

Dyadisches Einzeilen-Tetris



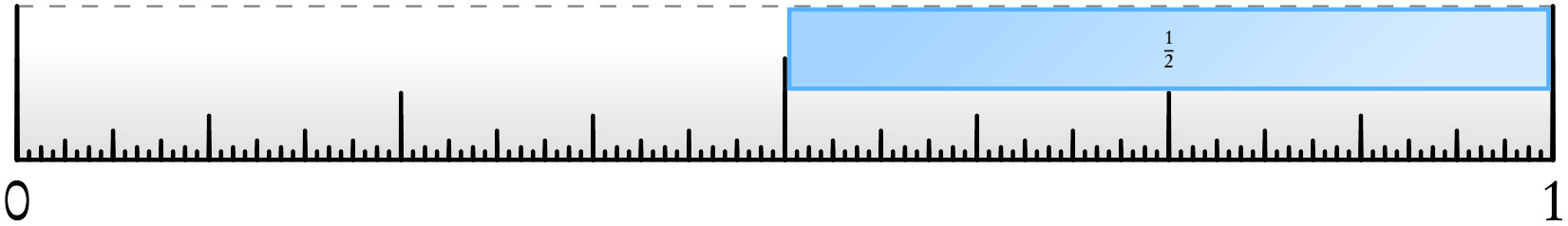
1. Die Breiten der Spielsteine sind negative Potenzen von 2.

Dyadisches Einzeilen-Tetris



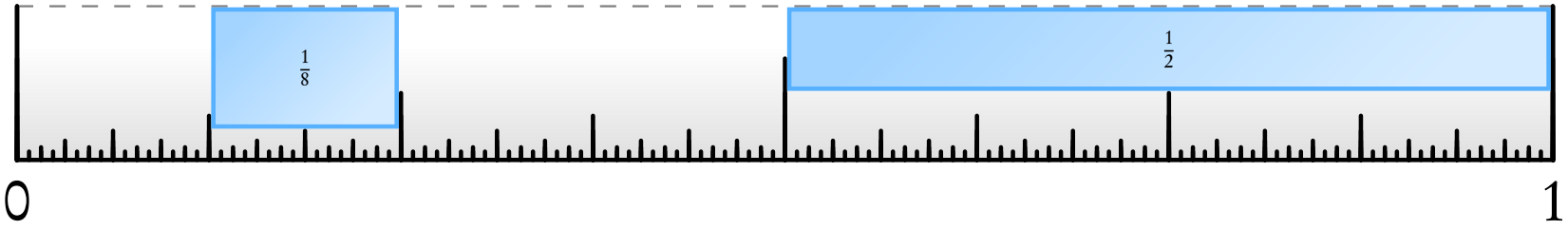
1. Die Breiten der Spielsteine sind negative Potenzen von 2.

Dyadisches Einzeilen-Tetris



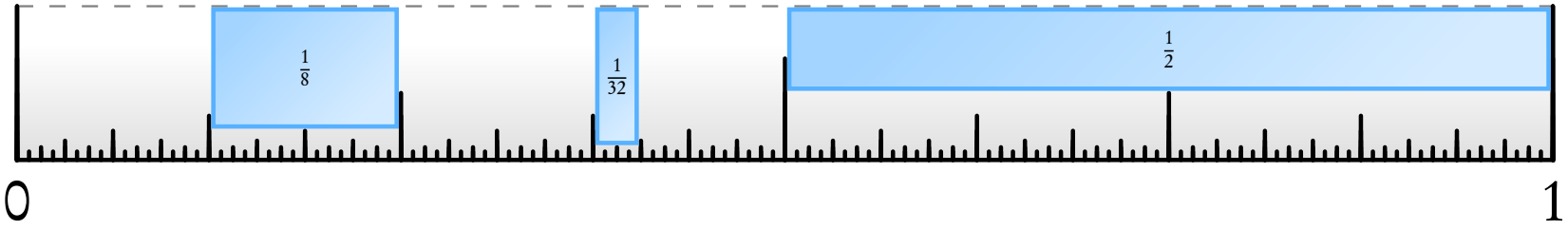
1. Die Breiten der Spielsteine sind negative Potenzen von 2.

Dyadisches Einzeilen-Tetris



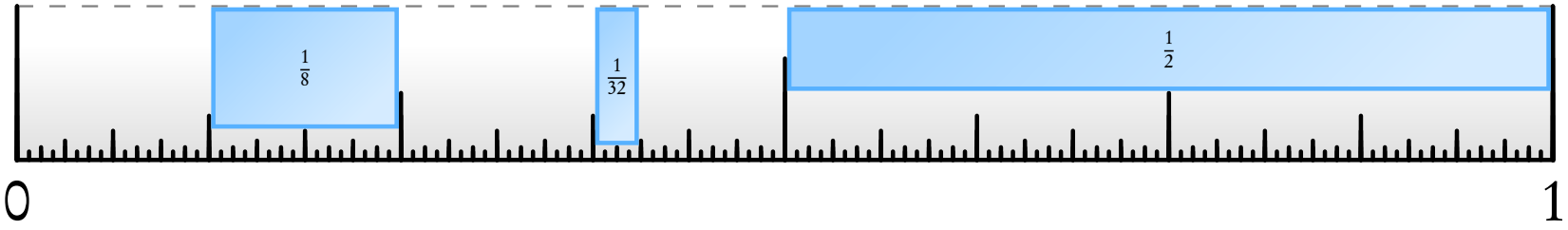
1. Die Breiten der Spielsteine sind negative Potenzen von 2.

Dyadisches Einzeilen-Tetris



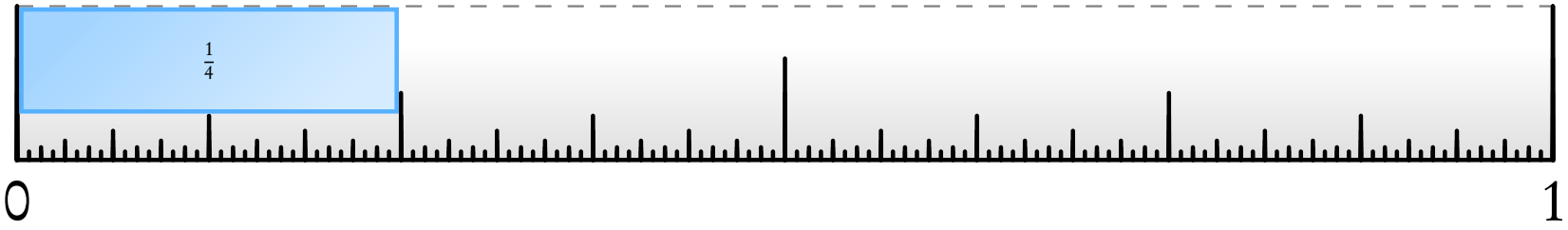
1. Die Breiten der Spielsteine sind negative Potenzen von 2.

Dyadisches Einzeilen-Tetris



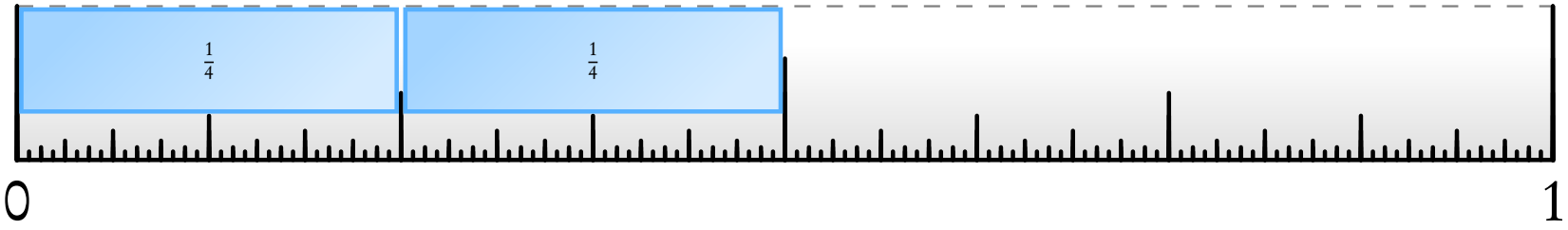
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.

Dyadisches Einzeilen-Tetris



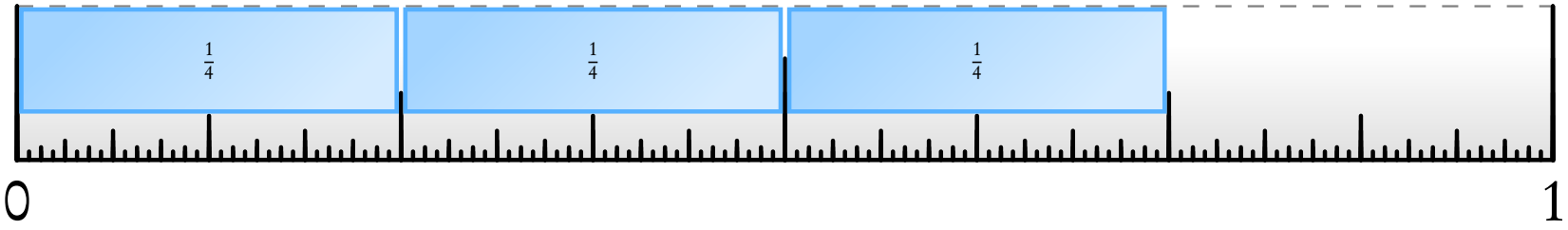
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.

Dyadisches Einzeilen-Tetris



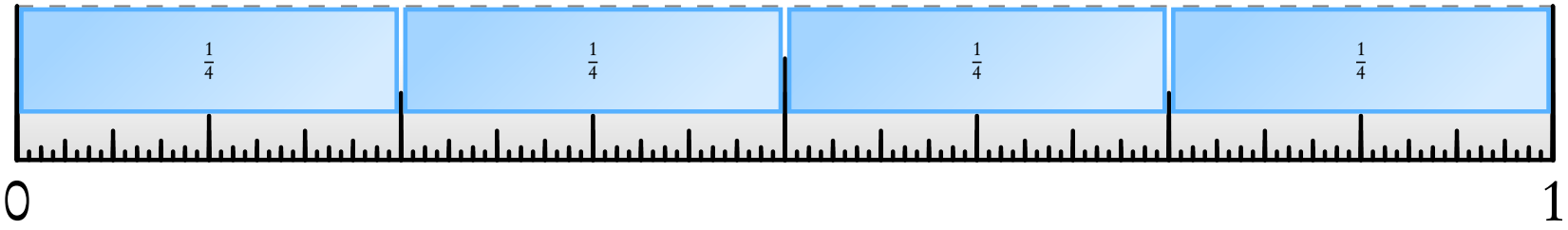
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.

Dyadisches Einzeilen-Tetris



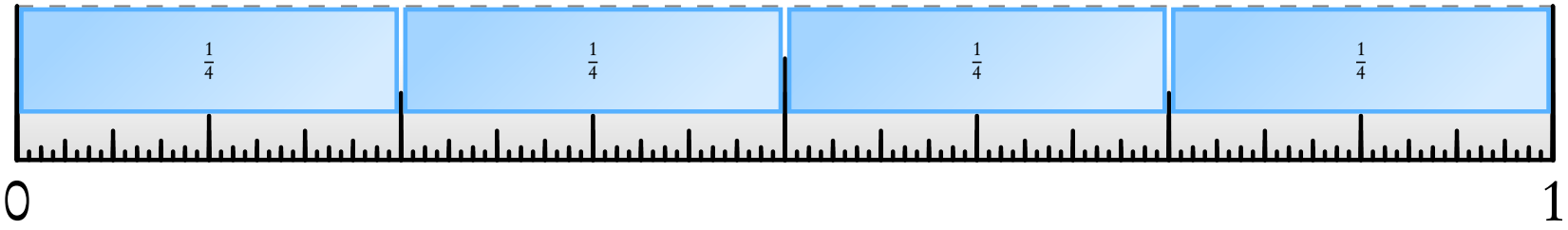
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.

Dyadisches Einzeilen-Tetris



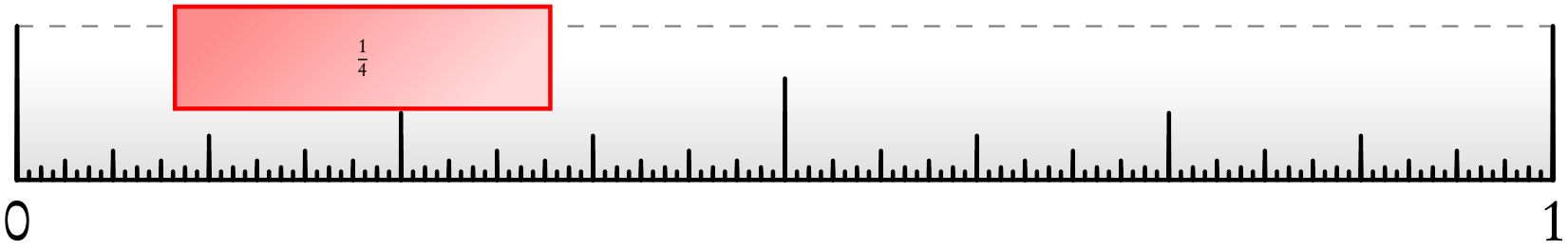
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.

Dyadisches Einzeilen-Tetris



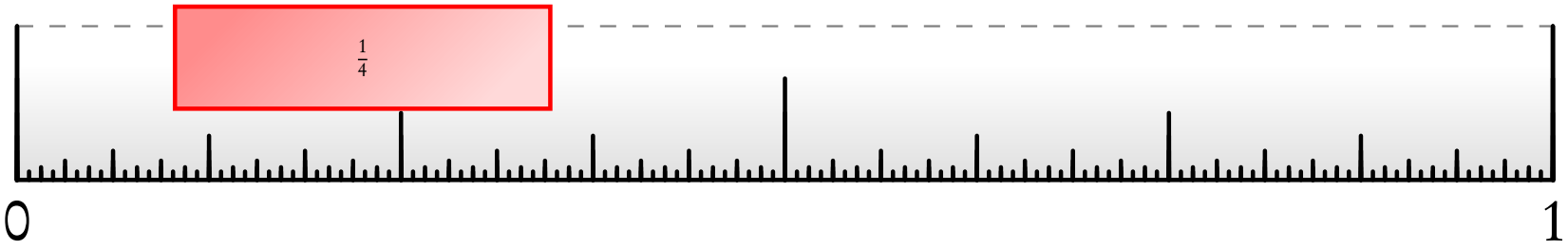
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.
3. An allen anderen Stellen würden sie auf einer der Spitzen landen, also oben herausstehen, und wir hätten verloren.

Dyadisches Einzeilen-Tetris



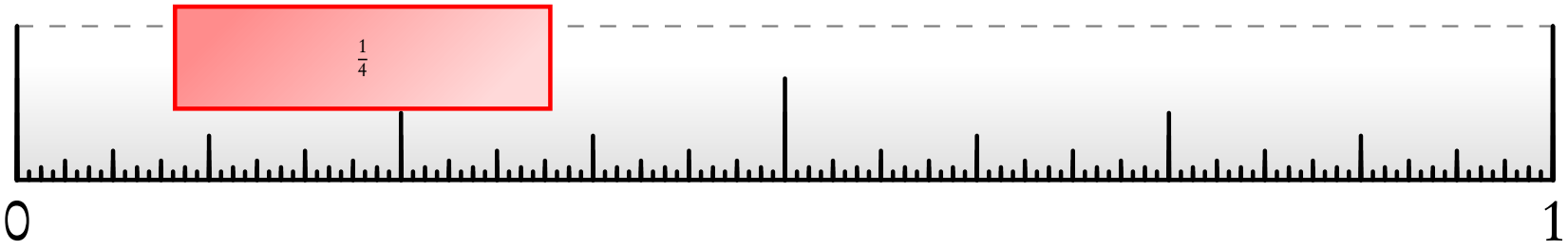
1. Die Breiten der Spielsteine sind negative Potenzen von 2.
2. Durch ihre Höhen passen sie nur an bestimmte Stellen; so gibt es für Steine der Breite $1/4$ nur vier mögliche Stellen.
3. An allen anderen Stellen würden sie auf einer der Spitzen landen, also oben herausstehen, und wir hätten verloren.

Dyadisches Einzeilen-Tetris



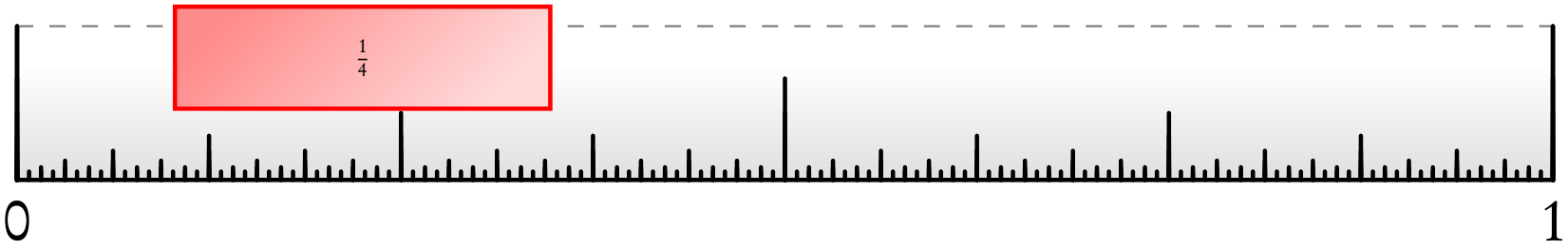
1. Wie im normalen Tetris...

Dyadisches Einzeilen-Tetris



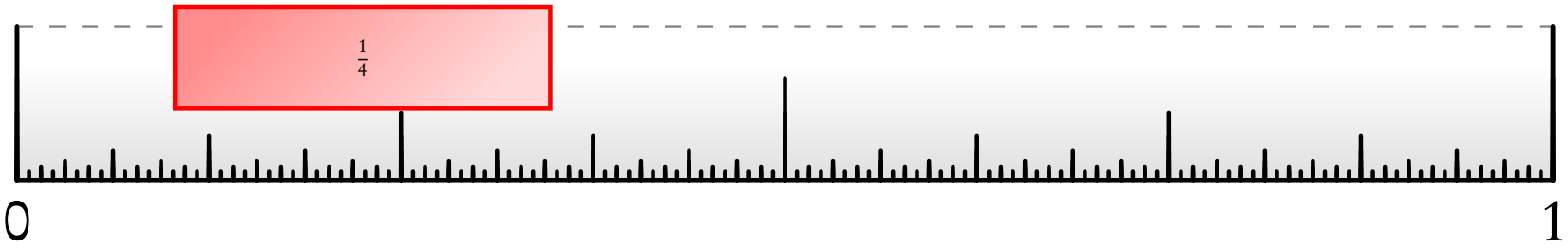
1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.

Dyadisches Einzeilen-Tetris



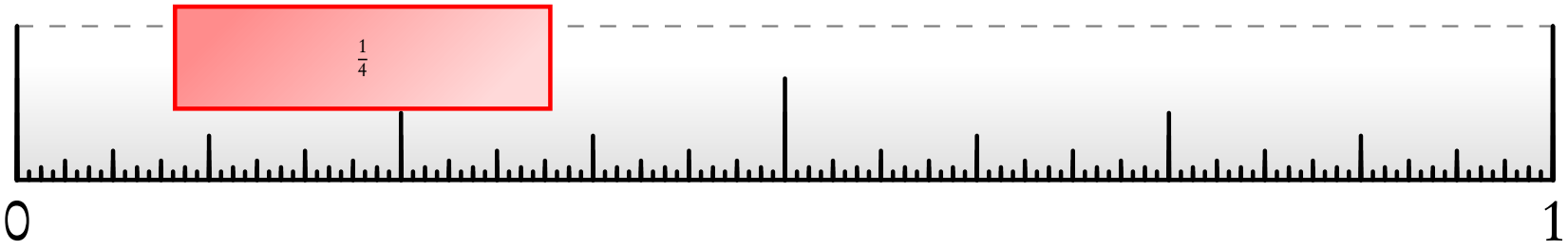
1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.

Dyadisches Einzeilen-Tetris



1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.
   - ... darf ein einmal abgelegter Stein nie wieder bewegt werden.

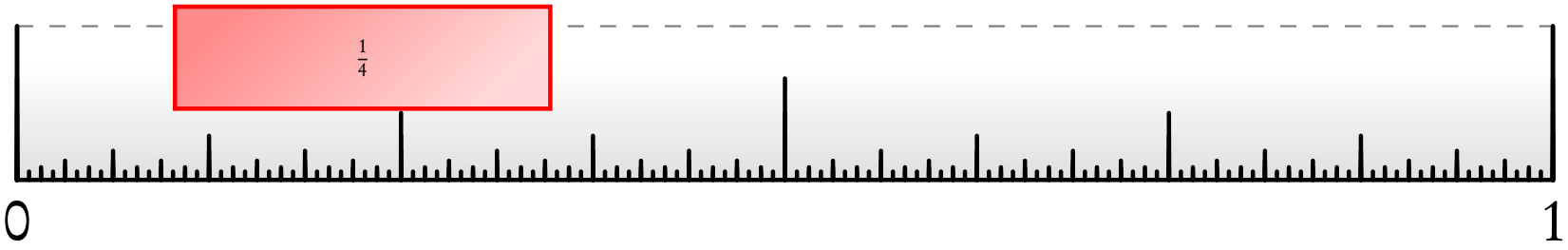
Dyadisches Einzeilen-Tetris



1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.
   - ... darf ein einmal abgelegter Stein nie wieder bewegt werden.
   - ... läuft das Spiel unendlich viele Runden.

     (Zumindest ist das der interessante Fall.)

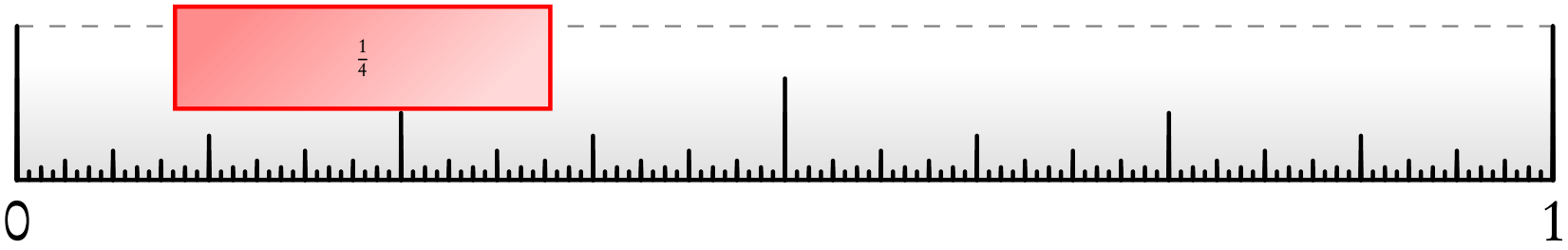
Dyadisches Einzeilen-Tetris



1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.
   - ... darf ein einmal abgelegter Stein nie wieder bewegt werden.
   - ... läuft das Spiel unendlich viele Runden.

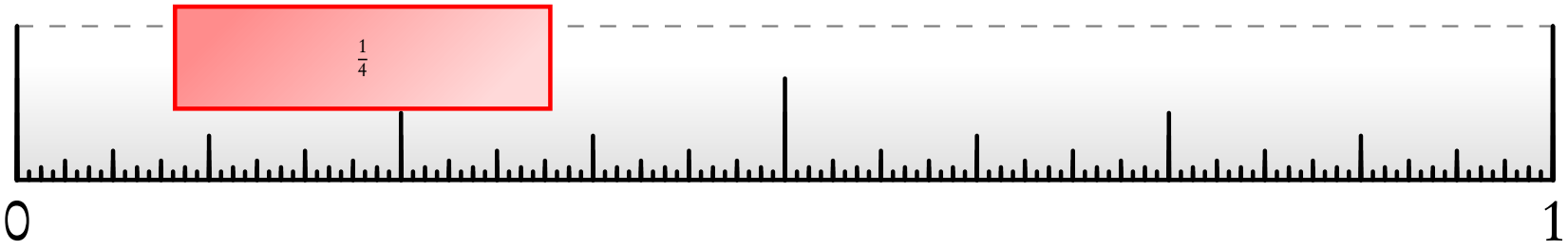
     (Zumindest ist das der interessante Fall.)
   - ... wollen wir unendlich lange überleben.

Dyadisches Einzeilen-Tetris



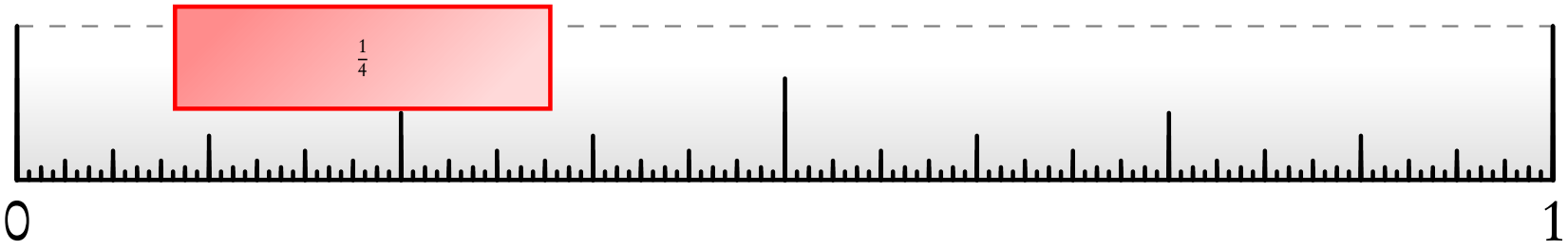
1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.
   - ... darf ein einmal abgelegter Stein nie wieder bewegt werden.
   - ... läuft das Spiel unendlich viele Runden.
     (Zumindest ist das der interessante Fall.)
   - ... wollen wir unendlich lange überleben.
2. **Testfrage.** Damit eine Chance besteht, muss der Gegner eine triviale Einschränkung bezüglich der Breiten beachten. Welche?

Dyadisches Einzeilen-Tetris



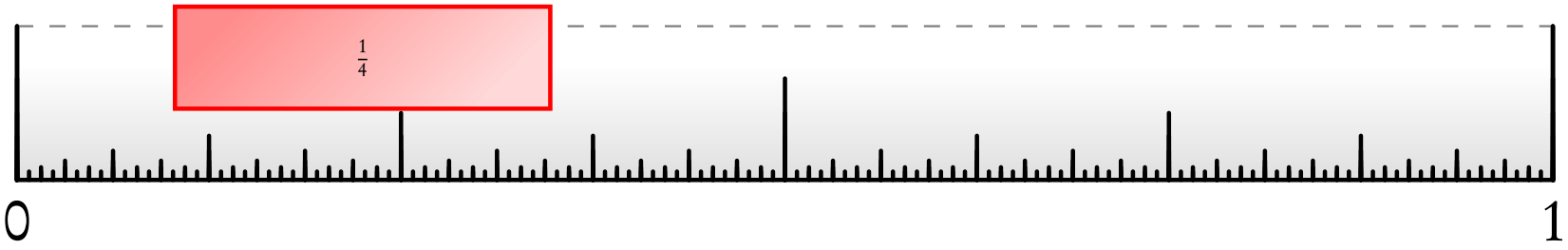
1. Wie im normalen Tetris...
   - ... wird uns ein Gegner nach und nach Steine liefern.
   - ... müssen wir uns sofort überlegen, wo wir sie ablegen wollen.
   - ... darf ein einmal abgelegter Stein nie wieder bewegt werden.
   - ... läuft das Spiel unendlich viele Runden.
     (Zumindest ist das der interessante Fall.)
   - ... wollen wir unendlich lange überleben.
2. **Testfrage.** Damit eine Chance besteht, muss der Gegner eine triviale Einschränkung bezüglich der Breiten beachten. Welche?
3. **Antwort.** Die Summe der Breiten der unendlich vielen Spielsteine, die er ausgibt, darf 1 nicht überschreiten.

Dyadisches Einzeilen-Tetris



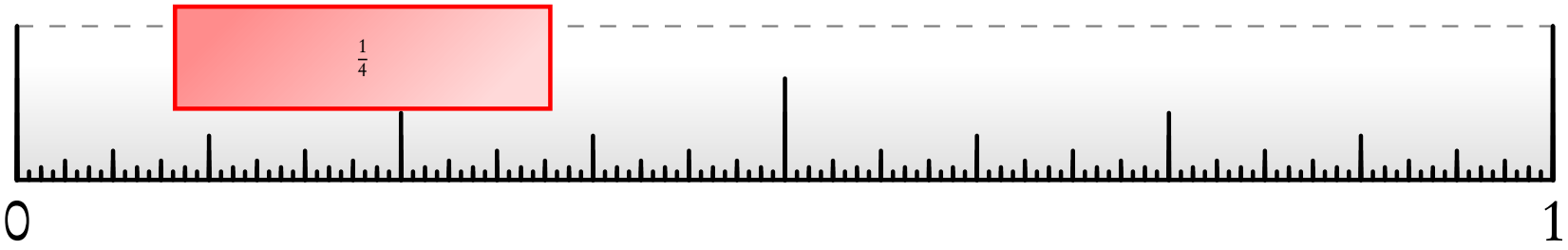
1. Nehmen wir also an, er halte sich an diese triviale Schranke.

Dyadisches Einzeilen-Tetris



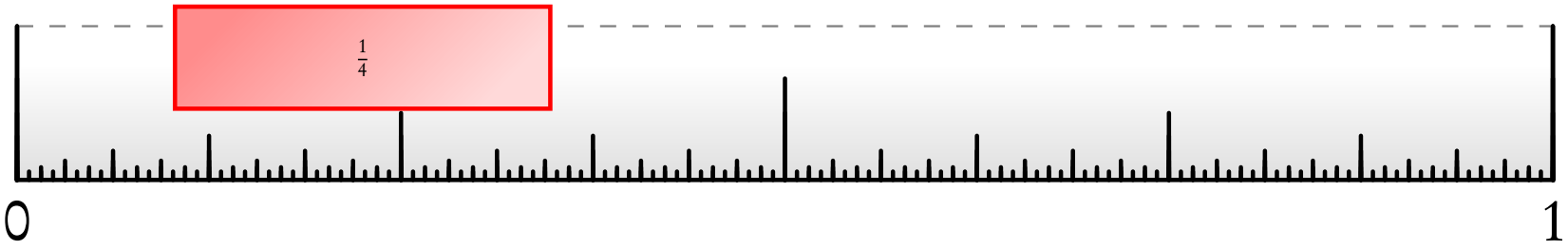
1. Nehmen wir also an, er halte sich an diese triviale Schranke.
2. Kann er uns eine Folge von Steinen geben, die wir nicht unterbekommen können?

Dyadisches Einzeilen-Tetris



1. Nehmen wir also an, er halte sich an diese triviale Schranke.
2. Kann er uns eine Folge von Steinen geben, die wir nicht unterbekommen können?
3. Hängt das von der Reihenfolge ab, in der er uns die Steine gibt?

Dyadisches Einzeilen-Tetris



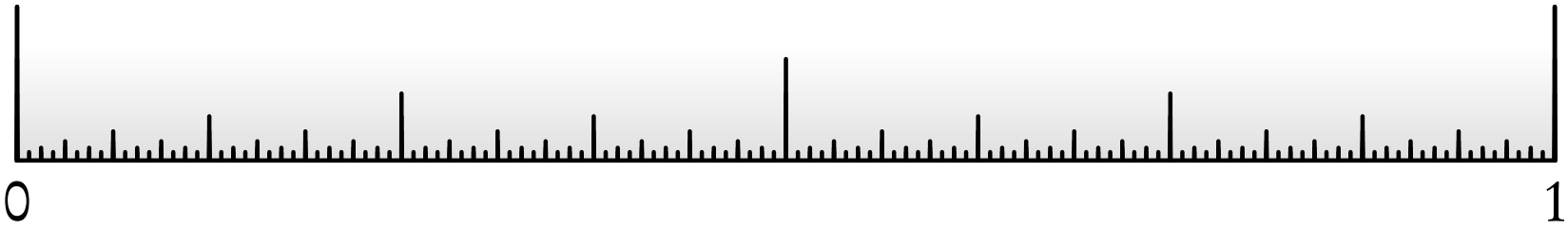
1. Nehmen wir also an, er halte sich an diese triviale Schranke.
2. Kann er uns eine Folge von Steinen geben, die wir nicht unterbekommen können?
3. Hängt das von der Reihenfolge ab, in der er uns die Steine gibt?
4. Hängt es von unserer Spielstrategie ab?

Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.

