Komplexitätstheorie

Markus Lohrey

Universität Siegen

Wintersemester 2013/2014

Teil 1: Grundlagen

Im folgenden behandeln wir einige Grundlagen:

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

Das meiste davon können Sie später wieder vergessen, denn:

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

Turingmaschinen: Definition

Notation: Mit $\mathcal{P}(A)$ bezeichnen wir die Potenzmenge (Menge aller Teilmengen) der Menge A.

Definition 1

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

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

Turingmaschinen: Definition

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

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

Außerdem gilt für alle $(q, a, b_1, \ldots, b_k) \in Q \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma^k$:

$$\delta(q, a, b_1, \ldots, b_k) = \emptyset \quad \Leftrightarrow \quad (q = q_J \text{ oder } q = q_N)$$

Bei einer deterministischen k-Band Turingmaschine M gilt zusätzlich $|\delta(\tilde{s})| \leq 1$ für alle $\tilde{s} \in Q \times (\Sigma \cup \{\triangleright, \triangleleft\}) \times \Gamma^k$.

Eine Turingmaschine mit Ausgabe ist wie eine deterministische Turingmaschine definiert, außer dass noch ein Ausgabealphabet Σ' existiert, und für δ gilt:

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

(λ ist das leere Wort).

Turingmaschinen: Konfigurationen

Definition 2

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

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

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

Turingmaschinen: Startkonfiguration, Übergänge, ...

1. Für die Eingabe $w \in \Sigma^*$ ist

$$\mathrm{Start}(w) = (q_0, 1, \square, 1, \ldots, \square, 1)$$

die zu w gehördende Startkonfiguration.
Beachte: $|\mathrm{Start}(w)| = 1$.

2. Für ein $\tilde{u} \in Q \times \Gamma^k \times \{-1, 1\}^{k+1}$ und Konfigurationen

$$\alpha = (q, i, u_1, i_1, \ldots, u_k, i_k) \text{ und } \beta$$

schreiben wir $\alpha \vdash_{\tilde{u}} \beta$, falls

$$\tilde{u} \in \delta(q, (\triangleright w \triangleleft)[i], u_1[i_1], \ldots, u_k[i_k])$$

und die Anwendung der "Anweisung" $\tilde{u}$ auf die Konfiguration α die Konfiguration β ergibt.

Übung: Definieren Sie dies formal.

3. Es gelte $\alpha \vdash_M \beta$ falls ein $\tilde{u} \in Q \times \Gamma^k \times \{-1, 1\}^{k+1}$ mit $\alpha \vdash_{\tilde{u}} \beta$ existiert.

Turingmaschinen: Rechnungen, Protokolle

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

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

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

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

Turingmaschinen: akzeptierte Menge, Zeit- und Platzbedarf

1. Der Zeitbedarf (bzw. Platzbedarf) der Rechnung $\alpha_0, \alpha_1, \ldots, \alpha_m$ ist m (bzw. $\max\{|\alpha_i| \mid 0 \leq i \leq m\}$).
2. M hat bei Eingabe w den Zeitbedarf (bzw. Platzbedarf) höchstens $N \in \mathbb{N}$, falls jede Rechnung von M bei Eingabe w Zeitbedarf (bzw. Platzbedarf) $\leq N$ hat.
3. Sei $f : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion.

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

   M is f-platzbeschränkt, falls M für jede Eingabe w Platzbedarf höchstens $f(|w|)$ hat.
4. $L(M) = \{w \in \Sigma^* \mid \exists \text{ akzeptierende Rechnung von } M \text{ bei Eingabe } w\}$ ist die von M akzeptierte Menge.

Turingmaschinen: Anzahl der Konfigurationen

Das folgende einfache Lemma werden wir häufig verwenden:

Lemma 3

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

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

Komplexitätsklassen

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

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

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

Komplexitätsklassen

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

Dies erlaubt, die gesamte Eingabe zu lesen.

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

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

Wichtige Komplexitätsklassen

Gebräuchliche Abkürzungen:

$$\mathbf{L} = \mathrm{DSPACE}(\log(n)) \tag{1}$$

$$\mathbf{NL} = \mathrm{NSPACE}(\log(n)) \tag{2}$$

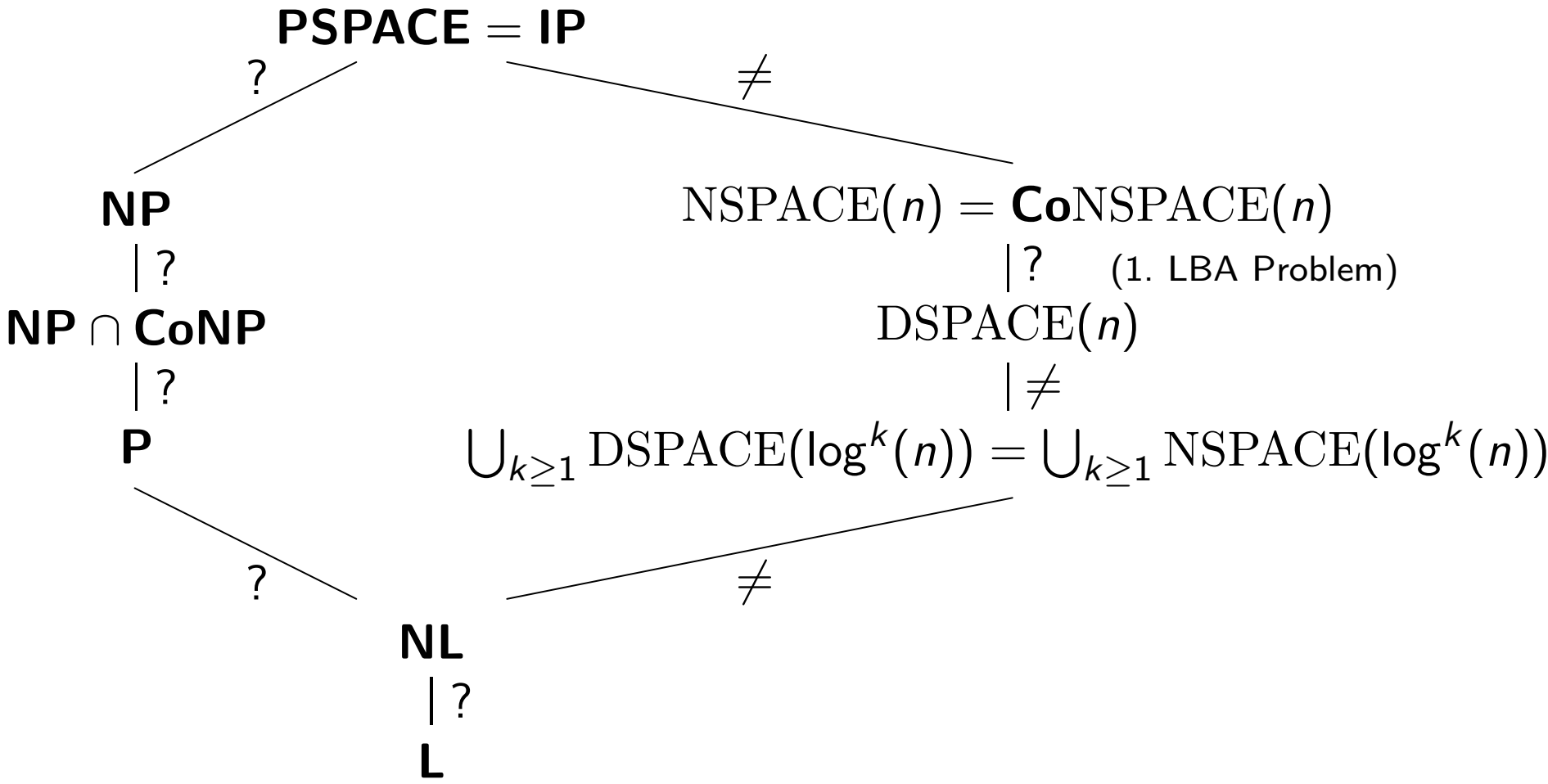
$$\mathbf{P} = \bigcup_{k\geq 1} \mathrm{DTIME}(n^k) \tag{3}$$

$$\mathbf{NP} = \bigcup_{k\geq 1} \mathrm{NTIME}(n^k) \tag{4}$$

$$\mathbf{PSPACE} = \bigcup_{k\geq 1} \mathrm{DSPACE}(n^k) = \bigcup_{k\geq 1} \mathrm{NSPACE}(n^k) \tag{5}$$