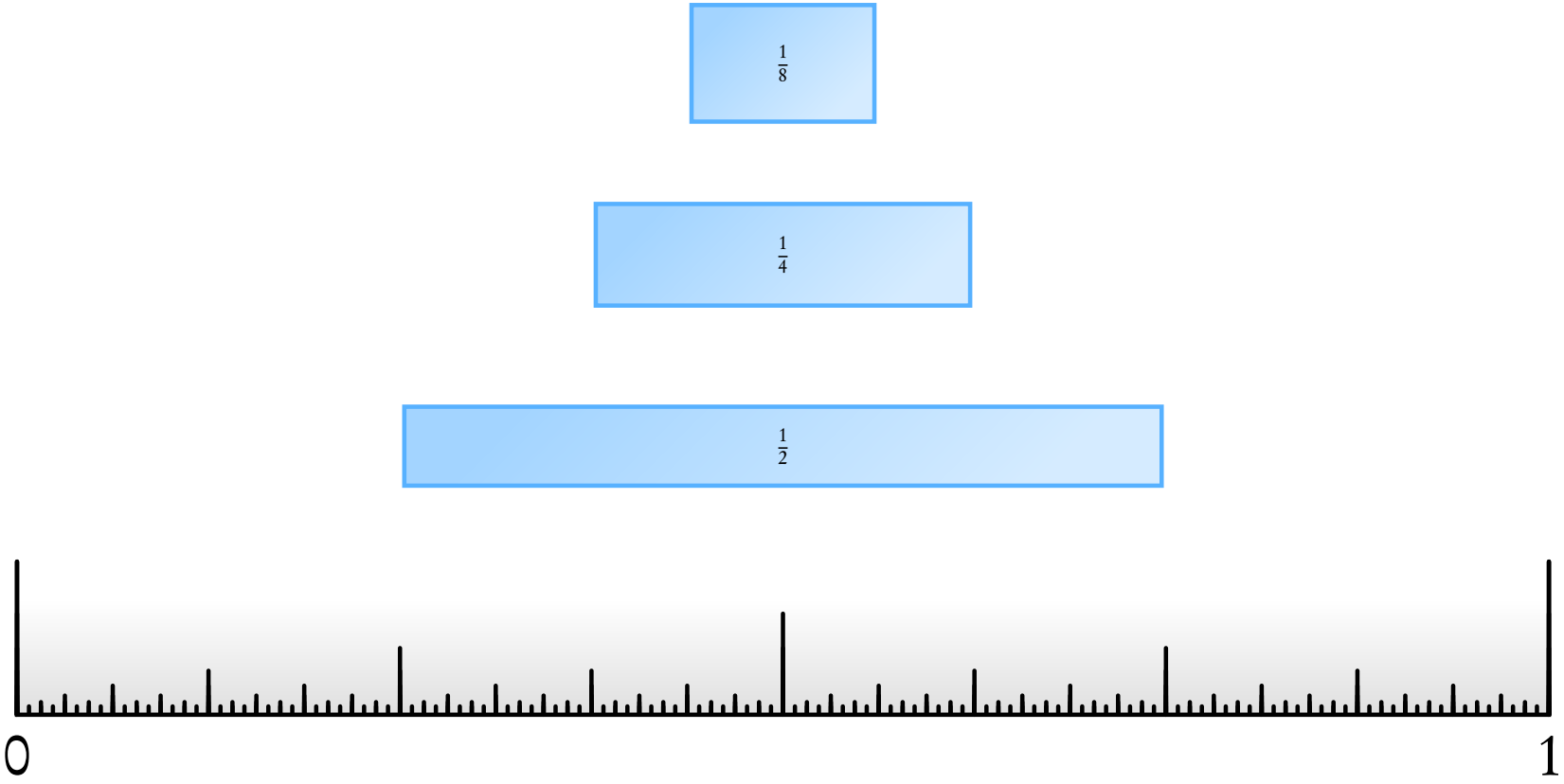
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



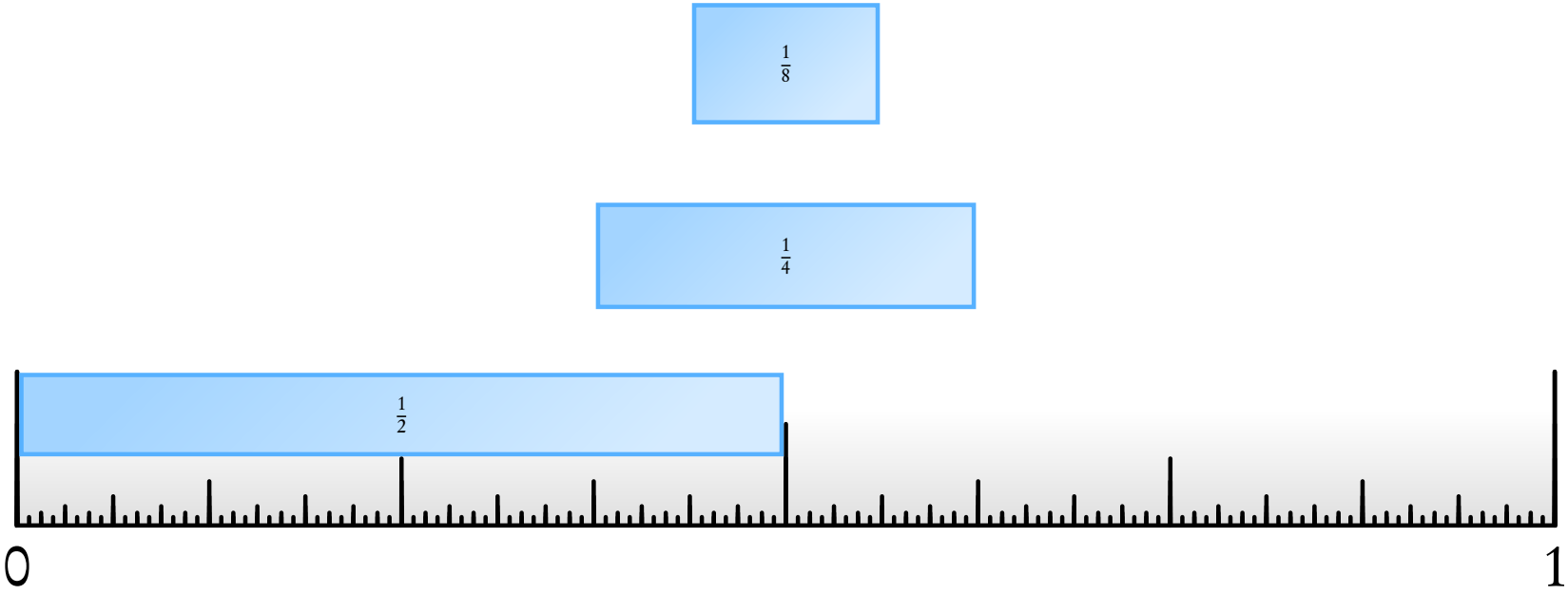
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



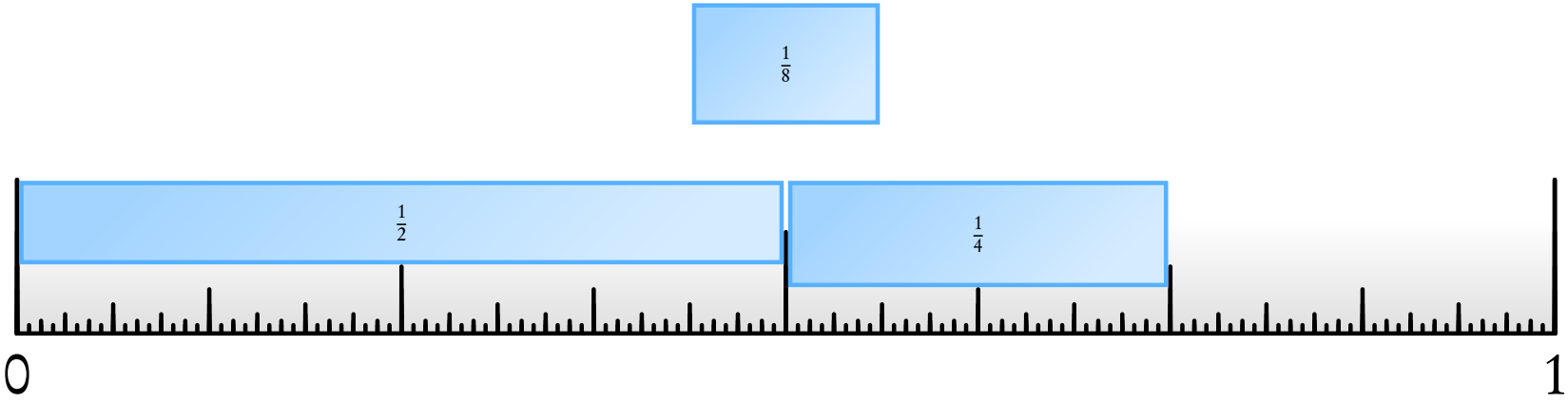
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



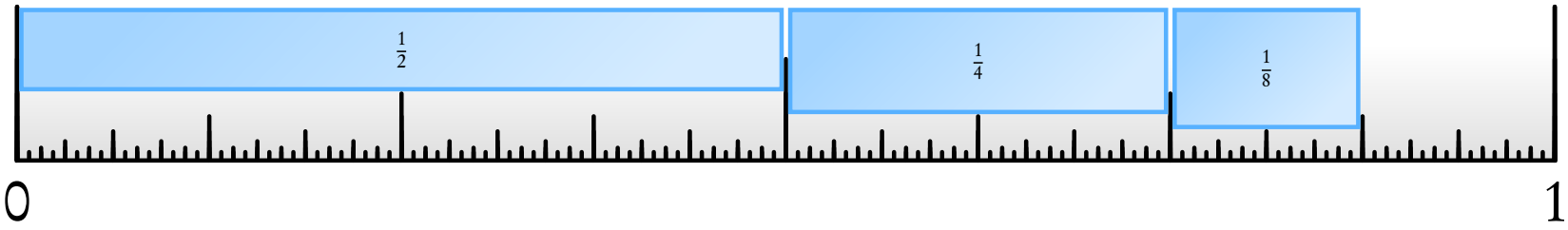
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



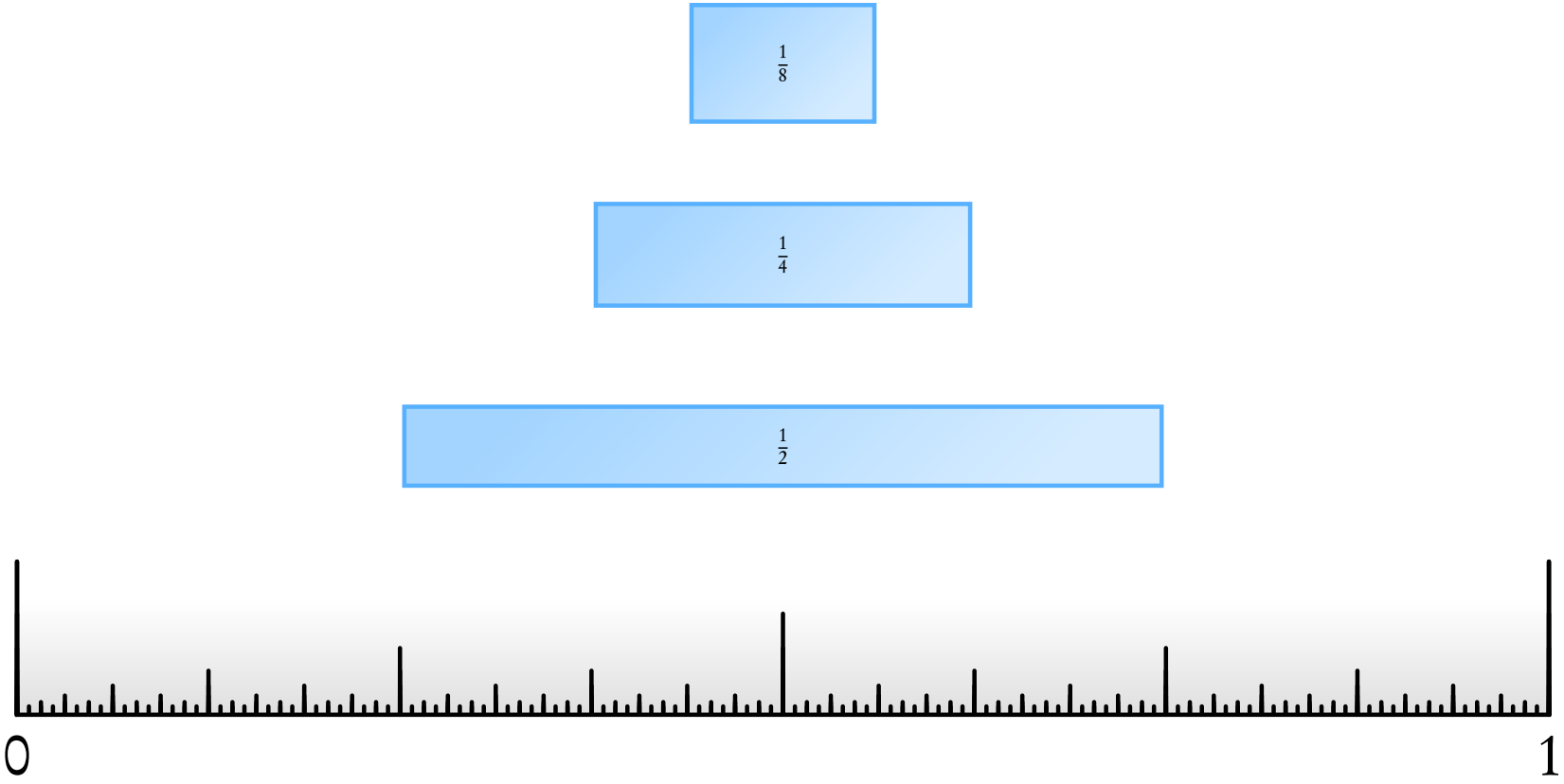
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



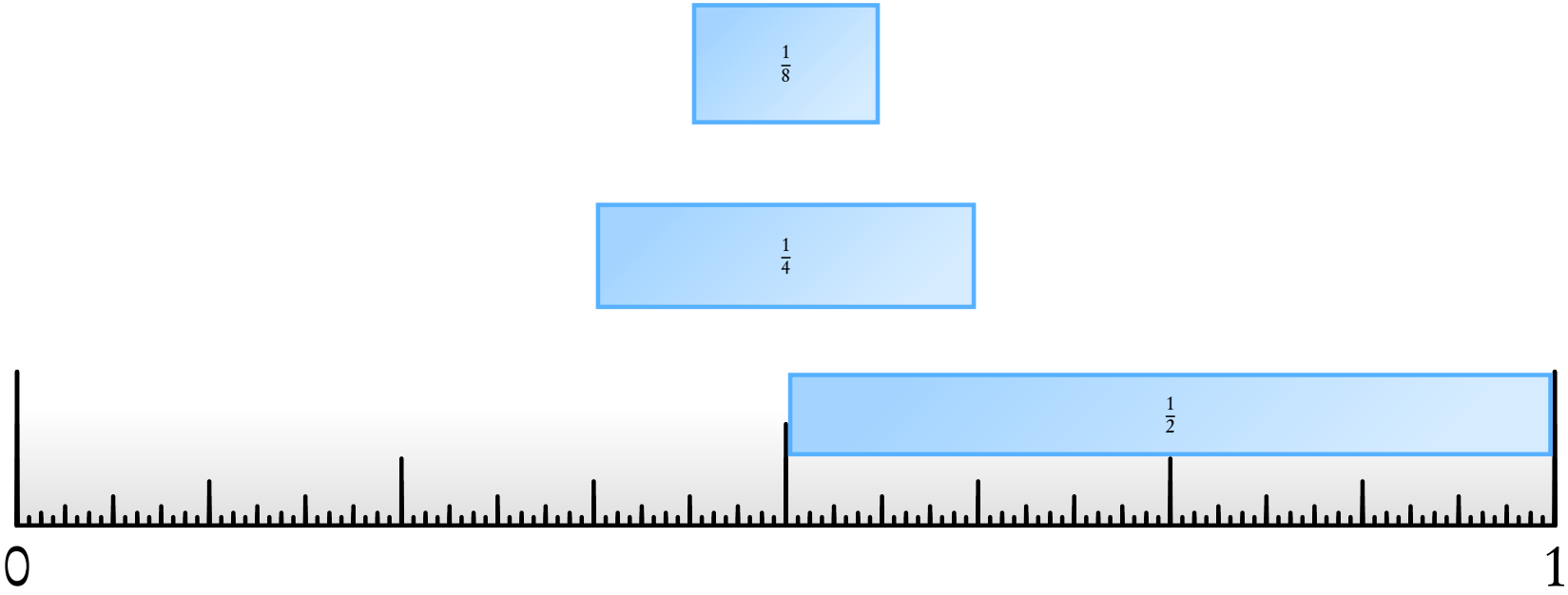
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



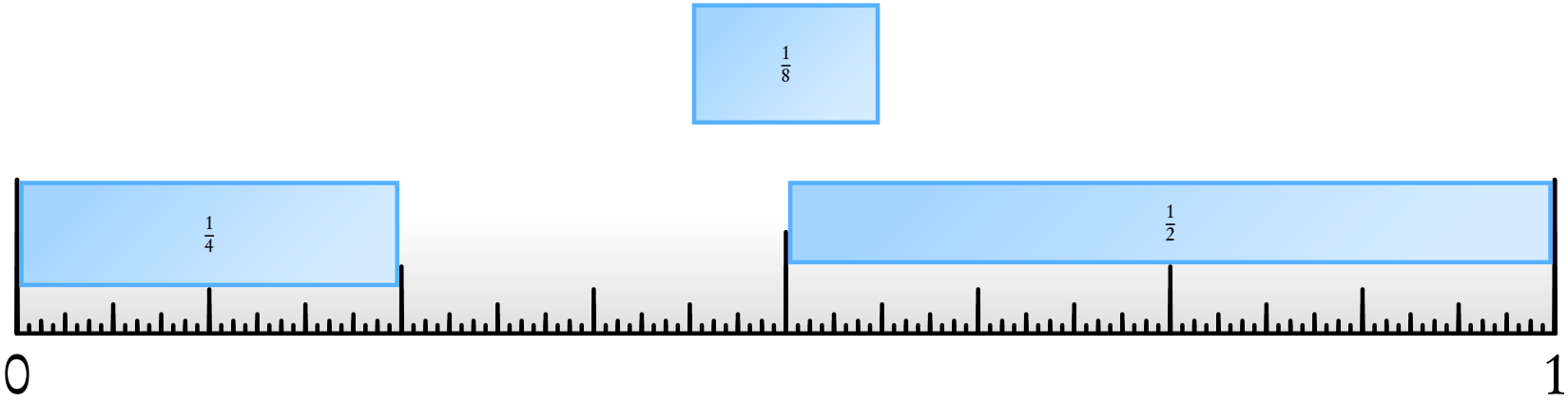
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



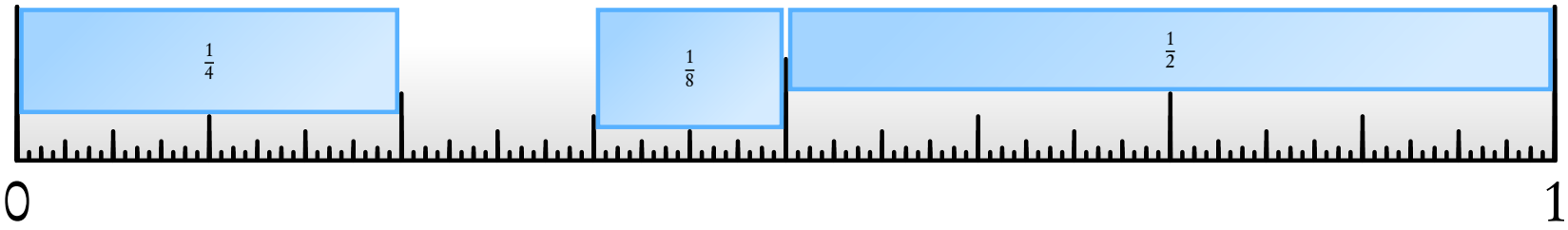
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



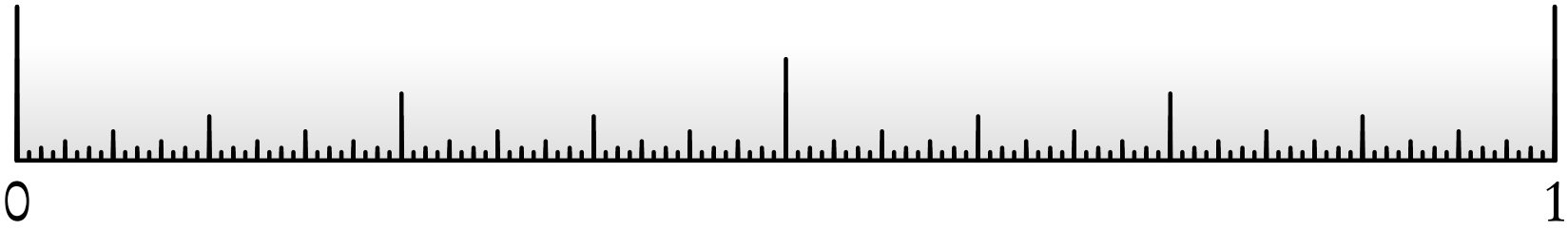
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
2. Bei dieser Reihenfolge können wir gar nichts falsch machen:



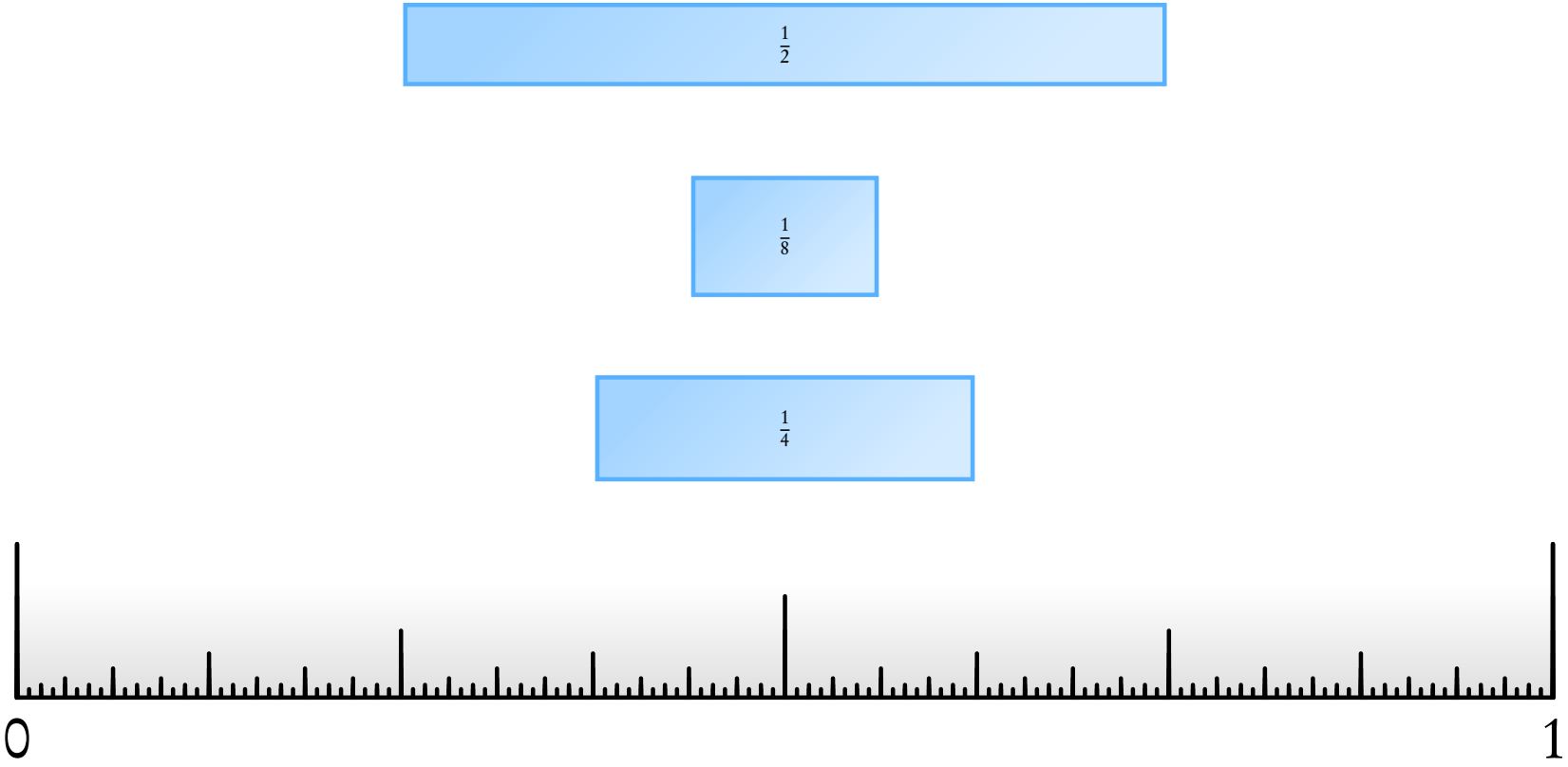
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



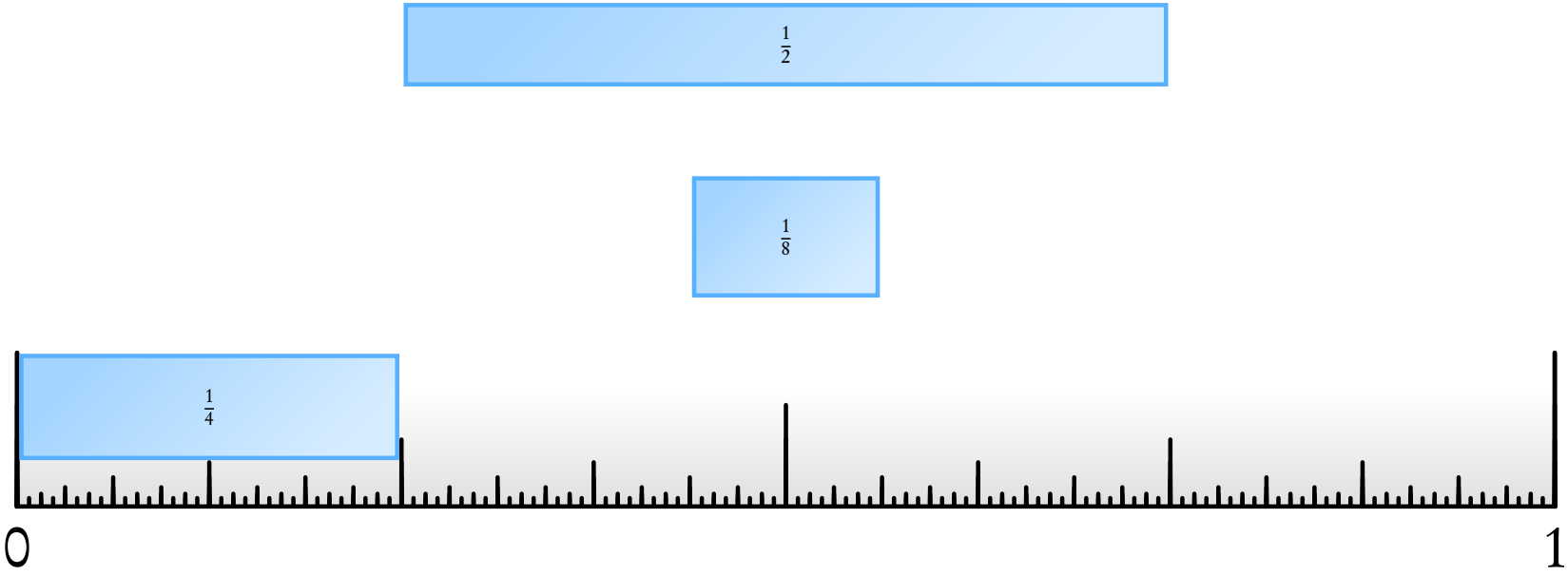
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



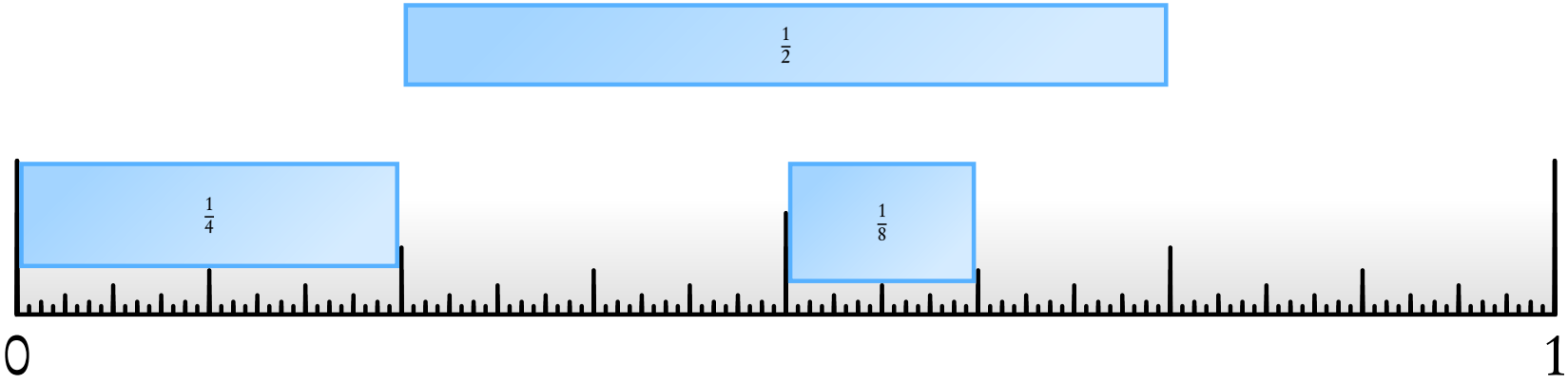
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



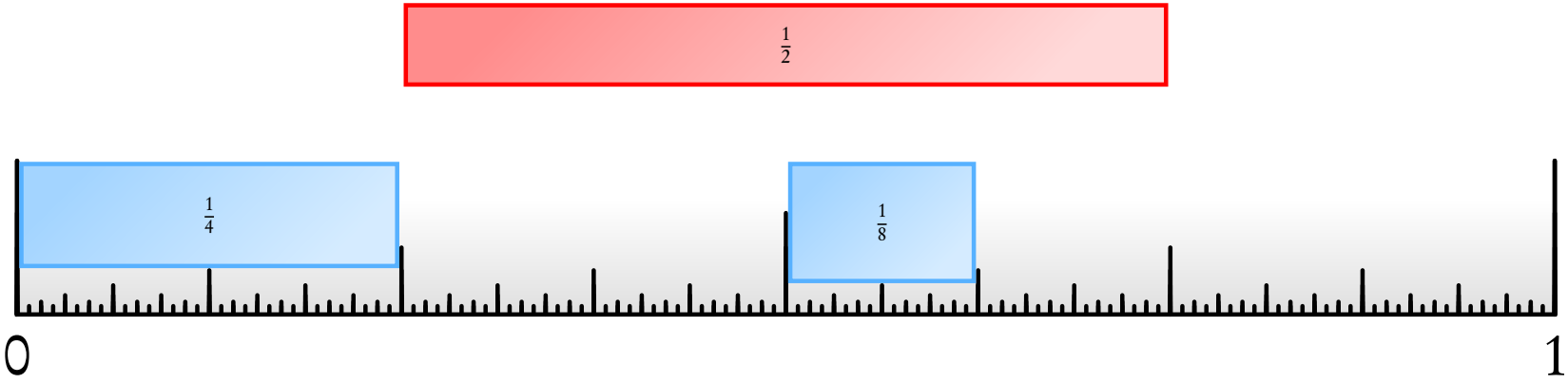
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



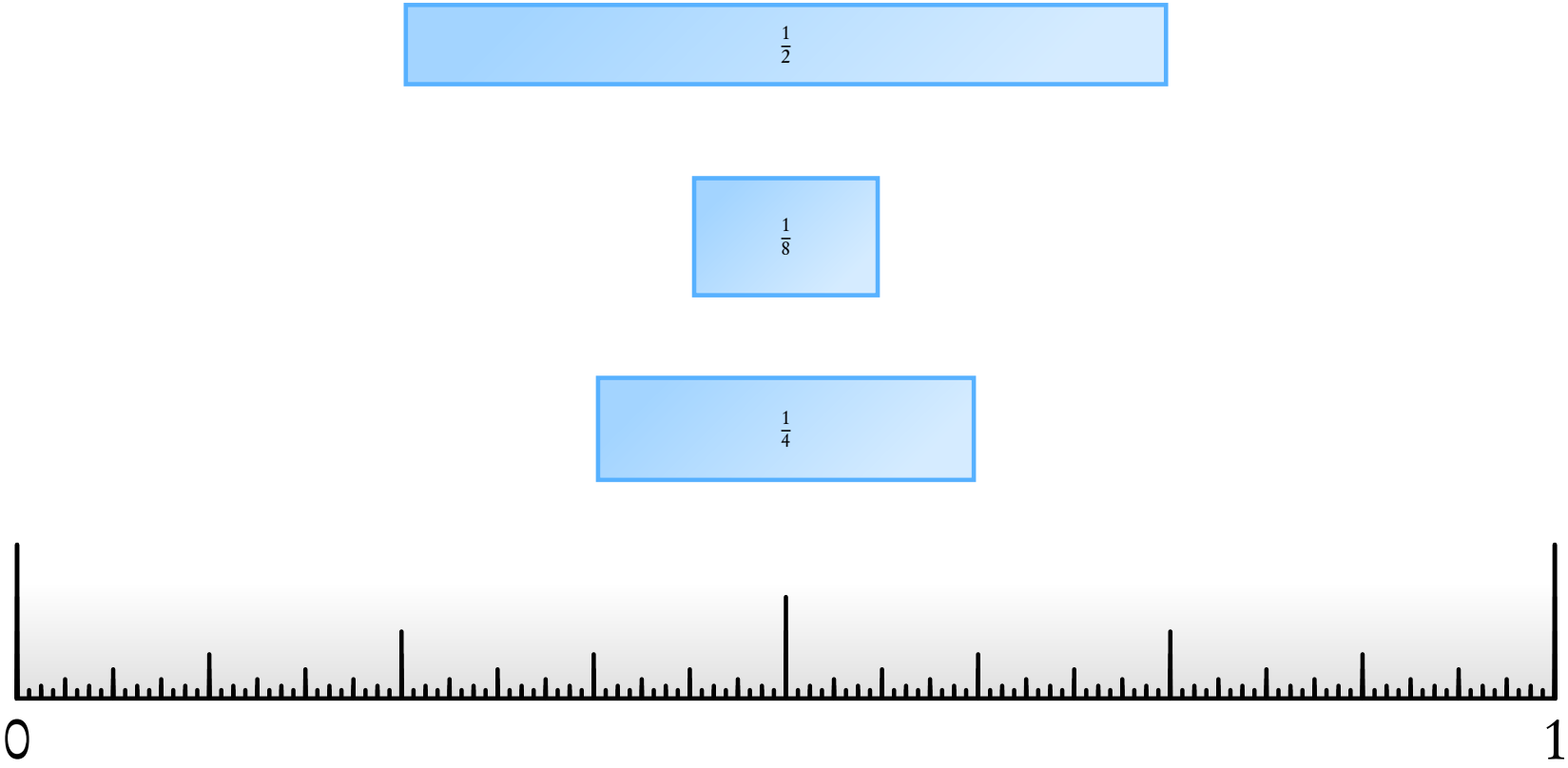
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



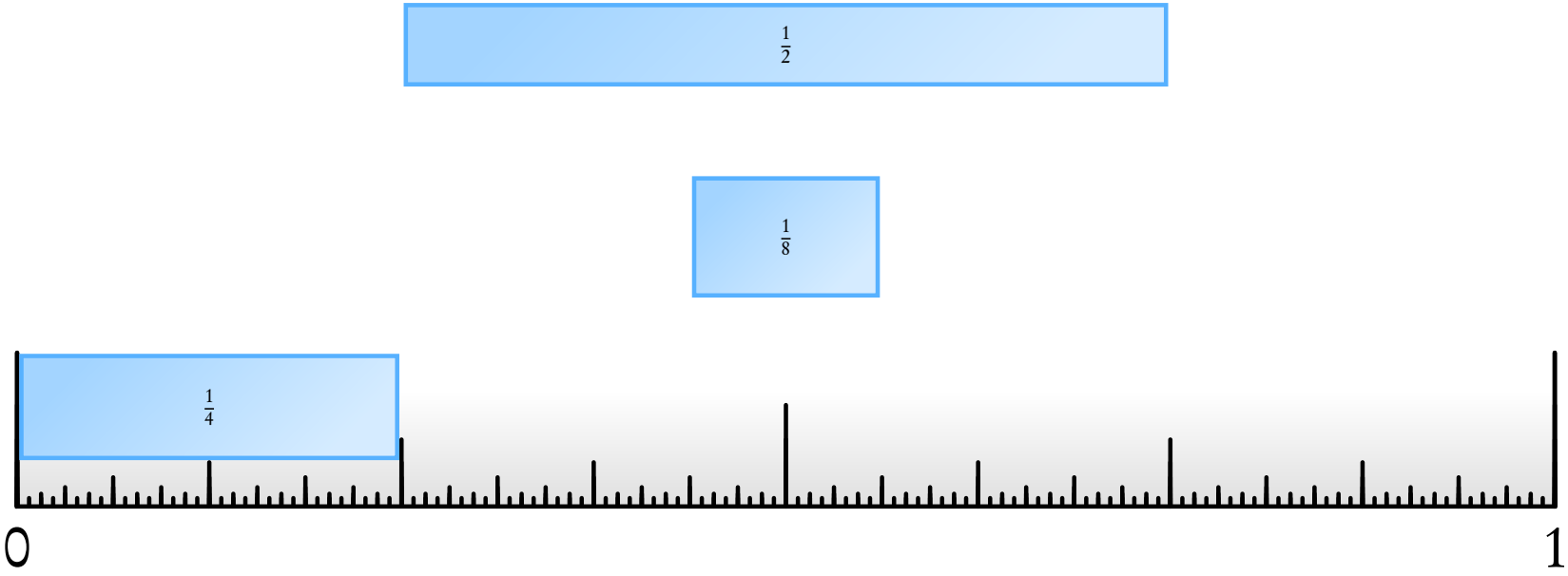
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



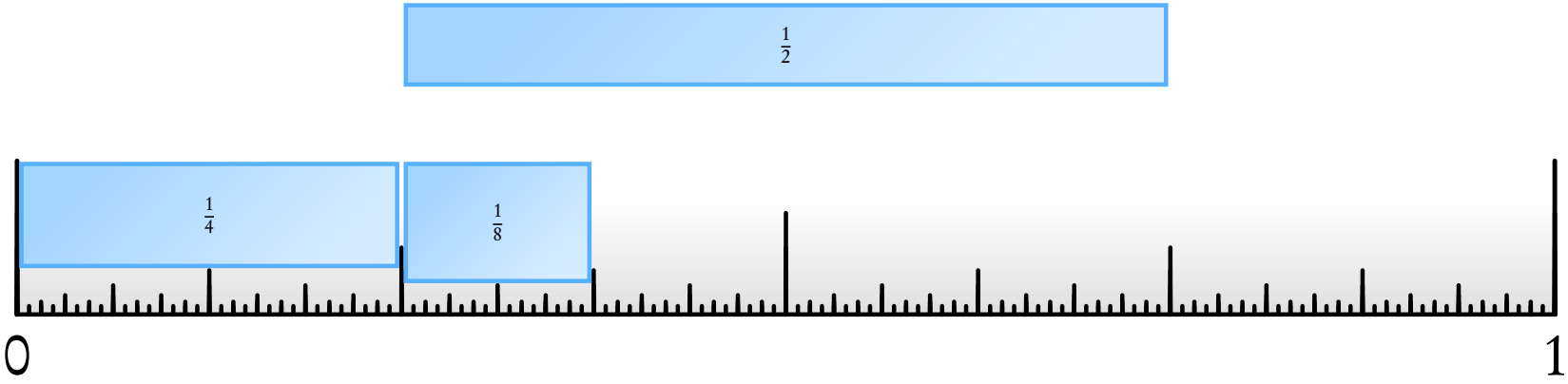
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



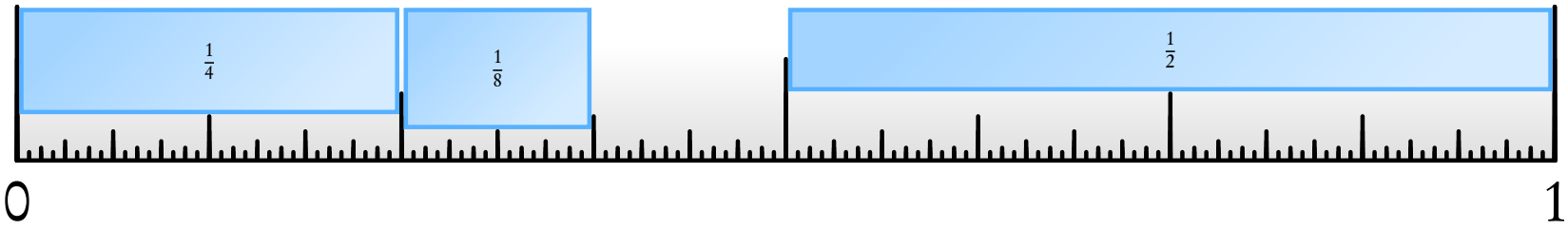
Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



Dumme Strategien

1. Die Antwort auf diese Fragen ist ganz klar „ja“.
3. In diesem Fall hingegen kommt es auf unsere Strategie an:



Kluge Strategien

1. Folgen wir einer dummen Strategie, sind wir leicht zu besiegen.
2. Nehmen wir also an, wir hätten eine bestmögliche Strategie.
   (Egal, was das bedeuten mag, und angenommen, dass eine solche überhaupt existiert.)
3. Nehmen wir weiter an, der Gegner sei allwissend. Er kann in unser Gehirn sehen, kennt also unsere Strategie und weiß schon im Voraus, wie wir auf seine Züge reagieren werden.
4. Kann er mit diesem Wissen erzwingen, dass wir verlieren, obwohl unsere Strategie so gut ist?