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

Beziehungen zwischen Komplexitätsklassen



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

Beispiele

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

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

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

Varianten algorithmischer Probleme

Beispiel 1: Traveling Salesman Problem (TSP)

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

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

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

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

Varianten algorithmischer Probleme

(A) Entscheidungsvariante:

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

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

(B) Berechnungsvariante:

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

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

(C) Optimierungsproblem:

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

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

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

Varianten algorithmischer Probleme

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

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

Beweis:

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

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

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

Im folgenden nehmen wir an, dass ein Rundweg existiert.

Varianten algorithmischer Probleme

2. Schritt:

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

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

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

Varianten algorithmischer Probleme

3. Schritt:

Berechne optimalen Rundweg wie folgt:

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

Varianten algorithmischer Probleme

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

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

Beweis:

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

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

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

Varianten algorithmischer Probleme

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

(A) Entscheidungsvariante:

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

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

(B) Berechnungsvariante:

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

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

(C) Optimierungsproblem:

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

Ziel: Berechne eine möglichst kleine Knotenüberdeckung von G.

Varianten algorithmischer Probleme

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

Zeigen Sie dies als Übung.

Das Grapherreichbarkeitsproblem

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

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

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

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

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

Das Grapherreichbarkeitsproblem

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

Warum benötigt obiger Algorithmus nur logarithmischen Platz?

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

Bemerkungen:

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

Teil 2: Beziehungen zwischen den Komplexitätsklassen

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

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

Satz 4

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

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

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

Der Satz von Hennie und Stearns (1966)

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

Satz 5

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

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

Satz 6

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

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

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

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

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

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

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

Beachte:

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

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

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

$\mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)})$

Satz 7

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

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

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

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

Wir berechnen jetzt die Menge R aller von Start(w) erreichbaren Konfigurationen:

$\mathrm{DSPACE}(f) \subseteq \mathrm{NSPACE}(f) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(f)})$

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

Zeitbedarf:

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

Folgerungen

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

Der Satz von Savitch (1970)

Satz 8

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

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

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

Beweis des Satz von Savitch

Beweis des Satzes von Savitch:

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

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

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

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

O.B.d.A. enthalte Accept_M nur eine einzige Konfiguration α_f, die von $\text{Start}(w)$ aus erreichbar sein kann.

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

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

Beweis des Satz von Savitch

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

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

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

Für $i > 0$ verwenden wir folgendes Rekursionschemata:

$$\mathrm{Reach}(\alpha, \beta, i) \iff \exists \gamma \in \mathsf{Conf}(M, w) : \mathrm{Reach}(\alpha, \gamma, i-1) \wedge \mathrm{Reach}(\gamma, \beta, i-1).$$

Umsetzung durch einen deterministischen Algorithmus:

Beweis des Satz von Savitch

FUNCTION Reach(α, β, i) (wobei $\alpha, \beta \in \text{Conf}(M, w)$ und $i \leq c \cdot s(|w|)$

var $b := \text{FALSE}$
if $i = 0$ **then**
$b := [(\alpha = \beta) \vee (\alpha \vdash_M \beta)]$
else
forall $\gamma \in \text{Conf}(M, w)$ **do**
if not b **and** Reach$(\alpha, \gamma, i-1)$ **then**
$b :=$ Reach$(\gamma, \beta, i-1)$
endif
endfor
endif
return b
ENDFUNC

Beweis des Satz von Savitch

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

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

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

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

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

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

Beweis des Satz von Savitch

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

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

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

Bemerkungen zum Satz von Savitch

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

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

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

Folgerungen aus dem Satz von Savitch

Satz 9

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

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

Satz 10

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

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

Hierarchiesätze

Satz 11 (Platzhierarchiesatz)

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

Bemerkungen:

- $s_1 \notin \Omega(s_2)$ bedeutet $\forall \epsilon > 0$ $\exists$ unendlich viele n mit $s_1(n) < \epsilon \cdot s_2(n)$.
  Seien etwa $s_1(n) = n$ und $s_2(n) = \begin{cases} n^2, & \text{falls } n \text{ gerade} \\ \log n, & \text{sonst} \end{cases}$,
  dann gilt: $s_2 \notin \Omega(s_1)$, $s_1 \notin \Omega(s_2)$.
- Aus dem Platzhierarchiesatz folgt etwa:

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

Beweis des Platzhierarchiesatzes

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

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

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

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

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

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

Beweis des Platzhierarchiesatzes

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

Wir beginnen mit einer (deterministischen) universellen Turingmaschine U.

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

U simuliert M_x auf der Eingabe w.

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

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

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

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

Beweis des Platzhierarchiesatzes

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

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

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

   Dies bedeutet, dass M_x auf y nicht terminiert.

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

Beweis des Platzhierarchiesatzes

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

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

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

Sei $M' = M_x$.

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

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

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

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

und somit

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

Beweis des Platzhierarchiesatzes

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

Also:

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

Zeithierarchiesatz

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

Analog zum Platzhierarchiesatz ergibt sich der deterministische Zeithierarchiesatz.

Satz 12 (Deterministischer Zeithierarchiesatz (ohne Beweis))

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

Als Folgerung hieraus ergibt sich:

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

Der Satz von Borodin

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

Satz 13 (Satz von Borodin (1972))

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

Bemerkungen:

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

Beweis des Satzes von Borodin

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

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

Betrachte nun die Menge

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

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

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

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

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

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

Beweis des Satzes von Borodin

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

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

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

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

Beweis des Satzes von Borodin

Beweis der Behauptung:

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

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

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

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

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

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

Der Satz von Immerman und Szelepcsényi (1987)

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

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

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

Satz 14 (Satz von Immerman und Szelepcsényi)

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

Beweis des Satzes von Immerman und Szelepcsényi

Beweismethode: Induktives Zählen

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

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

O.B.d.A. sei α_0 die einzige akzeptierende Konfiguration; etwa $\alpha_0 = (q_J, 1, \square, 1)$ (insbesondere $|\alpha_0| = 1$).

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

- α_0 ist die kleinste Konfiguration bezüglich $\prec$.
- $\alpha \prec \alpha'$ impliziert $|\alpha| \leq |\alpha'|$.
- $\alpha \prec \alpha'$ kann in Platz $|\alpha| + |\alpha'|$ überprüft werden.

Beweis des Satzes von Immerman und Szelepcsényi

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

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

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

Beweis des Satzes von Immerman und Szelepcsényi

Sei $k \geq 0$:

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

Beachte: Nach Lemma 3 gilt

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

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

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

Wir berechnen $r(*)$ in Platz $\mathcal{O}(f(n))$. Unser Algorithmus wird ohne Ergebnis abbrechen, falls $w \in L(M)$. Falls $w \notin L(M)$, wird der korrekte Wert $r(*)$ ausgegeben.

Beweis des Satzes von Immerman und Szelepcsényi

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

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

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

Beweis des Satzes von Immerman und Szelepcsényi

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

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

FUNCTION berechne-m(k, r)
  $\alpha := \alpha_0$; $m := 0 (= |\alpha_0|)$
  repeat r **times**
    berechne ein beliebiges $\alpha' \in R(k)$
    if $\alpha \prec \alpha'$ **then**
      $\alpha := \alpha'$
      $m := |\alpha'|$ **(*** $= \max\{m, |\alpha'|\}$ aufgr. der Ordnung ***)**
    else
      „FEHLER“ $\Rightarrow$ Programmabbruch
    endif
  endrepeat
  return m
ENDFUNC

Beweis des Satzes von Immerman und Szelepcsényi

Beachte:

- Wenn berechne-m(k, r) nicht mit „Fehler“ abbricht (und $r = r(k)$ gilt), dann wird der korrekte Wert $m(k)$ berechnet.
- Wenn $\alpha_0 \in R(k)$ (und damit $w \in L(M)$) dann muss berechne-m(k, r) mit „Fehler“ abbrechen, da in $R(k)$ nicht r viele Konfigurationen zur Verfügung stehen, die alle echt größer als α_0 sind.