Kluge Strategien

1. Folgen wir einer dummen Strategie, sind wir leicht zu besiegen.
2. Nehmen wir also an, wir hätten eine bestmögliche Strategie.
   (Egal, was das bedeuten mag, und angenommen, dass eine solche überhaupt existiert.)
3. Nehmen wir weiter an, der Gegner sei allwissend. Er kann in unser Gehirn sehen, kennt also unsere Strategie und weiß schon im Voraus, wie wir auf seine Züge reagieren werden.
4. Kann er mit diesem Wissen erzwingen, dass wir verlieren, obwohl unsere Strategie so gut ist?
5. **Satz.** Es gibt eine Spielstrategie mit der wir nie verlieren, unabhängig davon welche Spielsteine der Gegner in welcher Reihenfolge spielt — solange er die Summe 1 nicht überschreitet.
6. **Intuition.** Die triviale Schranke ist also bereits scharf.

Kluge Strategien

1. Folgen wir einer dummen Strategie, sind wir leicht zu besiegen.
2. Nehmen wir also an, wir hätten eine bestmögliche Strategie.
   (Egal, was das bedeuten mag, und angenommen, dass eine solche überhaupt existiert.)
3. Nehmen wir weiter an, der Gegner sei allwissend. Er kann in unser Gehirn sehen, kennt also unsere Strategie und weiß schon im Voraus, wie wir auf seine Züge reagieren werden.
4. Kann er mit diesem Wissen erzwingen, dass wir verlieren, obwohl unsere Strategie so gut ist?
5. **Satz.** Es gibt eine Spielstrategie mit der wir nie verlieren, unabhängig davon welche Spielsteine der Gegner in welcher Reihenfolge spielt — solange er die Summe 1 nicht überschreitet.
6. **Intuition.** Die triviale Schranke ist also bereits scharf.
7. Tatsächlich reicht eine Greedy-Strategie, aber wir müssen uns überlegen, was das hier eigentlich bedeutet.

Eine Greedy-Strategie

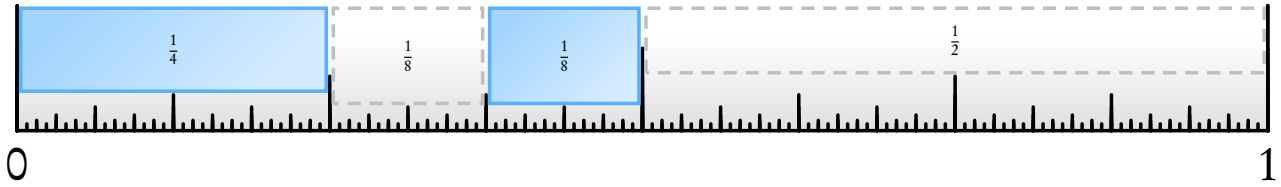
1. **Intuition.** „Greedy“ bedeutet, dass man in jedem Zug das macht, was lokal am günstigsten ist, ohne Langzeitfolgen zu beachten:

   „Mache immer den Zug mit dem sofortigen höchsten Gewinn!“

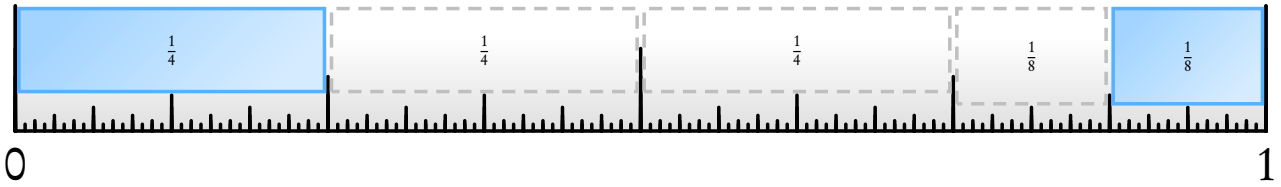
2. **In unserem Fall ist es aber umgekehrt:** Durch das Platzieren eines Steins werden bestimmte Plätze in unserer Kiste für bestimmte Steine unbenutzbar. Wir sollten also „geizig“ sein und versuchen, in jedem Zug den „wertlosesten“ Platz zu opfern.
3. **Frage.** Welche Platzierungen „kosten“ also am wenigsten?
4. **Testfrage.** Haben Sie eine Idee?

Eine Greedy-Strategie

1. **Intuition.** Ein großes freies Intervall ist wertvoller...



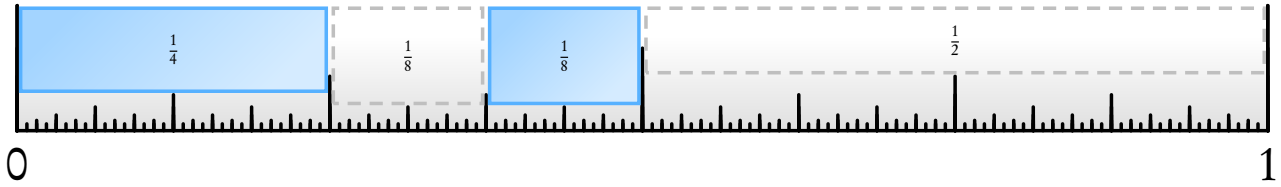
... als mehrere kleine, selbst wenn die Summen identisch sind.



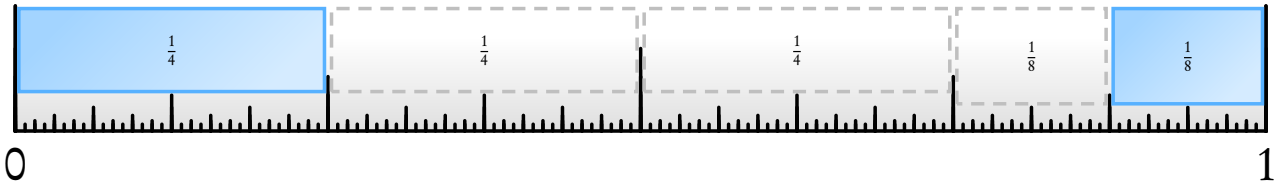
Denn ein großes kann notfalls immer noch einen kleinen Spielstein aufnehmen; umgekehrt geht dies jedoch nicht.

Eine Greedy-Strategie

1. **Intuition.** Ein großes freies Intervall ist wertvoller...



... als mehrere kleine, selbst wenn die Summen identisch sind.



Denn ein großes kann notfalls immer noch einen kleinen Spielstein aufnehmen; umgekehrt geht dies jedoch nicht.

2. Die Grundidee unserer Greedy-Strategie ist also:

> „*Ein neuer Spielstein wird immer in das* kleinste *Intervall platziert, das* groß genug *ist, um ihn aufzunehmen.*“

Äquivalente Strategien

1. Die Grundidee bestimmt die Strategie noch nicht eindeutig.
   Testfrage. Warum?

Äquivalente Strategien

1. Die Grundidee bestimmt die Strategie noch nicht eindeutig.
   Testfrage. Warum?
2. **Antwort.** Man kann zwischen gleich großen Intervallen wählen.
3. Strategien, die sich nur dadurch unterscheiden, sind gleich gut.
4. Um uns den Beweis zu erleichtern, wählen wir eine eindeutig bestimmte Strategie aus dieser Menge aus.

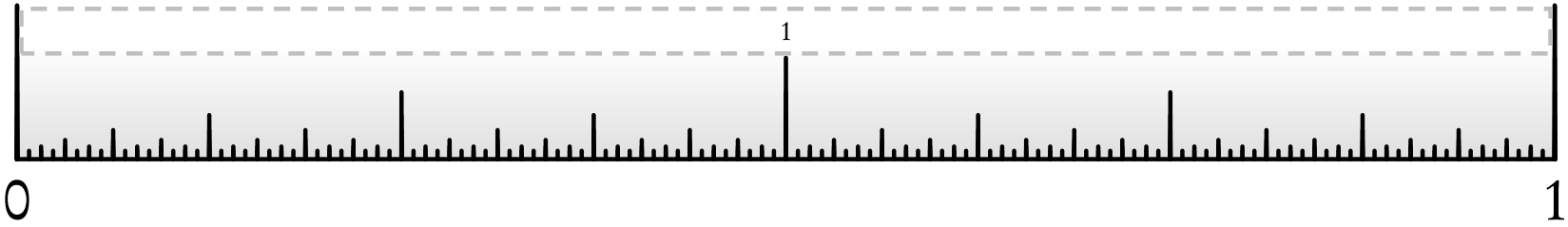
Eine feste Greedy-Strategie

1. **Behauptung.** Die Strategie

   „Jeder neue Spielstein wird so weit links wie möglich platziert.“

   gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

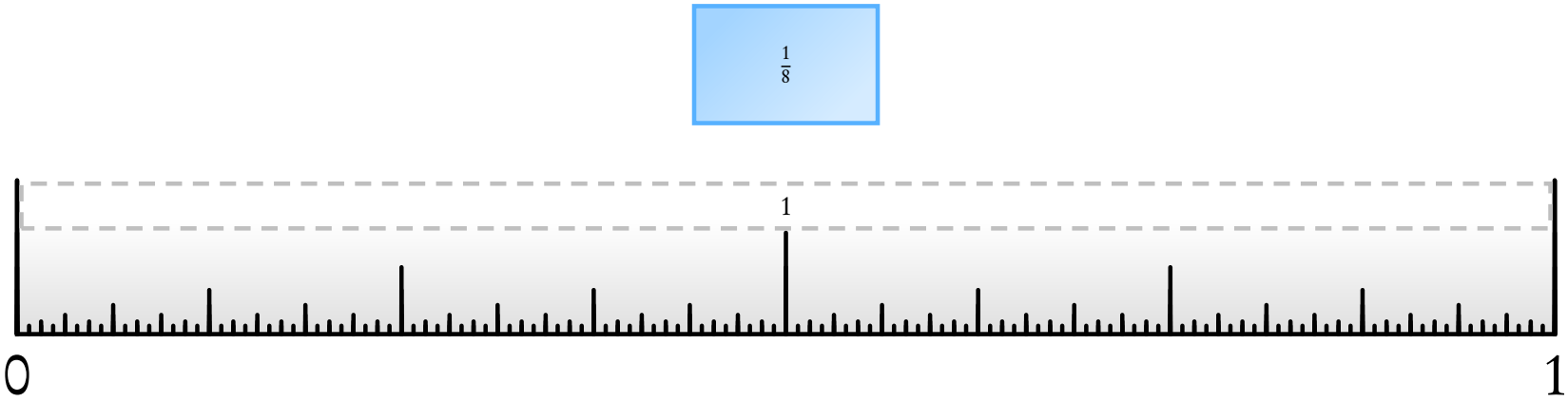


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

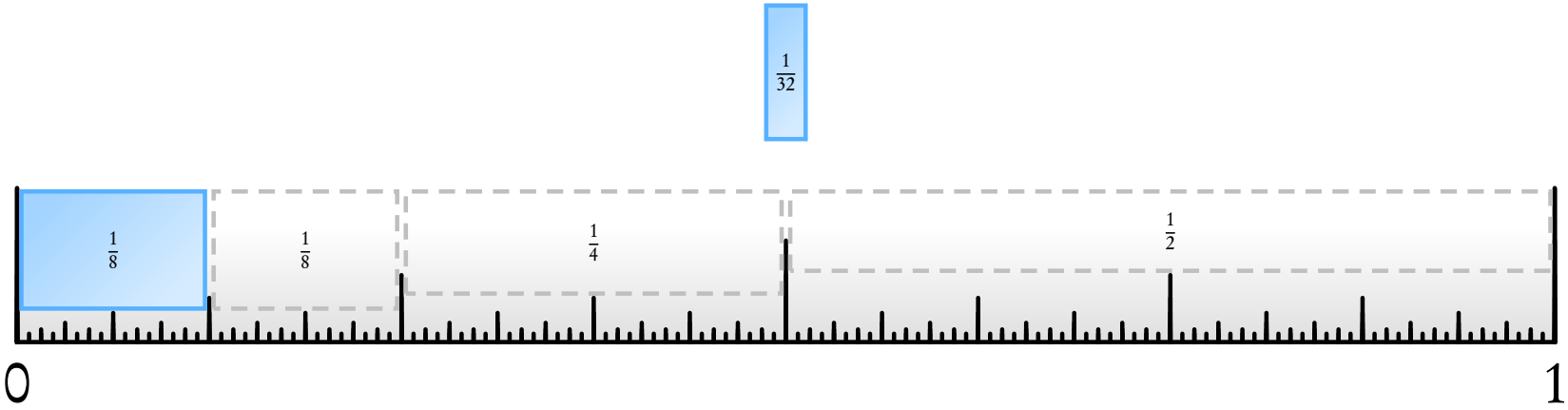


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

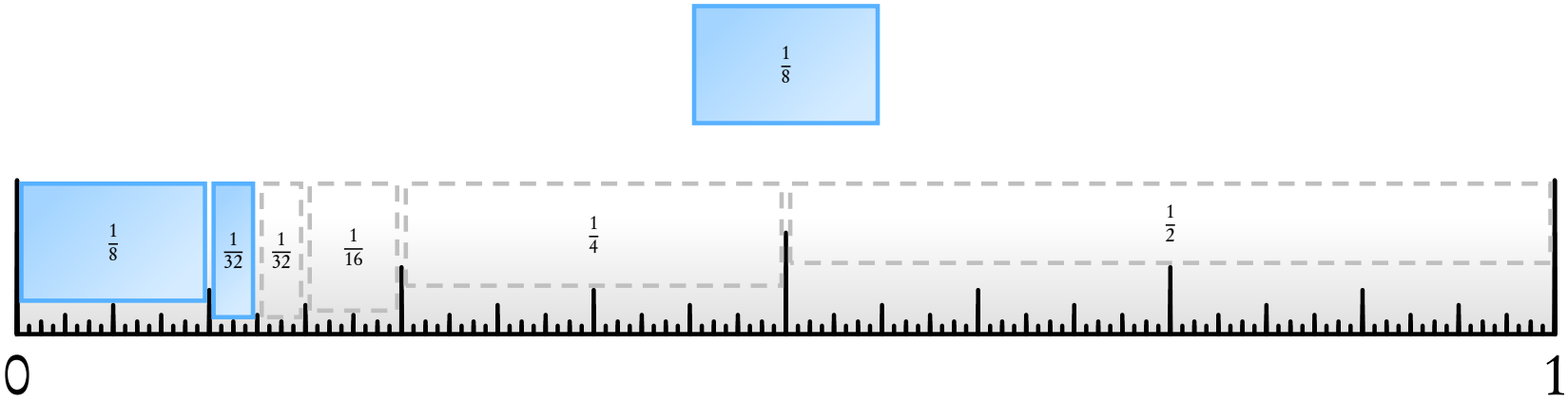


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

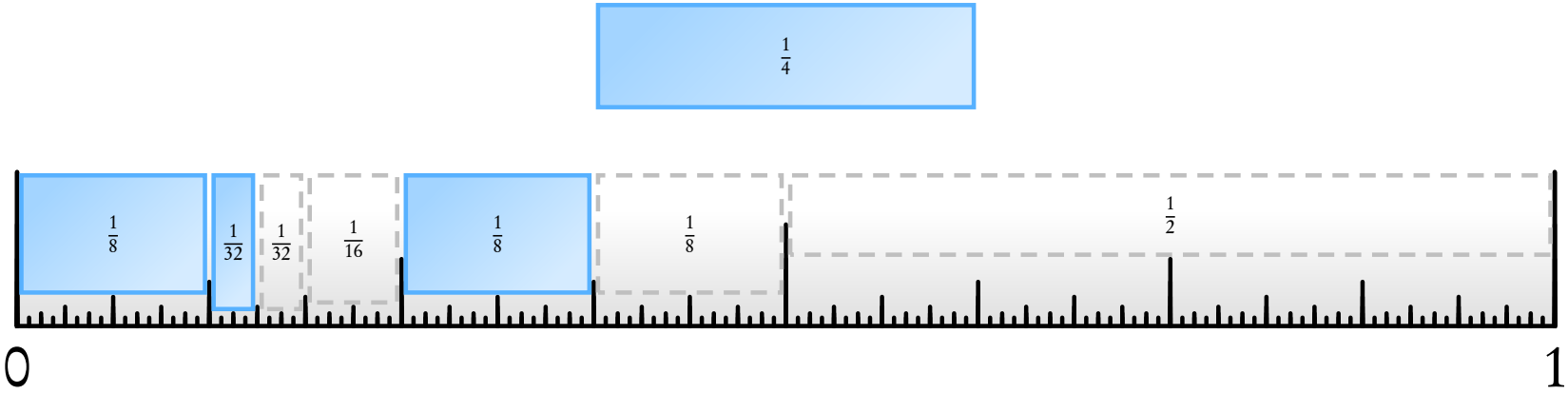


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

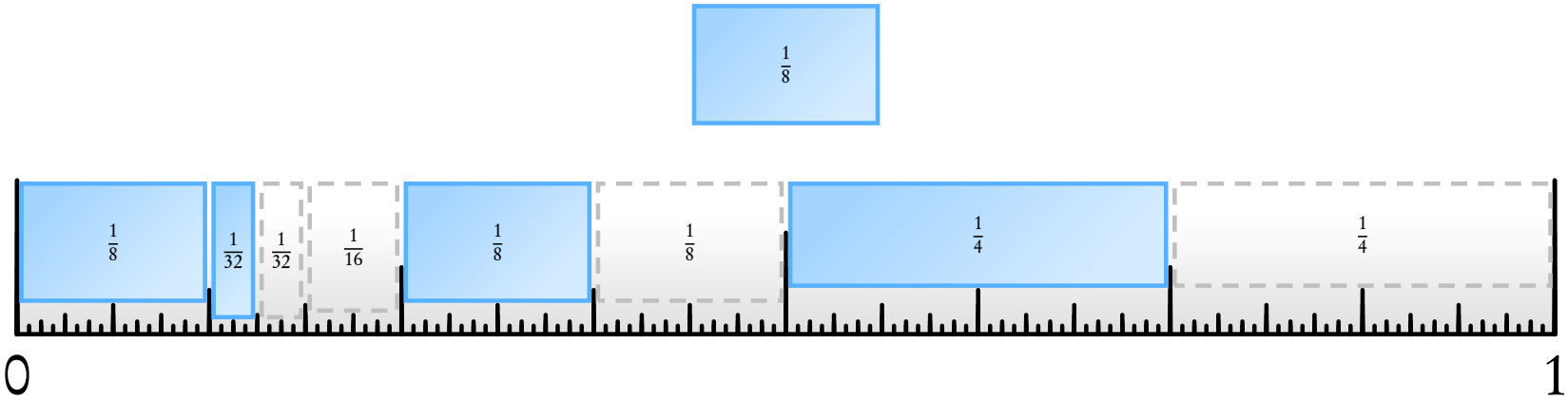


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

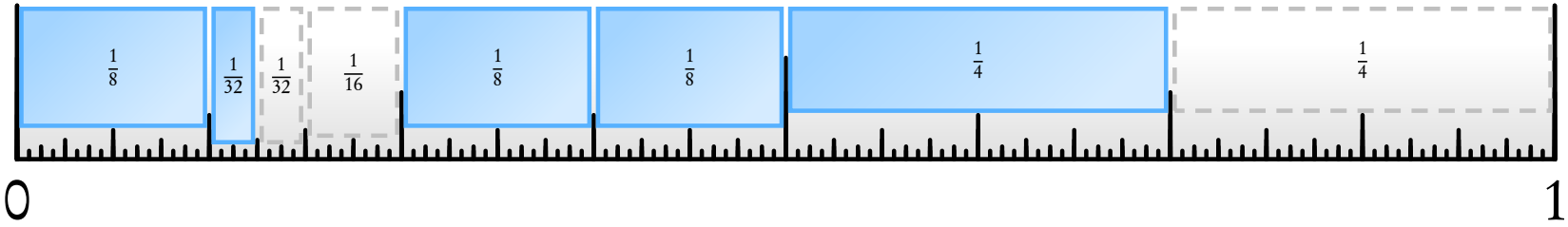


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

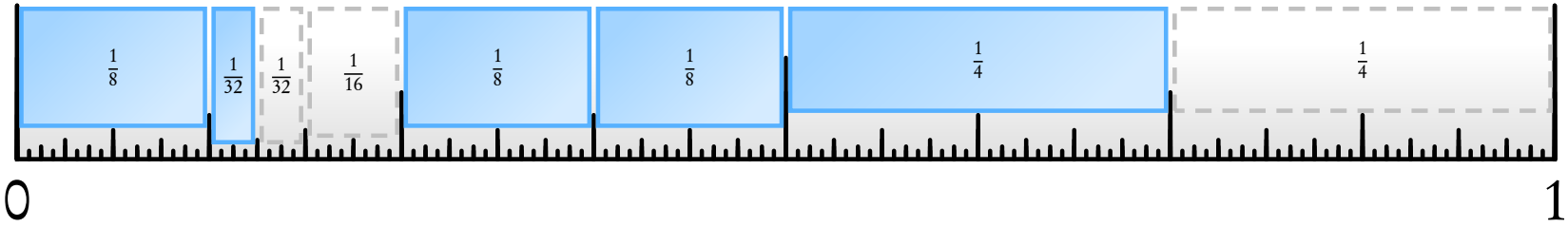


1 **Behauptung.** Die Strategie

„Jeder neue Spielstein wird so weit links wie möglich platziert.“

gehört zu den Strategien, die der Grundidee entsprechen.

Eine feste Greedy-Strategie

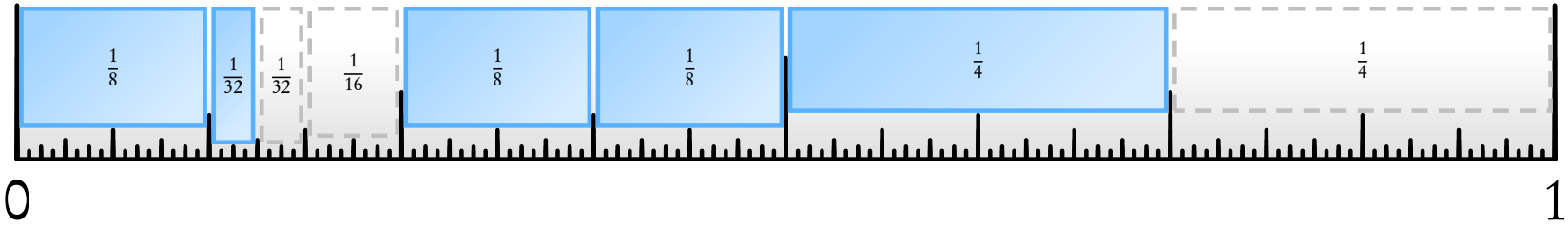


1. **Behauptung.** Die Strategie

   „Jeder neue Spielstein wird so weit links wie möglich platziert.“

   gehört zu den Strategien, die der Grundidee entsprechen.
2. **Beweis.**
   - **Testfrage.** Die Längen der freien Intervalle steigen zu jedem Zeitpunkt von links nach rechts *streng* monoton. Warum?

Eine feste Greedy-Strategie

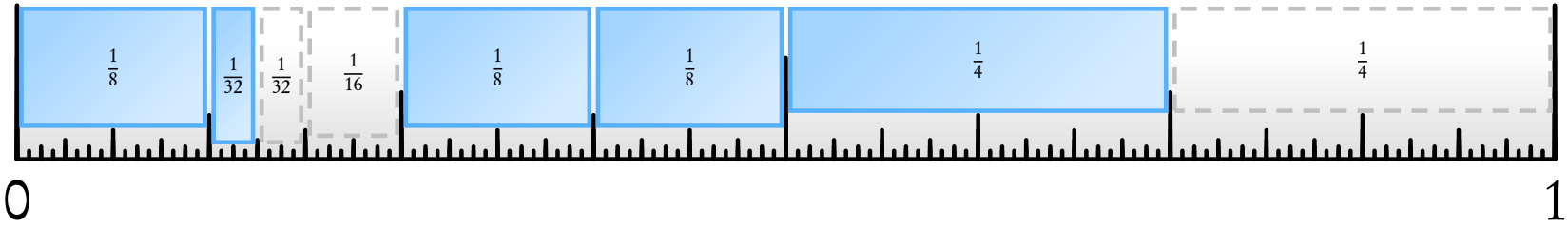


1. **Behauptung.** Die Strategie

   „Jeder neue Spielstein wird so weit links wie möglich platziert.“

   gehört zu den Strategien, die der Grundidee entsprechen.
2. **Beweis.**
   - **Testfrage.** Die Längen der freien Intervalle steigen zu jedem Zeitpunkt von links nach rechts *streng* monoton. Warum?
   - **Testfrage.** Warum folgt daraus die Behauptung?

Eine feste Greedy-Strategie

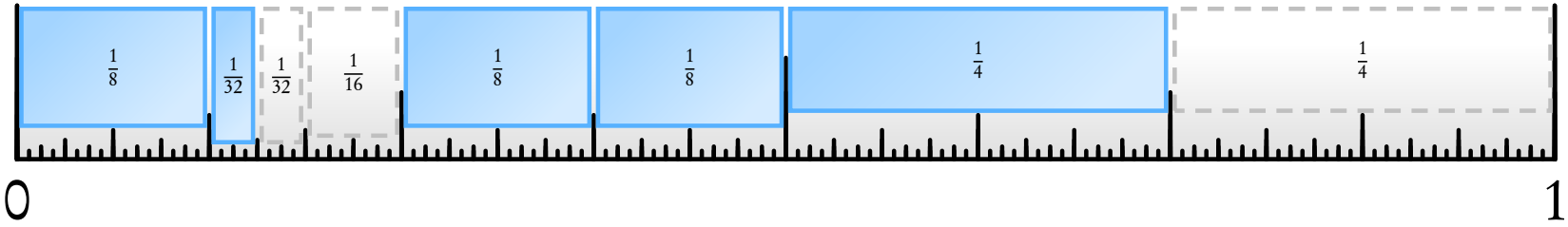


1. Wir können nun den Satz beweisen, in folgender Form.
2. **Satz.** Mit der Strategie

   „Jeder neue Spielstein wird so weit links wie möglich platziert.“

   verlieren wir nie, unabhängig davon welche Spielsteine der Gegner in welcher Reihenfolge spielt — solange er die Summe 1 nicht überschreitet.

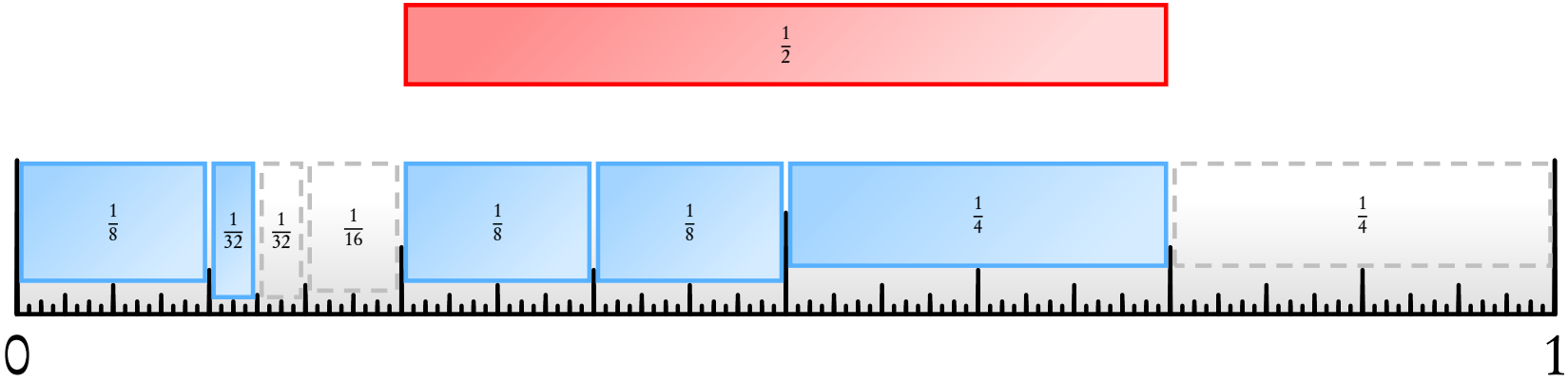
Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.

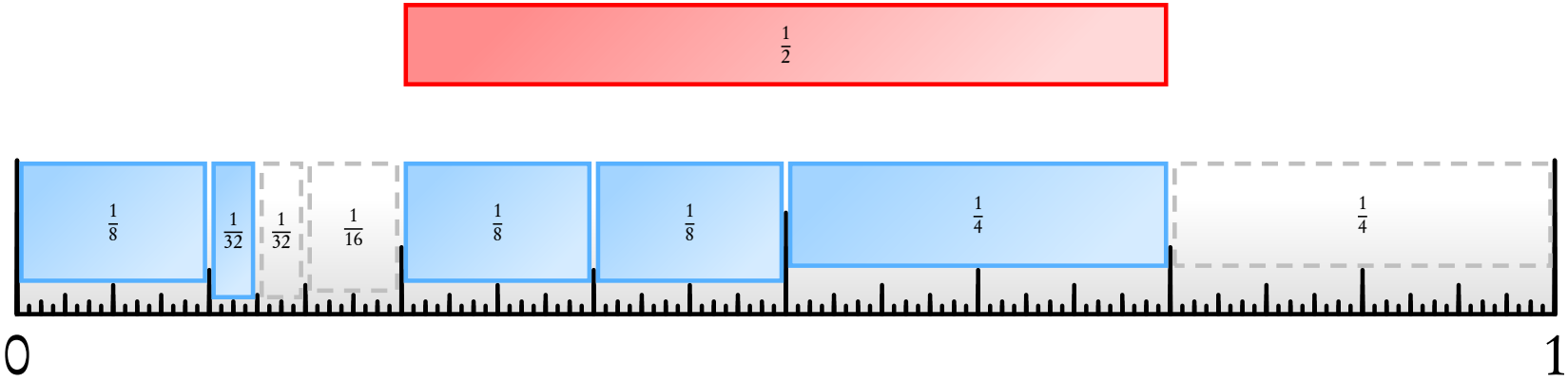
Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.

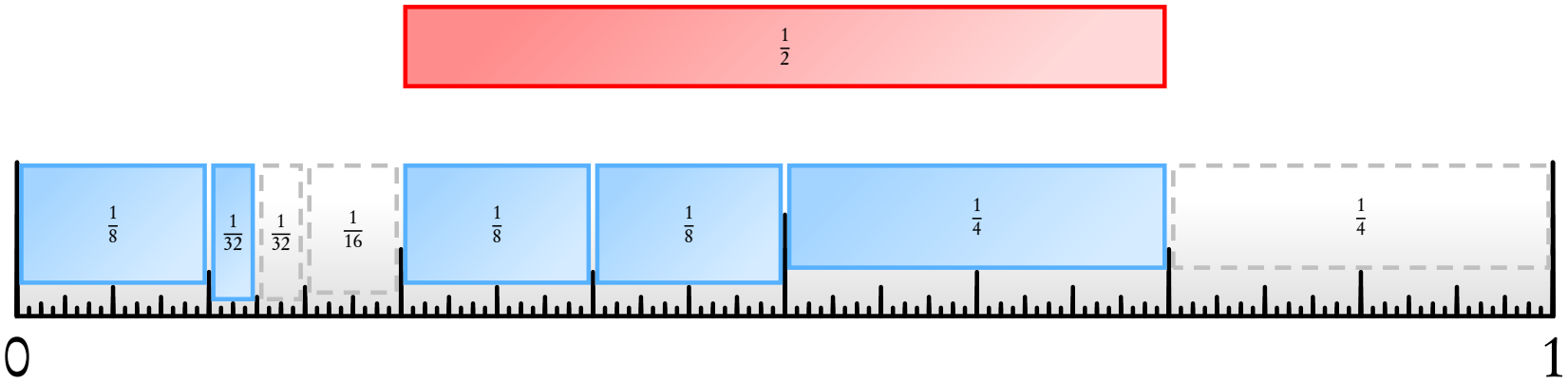
Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.
- Dann ist er größer, als das größte freie Intervall.

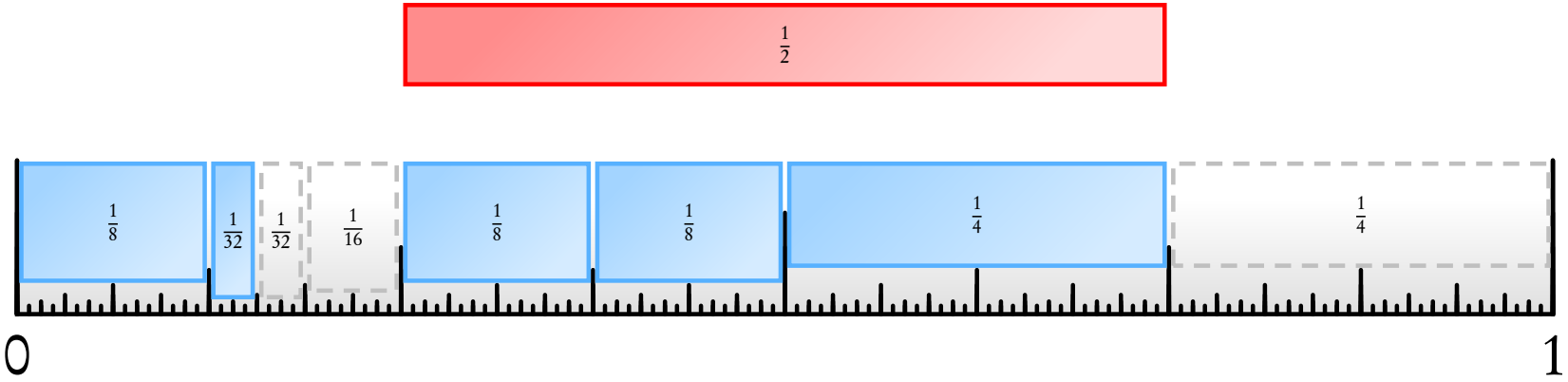
Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.
- Dann ist er größer, als das größte freie Intervall.
- Zu jedem Zeitpunkt gibt es aber nur endlich viele freie Intervalle.

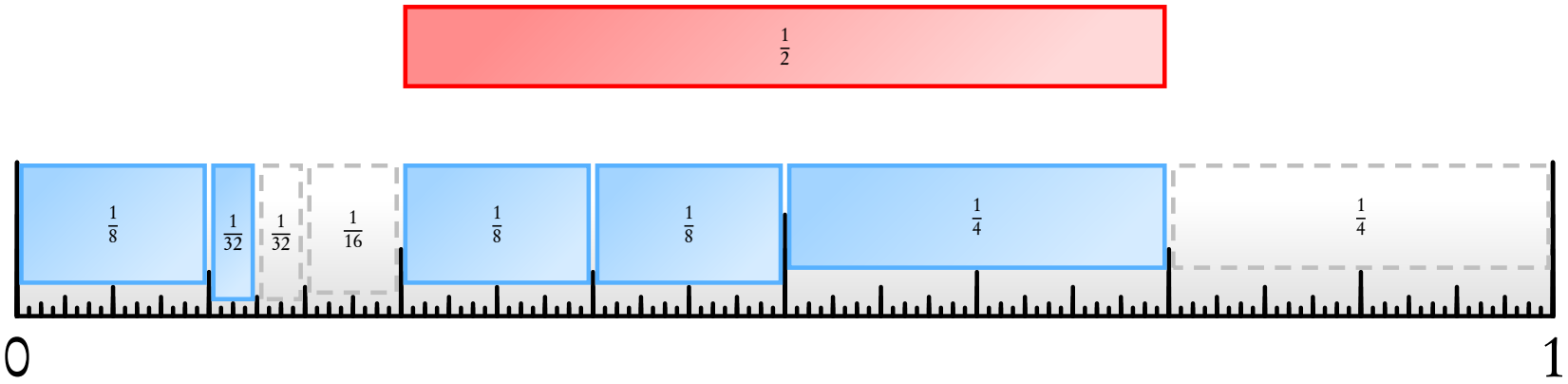
Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.
- Dann ist er größer, als das größte freie Intervall.
- Zu jedem Zeitpunkt gibt es aber nur endlich viele freie Intervalle.
- Da ihre Längen streng monoton steigend sind, und weil für alle k und ℓ gilt, dass $\sum_{i=k+1}^{\ell} 2^{-i} < 2^{-k}$, muss die Summe ihrer Breiten *echt* kleiner sein als die Breite des neuen Teils.

Eine feste Greedy-Strategie



1 **Beweis.**

- Nimm an, es gebe einen Zeitpunkt, an dem wir den neuen Spielstein nicht platzieren können.
- Dann ist er größer, als das größte freie Intervall.
- Zu jedem Zeitpunkt gibt es aber nur endlich viele freie Intervalle.
- Da ihre Längen streng monoton steigend sind, und weil für alle k und ℓ gilt, dass $\sum_{i=k+1}^{\ell} 2^{-i} < 2^{-k}$, muss die Summe ihrer Breiten *echt* kleiner sein als die Breite des neuen Teils.
- Aber dann hat der Gegner mit dem neuen Spielstein die Summe 1 überschritten. □

Präfixfreie Codes

1. **Definition.** Für zwei Wörter x und y schreiben wir $x \preceq y$, falls x ein Anfangsstück (oder *Präfix*) von y ist.
2. **Definition.** Eine Menge A von Wörtern heißt *präfixfrei*, wenn aus $x \neq y$ und $x \preceq y$ folgt, dass $x \notin A$ oder $y \notin A$.
3. **Beispiel.** Telefonnummern.

   (Streng genommen muss man sich dabei auf die jeweils *kürzesten* funktionierenden Telefonnummern jedes Anschlusses beschränken. Außerdem gibt es Ausnahmen.)

Präfixfreie Codes

1. Da Telefonnummern beliebig lang sein dürfen, gibt es unendlich viele. Sollen sie präfixfrei sein, müssen wir dennoch sparsam sein.
2. **Testfrage.** Können Sie hierzu ein Beispiel geben?

Präfixfreie Codes

1. Da Telefonnummern beliebig lang sein dürfen, gibt es unendlich viele. Sollen sie präfixfrei sein, müssen wir dennoch sparsam sein.
2. **Testfrage.** Können Sie hierzu ein Beispiel geben?

   Antwort. Angenommen, wir möchten, dass alle von

   - Feuerwehr
   - Polizei
   - Giftnotruf
   - Ärztlicher Bereitschaftsdienst
   - Einheitliche Behördennummer
   - Bankkarten-Sperrhotline
   - Taxizentrale
   - Auskunft
   - Zeitansage
   - Wetterauskunft

   mit einer *ein*stelligen Telefonnummer erreichbar sind. Wegen Präfixfreiheit gibt es dann *keine weiteren* Telefonnummern.
3. Um Platz zu sparen, beschränken wir uns ab jetzt auf *binäre* „Telefonnummern“. Das macht aber keinen echten Unterschied.

Präfixfreie Codes

1. Wir betrachten nun also binäre Codes statt Telefonnummern.
2. Für welche Objekte die Codes stehen, ist uns zunächst egal.
3. Man kann sich „Code-Wunschlisten“ vorstellen, z.B.
   - Ein Code der Länge 1 soll für Objekt 1 stehen.
   - Ein Code der Länge 2 soll für Objekt 2 stehen.
   - Ein Code der Länge 7 soll für Objekt 3 stehen.
   - Ein Code der Länge 2 soll für Objekt 4 stehen.

   $\vdots$
4. **Testfrage.** Ist die obige Wunschliste erfüllbar?

Präfixfreie Codes

1. Wir betrachten nun also binäre Codes statt Telefonnummern.
2. Für welche Objekte die Codes stehen, ist uns zunächst egal.
3. Man kann sich „Code-Wunschlisten“ vorstellen, z.B.
   - Ein Code der Länge 1 soll für Objekt 1 stehen.
   - Ein Code der Länge 2 soll für Objekt 2 stehen.
   - Ein Code der Länge 7 soll für Objekt 3 stehen.
   - Ein Code der Länge 2 soll für Objekt 4 stehen.

   $\vdots$
4. **Testfrage.** Ist die obige Wunschliste erfüllbar? **Antwort.** Nein.

Präfixfreie Codes

1. Wir betrachten nun also binäre Codes statt Telefonnummern.
2. Für welche Objekte die Codes stehen, ist uns zunächst egal.
3. Man kann sich „Code-Wunschlisten“ vorstellen, z.B.
   - Ein Code der Länge 1 soll für Objekt 1 stehen.
   - Ein Code der Länge 2 soll für Objekt 2 stehen.
   - Ein Code der Länge 7 soll für Objekt 3 stehen.
   - Ein Code der Länge 2 soll für Objekt 4 stehen.

   $\vdots$
4. **Testfrage.** Ist die obige Wunschliste erfüllbar? **Antwort.** Nein.
5. Andere Wunschlisten sind dagegen unproblematisch, z.B.
   - Ein Code der Länge 1 soll für Objekt 1 stehen.
   - Ein Code der Länge 2 soll für Objekt 2 stehen.
   - Ein Code der Länge 3 soll für Objekt 3 stehen.

   $\vdots$

Kolmogorovkomplexität

1. Sei M eine Turingmaschine, die Wörter einliest und ausgibt.
2. **Definition.** Falls der Definitionsbereich von M präfixfrei ist, so nennen wir auch M selbst *präfixfrei*.
3. **Definition.** Die *Kolmogorovkomplexität bezüglich M* eines Wortes x ist die Länge der kürzesten Eingabe mit der M die Ausgabe x produziert, also
$$\mathrm{C}_M(x) = \min\{|y| \colon M(y) = x\}.$$
4. **Intuition.** y ist eine mit dem Packprogramm M gepackte Version von x und $\mathrm{C}_M(x)$ die Länge der besten Packung.
5. **Definition.** Falls M präfixfrei ist schreibe K_M statt C_M und nenne dies die *präfixfreie Kolmogorovkomplexität bezüglich M*.

Kompressionswunschlisten

1. Eine berechenbar aufzählbare Liste von Wörtern $x_1, x_2, \ldots$ und Längen $n_1, n_2, \ldots$ können wir interpretieren als die Wunschliste
   - Ein Code der Länge n_1 soll für x_1 stehen.
   - Ein Code der Länge n_2 soll für x_2 stehen.
   - Ein Code der Länge n_3 soll für x_3 stehen.

   $\vdots$
2. Wir möchten ein entsprechendes Packprogramm M bauen.
3. Dazu kann man sich M als *berechenbar aufzählbare* Menge von Wortpaaren (y, x) vorstellen. **Testfrage.** Warum?
4. Also wollen wir die berechenbar aufzählbare Wunschliste umwandeln in eine berechenbare Aufzählung $(y_1, x_1), (y_2, x_2), \ldots$ von M, wobei $|y_i| = n_i$ für alle i.
5. Ob das geht, hängt davon ab, ob die Wunschliste erfüllbar ist.

Der Satz von Kraft & Chaitin

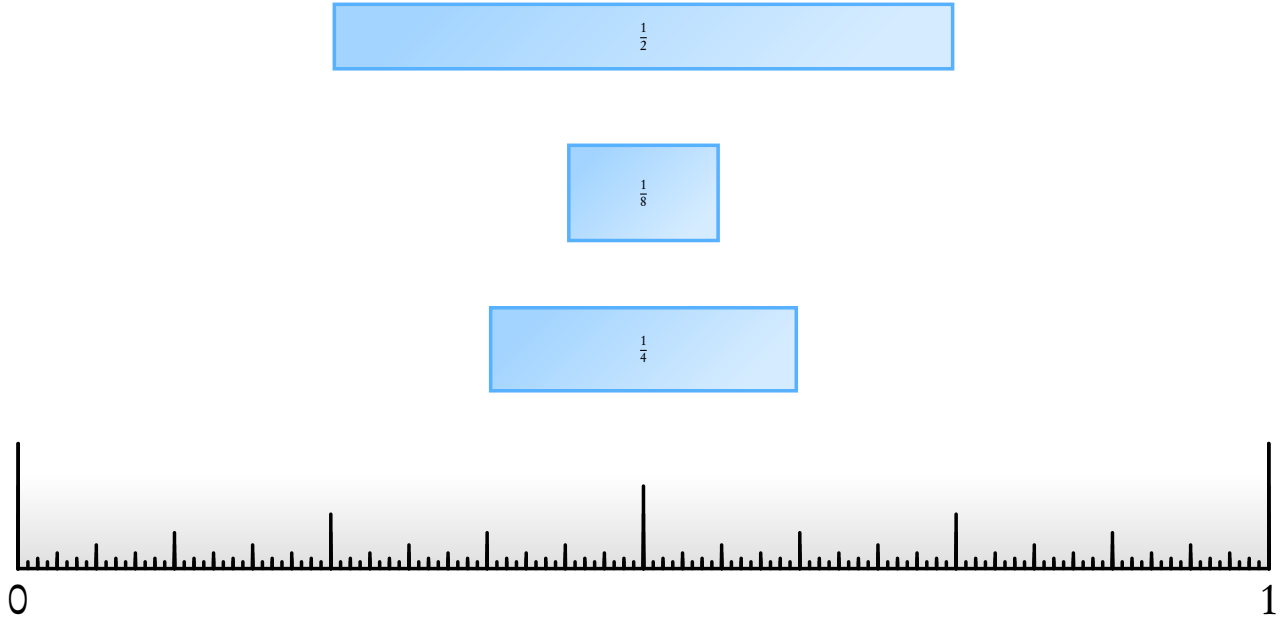
1. **Definition.** Ein *bounded request set* ist eine b.a. Menge
$$B = \{\langle x, n\rangle : x \text{ ist ein Wort und } n \in \mathbb{N}\},$$
so dass $\sum_{\langle x,n\rangle \in B} 2^{-n} \leq 1$.
2. **Satz.** Zu jedem bounded request set B gibt es präfixfreie M mit
$$\text{für alle } \langle x, n\rangle \in B \text{ gibt es ein } y \text{ mit } |y| = n \text{ und } M(y) = x,$$
insbesondere gilt also $\mathrm{K}_M(x) \leq n$.
3. **Intuition.** Zu jedem b.r.s. gibt es ein passendes Packprogramm.
4. **Testfrage.** Gilt auch die Umkehrung, das heißt gibt es für jede präfixfreie Turingmaschine M ein bounded request set B mit
$$\{\langle x, \mathrm{K}_M(x)\rangle : x \in \mathrm{img}(M)\} \subseteq B?$$

Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?

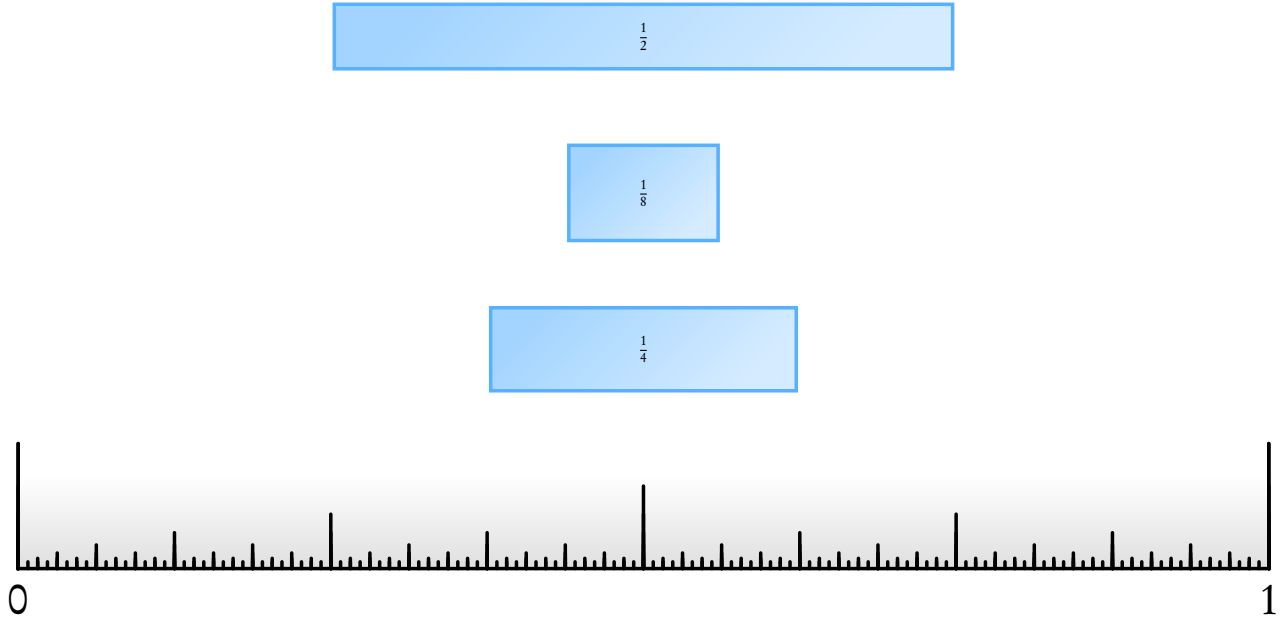
Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht



Der Satz von Kraft & Chaitin

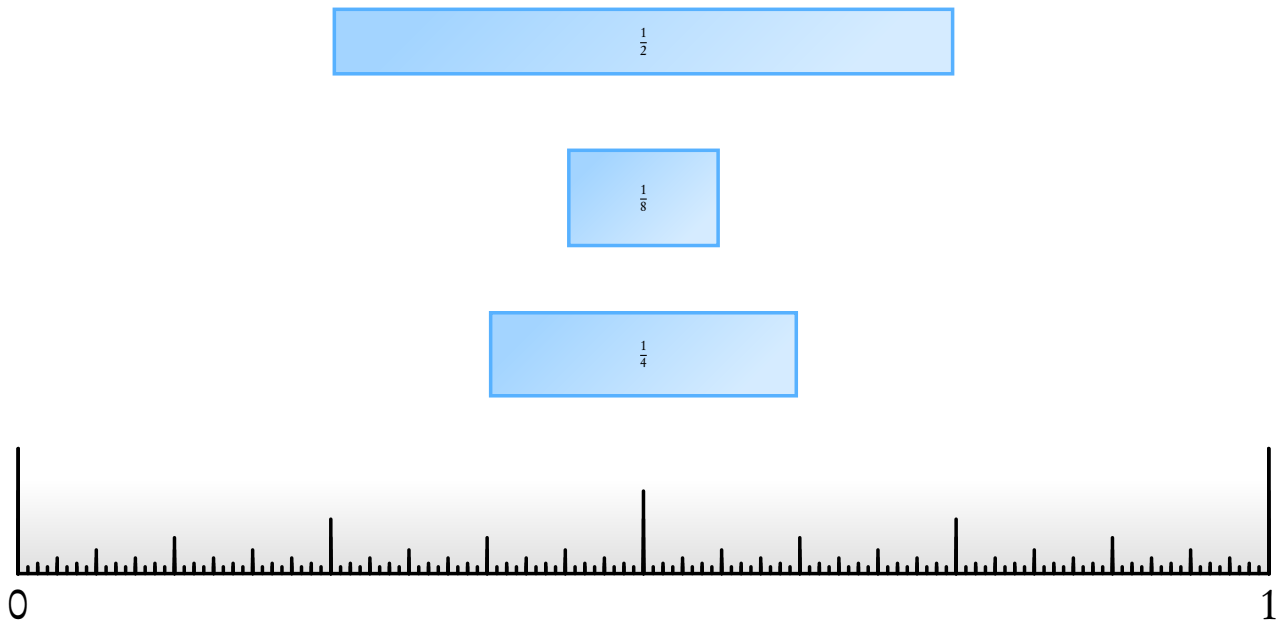
1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht



   - Dass B berechenbar aufzählbar ist, entspricht der Annahme, dass der Gegenspieler die Spielsteine nach und nach liefert.

Der Satz von Kraft & Chaitin

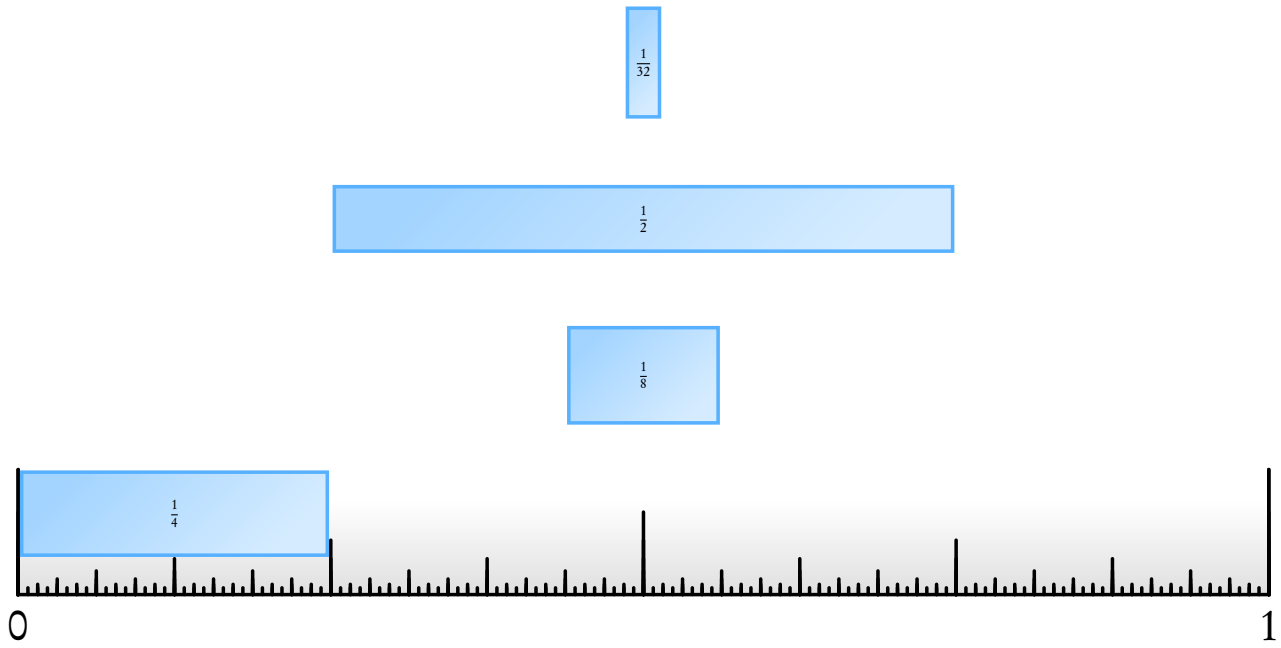
1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht



   - Codes für die x_i's auszuwählen, entspricht dem Platzieren.

Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht

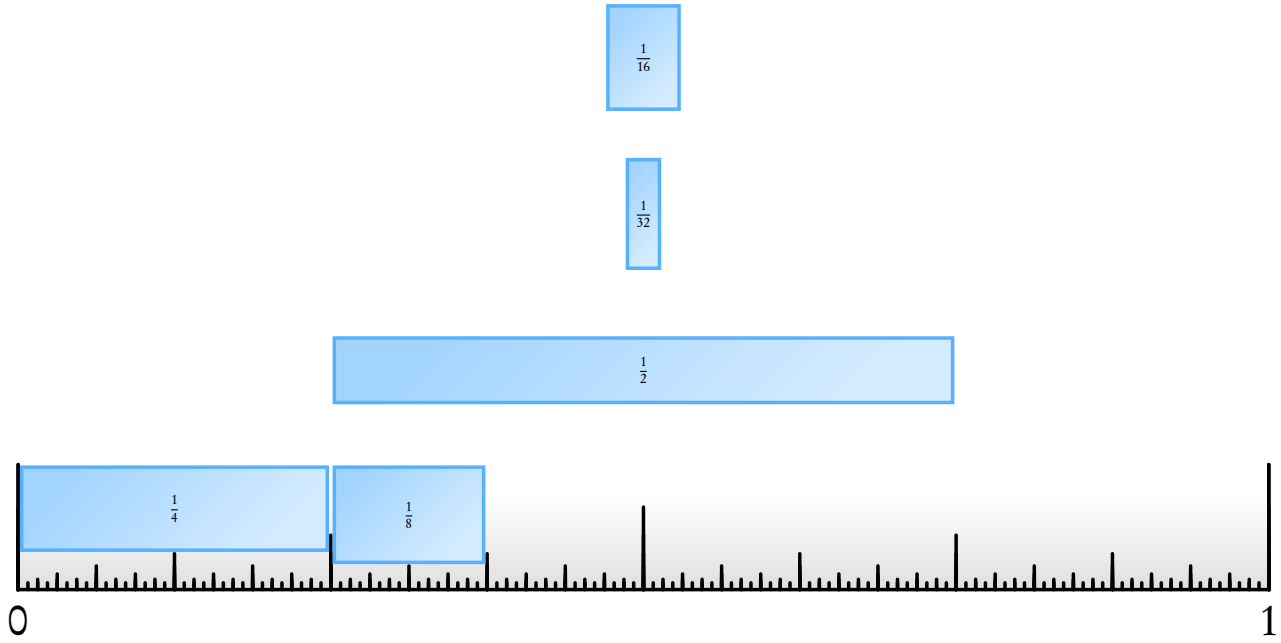


   - Codes für die x_i's auszuwählen, entspricht dem Platzieren.

$$\textit{Sei } M(00) = x_1 \quad \textit{bzw.} \quad \textit{zähle } (00, x_1) \textit{ nach } M \textit{ auf.}$$

Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht

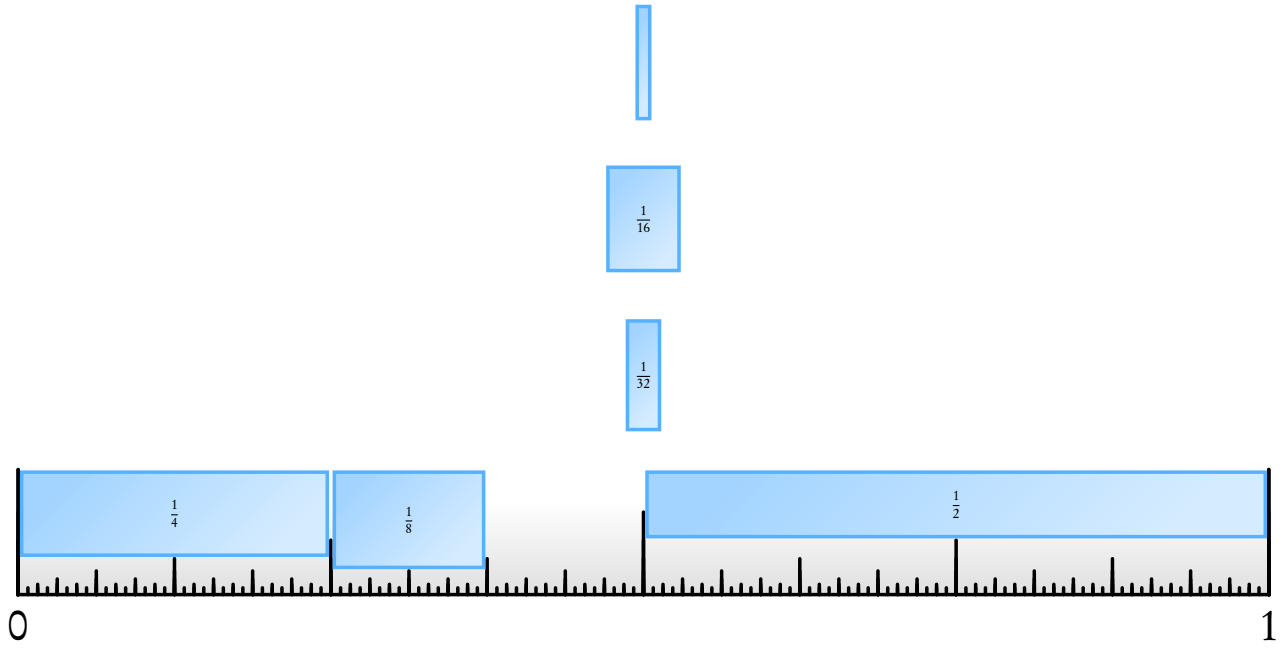


   - Codes für die x_i's auszuwählen, entspricht dem Platzieren.

$$\textit{Sei } M(010) = x_2 \quad \textit{bzw.} \quad \textit{zähle } (010, x_2) \textit{ nach } M \textit{ auf.}$$

Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht

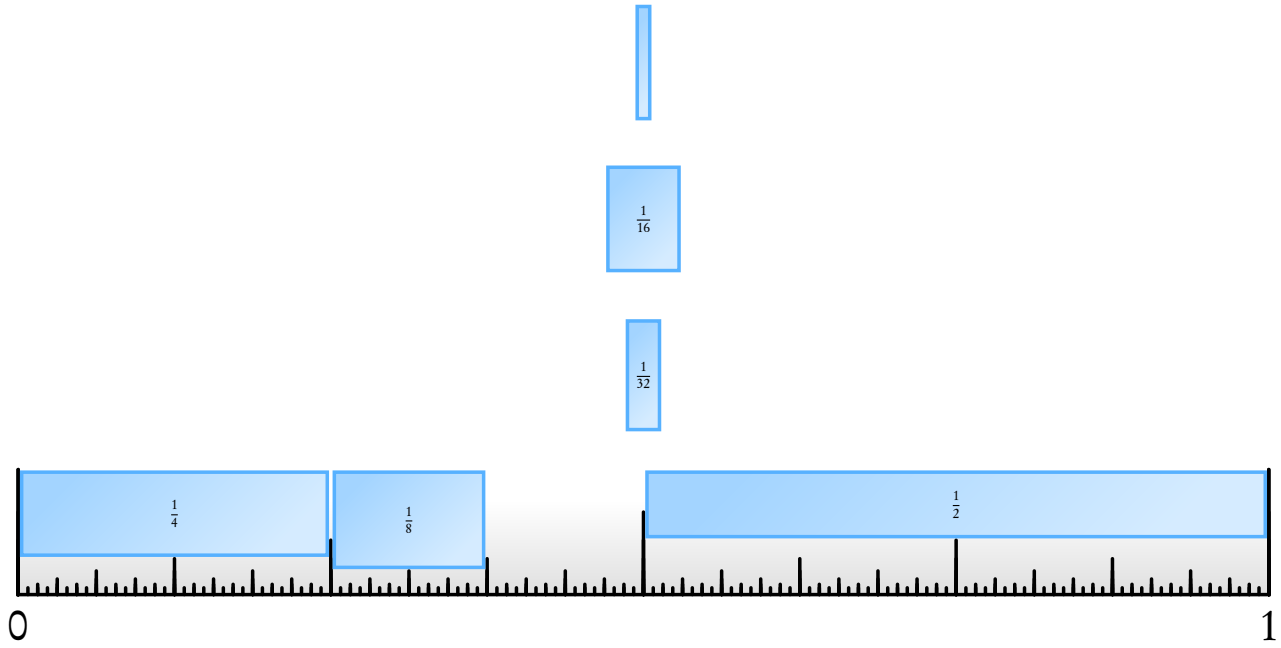


   - Codes für die x_i's auszuwählen, entspricht dem Platzieren.

$$\textit{Sei } M(1) = x_3 \quad \textit{bzw.} \quad \textit{zähle } (1, x_3) \textit{ nach } M \textit{ auf.}$$

Der Satz von Kraft & Chaitin

1. **Testfrage.** Haben Sie einen Verdacht, wie wir den Satz beweisen?
2. **Antwort.**
   - Die Aufzählung $\langle x_1, 2\rangle, \langle x_2, 3\rangle, \langle x_3, 1\rangle, \ldots$ von B entspricht



   - Dass $\sum_{\langle x,n\rangle \in B} 2^{-n} \leq 1$ ist, entspricht der Annahme, dass der Gegenspieler die Summe 1 nicht überschreitet. Also, falls B ein bounded request set ist, definieren wir ein korrektes M. $\square$

Die Kraft-Ungleichung

1. Aus der Vorlesung „Informations- und Codierungstheorie“ kennen Sie vielleicht die Kraft-Ungleichung.
2. Diese ist ein einfacher Spezialfall des Satzes von Kraft & Chaitin, in dem B endlich und von Anfang an bekannt ist.
3. **Der Beweis ist jetzt klar:** Sortiere die Spielsteine *ab*steigend nach Breite, und platziere sie dann von links nach rechts.

Präfixfreie Kolmogorovkomplexität

1. Wir definieren nun die Kolmogorovkomplexität unabhängig von einer bestimmten Maschine M.
2. Aus der Vorlesung „Berechenbarkeitstheorie“ wissen wir, dass es eine *optimal universelle* Turingmaschine gibt.
3. **Fakt.** Es gibt eine präfixfreie Maschine U, die *optimal universell* ist; also so, dass es für jede präfixfreie Maschine M ein c gibt mit
$$x \in \operatorname{img}(M) \implies \mathrm{K}_U(x) \leq \mathrm{K}_M(x) + c.$$
(Beweis → Übungszettel)
4. **Definition.** $\mathrm{K}_U(x)$ ist die *präfixfreie Kolmogorovkomplexität* von x. Da U fest gewählt ist, schreiben wir einfach $\mathrm{K}(x)$.
5. Da die Wahl von U etwas willkürlich ist, interessiert uns K nur bis auf eine additive Konstante. **Testfrage.** Woher kommt diese?

Präfixfreie Kolmogorovkomplexität

1. Da K nur bis auf additive Konstante sinnvoll definierbar ist, macht es wenig Sinn, $\mathrm{K}(x)$ für einzelne Wörter x zu betrachten.
2. Für eine unendliche Binärfolge $A \in \{0,1\}^{\mathbb{N}}$ sieht es anders aus:
   - Für $n \in \mathbb{N}$ bezeichne $A\restriction n$ die ersten n Bits von A.
   - **Testfrage.** Warum gilt auf jeden Fall $\liminf_{n\to\infty} \mathrm{K}(A\restriction n) = \infty$?
   - **Fragestellung.** Wie *genau* verhält sich $\mathrm{K}(A\restriction n)$ für $n \to \infty$?

Anwendbarkeit des Satzes von Kraft & Chaitin

1. **Idee.** Um zu zeigen, dass $\mathrm{K}(A\restriction n) \leq f(n) + \mathrm{O}(1)$ für alle n gilt,
   - zählen wir ein bounded request set B berechenbar auf, so dass für jedes n ein $(A\restriction n, f(n)) \in B$ existiert;
   - Aufgrund des Satzes von Kraft & Chaitin gibt es dann ein M mit
   $$\mathrm{K}(A\restriction n) \leq \mathrm{K}_M(A\restriction n) + \mathrm{O}(1) \leq f(n) + \mathrm{O}(1).$$
2. Man muss also garantieren, dass B ein bounded request set ist.
   - B muss b.a. sein. Das ist oft dadurch „geschenkt“, dass es eine effektive Konstruktion für A gibt, die parallel B auflistet.
   - Es muss $\sum_{\langle x,n\rangle \in B} 2^{-n} \leq 1$ gelten. Wir dürfen also während der Konstruktion von A nicht zu viele Paare nach B aufzählen.
   - **Wichtig:** Sobald ein $\langle x, n\rangle$ nach B aufgezählt wurde, kann man es nicht mehr entfernen, und es „verbraucht“ unwiderbringlich 2^{-n}.
   - Das kann dann zum Problem werden, wenn für ein $A\restriction m$ bereits ein $\langle A\restriction m, n\rangle$ nach B aufgezählt wurde, wir dann aber merken, dass A nicht die von uns gewünschten Eigenschaften hat, und wir deshalb eines der Bits in $A\restriction m$ noch einmal verändern müssen.
   - Solche Verschwendung darf also nicht zu oft auftreten.

Anwendbarkeit des Satzes von Kraft & Chaitin

1. Wir illustrieren die Verwendung an einem trivialen Beispiel.
2. Der Beweis mit dem Satz von Kraft & Chaitin ist „overkill“.
3. **Proposition.** Für alle $n \in \mathbb{N}$ gilt $\mathrm{K}(\emptyset\restriction n) = \mathrm{K}(\mathbb{N}\restriction n) + O(1)$.
4. **Beweis.**
   - Lasse U parallel auf allen Eingaben laufen. Jedesmal, wenn U für eine Eingabe y mit Ausgabe 1^n anhält, zähle $\langle 0^n, |y|\rangle$ nach B auf.
   - Dann ist B ein bounded request set, denn
   $$\sum_{\langle x,m\rangle\in B} 2^{-m} = \sum_{\langle 0^n,m\rangle\in B} 2^{-m} = \sum_{\substack{(y,n):\, y\in\mathrm{dom}(U)\\ \wedge U(y)=1^n}} 2^{-|y|} \leq \sum_{y\in\mathrm{dom}(U)} 2^{-|y|} \leq 1.$$
   - Nach Kraft & Chaitin gibt es ein präfixfreies M, so dass für alle n,
   $$\begin{aligned}\mathrm{K}(\emptyset\restriction n) &\leq \mathrm{K}_M(\emptyset\restriction n) + \mathrm{O}(1) = \mathrm{K}_M(0^n) + \mathrm{O}(1)\\ &= \mathrm{K}(1^n) + \mathrm{O}(1) = \mathrm{K}(\mathbb{N}\restriction n) + \mathrm{O}(1).\end{aligned}$$
   - Ein symmetrisches Argument zeigt $\mathrm{K}(\mathbb{N}\restriction n) \leq \mathrm{K}(\emptyset\restriction n) + O(1)$. □

Kapitelzusammenfassung

1. Wir haben die präfixfreie Kolmogorovkomplexität als die Länge der kürzesten präfixfreien Kodierung von Wörtern definiert.
2. Ein wichtiges Hilfsmittel, um obere Schranken auf diese Komplexität zu zeigen, ist der Satz von Kraft & Chaitin.
3. In ihm spielen *bounded request sets* eine wichtige Rolle.
4. Den Satzes haben wir mit Hilfe eines Spiels bewiesen.
5. Der Satz ist eine Verallgemeinerung der Kraft-Ungleichung.

Konstruktion K-trivialer Mengen

K-Trivialität

1. Der Satz von Kraft & Chaitin ist eines der wichtigsten Werkzeuge im Umgang mit Kolmogorovkomplexität.
2. Er wird oft ähnlich verwendet wie auf Folie 81 skizziert.
3. Wir wenden ihn nun auf ein konkretes Beispiel an.
4. **Definition.** Für $n \in \mathbb{N}$ mit Binärdarstellung $\overline{n}$ sei $\mathrm{K}(n) = \mathrm{K}(\overline{n})$.
5. **Definition.** Eine Binärfolge A heißt K-trivial falls, für alle n,
$$\mathrm{K}(A\restriction n) \leq \mathrm{K}(n) + \mathrm{O}(1).$$
6. **Intuition.** K-triviale Mengen enthalten minimale Information.
7. **Testfrage.** K-triviale Mengen haben unendlich viele *extrem* gut komprimierbare Anfangsstücke. Warum?
8. **Beispiel.** Berechenbare A sind K-trivial. **Testfrage.** Warum?
9. Um zu zeigen, dass es auch K-triviale Mengen gibt, die *nicht* berechenbar sind, benutzen wir den Satz von Kraft & Chaitin.

Approximation von berechenbar aufzählbaren Mengen

1. In der Vorlesung „Berechenbarkeitstheorie“ wurde eine Gödelnummerierung φ aller berechenbaren Funktionen definiert.
2. Durch $W_e = \mathrm{dom}(\varphi_e)$ wurde dann eine effektive Nummerierung W aller berechenbar aufzählbaren Teilmengen von $\mathbb{N}$ definiert.
3. **Definition.** Definiere $W_{e,s}$ als
$$\{n \in W_e : n < s \wedge \varphi_e(n) \text{ hält nach höchstens } s \text{ Schritten.}\}.$$
4. **Intuition.** $W_{e,s}$ sind die Elemente der berechenbar aufzählbaren Menge W_e, die schon in den ersten s Schritten aufgezählt werden.
5. **Testfrage.** Warum gilt $\lim_s W_{e,s} = W_e$?
6. **Testfrage.** Warum ist $W_{e,s}$ für alle e, s berechenbar?

Approximation von K

1. **Definition.** Lasse die Maschine U (die K definiert) auf den längen-lexikographisch kleinsten s Wörtern für je s Schritte laufen.

   Falls es existiert, sei y das kürzeste untersuchte Wort, für das U dabei mit Ausgabe x anhält; setze dann $\mathrm{K}_s(x) = |y|$.