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

Beweis des Satzes von Immerman und Szelepcsényi

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

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

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

Beweis des Satzes von Immerman und Szelepcsényi

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

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

Beweis des Satzes von Immerman und Szelepcsényi

Beachte:

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

Platzbedarf: Wir müssen speichern:

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

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

Beweis des Satzes von Immerman und Szelepcsényi

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

```
FUNCTION berechne-r(k+1, r)
   r' := 0    (* ist am Ende r(k+1) *)
   m := berechne-m(k, r)
   forall Konfigurationen β mit |β| ≤ m+1 do
      if Reach(r, k+1, β) then
         r' := r' + 1
      endif
   endforall
   return r'
ENDFUNC
```

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

Beweis des Satzes von Immerman und Szelepcsényi

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

Man beachte dazu: Wenn $w \in L(M)$, so bricht *jede* Rechnung mit einer Fehlermeldung ab, da die Funktion $m(k)$, sobald sie die akzeptierende Konfiguration α_0 in $R(k)$ vorfindet, nicht mehr erfolgreich durchlaufen werden kann.

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

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

Die Translationstechnik

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

Idee: Ausstopfen (Padding) von Sprachen.

Sei

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

Definiere die Sprache

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

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

Der Translationssatz für Zeitklassen

Satz 15 (Translationssatz für Zeitklassen)

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

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

Beweis: Der Beweis wird für DTIME geführt; der Beweis für NTIME verläuft analog.

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

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

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

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

Beweis des Translationssatzes

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

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

1. Teste in Zeit $m \leq g(m)$ ob $x \in w\$^*$ für ein $w \in \Sigma^*$ gilt.

   Sei $x = w\$^{m-n}$ mit $w \in \Sigma^*$, $|w| = n$.
2. Berechne $1^{g(m)}$ in Zeit $g(m)$ (g ist zeitkonstruierbar).
3. Teste nun wie folgt in Zeit $g(m)$, ob $f(n) = m$ gilt:

   Berechne $1^{f(n)}$ in Zeit $g(f(n))$. Falls die Maschine dabei mehr als $g(m)$ Schritte rechnen will, lehnen wir dabei ab (da g monoton ist, gilt $g(f(n)) > g(m) \rightarrow f(n) > m$).

   Falls $1^{f(n)}$ berechnet wird, können wir $1^{f(n)}$ mit 1^m vergleichen.

   Sei nun $x = w\$^{f(n)-n}$.
4. Teste in Zeit $\mathcal{O}(g(f(n))) = \mathcal{O}(g(m))$, ob $w \in L$ gilt. □

Der Translationssatz für Platzklassen

Satz 16 (Translationssatz für Platzklassen (ohne Beweis))

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

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

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

Folgerungen aus dem Translationssatz für Platzklassen

Satz 17 (Korollar aus Translationssatz für Platzklassen)

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

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

Sei $L \in \mathrm{NSPACE}(n)$.

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

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

Folgerungen aus den Translationssätzen

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

Satz 18 (Korollar aus den Translationssätzen)

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

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

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

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

Dies ist ein Widerspruch.