   Solange für x kein besseres y mit $U(y) = x$ gefunden wurde, sei $\mathrm{K}_s(x)$ eine triviale obere Schranke für $K(x)$. *(→ Übungszettel)*
2. **Intuition.**
   - $\mathrm{K}(x)$ ist die Länge der bestmöglichen Kompression von x.
   - Diesen Wert kennen wir nicht. Wir können nur immer weiter nach immer besseren Kompressionen von x suchen.
   - $\mathrm{K}_s(x)$ ist dann die Länge der besten Kompression, die wir innerhalb von s Zeitschritten finden können.
   - Je größer s ist, umso länger können wir suchen, und umso bessere Kompressionen können wir (möglicherweise) finden.
   - Also ist $\mathrm{K}_s(x)$ monoton fallend in s und $\mathrm{K}_s(x) \searrow \mathrm{K}(x)$.
3. Analog gilt für $n \in \mathbb{N}$, dass $\mathrm{K}_s(n) \searrow \mathrm{K}(n)$.

Nichtberechenbare K-triviale Mengen

1. **Satz.** Es gibt eine nicht-berechenbare, berechenbar aufzählbare, K-triviale Menge.
2. **Konstruktion.** Konstruiere das gesuchte $A = \lim_s A_s$ wie folgt:
   - Zum Zeitpunkt $s = 0$ sei $A_0 = \emptyset$.
   - Zu jedem Zeitpunkt $s \geq 1$ prüfe, ob es $e \leq s$ und $\underbrace{n > 2e}$ gibt mit:

$$\overbrace{W_{e,s} \cap A_{s-1} = \emptyset}^{2} \wedge \overbrace{n \in W_{e,s}}^{3} \wedge \overbrace{\sum_{n<\ell<2^s} 2^{-\mathrm{K}_s(\ell)} \leq 2^{-(e+1)}}^{4 \quad 1}$$

   - Falls ja, sei $A_s = A_{s-1} \cup \{n\}$ für das kleinste solche n.
   - Sonst sei $A_s = A_{s-1}$.
3. Wir verifizieren nun, dass die Konstruktion funktioniert. Die Argumentation erfolgt über eine Reihe von Behauptungen.

Verifikation

1. **Behauptung.** A ist berechenbar aufzählbar.

   Beweis. Dies ist offensichtlich, da 2 3 4 berechenbar sind.

2. **Behauptung.** Für jedes e gibt es höchstens ein n mit $n \in A \cap W_e$.

   Beweis. Für jedes e kann 2 ab dem Moment nicht mehr erfüllt sein, in dem das erste Element von W_e nach A aufgezählt wird.

3. **Behauptung.** $\overline{A}$ ist unendlich.

   Beweis. Betrachte ein beliebiges gerades ℓ. Wie viele Bits von $A\restriction\ell$ können 1 sein?

   Wegen der vorherigen Behauptung kann aufgrund eines e's nur ein Bit 1 werden. Es gibt aber wegen 1 nur $\ell/2$ viele e, die als Ursache dafür in Frage kommen, ein Bit in $A\restriction\ell$ zu 1 zu machen.

   Also hat mindestens die Hälfte der Bits von $A\restriction\ell$ den Wert 0.

Verifikation

1. **Behauptung.** Für jedes unendliche W_e gilt, dass $A \cap W_e \neq \emptyset$.

   Beweis. Wir müssen zeigen, dass für unendliche W_e und für ausreichend großes s die 2 3 4 irgendwann erfüllt werden.

   O.B.d.A. ist 2 wahr, denn sonst sind wir fertig.

   Wegen

$$\sum_{n<\ell<2^s} 2^{-\mathrm{K}_s(\ell)} \leq \sum_{n<\ell<2^s} 2^{-\mathrm{K}(\ell)} \leq \sum_{n<\ell} 2^{-\mathrm{K}(\ell)} \xrightarrow{n\to\infty} 0$$

   wird 4 von jedem ausreichend großen n erfüllt.

   Da W_e unendlich ist, ist für ausreichend großes s ein solches n auch in $W_{e,s}$, also wird für s groß genug auch 3 erfüllbar.

Verifikation

1. **Behauptung.** A ist nicht berechenbar.

   Beweis. Laut der letzten Behauptung schneidet jede unendliche berechenbar aufzählbare Menge A. Da $\overline{A}$ unendlich ist, ist es also nicht berechenbar aufzählbar. Also ist A nicht berechenbar.

Das bounded request set

1. Um nachzuweisen, dass A K-trivial ist, zählen wir ein geeignetes bounded request set $B = \lim_s B_s$ auf.
2. **Unser Ziel:** $\mathrm{K}(A\restriction n) \leq \mathrm{K}(n) + \mathrm{O}(1)$ für alle n.
3. **Problem.** Wir kennen nur die Approximationen $\mathrm{K}_s(n) \searrow \mathrm{K}(n)$.
4. **Lösung.**
   - Jedes Mal, wenn sich eine Approximation verbessert, also wenn ein $\mathrm{K}_s(\ell) < \mathrm{K}_{s-1}(\ell)$, wollen wir auch $\mathrm{K}_s(A\restriction \ell)$ absenken.
   - Dazu führe U parallel auf allen Eingaben aus. Wenn zur Zeit s die Berechnung mit Eingabe y einen Binärcode für eine Zahl ℓ ausgibt und $|y| < \mathrm{K}_{s-1}(\ell)$, zähle $\langle A_s\restriction \ell, |y|+1\rangle$ nach B auf.
   - Wir nennen diese Wünsche die *normalen Wünsche*.

Das bounded request set

1. **Unser Ziel:** $\mathrm{K}(A\restriction n) \leq \mathrm{K}(n) + \mathrm{O}(1)$ für alle n.
2. **Problem.** Manchmal geht einer der Wünsche „kaputt“.
   - Nimm an, zu einem Zeitpunkt s wird bei der Konstruktion von A eine Zahl n nach A aufgezählt.
   - Falls es in B_{s-1} bereits Wünsche der Form $\langle A_{s-1}\restriction \ell, \ldots\rangle$ mit $\ell \geq n$ gibt, so erfüllen diese ihre Aufgabe nicht mehr.
   - Denn sie betreffen „alte“ $A_{s-1}\restriction \ell$, die sich auf Bit n vom neuen A_s unterscheiden, und damit kein Anfangsstück von A mehr sind.
3. **Lösung.** Wir müssen diese „defekten“ Wünsche ersetzen.
   - Für alle ℓ mit $n < \underbrace{\ell < 2^s}_{5}$ sei k_ℓ minimal mit $\langle A_{s-1}\restriction \ell, k_\ell\rangle \in B_{s-1}$.
   - Zähle für alle solchen (ℓ, k_ℓ) den Wunsch $\langle A_s\restriction \ell, k_\ell\rangle$ nach B auf.
   - Wir nennen diese Wünsche die *Reparaturwünsche*.

Verifikation

1. **Behauptung.** A ist K-trivial.
2. **Noch zu zeigen:** B ist das gesuchte bounded request set.
   - B ist berechenbar aufzählbar. ✓

     Testfrage. Warum ist das klar?
   - Das Gewicht $\sum_{\langle x,n\rangle \in B} 2^{-n}$ ist höchstens 1. Dazu zeigen wir:
     - Die normalen Wünsche tragen höchstens $1/2$ bei. ✓

       Testfrage. Warum ist das klar?
     - Die Reparaturwünsche tragen höchstens $1/2$ bei.
   - B bezeugt tatsächlich $\mathrm{K}(A\restriction n) \leq \mathrm{K}(n) + \mathrm{O}(1)$ für alle n.
     - **Insbesondere:** Warum stört die Einschränkung 5 nicht?

Verifikation

1. **Behauptung.** Die Reparaturwünsche tragen höchstens $1/2$ bei.
2. **Beweis.** Reparaturwünsche entstehen als Konsequenz daraus, dass ein n zu A_s hinzugefügt wird.

   Testfrage. Wie sorgt 4 dafür, dass n nur hinzugefügt wird, wenn die dadurch ausgelösten Reparaturwünsche höchstens $2^{-(e+2)}$ beitragen?

   In der Summe über alle e ergibt sich damit ein Beitrag von $1/2$.

Verifikation

1. **Behauptung.** B bezeugt $\mathrm{K}(A\restriction n) \leq \mathrm{K}(n) + \mathrm{O}(1)$ für alle n.
2. **Beweis.**
   - Laut Kraft & Chaitin gibt es ein präfixfreies M zu B.
   - Wenn wir vorerst den Reparaturbedarf ignorieren, dann ist durch die normalen Wünsche erstmal klar, dass für alle n gilt, dass
   $$\mathrm{K}(A\restriction n) \leq \mathrm{K}_M(A\restriction n) + \mathrm{O}(1) \leq \mathrm{K}(n) + \underbrace{1 + \mathrm{O}(1)}_{\mathrm{O}(1)}.$$
   - Durch die Reparaturwünsche bleibt diese Eigenschaft auch dann erhalten, wenn sich A im Verlauf der Konstruktion ändert.
   - **Frage.** Warum ist die Einschränkung 5 kein Problem?
   - **Antwort.** Bis Zeitpunkt s kann U höchstens Wörter der Länge s erzeugen, die als Binärcodes nur Zahlen echt kleiner 2^s kodieren.

     Da normale Wünsche nur als Reaktion darauf erzeugt werden, gilt für jeden Wunsch $\langle x, n\rangle \in B_s$ dass $|x| < 2^s$.

     Es genügt beim Reparieren also, sich auf diese zu beschränken.
3. Damit ist der Beweis des Satzes vollständig. □

8 Interaktive Beweise

Beweise kommunizieren

1. Ein Mathematiker sucht nach einem Beweis für einen neuen Satz und findet ihn schließlich nach langer Suche.
2. Nun soll ein anderer Mathematiker ebenfalls von der Wahrheit dieses Satzes überzeugt werden. Dazu muss ihm nur der am Ende gefundene Beweis mitgeteilt werden.
3. Normalerweise wird der zweite Schritt wesentlich weniger Zeit erfordern, als der erste. Dem zweiten Mathematiker bleibt die lange Suche des ersten erspart.
4. Das erinnert an den Unterschied zwischen **NP** und **P**:
   Eine Lösung zu finden dauert viel länger als eine zu prüfen.
5. Was, wenn außerdem noch Rückfragen, Antworten darauf, und weitere Rückfragen erlaubt sind, also beliebige Interaktionen?

Interaktion formal definiert

1. **Definition.** Seien $f, g\colon \{0,1\}^{<\mathbb{N}} \to \{0,1\}^{<\mathbb{N}}$ und $k \geq 0$.

   Eine *k-Runden-Interaktion zwischen f und g bei Eingabe x* ist eine Folge von Wörtern $\langle f, g\rangle(x) := (a_1, \ldots, a_k)$, so dass

   - $a_1 = f(x)$,
   - $a_2 = g(x, a_1)$,

     $\vdots$
   - $a_{k-1} = f(x, a_1, \ldots, a_{k-2})$,
   - $a_k = g(x, a_1, \ldots, a_{k-2}, a_{k-1})$.

2. **Definition.** Das *Ergebnis der Interaktion* $\langle f, g\rangle(x)$ ist definiert als

$$\mathrm{Erg}_f \langle f, g\rangle(x) = f(x, a_1, \ldots, a_k).$$

3. **Intuition.**

   - Ausgehend von der Eingabe x arbeiten f und g abwechselnd und fügen neue Informationen a_i für das Gegenüber hinzu.
   - Da f und g möglicherweise ganz unterschiedliche Fähigkeiten haben, kann die jeweilige Hilfe des Gegenübers wichtig sein.
   - Am Ende fällt f eine Entscheidung.

Interaktive Beweise

1. Uns interessiert der Fall, in dem
   - g, der *Beweiser*, unbegrenzte Rechenleistung hat,
   - f, der *Prüfer*, nur polynomielle Rechenzeit hat,
   - und g versucht, f von der Wahrheit einer Aussage zu überzeugen.
2. **Intuition.** Wegen seiner unbegrenzten Rechenleistung kann g sehr aufwändig nach „guten Argumenten“ suchen, um f zu überzeugen. Diese Argumente müssen dann aber für das „relativ dumme“ f nachvollziehbar sein.
3. Ab sofort schreiben wir statt g meist P („prover“, Beweiser) und statt f meist V („verifier“, Prüfer).

Determininistische interaktive Beweise

1. **Definition.** Sei $k\colon \mathbb{N} \to \mathbb{N}$. Eine Sprache L hat ein *k-Runden deterministisches interaktives Beweissystem* falls eine deterministische Maschine V existiert, die bei Eingabe $(x, a_1, \ldots, a_i)$ in $|x|$-polynomieller Zeit läuft und für die gilt, dass
   - $x \in L \Longrightarrow (\exists P)(\mathrm{Erg}_V\langle V, P\rangle(x) = 1)$, *(Vollständigkeit)*
   - $x \notin L \Longrightarrow (\forall P)(\mathrm{Erg}_V\langle V, P\rangle(x) = 0)$, *(Korrektheit)*

   wobei die Interaktion eine $k(|x|)$-Runden-Interaktion ist.
2. **Definition. dIP** sei die Menge der Sprachen mit einem k-Runden deterministischen interaktiven Beweissystem für ein Polynom k.
3. **Bemerkung.** Es werden keinerlei Einschränkungen auf die Rechenleistung von P gefordert. Für eine falsche Aussage gibt es also keinen Beweis, egal mit wieviel Aufwand gesucht wird.

Determininistische interaktive Beweise

1. **Satz.** $\mathbf{dIP} = \mathbf{NP}$.
2. **Intuition.** Deterministische Interaktion ist uninteressant.
3. **Beweis. Testfrage.** Warum ist $\mathbf{NP} \subseteq \mathbf{dIP}$ trivial? ✓

   Umgekehrt sei $L \in \mathbf{dIP}$. Also gibt es für L ein k-Runden deterministisches interaktives Beweissystem, wobei k ein Polynom ist. Sei V der Prüfer.

   Folgende nichtdeterministische Maschine N akzeptiert $x \in L$:
   - Rate ein Protokoll $(a_1, a_2, \ldots, a_{k(|x|)})$ des interaktiven Beweises.
   - Simuliere V mehrfach, um zu prüfen ob
   $$V(x) = a_1 \wedge V(x, a_1, a_2) = a_3 \wedge \cdots \wedge V(x, a_1, \ldots, a_{k(|x|)}) = 1.$$
   - Falls ja, akzeptiere x. Sonst verwirf.

   Testfrage. Wieso arbeitet N korrekt und in Polynomialzeit? □

Probabilistische Prüfer

1. Da deterministische Interaktion nichts Neues bietet verglichen mit **NP**, randomisieren wir nun den Prüfer. Das heißt, er darf Münzen werfen, und die Ergebnisse beeinflussen sein Vorgehen.
2. Einen korrekten Beweis soll er mit hoher Wahrscheinlichkeit akzeptieren.
3. Einen von einem „beliebig bösartigen“ Beweiser vorgeschlagenen falschen Beweis soll er nur mit geringer Wahrscheinlichkeit akzeptieren.

Beispiel

1. **Testfrage.** Merlin hat je eine rote und grüne Socke. Artus ist farbenblind. Wie kann Merlin ihn von der Verschiedenheit überzeugen? Artus darf dabei 100 Mal eine Münze werfen.

Beispiel

1. **Testfrage.** Merlin hat je eine rote und grüne Socke. Artus ist farbenblind. Wie kann Merlin ihn von der Verschiedenheit überzeugen? Artus darf dabei 100 Mal eine Münze werfen.
2. **Antwort.** Wiederhole folgendes Spiel 100 Mal.
   - Merlin gibt Artus die Socken und schließt die Augen.
   - Artus wirft eine Münze; je nach Ergebnis vertauscht er die Socken, oder auch nicht.
   - Merlin öffnet die Augen.
   - Kann Merlin korrekt sagen, ob die Socken vertauscht wurden?

   Falls die Socken unterschiedlich sind, sollte Merlin 100 Mal richtig antworten. Sonst um die 50 Mal.

Probabilistische Interaktionen

1. Die Werte der geworfenen Münzen seien nun in $r \in \{0,1\}^{<\mathbb{N}}$ festgehalten. Diese Bits kann nur V sehen, der Beweiser P nicht.
2. **Definition.** Eine *probabilistische k-Runden-Interaktion zwischen V und P bei Eingabe x* ist ein $\langle V,P\rangle(x) := (r,a_1,\ldots,a_k)$, so dass
   - $a_1 = V(x,r)$,
   - $a_2 = P(x, a_1)$,
   - $a_3 = V(x,r,a_1,a_2)$,
   - $\vdots$
   - $a_{k-1} = P(x, a_1,\ldots,a_{k-2})$,
   - $a_k = V(x,r,a_1,\ldots,a_{k-2},a_{k-1})$.
3. **Definition.** Das *Ergebnis der Interaktion* $\langle V,P\rangle(x)$ ist definiert als

$$\mathrm{Erg}_V\langle V,P\rangle(x) = V(x,r,a_1,\ldots,a_k).$$

Probabilistische interaktive Beweise

1. **Definition.** Sei $k\colon \mathbb{N} \to \mathbb{N}$. Eine Sprache L hat ein *k-Runden probabilistisches interaktives Beweissystem* falls eine probabilistische Maschine V existiert, die bei Eingabe $(x, r, a_1, \ldots, a_i)$ in $|x|$-polynomieller Zeit läuft und für die gilt, dass
   - $x \in L \Longrightarrow (\exists P)\, \Pr[\mathrm{Erg}_V\langle V, P\rangle(x) = 1] \geq 2/3$, *(Vollständigkeit)*
   - $x \notin L \Longrightarrow (\forall P)\, \Pr[\mathrm{Erg}_V\langle V, P\rangle(x) = 1] \leq 1/3$, *(Korrektheit)*

   wobei die Interaktion eine probabilistische $k(|x|)$-Runden-Interaktion ist, und sich die Wahrscheinlichkeiten auf die Verteilung der von V benutzten Zufallsbits beziehen.
2. **Definition.** $\mathbf{IP}[k]$ sei die Menge aller Sprachen mit einem k-Runden probabilistischen interaktiven Beweissystem.
3. **Definition.** Sei $\mathbf{IP} = \bigcup_{c \geq 1} \mathbf{IP}[n \mapsto n^c]$.

Robustheit der Definition

1. **Lemma.** Sei $s \geq 1$. Die Klasse **IP** bleibt gleich, wenn man in der Definition die Parameter $2/3$ und $1/3$ durch $1 - 2^{-n^s}$ bzw. 2^{-n^s} ersetzt, wobei n die Eingabelänge sei. *(Beweis → Übungszettel)*
2. Wir haben die Rechenleistung des Beweisers nicht eingeschränkt. Er könnte sogar eine nichtberechenbare Funktion sein!

   Man kann aber zeigen, dass sich für jeden Prüfer in $\mathrm{poly}(|x|)$ Platz ein „bestmöglicher" Beweiser errechnen lässt, der Beweise vorschlägt, die mit maximaler Wahrscheinlichkeit vom Prüfer akzeptiert werden.

   Inbesondere gilt **IP** $\subseteq$ **PSPACE**. *(Beweis → Übungszettel)*

Isomorphismus von Graphen

1. **Testfrage.** Wann würden Sie zwei Graphen *isomorph* nennen?
2. **Definition.** Das *Graphen-Isomorphismus-Problem* GI ist so definiert, dass uns zwei Graphen (in geeigneter Kodierung) gegeben werden. Wir sollen dann entscheiden, ob sie isomorph sind.
3. **Testfrage.** Warum ist klar, dass $\mathrm{GI} \in \mathbf{NP}$?
4. Es ist unbekannt, ob $\mathrm{GI} \in \mathbf{P}$ oder ob GI **NP**-vollständig ist.

Nichtisomorphismus von Graphen

1. **Definition.** GNI ist das Komplement von GI.
2. **Satz.** GNI $\in$ **IP**[2].
3. **Idee.** Der Beweis benutzt die Idee des Sockenbeispiels.
   - P versucht, V davon zu überzeugen, dass er die beiden Graphen G_1 und G_2 auseinanderhalten kann (sogar modulo Isomorphie!).
   - V „vermengt“ die beiden Graphen.
   - P müsste mit seiner unbegrenzten Rechenleistung die beiden Graphen weiterhin unterscheiden können, falls sie sich *wirklich* unterscheiden.

Nichtisomorphismus von Graphen

1. **Satz.** GNI $\in$ **IP**[2].
2. **Beweis.** Seien G_1 und G_2 Graphen mit gleich vielen Knoten.
   - V: Wähle zufällig $i \in \{1,2\}$. Wähle eine zufällige Permutation und wende sie auf G_i an. Schicke das so entstandene H an P.
   - P: Finde j, so dass G_j zu H isomorph ist. Schicke j an V.
   - V: Falls $i = j$, akzeptiere; sonst verwirf.
3. Falls G_1 und G_2 wirklich verschieden sind, kann P problemlos immer das korrekte j schicken. Sonst kann P das richtige j nur mit 50% Wahrscheinlichkeit raten.
4. Die Wahrscheinlichkeit kann man durch mehrfache Ausführung des Protokolls unter den *Korrektheit*sparameter $1/3$ drücken. □

Nichtisomorphismus von Graphen

1. **Testfrage.** Warum reicht es nicht, wenn V einfach zufällig ein G_i auswählt und es unverändert an P schickt? Warum ist es wichtig, G_i vorher noch zu H zu permutieren?

Nichtisomorphismus von Graphen

1. **Testfrage.** Warum reicht es nicht, wenn V einfach zufällig ein G_i auswählt und es unverändert an P schickt? Warum ist es wichtig, G_i vorher noch zu H zu permutieren?
2. **Tipp.** Falls wir nur prüfen wollten, ob G_1 und G_2 *verschieden* sind, würde das reichen. Wir wollen aber prüfen, ob sie *nicht isomorph* sind.

Nichtisomorphismus von Graphen

1. **Testfrage.** Warum reicht es nicht, wenn V einfach zufällig ein G_i auswählt und es unverändert an P schickt? Warum ist es wichtig, G_i vorher noch zu H zu permutieren?
2. **Tipp.** Falls wir nur prüfen wollten, ob G_1 und G_2 *verschieden* sind, würde das reichen. Wir wollen aber prüfen, ob sie *nicht isomorph* sind.
3. **Antwort.** Damit P uns davon überzeugen kann, muss er die Graphen auch dann noch unterscheiden können, wenn jegliche Information entfernt wurde, die unter Isomorphie nicht erhalten bleibt. Durch die Anwendung einer zufälligen Permutation auf G_i wird all diese Information „verwischt“.

Öffentliche Münzwürfe

1. Das letzte Protokoll scheint nur zu funktionieren, solange wir die Ergebnisse der Münzwürfe geheimhalten. Was, wenn die Münzen öffentlich sind?
2. **Definition.** Für jedes k sei $\mathbf{AM}[k]$ die Menge von Sprachen in $\mathbf{IP}[k]$, für die der Prüfer nur Zufallsbits verschickt, und alle verwendeten Zufallsbits mitschicken *muss*.
3. Für $k > 2$ sieht der Beweiser *nicht* von Anfang an alle Zufallsbits.
4. **Bemerkung.** „**A**“ und „**M**“ stehen für „Artus“ und „Merlin“.

Öffentliche Münzwürfe

1. **Definition.** Wir schreiben **AM** für $\mathbf{AM}[2]$.
2. **Intuition.** $\mathbf{AM} = \mathbf{AM}[2]$ sind die Sprachen, die von einem Protokoll erkannt werden, bei dem
   - der Prüfer zunächst Münzen wirft,
   - die Ergebnisse dann an den Beweiser schickt,
   - der Beweiser dann mit einem Beweis antwortet,
   - und der Prüfer nun *deterministisch* seine Entscheidung fällt.
3. Analoges gilt für $\mathbf{AM}[3] = \mathbf{MAM}$ und $\mathbf{AM}[4] = \mathbf{AMAM}$, usw.
4. **Proposition.** Es gilt $\mathbf{AM}[k] = \mathbf{AM}[2]$ für alle $k \geq 2$.

(Beweis → Übungszettel)

Nichtisomorphismus von Graphen

1. Wie stark „schadet“ es, wenn die Zufallsbits öffentlich sind?

Nichtisomorphismus von Graphen

1. Wie stark „schadet“ es, wenn die Zufallsbits öffentlich sind?
2. **Satz.** $\mathrm{GNI} \in \mathbf{AM}[2]$.

   (Beweis siehe unten.)
3. Betrachte nur Graphen deren n Knoten mit $1, \ldots, n$ beschriftet sind. Jeder solche Graph hat höchstens $n!$ viele isomorphe solche Graphen. **Testfrage.** Warum?

Nichtisomorphismus von Graphen

1. Wie stark „schadet“ es, wenn die Zufallsbits öffentlich sind?
2. **Satz.** $\mathrm{GNI} \in \mathbf{AM}[2]$.

   (Beweis siehe unten.)
3. Betrachte nur Graphen deren n Knoten mit $1, \ldots, n$ beschriftet sind. Jeder solche Graph hat höchstens $n!$ viele isomorphe solche Graphen. **Testfrage.** Warum?
4. Nimm zunächst an, es seien *genau* $n!$. Betrachte

   $$S = \{H : H \text{ ist isomorph zu } G_1 \text{ oder } G_2\}.$$

   Testfrage. Was ist die Kardinalität von S? Wie hängt die Antwort davon ab, ob G_1 und G_2 isomorph sind?

Nichtisomorphismus von Graphen

1. Wie stark „schadet“ es, wenn die Zufallsbits öffentlich sind?

2. **Satz.** $\mathrm{GNI} \in \mathbf{AM}[2]$.

   (Beweis siehe unten.)

3. Betrachte nur Graphen deren n Knoten mit $1, \ldots, n$ beschriftet sind. Jeder solche Graph hat höchstens $n!$ viele isomorphe solche Graphen. **Testfrage.** Warum?

4. Nimm zunächst an, es seien *genau* $n!$. Betrachte

   $$S = \{H : H \text{ ist isomorph zu } G_1 \text{ oder } G_2\}.$$

   Testfrage. Was ist die Kardinalität von S? Wie hängt die Antwort davon ab, ob G_1 und G_2 isomorph sind?

   Antwort. Bei Isomorphie $\#S = n!$, sonst $\#S = 2 \cdot n!$. Das möchten wir gleich als Kriterium für Isomorphie verwenden!

Nichtisomorphismus von Graphen

1. Wie stark „schadet“ es, wenn die Zufallsbits öffentlich sind?

2. **Satz.** $\mathrm{GNI} \in \mathbf{AM}[2]$.

   (Beweis siehe unten.)

3. Betrachte nur Graphen deren n Knoten mit $1, \ldots, n$ beschriftet sind. Jeder solche Graph hat höchstens $n!$ viele isomorphe solche Graphen. **Testfrage.** Warum?

4. Nimm zunächst an, es seien *genau* $n!$. Betrachte

   $$S = \{H : H \text{ ist isomorph zu } G_1 \text{ oder } G_2\}.$$

   Testfrage. Was ist die Kardinalität von S? Wie hängt die Antwort davon ab, ob G_1 und G_2 isomorph sind?

   Antwort. Bei Isomorphie $\#S = n!$, sonst $\#S = 2 \cdot n!$. Das möchten wir gleich als Kriterium für Isomorphie verwenden!

5. Es gibt auch Graphen, die echt weniger als $n!$ viele isomorphe Graphen haben. **Testfrage.** Geben Sie ein Beispiel an!

Nichtisomorphismus von Graphen

1. Ein Graph G hat echt weniger als $n!$ viele isomorphe Graphen, wenn es eine „echte“ Permutation $\sigma \neq \mathrm{id}$ gibt mit $\sigma(G) = G$.

   Intuition. Wenn man Knoten umbenennen kann, ohne dass sich dabei der Graph ändert, gibt es weniger isomorphe Graphen.
2. Solche Permutationen nennt man die *Automorphismen von* G.
3. Wir ändern die Definition von S zu
$$S = \left\{ (H, \pi) \colon \begin{array}{c} \pi \text{ ist Automorphismus von } H \text{ und} \\ H \text{ ist isomorph zu } G_1 \text{ oder } G_2 \end{array} \right\}.$$
   Dann gilt immer $\#S \in \{n!,\ 2 \cdot n!\}$. *(Beweis → Übungszettel)*
4. **Testfrage.** Wie sieht ein Zertifikat für „$(H, \pi) \in S$“ aus?

Das Mengen-Unterschranken-Protokoll

1. Um den Prüfer vom Nichtisomorphismus zu überzeugen, muss der Beweiser den Prüfer davon überzeugen, dass $\#S = 2 \cdot n!$.
2. Dafür benutzen wir das *Mengen-Unterschranken-Protokoll*.
3. Es ist für S anwendbar, für die es Zertifikate für Mitgliedschaft gibt, die der Prüfer (mit seinen Ressourcen) überprüfen kann.
4. Der Beweiser kann mit dem Mengen-Unterschranken-Protokoll den Prüfer von der ungefähren Größe solcher S überzeugen:
   - Nimm an $\#S \geq K$:
     - Dann gibt es ein Zertifikat für die Behauptung „$\#S \geq K$“, das der Prüfer mit hoher Wahrscheinlichkeit akzeptieren wird.
     - Er kann also von korrekten Unterschranken überzeugt werden.
   - Nimm an $\#S < K/2$.
     - Dann gibt es *kein* Zertifikat für die Behauptung „$\#S \geq K$“, das der Prüfer mit hoher Wahrscheinlichkeit akzeptieren wird.
     - *Sehr* falsche Unterschranken werden also nicht akzeptiert.

Paarweise unabhängige Hashfunktionen

1. Wir müssen also das Mengen-Unterschranken-Protokoll beschreiben und beweisen, dass es tatsächlich funktioniert.
2. **Definition.** Sei $\mathscr{H}_{n,k}$ eine Menge von Funktionen von $\{0,1\}^n$ nach $\{0,1\}^k$. Wir nennen $\mathscr{H}_{n,k}$ eine *Menge paarweise unabhängiger Hashfunktionen* falls für alle $x, x' \in \{0,1\}^n$ mit $x \neq x'$ und alle $y, y' \in \{0,1\}^k$ gilt, dass

$$\Pr_{h \in \mathscr{H}_{n,k}} [h(x) = y \wedge h(x') = y'] = 2^{-2k}.$$

   Dabei wird h zufällig aus $\mathscr{H}_{n,k}$ gezogen.
3. **Intuition.** Im Schnitt sagt das Wissen, wohin ein x abgebildet wird, *gar nichts* darüber aus, wohin ein $x' \neq x$ abgebildet wird.

Paarweise unabhängige Hashfunktionen

1. **Proposition.** Für Mengen $\mathscr{H}_{n,k}$ paarweise unabhängiger Hashfunktionen gilt auch für alle $x \in \{0,1\}^n$ und $y \in \{0,1\}^k$, dass
$$\Pr_{h \in \mathscr{H}_{n,k}}[h(x) = y] = 2^{-k}.$$
(Beweis → Übungszettel)

2. **Intuition.** Die Gleichverteiltheit bezüglich zweier Argumente x und x', die in der Definition gefordert wird, überträgt sich auch auf den Fall nur eines Arguments x.

Paarweise unabhängige Hashfunktionen

1. Wir zeigen, dass solche Mengen von Funktionen existieren.
2. **Satz.** Sei n beliebig. Für $a, b \in \mathbb{F}_{2^n}$ sei $h_{a,b} \colon \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ definiert via $x \mapsto ax + b$. Dann ist $\mathscr{H}_{n,n} = \{h_{a,b}\}_{a,b \in \mathbb{F}_{2^n}}$ eine Menge paarweise unabhängiger Hashfunktionen.

   (Beweis siehe unten.)
3. **Korollar.** Es gibt $\mathscr{H}_{n,k}$ für alle n, k.

   Beweis.
   - Falls $n < k$, benutze $\mathscr{H}_{k,k}$, indem an die zu kurzen n-Bit-Eingaben $k - n$ konstante Bits angehängt werden.
   - Falls $n > k$, benutze $\mathscr{H}_{n,n}$, indem von den zu langen n-Bit-Ausgaben $n - k$ Bits abgeschnitten werden.
4. **Testfrage.** Wie aufwändig ist es, die $h_{a,b}$ zu berechnen?

Paarweise unabhängige Hashfunktionen

1. **Satz.** Sei n beliebig. Für $a, b \in \mathbb{F}_{2^n}$ sei $h_{a,b} \colon \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}$ definiert via $x \mapsto ax + b$. Dann ist $\mathscr{H}_{n,n} = \{h_{a,b}\}_{a,b \in \mathbb{F}_{2^n}}$ eine Menge paarweise unabhängiger Hashfunktionen.
2. **Beweis.**
   - Wir bestimmen, für wie viele $h \in \mathscr{H}_{n,n}$ es für feste $x, x' \in \mathbb{F}_{2^n}$ mit $x \neq x'$ und $y, y' \in \mathbb{F}_{2^n}$ passiert, dass $h(x) = y$ und $h(x') = y'$.
   - Das gilt genau dann, wenn 1 $ax + b = y$ und 2 $ax' + b = y'$.
   - „1 − 2“ liefert $a = \frac{y-y'}{x-x'}$. Folglich $b = y - ax = y - \frac{y-y'}{x-x'} \cdot x$.
   - Man sieht, dass es genau eine geeignete Wahl von a und b gibt, also genau ein passendes $h \in \mathscr{H}_{n,n}$.
   - Andererseits gilt $\#\mathscr{H}_{n,n} = 2^{2n}$, da es je 2^n Wahlmöglichkeiten für a und b gibt. Es folgt, dass $\Pr_{h \in \mathscr{H}_{n,k}}[h(x) = y \wedge h(x') = y'] = 2^{-2n}$.

□

Das Mengen-Unterschranken-Protokoll

1. **Protokoll.** Wähle $k \in \mathbb{N}$ mit $2^{k-2} < K \leq 2^{k-1}$.
   - V: Wähle zufällig $h \in \mathscr{H}_{n,k}$ und $y \in \{0,1\}^k$ und sende sie an P.
   - P: Finde $x \in S$ mit $h(x) = y$ und sende x und ein Zertifikat für „$x \in S$“ an V.
   - V: Akzeptiere, falls $h(x) = y$ und das Zertifikat für „$x \in S$“ gültig ist. Sonst verwirf.
2. Falls ein geeignetes x existiert, dann kann P damit V überzeugen.
3. Um zu zeigen, dass das Protokoll *korrekt* funktioniert, müssen wir also zeigen, dass die Wahrscheinlichkeit, dass x existiert, im Fall $\#S \geq K$ erheblich größer ist, als im Fall $\#S < K/2$.

Das Mengen-Unterschranken-Protokoll

1. **Behauptung.** Sei $S \subseteq \{0,1\}^n$ mit $\#S \leq 2^{k-1}$ und $p = {}^{\#S}/_{2^k}$. Dann
$$p \geq \Pr_{h \in \mathscr{H}_{n,k},\, y \in \{0,1\}^k}[\exists x \in S : h(x) = y] \geq {}^3/_4 \cdot p.$$
(Beweis siehe unten.)
2. Betrachte dann S_1 und S_2 mit $\#S_1 \geq K$ und $\#S_2 < {}^K/_2$:
$$\Pr_{h \in \mathscr{H}_{n,k},\, y \in \{0,1\}^k}[\exists x \in S_1 : h(x) = y] \geq 3 \cdot {}^{\#S_1}/_{2^{k+2}} \geq 3 \cdot {}^K/_{2^{k+2}}$$
Im Gegensatz dazu,
$$\Pr_{h \in \mathscr{H}_{n,k},\, y \in \{0,1\}^k}[\exists x \in S_2 : h(x) = y] \leq {}^{\#S_2}/_{2^k} \leq {}^K/_{2^{k+1}} = 2 \cdot {}^K/_{2^{k+2}}.$$
3. Der Unterschied um den Faktor ${}^3/_2$ reicht durch wiederholte Ausführung des Protokolls aus, um beide Fälle zu unterscheiden.
4. **Testfrage.** Warum stört die Annahme „$\#S \leq 2^{k-1}$“ nicht?

Beweis der Behauptung

1. **Behauptung.** Sei $S \subseteq \{0,1\}^n$ mit $\#S \leq 2^{k-1}$ und $p = {}^{\#S}/_{2^k}$. Dann
$$p \geq \Pr_{h \in \mathscr{H}_{n,k},\, y \in \{0,1\}^k}[\exists x \in S \colon h(x) = y] \geq {}^3/_4 \cdot p.$$

2. **Beweis.**
   - **Testfrage.** Warum gilt die obere Schranke? Betrachte $\#h(S)$!
   - Für die untere Schranke, wähle zunächst ein festes y.
   - Für $x \in S$, schreibe E_x für das Ereignis „$h(x) = y$“.
   - Durch Verwendung des Inklusions/Exklusions-Prinzips, der Proposition von Folie 116, und der Definition von Mengen paarweise unabhängiger Hashfunktionen erhalten wir
$$\begin{aligned} \Pr_{h \in \mathscr{H}_{n,k}}[\exists x \in S \colon h(x) = y] &= \Pr_{h \in \mathscr{H}_{n,k}}[\textstyle\bigcup_{x \in S} E_x] \\ &\geq \textstyle\sum_{x \in S} \Pr_h[E_x] - {}^1/_2 \sum_{x, x' \in S,\, x \neq x'} \Pr_h[E_x \cap E_{x'}] \\ &\geq {}^{\#S}/_{2^k} - {}^1/_2 \cdot {}^{\#S^2}/_{2^{2k}} \\ &= {}^{\#S}/_{2^k} \cdot (1 - {}^{\#S}/_{2^{k+1}}) \qquad \geq {}^3/_4 \cdot p. \end{aligned}$$
   - Durchschnitt bilden über alle y liefert die Behauptung. □

Nichtisomorphismus von Graphen

1. **Satz (Wiederholung).** $\mathrm{GNI} \in \mathbf{AM} = \mathbf{AM}[2]$.
2. **Beweis.**
   - Wir führen das Mengen-Unterschranken-Protokoll mehrfach aus.
   - Laut Chernoff-Ungleichung erreicht man durch konstant viele Wiederholungen die erforderlichen Fehlerschranken $2/3$ bzw. $1/3$.
   - Die genaue Rechnung sparen wir uns.

   - **Wichtiger:** Warum ist das überhaupt ein **AM**-Protokoll?

Nichtisomorphismus von Graphen

1. **Satz (Wiederholung).** $\mathrm{GNI} \in \mathbf{AM} = \mathbf{AM}[2]$.
2. **Beweis.**
   - Wir führen das Mengen-Unterschranken-Protokoll mehrfach aus.
   - Laut Chernoff-Ungleichung erreicht man durch konstant viele Wiederholungen die erforderlichen Fehlerschranken $2/3$ bzw. $1/3$.
   - Die genaue Rechnung sparen wir uns.
   - **Wichtiger:** Warum ist das überhaupt ein **AM**-Protokoll?
     - **Testfrage.** Warum reicht es, wenn P nur seine Zufallsbits schickt?

Nichtisomorphismus von Graphen

1. **Satz (Wiederholung).** $\mathrm{GNI} \in \mathbf{AM} = \mathbf{AM}[2]$.
2. **Beweis.**
   - Wir führen das Mengen-Unterschranken-Protokoll mehrfach aus.
   - Laut Chernoff-Ungleichung erreicht man durch konstant viele Wiederholungen die erforderlichen Fehlerschranken $2/3$ bzw. $1/3$.
   - Die genaue Rechnung sparen wir uns.
   - **Wichtiger:** Warum ist das überhaupt ein **AM**-Protokoll?
     - **Testfrage.** Warum reicht es, wenn P nur seine Zufallsbits schickt?

       Antwort. Weil die Zufallsbits h und y vollständig beschreiben.

Nichtisomorphismus von Graphen

1. **Satz (Wiederholung).** $\mathrm{GNI} \in \mathbf{AM} = \mathbf{AM}[2]$.
2. **Beweis.**
   - Wir führen das Mengen-Unterschranken-Protokoll mehrfach aus.
   - Laut Chernoff-Ungleichung erreicht man durch konstant viele Wiederholungen die erforderlichen Fehlerschranken $2/3$ bzw. $1/3$.
   - Die genaue Rechnung sparen wir uns.
   - **Wichtiger:** Warum ist das überhaupt ein **AM**-Protokoll?
     - **Testfrage.** Warum reicht es, wenn P nur seine Zufallsbits schickt?

       Antwort. Weil die Zufallsbits h und y vollständig beschreiben.
     - **Testfrage. AM** erlaubt nur zwei Runden. Wir benötigen aber mehrere Wiederholungen des Protokolls. Wie geht es trotzdem?

Nichtisomorphismus von Graphen

1. **Satz (Wiederholung).** $\mathrm{GNI} \in \mathbf{AM} = \mathbf{AM}[2]$.
2. **Beweis.**
   - Wir führen das Mengen-Unterschranken-Protokoll mehrfach aus.
   - Laut Chernoff-Ungleichung erreicht man durch konstant viele Wiederholungen die erforderlichen Fehlerschranken $2/3$ bzw. $1/3$.
   - Die genaue Rechnung sparen wir uns.
   - **Wichtiger:** Warum ist das überhaupt ein **AM**-Protokoll?
     - **Testfrage.** Warum reicht es, wenn P nur seine Zufallsbits schickt?

       Antwort. Weil die Zufallsbits h und y vollständig beschreiben.
     - **Testfrage. AM** erlaubt nur zwei Runden. Wir benötigen aber mehrere Wiederholungen des Protokolls. Wie geht es trotzdem?

       Antwort. Da die Wahlen von h und y in jeder Runde rein zufällig sind, und nicht von den bisherigen Ereignissen abhängen, kann der Prüfer auch gleich in der *ersten* Runde *mehrere* h's und *mehrere* y's wählen, und sie alle auf einmal an den Beweiser senden. □

Geheime Münzwürfe durch öffentliche simulieren

1. Wir haben gesehen, dass es manchmal keinen Unterschied macht, ob die Münzwürfe geheim oder öffentlich sind.
2. Das kann selbst für Probleme wie Graph-Nichtisomorphismus gelten, deren natürlichstes Protokoll eindeutig auf geheime Münzwürfe angewiesen ist.
3. Allgemein kann man mit wenigen zusätzlichen Runden geheime Münzwürfe durch öffentliche ersetzen.
4. **Satz.** Sei $k\colon \mathbb{N} \to \mathbb{N}$ mit $k(n)$ berechenbar in $\mathrm{poly}(n)$. Dann gilt
$$\mathbf{IP}[k] \subseteq \mathbf{AM}[k+2].$$
5. Der Beweis ist für diese Vorlesung zu komplex.

Kapitelzusammenfassung

1. Wir haben interaktive Protokolle zum Erkennen von Sprachen definiert. Dabei überzeugt ein leistungsstarker Beweiser einen leistungsschwachen Prüfer, dass Wörter in der Sprache sind.
2. Deterministische Protokolle bringen keinen Mehrwert. Deshalb haben wir erlaubt, in den Protokollen Zufallsbits zu verwenden.
3. **IP** enthält die Sprachen, die von polynomiellen randomisierten Protokollen erkannt werden.
4. Für jedes k enthält $\mathbf{AM}[k] \subseteq \mathbf{IP}[k]$ die Sprachen, für die es Protokolle mit öffentlichen Münzwürfen gibt.
5. Graph-Nichtisomorphismus liegt in $\mathbf{AM} = \mathbf{AM}[2]$. Zum Beweis haben wir das Mengen-Unterschranken-Protokoll und Familien von Hashfunktionen verwendet.
6. Öffentliche Münzen können geheime ersetzen, wenn man bereit ist, wenige zusätzliche Interaktionsrunden in Kauf zu nehmen.

9

Interaktive Beweise und die Polynomialzeithierarchie

Ist Graph-Isomorphismus **NP**-vollständig?

1. Wie erwähnt ist unbekannt, ob GI **NP**-vollständig ist.
2. Man kann jedoch zeigen, dass die Polynomialzeit-Hierarchie kollabiert, falls dem so sein sollte.
3. **Satz.** Falls GI **NP**-vollständig ist, so gilt $\mathbf{\Sigma}_2 = \mathbf{\Pi}_2$.
4. Der Beweis wird auf einem interaktiven Protokoll aufbauen.

Vorbereitungen

1. **Definition.** Sei TAUT die Menge der Booleschen Formeln, die mit jeder Belegung erfüllt sind.
2. **Testfrage.** Warum ist TAUT **coNP**-vollständig für $\leq_{\mathrm{m}}^{\mathrm{p}}$?

3. **Proposition.** 2QBF ist Σ_2-vollständig für $\leq_{\mathrm{m}}^{\mathrm{p}}$.

   (Beweis → Übungszettel)
4. Somit genügt es, zu zeigen, dass falls GI **NP**-vollständig ist, ein $B \in \mathbf{P}$ und Polynome p_1, p_2 existieren, so dass

   $$F \in \text{2QBF wahr} \iff \forall^{p_1(|F|)} x \, \exists^{p_2(|F|)} y \, \langle F, x, y \rangle \in B.$$

   Testfrage. Warum beweist das den Satz?

Vorbereitungen

1. Im Beweis verwenden wir ein **AM**-Protokoll für GNI.
2. Wir werden annehmen, dass das **AM**-Protokoll für GNI bezüglich *Vollständigkeit* Fehlerrate 0 hat.
3. **Zur Erinnerung:**
   - Unser **IP**-Protokoll für GNI hatte zwar diese Eigenschaft,
   - unser **AM**-Protokoll jedoch nicht.
   - Laut dem folgendem Satz existieren solche **AM**-Protokolle aber.
4. **Satz.** Die Klasse **AM** bleibt unverändert, wenn man *perfekte Vollständigkeit* fordert, also dass das Protokoll Wörter, die in der fraglichen Sprache enthalten sind, *immer* akzeptiert. (ohne Beweis)

Beweis

1. **Zu zeigen:** Es gibt $B \in \mathbf{P}$ und Polynome p_1, p_2, so dass
$$F \in \text{2QBF wahr} \iff \forall^{p_1(|F|)} x \, \exists^{p_2(|F|)} y \, \langle F, x, y \rangle \in B.$$
2. **Beweis.**
   - Wir nehmen an, GI sei **NP**-vollständig.
   - Dann ist GNI **coNP**-vollständig.
   - Also gilt TAUT $\leq_{\mathrm{m}}^{\mathrm{p}}$ GNI via $f \in \mathbf{FP}$.
   - Ein $F \in$ 2QBF hat o.E. die Form $\exists x \in \{0,1\}^n \, \forall y \in \{0,1\}^{n'} \, G(x,y)$.
   - Für jedes *feste* x kann $\forall y \, G(x,y)$ als Instanz von TAUT verstanden und somit via f in eine Instanz von GNI kodiert werden.
   - Also gilt, wenn f' eine geeignet modifizierte Version von f ist,
   $$F \text{ ist wahr} \Leftrightarrow \exists x \in \{0,1\}^n \, f'(F,x) \in \text{GNI}.$$
   - **Unser Ziel:** Die Bedingung auf der rechten Seite umformen in eine Bedingung der Form „$\forall^{p_1(|F|)} x \, \exists^{p_2(|F|)} y \, \langle F, x, y \rangle \in B$“ für ein $B \in \mathbf{P}$ und Polynome p_1, p_2.

Beweis

1. **Unser Ziel:** Die Bedingung „$\exists x \in \{0,1\}^n f'(F,x) \in \mathrm{GNI}$“ in die Form „$\forall^{p_1(|F|)} x \, \exists^{p_2(|F|)} y \, \langle F,x,y\rangle \in B$“ äquivalent umformen für ein $B \in \mathbf{P}$ und Polynome p_1, p_2.
2. Dazu verwenden wir, dass es ein **AM**-Protokoll für GNI gibt, dessen Fehlerrate für *Vollständigkeit* 0 ist.
3. Weiter können wir durch wiederholtes Ausführen des Protokolls annehmen, die Fehlerrate für *Korrektheit* sei echt kleiner als 2^{-n}.
4. Sei V der Prüfer in diesem Protokoll, m die Anzahl von V's Münzwürfen und m' die Länge der Antwort des Beweisers.
5. **Behauptung.** F ist wahr genau dann, wenn

$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x), r, a) = 1.$$

Beweis

1. **Behauptung (Wiederholung).** F ist wahr genau dann, wenn
$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x), r, a) = 1.$$
2. **Beweis.**
   - **Testfrage.** Warum gilt „$\Longrightarrow$“? Wie kommt „$\exists x$“ *hinter* „$\forall r$“?
   - Für „$\Longleftarrow$“, sei F unwahr. Dann $\forall x \in \{0,1\}^n f'(F,x) \notin \mathrm{GNI}$.
   - Die Fehlerrate für Korrektheit, d.h. die Wahrscheinlichkeit für ein „falsch positives“ Ergebniss des Protokolls, ist $< 2^{-n}$.
   - **Es gilt:** Es gibt mindestens ein festes $r \in \{0,1\}^m$, für das das Protokoll für *alle* x das richtige Ergebnis „nein“ liefert.

Beweis

1. **Behauptung (Wiederholung).** F ist wahr genau dann, wenn
$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x), r, a) = 1.$$
2. **Beweis.**
   - **Testfrage.** Warum gilt „$\Longrightarrow$“? Wie kommt „$\exists x$“ *hinter* „$\forall r$“?
   - Für „$\Longleftarrow$“, sei F unwahr. Dann $\forall x \in \{0,1\}^n f'(F,x) \notin \mathrm{GNI}$.
   - Die Fehlerrate für Korrektheit, d.h. die Wahrscheinlichkeit für ein „falsch positives“ Ergebniss des Protokolls, ist $< 2^{-n}$.
   - **Es gilt:** Es gibt mindestens ein festes $r \in \{0,1\}^m$, für das das Protokoll für *alle* x das richtige Ergebnis „nein“ liefert.
     - Es gibt $2^m \cdot 2^n$ Paare (r,x). Wenn es für jedes r mindestens ein x gibt, so dass (r,x) zu „falsch positiv“ führt, so sind das $\geq 2^m$ Paare.
     - Diese verteilen sich auf die 2^n x'e. Für ein durchschnittliches x sind das $2^m/2^n$ Paare von allen 2^m Paaren, in denen x auftaucht.
     - Also gibt es ein x mit $\Pr[\text{„falsch positiv“}] \geq 2^{m-n}/2^m = 2^{-n}$.
     - Dies widerspricht der obigen Wahl des Protokolls.

Beweis

1. **Behauptung (Wiederholung).** F ist wahr genau dann, wenn
$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x), r, a) = 1.$$
2. **Beweis.**
   - **Testfrage.** Warum gilt „$\Longrightarrow$“? Wie kommt „$\exists x$“ *hinter* „$\forall r$“?
   - Für „$\Longleftarrow$“, sei F unwahr. Dann $\forall x \in \{0,1\}^n f'(F,x) \notin \mathrm{GNI}$.
   - Die Fehlerrate für Korrektheit, d.h. die Wahrscheinlichkeit für ein „falsch positives“ Ergebniss des Protokolls, ist $< 2^{-n}$.
   - **Es gilt:** Es gibt mindestens ein festes $r \in \{0,1\}^m$, für das das Protokoll für *alle* x das richtige Ergebnis „nein“ liefert.

Beweis

1. **Behauptung (Wiederholung).** F ist wahr genau dann, wenn
$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x),r,a) = 1.$$
2. **Beweis.**
   - **Testfrage.** Warum gilt „$\Longrightarrow$“? Wie kommt „$\exists x$“ *hinter* „$\forall r$“?
   - Für „$\Longleftarrow$“, sei F unwahr. Dann $\forall x \in \{0,1\}^n f'(F,x) \notin \mathrm{GNI}$.
   - Die Fehlerrate für Korrektheit, d.h. die Wahrscheinlichkeit für ein „falsch positives“ Ergebniss des Protokolls, ist $< 2^{-n}$.
   - **Es gilt:** Es gibt mindestens ein festes $r \in \{0,1\}^m$, für das das Protokoll für *alle* x das richtige Ergebnis „nein“ liefert.
   - **Also:** $\exists r \in \{0,1\}^m \, \forall x \in \{0,1\}^n \, \forall a \in \{0,1\}^{m'} \; V(f'(F,x),r,a) = 0$.
   - Da dies die Negation der Behauptung ist, sind wir fertig. $\square$

Beweis

1. **Behauptung (Wiederholung).** F ist wahr genau dann, wenn
$$\forall r \in \{0,1\}^m \, \exists x \in \{0,1\}^n \, \exists a \in \{0,1\}^{m'} \; V(f'(F,x),r,a) = 1.$$
2. Es bleibt noch zu begründen, warum die Bedingung in der Behauptung die gesuchte Form „$\forall^{p_1(|F|)} x \, \exists^{p_2(|F|)} y \, \langle F,x,y\rangle \in B$“ für ein $B \in \mathbf{P}$ und Polynome p_1, p_2 hat.
3. Da V in Polynomialzeit arbeitet und $f' \in \mathbf{FP}$ gilt,
   - liegt $B := \{\langle F,r,\langle x,a\rangle\rangle : V(f'(F,x),r,a) = 1\}$ in $\mathbf{P}$,
   - sind m und $n+m'$ o.E. jeweils polynomiell in $|f'(F,x)|$, und
   - sind m und $n+m'$ auch polynomiell in $|F|$.
4. Insgesamt haben wir $\mathrm{2QBF} \in \mathbf{\Pi}_2$ gezeigt.
5. Da $\mathbf{\Pi}_2$ unter $\leq_{\mathrm{m}}^{\mathrm{p}}$ abgeschlossen ist, und 2QBF $\mathbf{\Sigma}_2$-vollständig ist, folgt $\mathbf{\Sigma}_2 \subseteq \mathbf{\Pi}_2$, und schlussendlich $\mathbf{\Sigma}_2 = \mathbf{\Pi}_2$. □

10

IP = PSPACE

IP = PSPACE

1. **Satz. IP = PSPACE**
2. Wir haben bereits gesehen, dass **IP** $\subseteq$ **PSPACE** gilt.
3. Wir müssen also noch **PSPACE** $\subseteq$ **IP** zeigen.
4. Dafür genügt es, QBF $\in$ **IP** zu zeigen.

   Testfrage. Welche Abschlusseigenschaft von **IP** ist hier wichtig?

Annäherung

1. Zunächst zeigen wir, dass es ein **IP**-Protokoll gibt, mit dem ein Prüfer von der Anzahl der erfüllenden Belegungen einer Formel in 3SAT überzeugt werden kann.
2. Ähnliche Ideen können benutzt werden, um QBF $\in$ **IP** zu zeigen. Es kommen nur einige lästige technische Komplikationen hinzu.

Erfüllende Belegungen zählen

1. **Definition.** Sei #3SAT die Sprache

   $$\{\langle F, k\rangle : F \in 3\mathrm{SAT} \text{ mit genau } k \in \mathbb{N} \text{ erfüllenden Belegungen}\}.$$

2. **Satz.** $\#3\mathrm{SAT} \in \mathbf{IP}$.
3. **Korollar.** $\overline{3\mathrm{SAT}} \in \mathbf{IP}$.

   Testfrage. Wie folgt das?
   Testfrage. Was folgt für das Verhältnis von **coNP** zu **IP**?

Erfüllende Belegungen zählen

1. **Definition.** Sei #3SAT die Sprache
$$\{\langle F,k\rangle : F \in \mathrm{3SAT} \text{ mit genau } k \in \mathbb{N} \text{ erfüllenden Belegungen}\}.$$
2. **Satz.** $\#\mathrm{3SAT} \in \mathbf{IP}$.
3. **Korollar.** $\overline{\mathrm{3SAT}} \in \mathbf{IP}$.

   Testfrage. Wie folgt das?
   Testfrage. Was folgt für das Verhältnis von **coNP** zu **IP**?

4. Für den Beweis des Satzes werden Formeln mit den Variablen $X_1,\dots,X_n$ in Polynome in den Variablen $X_1,\dots,X_n$ verwandelt.

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.
5. **Testfrage.** Wie hoch ist der Grad von P_F maximal?

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.
5. **Testfrage.** Wie hoch ist der Grad von P_F maximal?

   Antwort. Höchstens $3m$.

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.
5. **Testfrage.** Wie hoch ist der Grad von P_F maximal?

   Antwort. Höchstens $3m$.
6. **Testfrage.** Wie lässt sich P_F polynomiell in m speichern?

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.
5. **Testfrage.** Wie hoch ist der Grad von P_F maximal?

   Antwort. Höchstens $3m$.
6. **Testfrage.** Wie lässt sich P_F polynomiell in m speichern?

   Antwort. Indem wir auf das Ausmultiplizieren verzichten.

Polynome aus 3SAT-Formeln

1. Eine 3SAT-*Klausel* lässt sich als Polynom darstellen.
2. **Testfrage.** Zeigen Sie am Beispiel $C = (X_1 \vee \neg X_2 \vee X_3)$, wie!

   Antwort. C wird von Belegung (b_1, b_2, b_3) erfüllt genau dann, wenn das Polynom $P_C(X_1, X_2, X_3) := 1 - (1 - X_1) \cdot X_2 \cdot (1 - X_3)$ mit Belegung (b_1, b_2, b_3) den Wert 1 annimmt.
3. Auch 3SAT-*Formeln* entsprechen Polynome.
4. **Testfrage.** Sei $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$. Was ist P_F?

   Antwort. $P_F = \prod_{i=1}^{m} P_{C_i}$.
5. **Testfrage.** Wie hoch ist der Grad von P_F maximal?

   Antwort. Höchstens $3m$.
6. **Testfrage.** Wie lässt sich P_F polynomiell in m speichern?

   Antwort. Indem wir auf das Ausmultiplizieren verzichten.
7. **Testfrage.** Wieso ist das dann auch polynomiell in n?

Erfüllende Belegungen zählen

1. **Satz (Wiederholung).** #3SAT $\in$ **IP**.
2. Eine Formel F mit Variablen $X_1, \ldots, X_n$ hat offensichtlich
$$\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} P_F(b_1, \ldots, b_n)$$
viele erfüllende Belegungen.
3. **Testfrage.** Warum kann der Prüfer diese Zahl nicht ausrechnen? Warum brauchen wir die höhere Rechenleistung des Beweisers?
4. Für eine Primzahl $p \in (2^n, 2^{2n}]$ schadet es nicht, alle Rechenschritte nur modulo p durchzuführen. **Testfrage.** Warum?

Das Sumcheck-Protokoll

1. Wir abstrahieren nun von dem speziellen Fall #3SAT.
2. Es sei eine Darstellungsweise von Polynomen als Binärwörter gewählt. Sei $d \in \mathbb{N}$ und $g(X_1, \ldots, X_n)$ ein Polynom vom Grad $d \leq n$, das sich polynomiell in n darstellen lässt.
3. Wir beschreiben ein **IP**-Protokoll, mit dem ein Prüfer für $k \leq 2^n$ davon überzeugt werden kann, dass

$$\left(\sum_{b_1 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(b_1, \ldots, b_n) \right) \bmod p = k. \qquad (\dagger)$$

4. Alle weiteren Rechnunganweisungen beziehen sich auf $\mathbb{F}_p$.

Das Sumcheck-Protokoll

1. Wir definieren ein Polynom in nur einer Variable via
$$h_g(X_1) = \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_n \in \{0,1\}} g(X_1, b_2, \ldots, b_n).$$
2. **Testfrage.** Hat h_g eine in n polynomielle Darstellung?
3. **Testfrage.** Welchen Wert hat $h_g(0) + h_g(1)$, falls (†) gilt?

Das Sumcheck-Protokoll

Eingabe: $g(X_1,\ldots,X_n)$ in polynomieller Darstellung, $k \in \mathbb{F}_p$.

1. V: Falls $n = 1$, berechne $g(1) + g(0)$.
   - Falls dies $= k$ ist, akzeptiere, sonst verwerfe;
   - dann beende das Protokoll.
2. P: Falls $n \geq 2$, sende ein Polynom $s(X_1)$ mit Grad $\leq d$ an V.

   (Der „richtige“ Beweiser, für den das Protokoll gedacht ist, schickt hier $h_g(X_1)$.)
3. V: Berechne $s(0) + s(1)$. Falls dies $\neq k$ ist, verwerfe und beende.
4. V: Sonst wähle eine Zufallszahl $a \in \mathbb{F}_p$. Berechne $k' = s(a)$ und eine polynomielle Darstellung des Polynoms
   $$g'(X_2,\ldots,X_n) := g(a, X_2,\ldots,X_n).$$
5. Wiederhole das Protokoll für die Eingaben (g', k').

Verständnisfragen

1. Die Korrektheit des Protokoll beweisen wir in Kürze.
2. **Testfrage.** Welche schwierige Aufgabe übernimmt der Beweiser?
3. **Testfrage.** Welche Eigenschaft wird rekursiv überprüft?
4. **Testfrage.** Warum arbeitet V in Polynomialzeit?

Korrektheitsbeweis des Protokolls

1. Falls (†) gilt, so kann der „richtige“ Beweiser den Prüfer mit Wahrscheinlichkeit 1 davon überzeugen. **Testfrage.** Wie?
2. Es bleibt, falls (†) *nicht* gilt, zu beweisen, dass es keinen Beweiser gibt, der dem Prüfer erfolgreich vorspielen kann, (†) gelte doch.
3. **Testfrage.** Warum muss ein solcher „boshafter“ Beweiser in 2 statt h_g geschickt ausgewählte falsche s schicken?

Korrektheitsbeweis des Protokolls

1. Falls (†) gilt, so kann der „richtige“ Beweiser den Prüfer mit Wahrscheinlichkeit 1 davon überzeugen. **Testfrage.** Wie?
2. Es bleibt, falls (†) *nicht* gilt, zu beweisen, dass es keinen Beweiser gibt, der dem Prüfer erfolgreich vorspielen kann, (†) gelte doch.
3. **Testfrage.** Warum muss ein solcher „boshafter“ Beweiser in 2 statt h_g geschickt ausgewählte falsche s schicken?

   Antwort.

   - Würde er im ersten Durchlauf das *korrekte* h_g senden, so würde V, da (†) nicht gilt, in 3 verwerfen.
   - Analoges gilt in den rekursiven Aufrufen für g' und $h_{g'}$.
   - Der „boshafte“ Beweiser muss also sicherstellen, dass *in jeder Runde* gilt, dass $s(0) + s(1) = k$.
4. Zeigt man, dass ihm das nur mit kleiner Wahrscheinlichkeit in *allen* Runden gelingen kann, folgt die Korrektheit.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - **Testfrage.** Warum verwirft V immer falls $n = 1$?

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}\!/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.
   - Obwohl s „falsch“ ist, könnte durch Zufall $s(a) = h_g(a)$ gelten.
     (Das wäre gut für den „boshaften“ P, denn dann müsste er in der nächsten Rekursion V von einer *wahren* Aussage überzeugen, und das geht immer.)

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}\!/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.
   - Obwohl s „falsch“ ist, könnte durch Zufall $s(a) = h_g(a)$ gelten. (Das wäre gut für den „boshaften“ P, denn dann müsste er in der nächsten Rekursion V von einer *wahren* Aussage überzeugen, und das geht immer.)
   - $s - h_g$ ist ein Polynom vom Grad höchstens d und hat somit höchstens d Nullstellen a an denen $s(a) = h_g(a)$ gilt. Also gilt
     $$\Pr[\,V \text{ wählt ein } a \text{ mit } s(a) = h_g(a)] \leq {}^{d}\!/_{p}.$$

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[\,V \text{ verwirft}] \geq (1 - {}^{d}\!/_{p})^{n}$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.
   - Obwohl s „falsch“ ist, könnte durch Zufall $s(a) = h_g(a)$ gelten. (Das wäre gut für den „boshaften“ P, denn dann müsste er in der nächsten Rekursion V von einer *wahren* Aussage überzeugen, und das geht immer.)
   - $s - h_g$ ist ein Polynom vom Grad höchstens d und hat somit höchstens d Nullstellen a an denen $s(a) = h_g(a)$ gilt. Also gilt
     $$\Pr[\,V \text{ wählt ein } a \text{ mit } s(a) = h_g(a)] \leq {}^{d}\!/_{p}.$$
   - Somit $\Pr[\,V \text{ wählt ein } a \text{ mit } s(a) \neq h_g(a)] \geq 1 - {}^{d}\!/_{p}$. Dann ist die neue Aussage, die P in der Rekursion beweisen muss, *unwahr*.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[V \text{ verwirft}] \geq (1 - d/p)^n$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.
   - Obwohl s „falsch" ist, könnte durch Zufall $s(a) = h_g(a)$ gelten. (Das wäre gut für den „boshaften" P, denn dann müsste er in der nächsten Rekursion V von einer *wahren* Aussage überzeugen, und das geht immer.)
   - $s - h_g$ ist ein Polynom vom Grad höchstens d und hat somit höchstens d Nullstellen a an denen $s(a) = h_g(a)$ gilt. Also gilt
     $$\Pr[V \text{ wählt ein } a \text{ mit } s(a) = h_g(a)] \leq d/p.$$
   - Somit $\Pr[V \text{ wählt ein } a \text{ mit } s(a) \neq h_g(a)] \geq 1 - d/p$. Dann ist die neue Aussage, die P in der Rekursion beweisen muss, *unwahr*.
   - Nach Induktionsannahme ist die Wahrscheinlichkeit, dass V diese neue falsche Aussage verwirft, mindestens $(1 - d/p)^{n-1}$.

Korrektheitsbeweis des Protokolls

1. **Behauptung.** Gilt (†) nicht, so ist $\Pr[V \text{ verwirft}] \geq (1 - d/p)^n$.
2. **Beweis.** Beweis per Induktion über die Anzahl der Variablen n.
   - Nimm also an, die Aussage sei wahr für bis zu $n-1$ Variablen.
   - Falls P das korrekte h_g liefert, verwirft V mit Sicherheit, wie vorhin diskutiert. Nimm also an, P liefere $s(X_1) \neq h_g(X_1)$.
   - V wählt nun zufällig a und rekurriert auf $(g', k') = (g(a, \dots), s(a))$.
   - Obwohl s „falsch" ist, könnte durch Zufall $s(a) = h_g(a)$ gelten. (Das wäre gut für den „boshaften" P, denn dann müsste er in der nächsten Rekursion V von einer *wahren* Aussage überzeugen, und das geht immer.)
   - $s - h_g$ ist ein Polynom vom Grad höchstens d und hat somit höchstens d Nullstellen a an denen $s(a) = h_g(a)$ gilt. Also gilt
     $$\Pr[V \text{ wählt ein } a \text{ mit } s(a) = h_g(a)] \leq d/p.$$
   - Somit $\Pr[V \text{ wählt ein } a \text{ mit } s(a) \neq h_g(a)] \geq 1 - d/p$. Dann ist die neue Aussage, die P in der Rekursion beweisen muss, *unwahr*.
   - Nach Induktionsannahme ist die Wahrscheinlichkeit, dass V diese neue falsche Aussage verwirft, mindestens $(1 - d/p)^{n-1}$.
   - Insgesamt ergibt sich $\Pr[V \text{ verwirft}] \geq (1 - d/p)^n$. □

Von #3SAT zu QBF

1. Der Korrektheitsbeweis des Sumcheck-Protokolls vervollständigt auch den Beweis, dass #3SAT $\in$ **IP**.
2. Jetzt wollen wir die stärkere Aussage QBF $\in$ **IP** beweisen.
   (Daraus folgt dann insbesondere **IP** $=$ **PSPACE**.)
3. Wir überlegen uns zunächst, welche neuen Herausforderungen der Fall QBF mit sich bringt.

Von #3SAT zu QBF

1. Sei z.B. eine Formel $F = \forall X\, \exists Y\, \forall Z\, G(X, Y, Z)$ in QBF gegeben.
2. **Testfrage.** Wie sieht ein dazu äquivalentes Polynom aus?

Von #3SAT zu QBF

1. Sei z.B. eine Formel $F = \forall X\, \exists Y\, \forall Z\, G(X, Y, Z)$ in QBF gegeben.
2. **Testfrage.** Wie sieht ein dazu äquivalentes Polynom aus?

   Antwort. $F \in \mathrm{QBF}$ genau dann, wenn

$$\prod_{a \in \{0,1\}} \sum_{b \in \{0,1\}} \prod_{c \in \{0,1\}} P_G(a, b, c) \neq 0.$$

   (Wir hatten P_F für 3SAT-Formeln F definiert. Für beliebige G sei P_G analog definiert.)

Von #3SAT zu QBF

1. Sei z.B. eine Formel $F = \forall X\, \exists Y\, \forall Z\, G(X, Y, Z)$ in QBF gegeben.
2. **Testfrage.** Wie sieht ein dazu äquivalentes Polynom aus?

   Antwort. $F \in \mathrm{QBF}$ genau dann, wenn
   $$\prod_{a \in \{0,1\}} \sum_{b \in \{0,1\}} \prod_{c \in \{0,1\}} P_G(a, b, c) \neq 0.$$
   (Wir hatten P_F für 3SAT-Formeln F definiert. Für beliebige G sei P_G analog definiert.)
3. Das gleiche Schema lässt sich natürlich auf beliebige Formeln $F \in \mathrm{QBF}$ anwenden, wobei jedem $\forall$-Quantor ein Π-Operator entspricht, und jedem $\exists$-Quantor ein Σ-Operator.

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

   Antwort.

   - Die Grade zweier Polynome addieren sich beim Multiplizieren.

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

   Antwort.

   - Die Grade zweier Polynome addieren sich beim Multiplizieren.
   - Deshalb könnte h_g Grad bis zu 2^n haben, denn es kann bis zu n Π-Operatoren geben, und jeder davon kann den Grad verdoppeln.

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

   Antwort.

   - Die Grade zweier Polynome addieren sich beim Multiplizieren.
   - Deshalb könnte h_g Grad bis zu 2^n haben, denn es kann bis zu n Π-Operatoren geben, und jeder davon kann den Grad verdoppeln.
   - Dann hat h_g möglicherweise keine polynomiell lange Darstellung.

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

   Antwort.

   - Die Grade zweier Polynome addieren sich beim Multiplizieren.
   - Deshalb könnte h_g Grad bis zu 2^n haben, denn es kann bis zu n Π-Operatoren geben, und jeder davon kann den Grad verdoppeln.
   - Dann hat h_g möglicherweise keine polynomiell lange Darstellung.
   - V müsste aber mit (einem Kandidaten für) h_g arbeiten können.

Eine naive Idee

1. **Naive Idee.** Das Sumcheck-Protokoll so erweitern, dass es auch mit Π-Operatoren zurecht kommt, indem

   V auf „$s(0) \cdot s(1) = k$“ prüft anstatt auf „$s(0) + s(1) = k$“.

2. Die Idee ist aber aus Ressourcengründen nicht umsetzbar.
3. **Testfrage.** Gehen Sie das Sumcheck-Protokoll durch. Wo scheitert die Idee an der Polynomialzeitbeschränkung von V?

   Antwort.

   - Die Grade zweier Polynome addieren sich beim Multiplizieren.
   - Deshalb könnte h_g Grad bis zu 2^n haben, denn es kann bis zu n Π-Operatoren geben, und jeder davon kann den Grad verdoppeln.
   - Dann hat h_g möglicherweise keine polynomiell lange Darstellung.
   - V müsste aber mit (einem Kandidaten für) h_g arbeiten können.

4. **Also:** Wir müssen uns Tricks überlegen, um den Grad niedrig zu halten, ohne die Aussagekraft des Protokolls zu beeinträchtigen.

Grad-Reduktion

1. In die Polynome, die wir betrachten, müssen wir für die Variablen nur die Werte 0 und 1 einsetzen.
2. **Idee:** Ersetze Polynome hohen Grades durch Polynome niedrigen Grades, die auf *binären* Belegungen übereinstimmen.
3. Wir definieren die folgenden Operatoren auf Polynomen P:

$$\begin{aligned}
\exists_{X_i} : P(X_1,\ldots,X_\ell) \mapsto \quad & P(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots,X_\ell) \\
& +P(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots,X_\ell) \\
\forall_{X_i} : P(X_1,\ldots,X_\ell) \mapsto \quad & P(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots,X_\ell) \\
& \cdot P(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots,X_\ell) \\
L_{X_i} : P(X_1,\ldots,X_\ell) \mapsto \quad & P(X_1,\ldots,X_{i-1},1,X_{i+1},\ldots,X_\ell) \cdot X_i \\
& +P(X_1,\ldots,X_{i-1},0,X_{i+1},\ldots,X_\ell) \cdot (1-X_i)
\end{aligned}$$

4. Nach Anwendung von L_{X_i} hat X_i nur noch Grad 1.
   Testfrage. Warum ist das äquivalent für binäre Belegungen?

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \forall_Z L_X L_Y L_Z P_G(X,Y,Z).$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \forall_Z L_X L_Y L_Z \underbrace{P_G(X,Y,Z)}_{\substack{\text{3 Variablen} \\ \deg(X)\leq n \\ \deg(Y)\leq n \\ \deg(Z)\leq n}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \forall_Z L_X L_Y \underbrace{L_Z P_G(X,Y,Z)}_{\substack{\text{3 Variablen} \\ \deg(X)\leq n \\ \deg(Y)\leq n \\ \deg(Z)=1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \forall_Z L_X \underbrace{L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{3 Variablen} \\ \deg(X)\leq n \\ \deg(Y)=1 \\ \deg(Z)=1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \forall_Z \underbrace{L_X L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{3 Variablen} \\ \deg(X)=1 \\ \deg(Y)=1 \\ \deg(Z)=1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X L_Y \underbrace{\forall_Z L_X L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{2 Variablen} \\ \deg(X)\leq 2 \\ \deg(Y)\leq 2}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y L_X \underbrace{L_Y \forall_Z L_X L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{2 Variablen} \\ \deg(X)\leq 2 \\ \deg(Y)=1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \exists_Y \underbrace{L_X L_Y \forall_Z L_X L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{2 Variablen} \\ \deg(X)=1 \\ \deg(Y)=1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird

$$\forall_X \underbrace{\exists_Y L_X L_Y \forall_Z L_X L_Y L_Z P_G(X,Y,Z)}_{\substack{\text{1 Variable} \\ \deg(X)\leq 1}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X, Y, Z)$ wird

$$\underbrace{\forall_X \exists_Y L_X L_Y \forall_Z L_X L_Y L_Z P_G(X, Y, Z)}_{0 \text{ Variablen}}.$$

(Für beliebige Formeln aus QBF geht es analog.)