Folgerungen aus den Translationssätzen

Bemerkung:

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

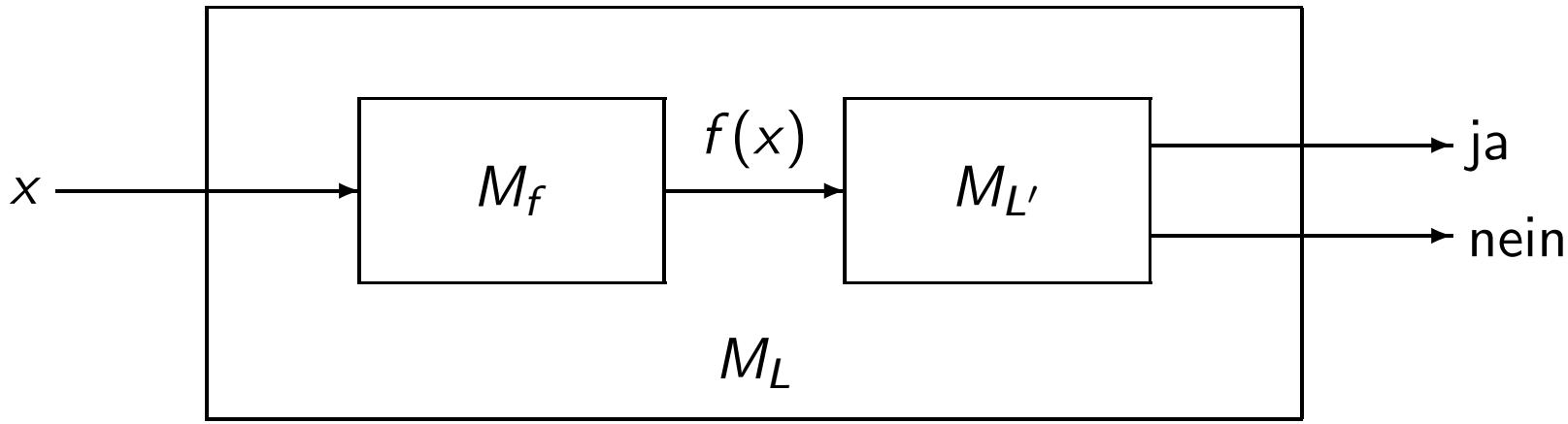
Teil 3: Reduktionen und vollständige Probleme

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

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

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

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



Polynomialzeitreduktionen

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

Proposition 19

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

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

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

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

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

Reduktionen in logarithmischen Platz

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

Definition Logspace-Transducer

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

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

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

Reduktionen in logarithmischen Platz

Definition

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

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

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

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

Reduktionen in logarithmischen Platz

Bemerkungen:

- Logspace-Reduktionen lassen sich auch auf Klassen unterhalb von **P** anwenden und erlaubt eine feinere Einteilung als unter Verwendung von Polynomialzeitreduktionen.
- Jede logspace berechenbare Abbildung $f : \Sigma^* \to \Sigma'^*$ ist in polynomialer Zeit berechenbar.
  Insbesondere: $\exists k \geq 0 \; \forall x \in \Sigma^* : |f(x)| \leq |x|^k$.
- Logspace-Reduktionen und Polynomialzeitreduktionen sind genau dann gleichmächtig, wenn $\mathbf{L} = \mathbf{P}$ gilt.

$\leq_m^{\log}$ ist transitiv

Proposition 20

($\leq_m^{\log}$ ist transitiv) $L \leq_m^{\log} L' \leq_m^{\log} L'' \implies L \leq_m^{\log} L''$

Beachte: Die analoge Aussage für Polynomialzeitreduktionen ist trivial.

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

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

$\leq_m^{\log}$ ist transitiv

Beweis von Proposition 20:

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

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

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

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

Beispiel: Sei $f(n) = n^k$. Dann ist f logspace berechenbar.

Also gilt mit Hilfe der Reduktion $w \mapsto w\$^{|w|^k-|w|}$, dass $L \leq_m^{\log} \mathrm{Pad}_f(L)$.

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

Vollständige Probleme

Definition

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

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

GAP ist **NL**-vollständig

Wir geben ein erstes Beispiel:

Satz 21

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

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

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

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

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

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

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

GAP ist **NL**-vollständig

Offensichtlich gilt:

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

f kann schließlich in logarithmischen Platz berechnet werden. □

Satz 22

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

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

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

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

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

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

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

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

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

Das generische **NP**-vollständige Problem

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

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

Satz 23

L_{Gen} ist **NP**-vollständig.