Grad-Reduktion

1. Statt Π und Σ können wir nun $\exists_{X_i}$ und $\forall_{X_i}$ benutzen.
2. **Beispiel.** Aus $F = \forall X\, \exists Y\, \forall Z\, G(X,Y,Z)$ wird
$$\underbrace{\forall_X \exists_Y L_X L_Y \forall_Z L_X L_Y L_Z P_G(X,Y,Z)}_{0 \text{ Variablen}}.$$
(Für beliebige Formeln aus QBF geht es analog.)
3. **Folgende Eigenschaften werden später wichtig:**
   - Ausgewertet ergibt das Obige einen langen Ausdruck über $(\mathbb{N}, +, \cdot, 0, 1)$ *ohne Variablen*.
   - *Nicht* ausgewertet ist es ein kurzer *Text*ausdruck, so dass in P_G noch alle Variablen vorhanden sind.
   - Links von L_W steht immer ein $\exists_W$ oder ein $\forall_W$.

Grad-Reduktion

1. Als Eingabe erhält der Prüfer später (unter anderem) F.
2. Er bringt dies in die obige Form (als Text, ohne Rechnung):
   - G wird wie gehabt in P_G umgewandelt.
   - Die Quantifizierung wird wie gerade besprochen in einen *Textausdruck* von $O(n^2)$ Operatoren umgewandelt.
3. Um zu entscheiden, ob F wahr ist, müsste man nun
   - die Operatoren von rechts nach links abarbeiten,
   - einen Ausdruck über $(\mathbb{N}, +, \cdot, 0, 1)$ ohne Variablen erhalten,
   - und nachrechnen, ob dieser 0 ergibt.

   (Denn F ist wahr genau dann, wenn nicht 0 herauskommt.)
4. Doch das ist für den Prüfer unmöglich, denn der Ausdruck über $(\mathbb{N}, +, \cdot, 0, 1)$ ist im Allgemeinen exponentiell groß.

Testfrage. Warum?

Grad-Reduktion

1. Also benutzen wir wieder die Rechenleistung des Beweisers.
2. **Also:**
   - Wir nutzen ein interaktives Protokoll mit $O(n^2)$ Runden.
   - In jeder Runde wird ein Operator abgearbeitet.
   - Dabei wird die Hilfe des Beweisers benutzt, damit die auftretenden Polynome jederzeit handhabbar bleiben.

Idee der Rekursion

1. Der Prüfer kann den Textausdruck nicht auswerten, aber
   - Operatoren *als Text* löschen
   - und P_G beliebig verändern.
2. Wenn ein Operator weggelassen wird, müssten wir, um das zu kompensieren, gemäß der Operator-Regeln die Länge des Ausdrucks verdoppeln. Das kann der Prüfer nicht leisten.
3. Stattdessen lässt er sich vom Beweiser eine (angeblich korrekte) einfache Darstellung der "Hälfte" des Polynoms liefern, also eine Hälfte, in der die vom Operator gebundene Variable *frei* ist.
4. Diese prüfen wir (wie beim Sumcheck-Protokoll) durch Einsetzen von 0 und 1 darauf, ob ein bestimmter Kontrollwert herauskommt.
5. Für die Rekursion lassen wir nun den ersten Operator weg, und setzen für die zugehörige Variable einen festen Wert $a \in \mathbb{F}_p$ ein.
6. Der Kontrollwert für die nächste Runde errechnet sich wieder durch Einsetzen von a in die vom Beweiser gelieferte "Hälfte".

Idee der Rekursion

1. Wie zuvor funktioniert mit dem „richtigen“ Beweiser alles.
2. Ein „boshafter“ Beweiser versucht aber, den Prüfer zu täuschen.
3. Dazu liefert er (wie beim Sumcheck-Protokoll) falsche Polynome.
4. Wie zuvor werden diese vom Prüfer rekursiv geprüft.
5. Der Beweiser muss also in jeder Runde erneut falsche Polynome liefern, um die Illusion aufrecht zu erhalten.
6. **Zu zeigen:** Das „Lügengebäude“ des Beweisers bricht mit hoher Wahrscheinlichkeit zusammen, bevor die Rekursion endet.

Idee der Rekursion

1. **Zu zeigen:** Das „Lügengebäude“ des Beweisers bricht mit hoher Wahrscheinlichkeit zusammen, bevor die Rekursion endet.
2. **Ein wichtiger Hebel für den Prüfer:**
   - Es genügt, dass das Polynom auf binären Eingaben das richtige Ergebnis liefert.
   - Aber um einen unehrlichen Beweiser zu erwischen, der ein falsches Polynom liefert, kann der Prüfer die Korrektheit des Polynoms auf beliebigen Eingaben $a \in \mathbb{F}_p$ überprüfen.
   - Das erschwert es, mit den Polynomen zu schummeln.
   - So hatten wir es auch beim Sumcheck-Protokoll gemacht.
3. Wir skizzieren das Protokoll nun rekursiv.

Idee der Rekursion

1. **Äußerste Rekursionsstufe:** Alle Variablen in P_G sind durch Operatoren gebunden. Der Ausdruck stellt also ein Polynom ohne Variablen dar, also eine konstante Zahl. Uns liegt eine Behauptung vor, welchen Wert diese hat; das wollen wir prüfen.
2. **Innerste Rekursionsstufe:** Hier gibt es keine Operatoren mehr; und für alle Variablen in P_G wurden Zahlen aus $\mathbb{F}_p$ eingesetzt. Der Prüfer kann ohne Hilfe des Beweisers nachrechnen, ob der aktuelle Kontrollwert herauskommt.
3. **Zwischendurch:** In P_G gibt es noch Variablen, die aber alle durch Operatoren gebunden sind; für alle übrigen Variablen wurden bereits Zahlen aus $\mathbb{F}_p$ nach P_G eingesetzt. Würde man diesen Ausdruck auswerten (teils gemäß der Operatorregeln, teils mit den eingesetzten Zahlen) käme auch hier ein konkreter Zahlenwert heraus.

Das Protokoll für QBF

1. **Behauptung:** Für die beschriebenen "gemischten" Ausdrücke, wie sie im Verlauf der Rekursion auftreten, und alle k gilt:
   - Falls der Ausdruck die Zahl k darstellt, kann der Beweiser den Prüfer mit Wahrscheinlichkeit 1 davon überzeugen;
   - sonst kann er den Prüfer nur mit kleiner Wahrscheinlichkeit (fälschlicherweise) überzeugen, der Ausdruck stelle k dar.
2. **Innerste Rekursionsstufe:** Wie beschrieben benötigt der Prüfer in diesem Fall für die Prüfung den Beweiser nicht. ✓

Das Protokoll für QBF

1. **Behauptung:** Für die beschriebenen "gemischten" Ausdrücke, wie sie im Verlauf der Rekursion auftreten, und alle k gilt:
   - Falls der Ausdruck die Zahl k darstellt, kann der Beweiser den Prüfer mit Wahrscheinlichkeit 1 davon überzeugen;
   - sonst kann er den Prüfer nur mit kleiner Wahrscheinlichkeit (fälschlicherweise) überzeugen, der Ausdruck stelle k dar.

3. **Rekursionsschritt:** Die Eingabe liegt vor
   - als Ausdruck „$\mathcal{O}_m \mathcal{O}_{m-1} \dots \mathcal{O}_1 h$"
   - mit einem h, das der Prüfer ohne Beweiser selbst auswerten kann,
   - mit $\mathcal{O}_m \in \{\forall_{X_i}, \exists_{X_i}, L_{X_i}\}$ für ein $i \in \mathbb{N}$, und übrige $\mathcal{O}_j$'s analog;
   - zusätzlich erhalten wir einen behaupteten Wert k.

Das Protokoll für QBF

Protokoll: Falls $\mathcal{O}_m = \exists_{X_i}$.

1. P: Sende ein Polynom $s(X_i)$ an V.
   (Der „richtige“ Beweiser schickt hier $(\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(X_i)$.)
2. V: Berechne $s(0) + s(1)$.
3. V: Falls das Ergebnis $\neq k$ ist, verwerfe und beende.
4. V: Sonst wähle eine Zufallszahl $a \in \mathbb{F}_p$. Berechne $k' = s(a)$.
5. Wiederhole das Protokoll für die Eingaben
$$((\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(a), k').$$

Das Protokoll für QBF

Protokoll: Falls $\mathcal{O}_m = \forall_{X_i}$.

1. P: Sende ein Polynom $s(X_i)$ an V.
   (Der „richtige“ Beweiser schickt hier $(\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(X_i)$.)
2. V: Berechne $s(0) \cdot s(1)$.
3. V: Falls das Ergebnis $\neq k$ ist, verwerfe und beende.
4. V: Sonst wähle eine Zufallszahl $a \in \mathbb{F}_p$. Berechne $k' = s(a)$.
5. Wiederhole das Protokoll für die Eingaben
   $$((\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(a), k').$$

Das Protokoll für QBF

Protokoll: Falls $\mathcal{O}_m = L_{X_i}$.

1. P: Sende ein Polynom $s(X_i)$ an V.

   (Der „richtige“ Beweiser schickt hier $(\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(X_i)$.)

2. V: Berechne $a_i \cdot s(1) + (1 - a_i) \cdot s(0)$.

   Hierbei ist a_i das a, welches wir in einer früheren Runde für X_i eingesetzt hatten, als wir $\exists_{X_i}$ bzw. $\forall_{X_i}$ abgearbeitet hatten.

3. V: Falls das Ergebnis $\neq k$ ist, verwerfe und beende.
4. V: Sonst wähle eine Zufallszahl $a \in \mathbb{F}_p$. Berechne $k' = s(a)$.
5. Wiederhole das Protokoll für die Eingaben

$$((\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(a), k').$$

Verifikation

1. **Zu zeigen war:** Das „Lügengebäude“ des Beweisers bricht mit hoher Wahrscheinlichkeit zusammen.
2. **Grund:** Falls er ein falsches s liefert, so passen $(\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(a)$ (welches der Prüfer selbst vom vertrauenswürdigen $\mathcal{O}_m \dots \mathcal{O}_1 h$ ableitet) und $k' = s(a)$ nur mit geringer Wahrscheinlichkeit zufälligerweise zusammen.
3. Dann muss der Beweiser in der Rekursion mit den Eingaben
$$((\mathcal{O}_{m-1} \dots \mathcal{O}_1 h)(a), k')$$
mit hoher Wahrscheinlichkeit eine falsche Aussage beweisen.
4. Nach Induktionsannahme gelingt das dem Beweiser erneut nur mit geringer Wahrscheinlichkeit.
5. Die genaue Rechnung ist analog zum Sumcheck-Protokoll; wir sparen sie uns. □

Der Beweis relativiert nicht

1. **Testfrage.** Wie würden Sie $\mathbf{IP}^A$ für ein Orakel A definieren?
2. Man kann zeigen, dass es Orakel A und B gibt, so dass
$$\mathbf{IP}^A = \mathbf{PSPACE}^A \quad \text{und} \quad \mathbf{IP}^B \neq \mathbf{PSPACE}^B.$$
Testfrage. Nennen Sie eine Komplexitätsklasse, aus der man A trivialerweise nehmen kann.
3. Tatsächlich gilt sogar für *fast alle* Orakel B, dass
$$\mathbf{IP}^B \subsetneq \mathbf{PSPACE}^B.$$

Kapitelzusammenfassung

1. Zu Boolschen Formeln gibt es gleichwertige Polynome.
2. Quantoren vor den Formeln entsprechen Σ- und Π-Operatoren.
3. In deterministischer Polynomialzeit ist es im Allgemeinen nicht möglich, Formeln mit diesen Operatoren auszuwerten.
4. Ein allmächtiger Beweiser könnte diese Aufgabe übernehmen, doch ein Prüfer hätte keinen Grund, dem Ergebnis zu vertrauen.
5. Es gibt aber Protokolle, bei denen der Beweiser die Komplexität für den Prüfer stark reduziert, ohne schummeln zu können.
6. Denn der Prüfer führt kleine, zufällig ausgewählte Kontrollrechnungen durch, durch die Schummeln auffallen würde.
7. Mit diesem Ansatz konnten wir zeigen, dass
$$\#3\mathrm{SAT} \in \mathbf{IP} \qquad \text{und} \qquad \mathrm{QBF} \in \mathbf{IP}.$$
8. Insbesondere folgt $\mathbf{IP} = \mathbf{PSPACE}$.

11 Das PCP-Theorem

Überblick

1. In diesem Abschnitt lernen wir das PCP-Theorem kennen.
2. Es handelt sich um eines der wichtigsten Ergebnisse der Theoretischen Informatik der letzten Jahrzehnte.
3. Der Satz kann auf zwei Weisen verstanden werden:
   - als eine Aussage über probabilistische Beweissysteme
   - als die Aussage, dass bestimmte Optimierungsprobleme nur schlecht approximierbar sind, falls $\mathbf{P} \neq \mathbf{NP}$.
4. Wir werden die Äquivalenz beider Versionen beweisen.
5. Aus praktischer Sicht sind die Folgerungen zu Approximationsalgorithmen besonders interessant. Wir betrachten Beispiele.
6. Das PCP-Theorem selbst werden wir nicht beweisen.

Optimierungsprobleme

1. Bei einem Optimierungsproblem soll zu einer Instanz x eine „möglichst gute“ Lösung y gefunden werden. Was heißt „gut“?
2. **Definition.** Eine Qualitätsfunktion für ein Optimierungsproblem ist eine partielle Abbildung $\mathrm{qua}\colon (\{0,1\}^{<\mathbb{N}})^2 \to \mathbb{Q}$.
3. **Intuition.** Das erste Argument soll hierbei die Instanz x sein, das zweite eine Beschreibung der Lösung y. Falls x keine gültige Beschreibung einer Instanz des Problems ist, oder y keine gültige Beschreibung einer Lösung zu x, so sei $\mathrm{qua}(x,y)$ undefiniert.

Optimierungsprobleme

1. **Definition.** Für ein Maximierungsproblem sei die Optimumsfunktion $\mathrm{opt}\colon \{0,1\}^{<\mathbb{N}} \to \mathbb{Q}$ definiert via

$$\mathrm{opt}(x) = \max_{y \text{ ist Lösung für } x} \mathrm{qua}(x,y).$$

   Für ein Minimierungsproblem ersetze „max“ durch „min“.

2. **Intuition.** Für eine Instanz x eines Optimierungsproblems gibt $\mathrm{opt}(x)$ den Wert der besten Lösung an, der für x möglich ist.

Optimierungsprobleme aus Entscheidungsproblemen

1. Zu vielen Entscheidungsprobleme kann man eine Variante als Optimierungsproblem definieren.
2. **Beispiel.** Max3SAT ist definiert als das Problem, zu einer (nicht notwendigerweise erfüllbaren) Formel F in 3-KNF eine Belegung zu finden, die *möglichst viele* Klauseln wahr macht.

   Wir wählen $\mathrm{qua}(x, y)$ als den *Anteil* der Klauseln in x, die durch y wahr gemacht werden. Dann gilt

   $$F \in 3\mathrm{SAT} \iff \mathrm{opt}(F) = 1,$$

   das heißt, alle Klauseln in F können gleichzeitig wahr sein.

Approximationsalgorithmen

1. **Beispiel.** Eine Instanz x von Max3SAT erlaubt eine „perfekte“ Lösung, wenn $\mathrm{opt}(x) = 1$.
2. Für viele x gibt es keine „perfekte“ Lösung. Aber auch dann sollte ein guter Approximationsalgorithmus eine Lösung y liefern, so dass $\mathrm{qua}(x, y)$ nur „ein wenig“ kleiner als $\mathrm{opt}(x)$ ist.
3. Diese Idee motiviert die folgenden Definitionen.

Approximationsalgorithmen

1. **Definition.** Sei $\rho \leq 1$. Für ein Maximierungsproblem ist A ein ρ-Approximationsalgorithmus, falls für alle gültigen Eingaben x gilt, dass $\mathrm{qua}(x, A(x)) \geq \rho \cdot \mathrm{opt}(x)$.
2. **Definition.** Sei $\rho \leq 1$. Für ein Minimierungsproblem ist A ein ρ-Approximationsalgorithmus, falls für alle gültigen Eingaben x gilt, dass $\mathrm{qua}(x, A(x)) \leq 1/\rho \cdot \mathrm{opt}(x)$.
3. Uns interessieren Polynomialzeit-Approximationsalgorithmen.
4. **Testfrage.** Nennen Sie einen $1/2$-Approximationsalgorithmus für das Problem der minimalen Knotenüberdeckung (MinVC)!

   (Erinnern Sie sich dazu an die Übung zur Vorlesung „Komplexitätstheorie“.)
5. **Testfrage.** Wie sieht es für Max3SAT aus?

Approximationsalgorithmen

1. Falls $\mathbf{P} \neq \mathbf{NP}$ sind viele Entscheidungsprobleme nicht in Polynomialzeit lösbar, z.B. 3SAT. Es wäre aber vorstellbar, dass Polynomialzeitalgorithmen gute Näherungen liefern.
2. Zum Beispiel könnte es einen Polynomialzeitalgorithmus geben, der zu jeder erfüllbaren 3SAT-Formel eine Belegung liefert, die *fast* alle Klauseln wahr macht. Das ist a priori kein Widerspruch.
3. In der Praxis könnte eine gute Approximation vielleicht reichen.
4. **Doch:** Das PCP-Theorem zeigt, dass viele **NP**-schwere Entscheidungsprobleme auch schlecht approximierbar sind, falls $\mathbf{P} \neq \mathbf{NP}$.

Beweise probabilistisch prüfen

1. **Definition.** Sei $r\colon \mathbb{N} \to \mathbb{N}$. Wir sagen L *besitzt einen* r-PCP-*Prüfer* falls es eine probabilistische Polynomialzeit-Orakel-Turingmaschine V und eine Konstante $q > 0$ gibt, so dass bei Eingabe x
   - V zunächst $r(|x|)$ Zufallsbits für den weiteren Gebrauch generiert,
   - V dann höchstens q Orakelfragen stellt,
   - V dann mit den Zufallsbits und Orakelantworten entscheidet.

   Für V soll gelten, dass
   - falls $x \in L$, so gibt es einen *korrekten Beweis für* x, das bedeutet ein Orakel π mit $\Pr[\,V^{\pi}(x) \text{ akzeptiert}] = 1$; *(Vollständigkeit)*
   - falls $x \notin L$, so $\Pr[\,V^{\pi}(x) \text{ akzeptiert}] \leq 1/2$ für *jedes* π. *(Korrektheit)*
2. **Definition.** $\mathbf{PCP}(r) = \{L\colon L \text{ hat einen } (O(r))\text{-PCP-Prüfer}\}$.
3. **Bemerkung.** „PCP“ heißt „probabilistically checkable proof“.

Anmerkungen zur Definition

1. Man kann π als Wort der Länge $q \cdot 2^{r(|x|)}$ betrachten, denn auch über alle möglichen Zufallsbits zusammengenommen können nicht mehr verschiedene Orakelfragen gestellt werden.
2. Die Länge der „Adresse“ (also die Orakelfrage, die gestellt werden muss) eines Bits im Beweis ist logarithmisch in der Länge des Beweises.
3. **Testfrage.** Warum gilt $\mathbf{PCP}(r) \subseteq \mathbf{NTIME}(2^{O(r)})$?
4. Der *Korrektheits*parameter $1/2$ ist willkürlich. Jedes andere $0 < p < 1$ ergibt die gleichen Komplexitätsklassen.

(Beweis → Übungszettel)

Beispiel für einen probabilistisch prüfbaren Beweis

1. **Beispiel.** GNI $\in$ **PCP**(poly).
2. **Beweis.** Der Prüfer V arbeitet wie folgt:
   - Seien G_0 und G_1 zwei Graphen mit n Knoten.
   - Der korrekte Beweis π ist ein exponentiell langes Wort so dass

$$\pi(\ulcorner H \urcorner) = \begin{cases} 0 & \text{falls } H \text{ isomorph zu } G_0 \text{ ist,} \\ 1 & \text{sonst.} \end{cases}$$

   - V wählt zufällig $b \in \{0, 1\}$ und eine zufällige Permutation σ.
   - Mithilfe einer Orakelfrage prüft V ob $\pi\left(\ulcorner \sigma(G_b) \urcorner\right) = b$.
   - Falls ja, akzeptiert V, sonst verwirft er.

Testfrage. Warum ist, falls G_0 und G_1 *nicht* isomorph sind, mit dem korrekten Beweis π die Akzeptanzwahrscheinlichkeit 1?

Testfrage. Warum ist die Akzeptanzwahrscheinlichkeit, falls G_0 und G_1 isomorph sind, für jeden *beliebigen* Beweis nur $1/2$? □

Das erste PCP-Theorem

1. **Satz.** $\mathbf{PCP}(\log) = \mathbf{NP}$.
2. **Testfrage.** Warum gilt $\mathbf{PCP}(\log) \subseteq \mathbf{NP}$?
3. Die umgekehrte Inklusion kann im Rahmen dieser Vorlesung nicht bewiesen werden. Stattdessen konzentrieren wir uns darauf, die Konsequenzen des Satzes zu verstehen.

Das zweite PCP-Theorem

1. **Satz.** Es gibt ein $\rho < 1$, so dass für jedes $L \in \mathbf{NP}$ gilt, dass $L \leq_{\mathrm{m}}^{\mathrm{p}} 3\mathrm{SAT}$ via ein $f \in \mathbf{FP}$, so dass,
   - falls $x \in L$, dann $\mathrm{opt}(f(x)) = 1$,
   - falls $x \notin L$, dann $\mathrm{opt}(f(x)) < \rho$.
2. **Testfrage.** Vergleichen Sie das mit dem Satz von Cook-Levin!

Das zweite PCP-Theorem

1. **Satz.** Es gibt ein $\rho < 1$, so dass für jedes $L \in \mathbf{NP}$ gilt, dass $L \leq_{\mathrm{m}}^{\mathrm{p}} 3\mathrm{SAT}$ via ein $f \in \mathbf{FP}$, so dass,
   - falls $x \in L$, dann $\mathrm{opt}(f(x)) = 1$,
   - falls $x \notin L$, dann $\mathrm{opt}(f(x)) < \rho$.
2. **Testfrage.** Vergleichen Sie das mit dem Satz von Cook-Levin!

   Antwort. Der Satz von Cook-Levin garantiert keine „Lücke“. Tatsächlich gibt es $x \notin L$ für die dort konstruierte Reduktion f, für die $\mathrm{opt}(f(x))$ beliebig nahe an (aber echt unter) 1 liegt.

   (Beweis → Übungszettel)

Das zweite PCP-Theorem

1. **Satz.** Es gibt ein $\rho < 1$, so dass für jedes $L \in \mathbf{NP}$ gilt, dass $L \leq_{\mathrm{m}}^{\mathrm{p}} 3\mathrm{SAT}$ via ein $f \in \mathbf{FP}$, so dass,
   - falls $x \in L$, dann $\mathrm{opt}(f(x)) = 1$,
   - falls $x \notin L$, dann $\mathrm{opt}(f(x)) < \rho$.
2. **Testfrage.** Vergleichen Sie das mit dem Satz von Cook-Levin!

   Antwort. Der Satz von Cook-Levin garantiert keine „Lücke“. Tatsächlich gibt es $x \notin L$ für die dort konstruierte Reduktion f, für die $\mathrm{opt}(f(x))$ beliebig nahe an (aber echt unter) 1 liegt.

   (Beweis → Übungszettel)
3. **Korollar.** Es gibt ein $\rho < 1$, so dass die Existenz eines in Polynomialzeit arbeitenden ρ-Approximationsalgorithmus' für Max3SAT implizieren würde, dass $\mathbf{P} = \mathbf{NP}$.
4. **Testfrage.** Wie folgt das aus dem Satz?

Äquivalenz der beiden PCP-Theoreme

1. Wir werden zeigen, dass beide PCP-Theoreme äquivalent sind.
2. Als „Übersetzungshilfe“ definieren wir eine Klasse von Sprachen und formulieren einen Satz dazu. Dann zeigen wir dass dieser Satz zu *beiden* PCP-Theoremen äquivalent ist.

Das Constraint Satisfaction Problem

1. **Definition.** Eine *Instanz des q-Constraint Satisfaction Problems* ist eine Liste von m Booleschen Funktionen in jeweils q Variablen
$$\varphi_1(X_{i_{1,1}},\dots,X_{i_{1,q}}) \quad \dots \quad \varphi_m(X_{i_{m,1}},\dots,X_{i_{m,q}})$$
mit $i_{j,k} \in \{1,\dots,n\}$ für alle j,k.
2. **Definition.** Eine Belegung σ der Variablen $X_1,\dots,X_n$ ist *erfüllend* für diese Instanz, falls $\varphi_i(\sigma) = 1$ für jedes i.
3. **Intuition.** Alle Constraints sollen gleichzeitig wahr werden.
4. **Definition.** Das *q-Constraint Satisfaction Problem* qCSP sei
$$q\mathrm{CSP} = \left\{ x \in \{0,1\}^{<\mathbb{N}} : \begin{array}{c} x \text{ kodiert eine Instanz des} \\ q\text{-Constraint Satisfaction Problem, für} \\ \text{die eine erfüllende Belegung existiert.} \end{array} \right\}.$$
5. **Testfrage.** Warum ist 3SAT ein Spezialfall eines qCSP?

qCSP mit Lücke

1. **Definition.** Für eine Instanz x und Belegung σ von qCSP sei
$$\mathrm{qua}(x,\sigma) = {}^{1}\!/_{m} \cdot \#\{1 \le i \le m \colon \varphi_i(\sigma) = 1\},$$
also der *Anteil* der Constraints, der durch σ wahr wird.
2. qCSP wird uns nicht als „normales“ Entscheidungsproblem interessieren, sondern nur in der folgenden Form „mit Lücke“.
3. **Definition.** ρGap qCSP ist die Aufgabe, für eine Instanz x von qCSP, so dass
   - entweder $\mathrm{opt}(x) = 1$
   - oder $\mathrm{opt}(x) < \rho$

   gilt, diese beiden Fälle zu unterscheiden.
4. **Testfrage.** Warum ist das kein „normales“ Entscheidungsproblem?
(Man spricht von „promise problems“.)

NP-Schwere von GapCSP

1. Wir passen die normale Definition von **NP**-Schwere für Entscheidungsprobleme an den Sonderfall ρGap$q$CSP an.
2. **Definition.** Wir sagen ρGap$q$CSP ist **NP**-schwer, falls für jede Sprache $L \in \mathbf{NP}$ eine Polynomialzeitreduktion f existiert, so dass
   - f Wörter x auf qCSP-Instanzen $f(x)$ abbildet;
   - falls $x \in L$, dann $\mathrm{opt}(f(x)) = 1$;
   - falls $x \notin L$, dann $\mathrm{opt}(f(x)) < \rho$.
3. **Testfrage.** Warum ist das eine sinnvolle Definition?

NP-Schwere von GapCSP

1. Wir passen die normale Definition von **NP**-Schwere für Entscheidungsprobleme an den Sonderfall ρGap$q$CSP an.
2. **Definition.** Wir sagen ρGap$q$CSP ist **NP**-schwer, falls für jede Sprache $L \in \mathbf{NP}$ eine Polynomialzeitreduktion f existiert, so dass
   - f Wörter x auf qCSP-Instanzen $f(x)$ abbildet;
   - falls $x \in L$, dann $\mathrm{opt}(f(x)) = 1$;
   - falls $x \notin L$, dann $\mathrm{opt}(f(x)) < \rho$.
3. **Testfrage.** Warum ist das eine sinnvolle Definition?
4. **Antwort.**
   - Die Definition verlangt, dass sich Fragen zu **NP**-Entscheidungsproblemen via f auf Fragen zu ρGap$q$CSP reduzieren lassen.
   - Ein polynomieller Algorithmus für ρGap$q$CSP ließe sich mit f kombinieren zu einem polynomiellen Algorithmus für L.
   - Das bildet die klassische **NP**-schwer-Definition nach.

NP-Schwere von GapCSP

1. **Satz.** Es gibt q und $\rho \in (0,1)$, so dass $\rho\mathrm{Gap}\,q\mathrm{CSP}$ **NP**-schwer ist.
2. Wir zeigen die Äquivalenz des Satzes zu *beiden* PCP-Theoremen.
3. Wir beginnen mit der Äquivalenz zur Existenz von **PCP**(log)-Prüfern für alle Sprachen in **NP**.

Beweis 1: **PCP**(log)-Prüfer $\rightsquigarrow$ $1/2$Gap qCSP **NP**-schwer

1. Nimm an $\mathbf{NP} \subseteq \mathbf{PCP}(\log)$.
2. Wir zeigen, dass $1/2$Gap qCSP **NP**-schwer ist, indem wir 3SAT-Instanzen in $1/2$Gap qCSP-Instanzen übersetzen.

   Testfrage. Warum genügt das?
3. Nach Annahme hat 3SAT einen Prüfer V der bei Eingabe x $c \log |x|$ Zufallsbits nutzt und $q \in O(1)$ viele Bits von π liest.

Beweis 1: **PCP**(log)-Prüfer $\rightsquigarrow$ $1/2$Gap qCSP **NP**-schwer

1. Für feste Eingabe x und Zufallsbits r sind die Anfragen, die V stellt, eindeutig bestimmt. Seien $\pi_1^{x,r}, \ldots, \pi_q^{x,r}$ Variablen, die die entsprechenden Antworten enthalten mögen.
2. Schreibe $\varphi_{x,r}(\pi_1^{x,r}, \ldots, \pi_q^{x,r})$ für die Ausgabe von V bei Eingabe x und Zufallsbits r in Abhängigkeit von den Werten $\pi_1^{x,r}, \ldots, \pi_q^{x,r}$.
3. Jedes $\varphi_{x,r}$ ist eine Boolesche Funktion in q Variablen.
4. Für jedes x ist dann $\varphi_x := \{\varphi_{x,r}\}_{r \in \{0,1\}^{c \log |x|}}$ eine qCSP-Instanz.
5. **Testfrage.** Warum ist $x \mapsto \varphi_x$ in Polynomialzeit berechenbar?
6. Wegen der Vollständigkeit des PCP-Systems sind, falls $x \in 3\mathrm{SAT}$, alle Constraints in φ_x gleichzeitig erfüllbar, also $\mathrm{opt}(\varphi_x) = 1$.
7. Falls $x \notin 3\mathrm{SAT}$, ist aufgrund der Korrektheit des PCP-Systems höchstens die Hälfte der Constraints erfüllbar, so $\mathrm{opt}(\varphi_x) \leq 1/2$.

Beweis 2: ρGap qCSP **NP**-schwer $\rightsquigarrow$ **PCP**(log)-Prüfer

1. ρGap qCSP sei **NP**-schwer für Konstanten $\rho < 1$, $q \in \mathbb{N}$.
2. Sei $L \in \mathbf{NP}$ beliebig und sei f eine Reduktion auf ρGap qCSP.
3. Wir müssen einen **PCP**(log)-Prüfer V für L konstruieren.

Beweis 2: ρGap qCSP **NP**-schwer $\rightsquigarrow$ **PCP**(log)-Prüfer

1. V arbeitet bei Eingabe x wie folgt:
   - V berechnet $f(x)$ und wählt zufällig einen Constraint darin aus.
   - V interpretiert π als eine Belegung der Variablen in $f(x)$.
   - V liest aus π die q Bits aus, von denen der Constraint abhängt.
   - Falls diese den Constraint erfüllen, akzeptiere. Sonst verwirf.
2. V arbeitet korrekt, denn nach Annahme gilt, dass…
   - falls $x \in L$ ist, alle Constraints in $f(x)$ gleichzeitig erfüllbar sind. Wenn π eine erfüllende Belegung kodiert, akzeptiert V sicher.
   - falls $x \notin L$, jede Belegung echt weniger als Anteil ρ der Constraints erfüllt. Also akzeptiert V mit Wahrscheinlichkeit $< \rho$.
3. Falls $\rho \leq 1/2$ sind wir fertig. Falls nicht benutzen wir die obige Anmerkung, dass der *Korrektheits*parameter $1/2$ durch beliebige $0 < p < 1$ ersetzt werden kann. *(Beweis → Übungszettel)*

Beweis 3: 3SAT mit Lücke $\rightsquigarrow$ ρGap 3CSP **NP**-schwer

1. 3SAT-Instanzen sind (leicht umkodiert) Instanzen von 3CSP.
2. Gibt es also für beliebige $L \in \mathbf{NP}$ eine Reduktion auf 3SAT mit Lücke ρ, so gibt es auch eine Reduktion von L auf ρGap 3CSP.
3. Nach Definition ist dann ρGap 3CSP **NP**-schwer.

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Sei $(1-\varepsilon)$Gap qCSP **NP**-schwer und $L \in \mathbf{NP}$.
2. Dann gibt es ein $f \in \mathbf{FP}$, so dass $f(x)$ eine qCSP-Instanz ist mit
$$\mathrm{opt}(f(x)) \begin{cases} = 1 & \text{falls } x \in L, \\ \leq (1-\varepsilon) & \text{falls } x \notin L. \end{cases}$$
3. Unser Ziel ist nun, $f(x)$ in eine 3SAT-Instanz umzuwandeln, aber dabei eine „Lücke“ zu erhalten. (Die Lücke darf dabei kleiner werden).
4. Wie kann man also eine qCSP- als 3SAT-Instanz darstellen?

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. **Unser Ziel:** Eine qCSP-Instanz φ als eine erfüllungsäquivalente 3SAT-Instanz φ'' darstellen.

2. Ein Constraint φ_i ist eine Boolesche Funktion in q Variablen.
3. **Behauptung.** Es gibt zu φ_i eine konjunktive Normalform mit höchstens 2^q Klauseln mit je q Literalen.

   Testfrage. Warum? (Siehe die Vorlesung „Mathematische Logik“!)

4. Wir ersetzen also alle φ_i's in φ durch je eine solche konjunktive Normalform, und fügen diese zu einer großen, zu φ erfüllungsäquivalenten konjunktiven Normalform φ' zusammen.
5. **Zu klären:** Was passiert dabei mit der Lücke?

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Falls in φ alle Constraints gleichzeitig erfüllbar waren, so sind auch alle Klauseln von φ' gleichzeitig erfüllbar. ✓
2. Nimm also an, dass jede Belegung σ in φ einen Anteil größer ε der Constraints *un*erfüllt lässt, und sei φ_i ein solcher Constraint.
3. Aus φ_i wurde in φ' eine *Liste* von $\leq 2^q$ Klauseln, von denen *alle* wahr sein müssen, damit φ_i wahr ist, und umgekehrt. Da φ_i aber unerfüllt ist, ist *mindestens eine* der $\leq 2^q$ Klauseln unerfüllt.
4. Das gleiche gilt für die anderen durch σ unerfüllten Constraints.
5. Da diese einen Anteil ε aller Constraints in φ ausmachen, ist in φ' mindestens Anteil $\varepsilon/2^q$ aller Klauseln durch σ unerfüllt. ✓
6. Die Lücke bleibt also (wenn auch etwas kleiner) erhalten.

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Aus einer Liste φ von m Constraints, die erfüllt werden sollen, wird also eine konjunktive Normalform φ' mit $\leq m2^q$ Klauseln.
2. Wir müssen noch die Literale pro Klausel von q auf 3 drücken.
3. **Testfrage.** Wie? (Siehe die Vorlesung „Komplexitätstheorie“!)
4. Die entstehende 3SAT-Instanz φ'' hat höchstens $qm2^q$ Klauseln.
5. **Zu klären:** Was passiert dabei mit der Lücke?

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Durch das Aufteilen der q-Literal- in mehrere 3-Literal-Klauseln verteilen sich unerfüllte Teile der Klauseln irgendwie.
2. Wie genau ist unwichtig, aber wenn eine Klausel vorher unerfüllt war, ist danach *mindestens eine* der $\leq q$ neuen Klauseln unerfüllt.
3. Deshalb gilt $\operatorname{opt}(\varphi'') < 1 - \varepsilon/q2^q$. ✓

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Durch das Aufteilen der q-Literal- in mehrere 3-Literal-Klauseln verteilen sich unerfüllte Teile der Klauseln irgendwie.
2. Wie genau ist unwichtig, aber wenn eine Klausel vorher unerfüllt war, ist danach *mindestens eine* der $\leq q$ neuen Klauseln unerfüllt.
3. Deshalb gilt $\operatorname{opt}(\varphi'') < 1 - \varepsilon/q2^q$. ✓
4. Wir haben also eine qCSP-Instanz in eine erfüllungsäquivalente 3SAT-Instanz umgewandelt, und dabei eine Lücke bewahrt.

Beweis 4: $(1-\varepsilon)$Gap qCSP **NP**-schwer $\rightsquigarrow$ 3SAT mit Lücke

1. Durch das Aufteilen der q-Literal- in mehrere 3-Literal-Klauseln verteilen sich unerfüllte Teile der Klauseln irgendwie.
2. Wie genau ist unwichtig, aber wenn eine Klausel vorher unerfüllt war, ist danach *mindestens eine* der $\leq q$ neuen Klauseln unerfüllt.
3. Deshalb gilt $\operatorname{opt}(\varphi'') < 1 - \varepsilon/q2^q$. ✓
4. Wir haben also eine qCSP-Instanz in eine erfüllungsäquivalente 3SAT-Instanz umgewandelt, und dabei eine Lücke bewahrt.
5. Die Äquivalenz der beiden PCP-Theoreme ist bewiesen. □

Kapitelzusammenfassung

1. Wir haben zwei Versionen des PCP-Theorems kennengelernt.
2. Einerseits besagt das PCP-Theorem, dass es für *alle* Sprachen in **NP** Zertifikate gibt, die nur an sehr wenigen Stellen inspiziert werden müssen, um sich probabilistisch von der Korrektheit des Zertifikats zu überzeugen.
3. Andererseits besagt das PCP-Theorem falls $\mathbf{P} = \mathbf{NP}$, dass sich Max3SAT nur schlecht approximieren lässt.
4. Wir habe die Äquivalenz dieser beiden Sichtweisen bewiesen.

12 Konsequenzen des PCP-Theorems

Konsequenzen des PCP-Theorems

1. Laut PCP-Theorem ist Max3SAT schwer zu approximieren.
2. Doch es folgen daraus in vielen Fällen auch für weitere **NP**-vollständige Probleme ähnliche Ergebnisse.
3. Wir illustrieren das an folgenden Graphenproblemen:
   - MaxIS, der Suche nach größtmöglichen unabhängigen Mengen
   - MinVC, der Suche nach kleinstmöglichen Knotenüberdeckungen

Vorbereitung

1. **Lemma.** Es gibt eine Abbildung $f \in \mathbf{FP}$, die 3SAT-Instanzen F auf Graphen $f(F)$ abbildet;
   - dabei hängt $\#V(f(F))$ nur von der Anzahl der Klauseln in F ab
     (insbesondere hängt $\#V(f(F))$ nicht von $\mathrm{opt}(F)$ ab)
   - und die größte unabhängige Menge in $f(F)$ hat die Größe

$$\mathrm{opt}(F) \cdot \#V(f(F)) \cdot {}^{1}/_{7}.$$

(Beweis → Übungszettel)

Größtmögliche unabhängige Mengen

1. **Satz.** Falls $\mathbf{P} \neq \mathbf{NP}$, dann gibt es eine Konstante $\rho < 1$, so dass MaxIS nicht in Polynomialzeit ρ-approximiert werden kann.
2. **Beweis.**
   - Sei L **NP**-vollständig.
   - Nimm an, es gebe einen Polynomialzeit-ρ-Approximations-algorithmus für MaxIS. Wir werden daraus ein deterministisches Polynomialzeit-Entscheidungsverfahren für L bauen.
   - Dies wäre ein Widerspruch, denn wegen der Annahme $\mathbf{P} \neq \mathbf{NP}$ kann es kein solches Entscheidungsverfahren für L geben.

Größtmögliche unabhängige Mengen

1. Wir wollen entscheiden, ob $x \in L$. Laut zweitem PCP-Theorem gibt es $\rho < 1$ und $g \in \mathbf{FP}$, so dass $g(x)$ eine 3SAT-Instanz ist mit
   - $\mathrm{opt}(g(x)) = 1$, falls $x \in L$;
   - $\mathrm{opt}(g(x)) < \rho$, falls $x \notin L$.
2. Darauf wenden wir f (aus der Vorbereitung) an und erhalten einen Graph mit n Knoten, dessen größte unabhängige Menge Größe
   - $\mathrm{opt}(f \circ g(x)) = \mathrm{opt}(g(x)) \cdot n/7 = \quad n/7$ hat, falls $x \in L$,
   - $\mathrm{opt}(f \circ g(x)) = \mathrm{opt}(g(x)) \cdot n/7 < \rho \cdot n/7$ hat, falls $x \notin L$.
3. Darauf wende den ρ-Approximationsalgorithmus für MaxIS an.
   - Falls $x \in L$, so findet dieser eine unabhängige Menge von der Größe mindestens $\rho \cdot \mathrm{opt}(f \circ g(x)) = \rho \cdot n/7$.
   - Falls $x \notin L$, dann ist *jede* unabhängige Menge echt kleiner.
4. An der Approximation zeigt sich also, ob $x \in L$ oder nicht. $\square$

Kleinstmögliche Knotenüberdeckungen

1. **Satz.** Falls $\mathbf{P} \neq \mathbf{NP}$, dann gibt es eine Konstante $\rho' < 1$, so dass MinVC nicht in Polynomialzeit ρ'-approximiert werden kann.
2. **Beweis.**
   - Gehe zunächst vor wie im letzten Beweis. **Testfrage.** Warum hat die kleinste Knotenüberdeckung in $f \circ g(x)$ Größe $n - \mathrm{opt}(g(x)) \cdot {}^{n}/_{7}$?
   - Sei $\rho' = \frac{6}{7-\rho}$.
   - Gibt es einen ρ'-Approximationsalgorithmus für MinVC, dann
     - falls $x \in L$, findet er eine Knotenüberdeckung der Größe höchstens
     $$ {}^{1}/_{\rho'} \cdot \Big(n - \underbrace{\mathrm{opt}(g(x))}_{1} \cdot {}^{n}/_{7}\Big) = \frac{7-\rho}{6} \cdot \frac{6n}{7} = n - \rho \cdot \frac{n}{7}; $$
     - falls $x \notin L$, ist die kleinste Knotenüberdeckung *echt* größer als
     $$ n - \mathrm{opt}(g(x)) \cdot {}^{n}/_{7} = n - \rho \cdot \frac{n}{7}. $$
   - Wieder wäre „$x \in L$?“ an der Approximation entscheidbar. □

Eine Stärkung der Aussage für MaxIS

1. Wir können die Aussage für MaxIS noch deutlich stärken:

   Sie gilt sogar für *alle* Konstanten ρ!

2. **Satz.** Falls $\mathbf{P} \neq \mathbf{NP}$, dann gilt für *jedes* $0 < \rho'' \leq 1$, dass MaxIS nicht in Polynomialzeit ρ''-approximiert werden kann.

3. **Idee.**
   - Verkleinert man das Zahlenverhältnis zwischen $\rho \cdot n/7$ und $n/7$ weiter auf $\rho'' \ll \rho$, würde bereits ein ρ''-Approximationsalgorithmus die Fälle „$x \in L$“ und „$x \notin L$“ unterscheidbar machen.
   - Die Annahme, ein solcher Algorithmus existiere, führt dann wieder zum gleichen Widerspruch.
   - Um die Lücke zu verstärken, wird der Graph $f \circ g(x)$ vor dem Ausführen des Approximationsalgorithmus noch mit einer Abbildung h weiterverarbeitet.

Ein Hilfsmittel aus der Graphentheorie

1. **Definition.** Für $k \in \mathbb{N}$ und einen Graph G mit n Knoten sei G_k der Graph mit $V(G_k) = \{S \subseteq V(G) \colon \#S = k\}$ und

$$E(G_k) = \{(S_1, S_2) \colon S_1 \cup S_2 \text{ ist keine unabhängige Menge in } G\}.$$

2. **Behauptung.** Jede größte unabhängige Menge J in G_k hat die Form $\{S \subseteq I \colon \#S = k\}$ mit I eine größte unabhängige Menge in G.
3. **Beweis.**
   - Sei J eine beliebige größte unabhängige Menge in G_k. Dann ist $I = \bigcup_{S \in J} S$ offensichtlich eine unabhängige Menge in G und folglich ist $\widehat{J} = \{S \subseteq I \colon \#S = k\}$ eine unabhängige Menge in G_k.
   - Sei $\{v_1, \ldots, v_k\} \in J$. Dann $v_1, \ldots, v_k \in I$, und so $\{v_1, \ldots, v_k\} \in \widehat{J}$. Also $J \subseteq \widehat{J}$, und wegen Maximalität $J = \widehat{J}$. Also hat J obige Form.
   - Nimm an, I sei *nicht* eine der *größten* unabhängigen Mengen in G. Sei also I' eine unabhängige Menge mit $\#I < \#I'$.
   - Dann ist $J' := \{S \subseteq I' \colon \#S = k\}$ eine unabhängige Menge in G_k. Dann $\#J = \binom{\#I}{k} < \binom{\#I'}{k} = \#J'$, J's Maximalität widersprechend. $\square$

Eine Stärkung der Aussage für MaxIS

1. Sei $h = (G \mapsto G_k)$. **Testfrage.** Warum ist $h \in$ **FP**?
2. Wir müssen uns überlegen, wie h die „Lücke“ verändert.
3. Zunächst bemerken wir, dass nur die „relative“ Größe der Lücke relevant ist, also das Zahlenverhältnis zwischen der Größe der größten unabhängigen Menge im Fall „$x \in L$“ verglichen mit dem Fall „$x \notin L$“.
4. Da in beiden Fällen G die gleiche Knotenzahl hat, hat auch in beiden Fällen G_k die gleiche Knotenzahl. Diese Zahl kann man also ignorieren, und nur den *Quotienten* der Größen der größten unabhängigen Mengen beider Fälle betrachten.

Eine Stärkung der Aussage für MaxIS

1. Aufgrund der Behauptung gilt, das h ein G mit größter unabhängiger Menge der Größe ℓ abbildet auf ein G_k mit größter unabhängige Menge der Größe $\binom{\ell}{k}$.
2. Bei mehrmaliger Anwendung entwickelt sich der Quotient so:

$$\frac{\rho \cdot n/7}{n/7} \overset{h}{\rightsquigarrow} \frac{\binom{\rho \cdot n/7}{k}}{\binom{n/7}{k}} \overset{h}{\rightsquigarrow} \frac{\binom{\binom{\rho \cdot n/7}{k}}{k}}{\binom{\binom{n/7}{k}}{k}} \overset{h}{\rightsquigarrow} \frac{\binom{\binom{\binom{\rho \cdot n/7}{k}}{k}}{k}}{\binom{\binom{\binom{n/7}{k}}{k}}{k}} \overset{h}{\rightsquigarrow} \dots$$

3. Durch wiederholte Anwendung von h kann man den Quotienten beliebig drücken. Die genaue Rechnung ersparen wir uns. $\square$

Kapitelzusammenfassung

1. Wir haben unter der Annahme $\mathbf{P} \neq \mathbf{NP}$ aus dem PCP-Theorem gefolgert, dass es eine Konstante $\rho < 1$ gibt, so dass das Problem der kleinsten Knotenüberdeckung MinVC nicht bis auf die Genauigkeit ρ in Polynomialzeit approximiert werden kann.
2. Eine analoge Aussage, jedoch sogar für *alle* $0 < \rho \leq 1$, gilt für MaxIS, das Problem der größten unabhängigen Mengen.
3. Die Beispiele illustrieren die Relevanz des PCP-Theorems für die praktische Informatik.

13 Schaltkreiskomplexität

Vorbemerkungen

1. Bisher haben wir Algorithmen durch Turingmaschinen (oder äquivalente Modelle) dargestellt. Dabei beschreibt *ein* Objekt, z.B. eine Übergangsfunktion, den Algorithmus für Eingaben *aller* Längen.
2. Jetzt betrachten wir ein anderes Rechenmodell: Schaltkreise.
3. Wir kennen sie bereits aus der Vorlesung „Komplexitätstheorie," allerdings nicht als Berechnungsmodell, sondern als *Eingaben*.
4. Wie jede Hardware ist ein Schaltkreis endlich groß, hat also nur endlich viele Eingänge und kann nur Eingaben passender Größe verarbeiten.

Schaltkreiskomplexität

1. **Testfrage.** Wie hatten wir Schaltkreise in der Vorlesung „Komplexitätstheorie“ definiert?

Schaltkreiskomplexität

1. **Testfrage.** Wie hatten wir Schaltkreise in der Vorlesung „Komplexitätstheorie“ definiert?
2. **Definition.** Für $L \subseteq \{0,1\}^{<\mathbb{N}}$ sei $c_L : \mathbb{N} \to \mathbb{N}$ die Abbildung

$$n \mapsto \min \left\{ \ell \in \mathbb{N} : \begin{array}{c} \text{Es gibt einen Schaltkreis } C \text{ mit } n \text{ Ein-} \\ \text{gabeleitungen und } \ell \text{ Gattern, so dass für} \\ \text{alle } x \in \{0,1\}^n \text{ gilt, dass } C(x) = L(x). \end{array} \right\}.$$

3. **Intuition.**
   - Für jede Eingabelänge gibt es einen eigenen Schaltkreis.
   - Es wird immer der kleinste Schaltkreis berücksichtigt, der für *alle* Eingaben der gleichen Länge die richtige Antwort gibt.
   - Es wird kein „Muster“ gefordert; Schaltkreise für verschiedene Eingabelängen müssen nichts miteinander zu tun haben. Man spricht von *nichtuniformer* Komplexität.
   - Statt *einer* Turingmaschine, die eine Sprache erkennt, gibt es hier nun also Schaltkreis*familien*, die das gemeinsam tun.

Polynomielle Schaltkreise

1. **Testfrage.** Ist c_L für *jedes* $L \subseteq \{0,1\}^{<\mathbb{N}}$ definiert?

Polynomielle Schaltkreise

1. **Testfrage.** Ist c_L für *jedes* $L \subseteq \{0,1\}^{<\mathbb{N}}$ definiert?
2. **Definition.** Für $L \subseteq \{0,1\}^{<\mathbb{N}}$ sagen wir L *hat polynomielle Schaltkreise*, falls c_L durch ein Polynom beschränkt wird.

Polynomielle Schaltkreise

1. **Testfrage.** Ist c_L für *jedes* $L \subseteq \{0,1\}^{<\mathbb{N}}$ definiert?
2. **Definition.** Für $L \subseteq \{0,1\}^{<\mathbb{N}}$ sagen wir L *hat polynomielle Schaltkreise*, falls c_L durch ein Polynom beschränkt wird.
3. **Testfrage.** Ist jedes solche L entscheidbar?

Polynomielle Schaltkreise

1. **Testfrage.** Ist c_L für *jedes* $L \subseteq \{0,1\}^{<\mathbb{N}}$ definiert?
2. **Definition.** Für $L \subseteq \{0,1\}^{<\mathbb{N}}$ sagen wir L *hat polynomielle Schaltkreise*, falls c_L durch ein Polynom beschränkt wird.
3. **Testfrage.** Ist jedes solche L entscheidbar?

   Antwort. Nein, es gibt offensichtliche Gegenbeispiele.

Adviceklassen

1. **Definition.** Sei $\mathbf{C}$ eine Menge von Sprachen und $\mathbf{F}$ eine Menge von Funktionen von $\mathbb{N}$ nach $\{0,1\}^{<\mathbb{N}}$. Dann sei

$$\mathbf{C}/\mathbf{F} := \{L\colon\ \exists A{\in}\mathbf{C}\ \exists f{\in}\mathbf{F}\,(x \in L \longleftrightarrow \langle x, f(|x|)\rangle \in A)\}.$$

2. **Intuition.**
   - Funktionen wie oben nennt man *Advice-Funktionen*.
   - Nur mit Hilfsinformationen („Advice“) läge L in $\mathbf{C}$.
   - Diese dürfen nur von der *Länge* einer Eingabe abhängen.

Polynomieller Advice

1. **Definition.** Sei **poly** die Menge der $f : \mathbb{N} \to \{0, 1\}^{<\mathbb{N}}$ für die ein Polynom p existiert mit $|f(n)| < p(n)$ für alle n.
2. **Beispiel.** Im Folgenden wird uns **P/poly** interessieren.
3. **Intuition.**
   - Für $L \in \mathbf{P/poly}$ lässt sich „$x \in L$?“ in polynomieller Zeit entscheiden, wenn man eine polynomiell lange Zusatzinformation erhält, die nur von der Länge von x abhält.

Polynomielle Adviceklassen

1. **Testfrage.** Wie verhalten sich **P** und **P/poly** zueinander?

Polynomielle Adviceklassen

1. **Testfrage.** Wie verhalten sich **P** und **P/poly** zueinander?
2. **Testfrage.** Sei **fp** die Menge der in Polynomialzeit berechenbaren Advice-Funktionen. Was ist dann **P/fp**?

Polynomielle Adviceklassen

1. **Testfrage.** Wie verhalten sich **P** und **P/poly** zueinander?
2. **Testfrage.** Sei **fp** die Menge der in Polynomialzeit berechenbaren Advice-Funktionen. Was ist dann **P/fp**?
3. **Testfrage.** Haben Sie einen Verdacht, wie man **P/poly** charakterisieren könnte?

Advice und Schaltkreise

1. **Satz.** Die Menge **P/poly** enthält genau die Sprachen, die polynomielle Schaltkreise haben.

Advice und Schaltkreise

1. **Satz.** Die Menge **P/poly** enthält genau die Sprachen, die polynomielle Schaltkreise haben.
2. **Testfrage.**
   - Wie hatten wir das *Circuit Value Problem* CVP in der Vorlesung „Komplexitätstheorie“ definiert?
   - Welche wichtige Eigenschaft hatte CVP?

Beweis: Schaltkreise $\rightsquigarrow$ Advice

1. Sei L eine Sprache, die polynomielle Schaltkreise hat.
2. Also gibt es ein Polynom p, so dass für jedes n ein Schaltkreis C_n existiert mit n Eingabeleitungen und höchstens $p(n)$ Gattern, so dass für alle $x \in \{0,1\}^n$ gilt, dass $C_n(x) = L(x)$.
3. Sei f die Abbildung $n \mapsto \overline{C_n}$; offensichtlich gilt $f \in \mathbf{poly}$.
4. **Behauptung.** $\{\langle x, f(|x|)\rangle : x \in L\} \in \mathbf{P}$.

   Testfrage. Warum? Reduzieren Sie auf eine Sprache in **P**!
5. Nach Definition ist dann auch $L \in \mathbf{P}/\mathbf{poly}$. ✓

Beweis: Advice $\rightsquigarrow$ Schaltkreise

1. Sei $L \in \mathbf{P}/\mathbf{poly}$. Also existiert ein $f \in \mathbf{poly}$, so dass
$$A := \{\langle x, f(|x|)\rangle : x \in L\} \in \mathbf{P}.$$
2. Da CVP **P**-schwer ist, gilt $A \leq_{\mathrm{m}}^{\mathrm{p}}$ CVP.
   - In der Vorlesung „Komplexitätstheorie“ hatten wir m-Reduktionen $g \in \mathbf{FL} \subseteq \mathbf{FP}$ von beliebigen $L \in \mathbf{P}$ auf CVP konstruiert.
   - Aus einer Eingabe y erzeugte g einen Schaltkreis und eine Belegung der Eingabeleitungen. Dabei hing der Schaltkreis nur von $|y|$ ab, nicht von y. Nur die Belegung hing auch von y ab.
3. Sei ein solches g für unsere Menge A gegeben.
   - Der erzeugte Schaltkreis hängt dann nur von $|\langle x, f(|x|)\rangle|$ ab, also o.E. nur von $|x|$ (ansonsten reduziere A zunächst auf geeignetes A').
   - Seine Größe ist polynomiell in $|\langle x, f(|x|)\rangle|$, also in $|x|$.
   - Die *konkreten* Werte von x und von $f(|x|)$ wirken sich o.E. nur auf die von g erzeugte Belegung aus.

Beweis: Advice $\rightsquigarrow$ Schaltkreise

1. Für alle n sei nun C_n der eindeutig bestimmte Schaltkreis, den g bei Eingaben der Form $\langle x, f(|x|)\rangle$ mit $x \in \{0,1\}^n$ erzeugt.
2. Wandle alle C_n so zu C'_n ab, dass alle Eingabeleitungen fest belegt werden mit den Bitwerten in der von f erzeugten Belegung.
3. *Ausgenommen* seien hiervon die Eingabeleitungen, die den Bits von x entsprechen; diese sollen freie Eingabeleitungen bleiben.
4. Dann gilt für alle C'_n, dass sie
   - n Eingabeleitungen haben,
   - polynomiell groß in n sind.
5. Weiterhin gilt
$$x \in L \Leftrightarrow \langle x, f(|x|)\rangle \in A \Leftrightarrow g(\langle x, f(|x|)\rangle) \in \mathrm{CVP} \Leftrightarrow C'_n(x) = 1.$$
6. Also hat L polynomiell große Schaltkreise via $(C'_n)_{n\in\mathbb{N}}$. $\square$

NP versus P/poly

1. **Testfrage.** Vergleichen Sie die Definition von **P/poly** mit der Charakterisierung von **NP** durch den Projektionssatz, den wir in der Vorlesung „Komplexitätstheorie“ bewiesen hatten.

NP versus P/poly

1. **Testfrage.** Vergleichen Sie die Definition von **P/poly** mit der Charakterisierung von **NP** durch den Projektionssatz, den wir in der Vorlesung „Komplexitätstheorie“ bewiesen hatten.
2. Oberflächlich betrachtet sehen sich beide Aussagen ähnlich.

   (... aber wirklich nur oberflächlich betrachtet!)

NP versus P/poly

1. **Testfrage.** Vergleichen Sie die Definition von **P/poly** mit der Charakterisierung von **NP** durch den Projektionssatz, den wir in der Vorlesung „Komplexitätstheorie“ bewiesen hatten.
2. Oberflächlich betrachtet sehen sich beide Aussagen ähnlich.

   (... aber wirklich nur oberflächlich betrachtet!)
3. Man kann sich also fragen, welcher Zusammenhang zwischen beiden Komplexitätsklassen besteht.
4. **Testfrage.** Kann **P/poly** $\subseteq$ **NP** gelten?

NP versus P/poly

1. **Testfrage.** Vergleichen Sie die Definition von **P/poly** mit der Charakterisierung von **NP** durch den Projektionssatz, den wir in der Vorlesung „Komplexitätstheorie“ bewiesen hatten.
2. Oberflächlich betrachtet sehen sich beide Aussagen ähnlich.

   (... aber wirklich nur oberflächlich betrachtet!)
3. Man kann sich also fragen, welcher Zusammenhang zwischen beiden Komplexitätsklassen besteht.
4. **Testfrage.** Kann **P/poly** $\subseteq$ **NP** gelten?

   Antwort. Nein. Es gibt ja sogar unentscheidbare $L \in$ **P/poly**.

NP versus P/poly

1. **Testfrage.** Vergleichen Sie die Definition von **P/poly** mit der Charakterisierung von **NP** durch den Projektionssatz, den wir in der Vorlesung „Komplexitätstheorie“ bewiesen hatten.
2. Oberflächlich betrachtet sehen sich beide Aussagen ähnlich.

   (... aber wirklich nur oberflächlich betrachtet!)
3. Man kann sich also fragen, welcher Zusammenhang zwischen beiden Komplexitätsklassen besteht.
4. **Testfrage.** Kann $\mathbf{P/poly} \subseteq \mathbf{NP}$ gelten?

   Antwort. Nein. Es gibt ja sogar unentscheidbare $L \in \mathbf{P/poly}$.
5. **Offene Frage.** Kann $\mathbf{NP} \subseteq \mathbf{P/poly}$ gelten?

Der Satz von Karp & Lipton

1. **Satz.** Falls $\mathbf{NP} \subseteq \mathbf{P/poly}$, dann $\mathbf{PH} = \Sigma_2 = \Pi_2$, also kollabiert die Polynomialzeithierarchie auf der zweiten Stufe.
2. Bevor wir den Satz beweisen, betrachten wir noch kurz ein Phänomen, das im Beweis ausgenutzt wird.

Selbstreduzierbarkeit

1. **Definition.** Eine Menge A heißt *selbstreduzierbar*, wenn es eine in Polynomialzeit arbeitende, deterministische Orakel-Turing-maschine M gibt, so dass
   - $A = L(M, A)$, und
   - bei Eingaben der Länge n befragt M das Orakel nur nach Wörtern, deren Länge höchstens $n - 1$ ist.

Selbstreduzierbarkeit

1. **Definition.** Eine Menge A heißt *selbstreduzierbar*, wenn es eine in Polynomialzeit arbeitende, deterministische Orakel-Turing-maschine M gibt, so dass
   - $A = L(M, A)$, und
   - bei Eingaben der Länge n befragt M das Orakel nur nach Wörtern, deren Länge höchstens $n - 1$ ist.

2. **Beispiel.** SAT ist selbstreduzierbar.

   (Streng genommen setzt das voraus, dass SAT-Instanzen so kodiert werden, dass kürzere Formeln kürzere Codes erhalten. Dies sollte für vernünftige Kodierungen aber gelten.)

Selbstreduzierbarkeit

1. **Definition.** Eine Menge A heißt *selbstreduzierbar*, wenn es eine in Polynomialzeit arbeitende, deterministische Orakel-Turing-maschine M gibt, so dass
   - $A = L(M, A)$, und
   - bei Eingaben der Länge n befragt M das Orakel nur nach Wörtern, deren Länge höchstens $n - 1$ ist.

2. **Beispiel.** SAT ist selbstreduzierbar.

   (Streng genommen setzt das voraus, dass SAT-Instanzen so kodiert werden, dass kürzere Formeln kürzere Codes erhalten. Dies sollte für vernünftige Kodierungen aber gelten.)

   Testfrage. Haben Sie eine Idee, warum?

Selbstreduzierbarkeit von SAT

1. **Beispiel.** SAT ist selbstreduzierbar.
2. **Beweis.** Folgender Algorithmus bezeugt die Aussage:

$F \leftarrow$ Eingabe;
Wenn F *enthält keine Variablen* **dann**
  Vereinfache F;
  Wenn $F = 1$ **dann** akzeptiere **sonst** verwirf;
sonst
  Sei X die Variable in F mit kleinstem Index;
  Wenn $\text{Orakel}(F[X/0]) = 1 \vee \text{Orakel}(F[X/1]) = 1$ **dann**
    Akzeptiere;
  sonst
    Verwirf;
  Ende
Ende

□

Erfüllende Belegungen bestimmen

1. **Testfrage.** Man kann die Selbstreduzierbarkeit auch nutzen, um zu einer Formel, von der wir nur wissen, dass sie erfüllbar ist, eine erfüllende Belegung zu finden. Haben Sie eine Idee wie?

Erfüllende Belegungen bestimmen

1. **Testfrage.** Man kann die Selbstreduzierbarkeit auch nutzen, um zu einer Formel, von der wir nur wissen, dass sie erfüllbar ist, eine erfüllende Belegung zu finden. Haben Sie eine Idee wie?
2. **Beweis.** Folgender Algorithmus leistet das Gewünschte:

$F \leftarrow$ Eingabe;
Seien $X_{i_1}, \ldots, X_{i_\ell}$ die Variablen in F;
Für $j = 1, 2, 3, \ldots, \ell$ **tu**
  Wenn Orakel$(F[X_{i_j}/1]) = 1$ **dann** $\sigma_j \leftarrow 1$ **sonst** $\sigma_j \leftarrow 0$;
  $F \leftarrow F[X_{i_j}/\sigma_j]$;
Ende
Gib $(\sigma_1, \ldots, \sigma_\ell)$ aus; □

Erfüllende Belegungen bestimmen

1. **Testfrage.** Man kann die Selbstreduzierbarkeit auch nutzen, um zu einer Formel, von der wir nur wissen, dass sie erfüllbar ist, eine erfüllende Belegung zu finden. Haben Sie eine Idee wie?
2. **Beweis.** Folgender Algorithmus leistet das Gewünschte:

   $F \leftarrow$ Eingabe;
   Seien $X_{i_1}, \ldots, X_{i_\ell}$ die Variablen in F;
   Für $j = 1, 2, 3, \ldots, \ell$ **tu**
   | **Wenn** Orakel$(F[X_{i_j}/1]) = 1$ **dann** $\sigma_j \leftarrow 1$ **sonst** $\sigma_j \leftarrow 0$;
   | $F \leftarrow F[X_{i_j}/\sigma_j]$;
   Ende
   Gib $(\sigma_1, \ldots, \sigma_\ell)$ aus; □

3. **Idee.** In abgewandelter Form werden wir diesen Algorithmus im Beweis des Satzes von Karp & Lipton nutzen.

Der Satz von Karp & Lipton

1. **Satz (Wiederholung).** Falls $\mathbf{NP} \subseteq \mathbf{P/poly}$, dann $\mathbf{PH} = \Sigma_2 = \Pi_2$.
2. **Beweis.**
   - Falls $\mathbf{NP} \subseteq \mathbf{P/poly}$, dann hätte SAT polynomielle Schaltkreise.
   - Laut Abschnitt 5 genügt es, zu zeigen, dass dann $\Pi_2 \subseteq \Sigma_2$.

Der Satz von Karp & Lipton

1. Sei also $L \in \mathbf{\Pi}_2$ beliebig.
2. Dann gibt es laut Abschnitt 5 ein $B \in \mathbf{P}$ und $p_1, p_2 \in \text{poly}$ mit
$$x \in L \iff \forall^{p_1(|x|)} y \, \exists^{p_2(|x|)} z \;\; \langle x, y, z \rangle \in B.$$
3. Für gegebene x und y betrachte davon die Teilformel
$$\exists^{p_2(|x|)} z \;\; \langle x, y, z \rangle \in B.$$
Diese Eigenschaft zu entscheiden ist ein Problem in **NP**.
4. Beim Satz von Cook wird zu jeder **NP**-Maschine M eine von der Eingabe w abhängige Formel konstruiert, die erfüllbar ist genau dann, wenn M für w einen akzeptierenden Rechenpfad hat.
5. Auf unsere Situation angewendet liefert diese Konstruktion eine in polynomieller Zeit berechenbare Funktion f, so dass
$$x \in L \iff \forall^{p_1(|x|)} y \;\; f(x, y) \in \text{SAT}$$

Der Satz von Karp & Lipton

1. Nimm an, wir hätten eine Schaltkreisfamilie $(C_n)_{n\in\mathbb{N}}$ nicht-deterministisch geraten, und wollen, dass sie SAT entscheide.
2. Ein C_n könnte bei Eingabe F zwei Arten von Fehlern machen:
   - **falsch negativ:** F wird verworfen, obwohl $F \in \mathrm{SAT}$.
   - **falsch positiv:** F wird akzeptiert, obwohl $F \notin \mathrm{SAT}$.

Der Satz von Karp & Lipton

1. Nimm an, wir hätten eine Schaltkreisfamilie $(C_n)_{n\in\mathbb{N}}$ nicht-deterministisch geraten, und wollen, dass sie SAT entscheide.
2. Ein C_n könnte bei Eingabe F zwei Arten von Fehlern machen:
   - **falsch negativ:** F wird verworfen, obwohl $F \in \mathrm{SAT}$.
   - **falsch positiv:** F wird akzeptiert, obwohl $F \notin \mathrm{SAT}$.
3. **Idee.**
   - Falls $(C_n)_{n\in\mathbb{N}}$ zufälligerweise korrekt ist, also wirklich SAT entscheidet, so können wir den oben beschriebenen Algorithmus nutzen, um erfüllende Belegungen σ für ein F zu bestimmen.
   - Dabei werden Orakelaufrufe durch C_m's passender Größe ersetzt.
   - Durch Einsetzen können wir prüfen, ob ein σ ein F tatsächlich erfüllt. Falls ja, können wir *sicher* sein, dass F erfüllbar ist.
   - Ist F hingegen *nicht* erfüllbar, so kann auch eine beliebig falsche Schaltkreisfamilie keine erfüllende Belegung liefern.
   - So können Fehler vom Typ „falsch positiv“ *verhindert* werden.

Der Satz von Karp & Lipton

1. Kehren wir zurück zur oben ermittelten Bedingung
$$x \in L \iff \forall^{p_1(|x|)} y \; f(x,y) \in \mathrm{SAT}.$$
2. **Behauptung.** Das ist für ein $A \in \mathbf{P}$ und $p_3 \in \mathrm{poly}$ äquivalent zu
$$x \in L \iff \exists^{p_3(|x|)} c \; \forall^{p_1(|x|)} y \; \langle x, c, y \rangle \in A.$$
3. Das wäre die gesuchte Σ_2-Charakterisierung von L.
4. **Es bleibt zu zeigen...**
   - ... dass solche $A \in \mathbf{P}$ und $p_3 \in \mathrm{poly}$ existieren, und
   - ... dass sie tatsächlich L wie behauptet charakterisieren.

Der Satz von Karp & Lipton

1. **Definition.** $A := \{\langle x, c, y\rangle : M \text{ akzeptiert } \langle x, c, y\rangle\}$,

   wobei M bei Eingabe $\langle x, c, y\rangle$ wie folgt arbeiten möge:

   - Berechne in Polynomialzeit die Formel $F := f(x, y)$.
   - Beginnend vom Anfang, interpretiere c's Bits als Codes der Schaltkreise $C_1, \ldots, C_{|F|}$. Entnimm $C_{|F|}$ aus c und simuliere ihn mit Eingabe F in Polynomialzeit. Falls $C_{|F|}(F) = 0$, verwirf.
   - Sonst (falls $C_{|F|}$ glaubt, $F \in \mathrm{SAT}$), versuche eine erfüllende Belegung σ für F zu finden. Nutze dazu den obigen Algorithmus, ersetze aber jeden Orakelaufruf durch einen Schaltkreis C' passender Größe.
   - Entnimm dazu aus c jeweils eine Beschreibung von C' an den entsprechenden Positionen; um die Orakelantworten zu erhalten, simuliere dann C' in Polynomialzeit.
   - Sobald σ ganz bestimmt wurde, prüfe in Polynomialzeit, ob F von σ tatsächlich erfüllt wird. Falls ja, akzeptiere; sonst verwerfe.

   Nach Konstruktion gilt offensichtlich $A \in \mathbf{P}$. ✓

Der Satz von Karp & Lipton

1. Nach Annahme gibt es eine polynomiell große Schaltkreisfamilie $(C_n)_{n\in\mathbb{N}}$, die SAT korrekt entscheidet.
2. Sei p_3 ein ausreichend großes Polynom, so dass für alle x,
$$p_3(|x|) \geq d \cdot |F| \cdot \max\{\overline{C_1}, \ldots, \overline{C_{|F|}}\},$$
für eine geeignete Konstante d. Ein solches p_3 existiert, da
   - $|y|$ polynomiell beschränkt ist in $|x|$,
   - $|F|$ polynomiell beschränkt ist in $|x|$ und $|y|$,
   - jedes $\overline{C_i}$ mit $1 \leq i \leq |F|$ polynomiell beschränkt ist in $|F|$.
3. Also genügen die $p_3(|x|)$ Bits von c, um ein ausreichend langes Anfangsstück der Folge $(C_n)_{n\in\mathbb{N}}$ zu beschreiben, damit die Rechnung auf der letzten Folie durchgeführt werden kann. ✓

Der Satz von Karp & Lipton

1. **Behauptung.** Es gilt
$$x \in L \iff \exists^{p_3(|x|)} c \, \forall^{p_1(|x|)} y \;\; \langle x, c, y \rangle \in A.$$
2. **Angenommen $x \in L$:**
   - Dann gilt für alle y mit $|y| = p_1(|x|)$, dass $f(x,y) \in \mathrm{SAT}$.
   - Für ein c, das Schaltkreise $(C_n)_{n \in \mathbb{N}}$ wie auf der letzten Folie beschreibt, gilt dann für alle solchen y, dass $\langle x, c, y \rangle \in A$. ✓
3. **Angenommen $x \notin L$:**
   - Dann gibt es ein y mit $|y| = p_1(|x|)$ mit $f(x,y) \notin \mathrm{SAT}$.
   - Da wir Fehler vom Typ „falsch positiv“ ausgeschlossen haben, gilt $\langle x, c, y \rangle \notin A$ unabhängig von c. ✓
4. Die oben genannte Σ_2-Bedingung charakterisiert also L. □

Kapitelzusammenfassung

1. Wir haben Schaltkreiskomplexitität als ein nichtuniformes Komplexitätsmaß definiert.
2. Adviceklassen ähneln gewöhnlichen Komplexitätsklassen, aber ergänzt um nichtuniforme Zusatzinformationen.
3. Wir haben gezeigt, dass polynomiell große Schaltkreise die gleichen Sprachen erkennen wie Polynomialzeit-Turing-maschinen, die zusätzlich polynomiell großen Advice erhalten.
4. Wir haben gezeigt, dass SAT selbstreduzierbar ist, und dies genutzt, um den Satz von Karp & Lipton zu beweisen.
5. Dieser besagt, dass die Annahme $\mathbf{NP} \subseteq \mathbf{P/poly}$ impliziert, dass die Polynomialzeithierarchie auf der zweiten Stufe kollabiert.

14 Eine letzte Sache…

Frohe Weihnachten!

(**NP**-Witz ausgedacht von GPT-4,
Cartoon erzeugt von DALL·E 3)