Beweis:

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

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

Dies ist ein nichtdeterministischer Polynomialzeitalgorithmus für L_{Gen}.

Das generische **NP**-vollständige Problem

L_{Gen} ist **NP**-schwierig:

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

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

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

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

Der Satz von Cook

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

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

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

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

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

Satz 24 (Satz von Cook)

SAT ist **NP**-vollständig.

Beweis des Satzes von Cook

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

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

(B) SAT ist **NP**-schwierig.

Sei $L \in$ **NP**.

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

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

Die Abbildung f wird logspace berechenbar sein.

Beweis des Satzes von Cook

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

Sei $w = w_1 w_2 \cdots w_n \in \Sigma^*$ eine Eingabe der Länge n.

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

- M hat nur ein Band, auf dem die Eingabe zu Beginn steht, und auf das links von der Eingabe geschrieben werden darf.
- Die Endmarker $\triangleright$ und $\triangleleft$ brauchen wir nicht.
- M akzeptiert w genau dann, wenn sich M nach genau $p(n)$ vielen Schritten im Zustand q_J befindet, und der Schreib-Lesekopf wieder in der Ausgangsposition ist.
- Aus $(p_1, a_1, d_1), (p_2, a_2, d_2) \in \delta(q, a)$ folgt $a_1 = a_2$ und $d_1 = d_2$.

Beweis des Satzes von Cook

Jede von der Startkonfiguration erreichbare Konfiguration kann durch ein Wort aus

$$\text{Conf} = \{\Box uqv\Box\Box \mid q \in Q,\ uv \in \Gamma^{p(n)}\}$$

beschrieben werden.

Die Startkonfiguration ist $\Box q_0 w \Box^{p(n)-n+2}$.

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

$$\alpha = \alpha[-1]\alpha[0]\cdots\alpha[p(n)]\alpha[p(n)+1]\alpha[p(n)+2]$$

wobei $\alpha[-1] = \alpha[p(n)+1] = \alpha[p(n)+2] = \Box$, $\alpha[0], \ldots, \alpha[p(n)] \in Q \cup \Gamma$.

Beweis des Satzes von Cook

Definiere die Menge der 5-Tupel $\Delta \subseteq (Q \cup \Gamma)^5$ als alle 5-Tupel, die von einem der folgenden 3 Typen sind:

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

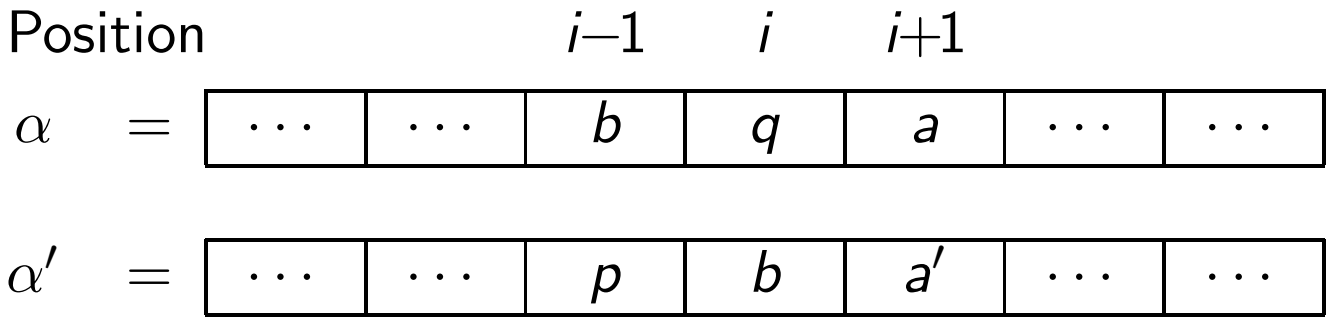
Dann gilt für alle $\alpha, \alpha' \in \square(Q \cup \Gamma)^*\square\square$ mit $|\alpha| = |\alpha'|$:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \text{Conf und}$$
$$\forall i \in \{0, \ldots, p(n)\} : (\alpha[i-1], \alpha[i], \alpha[i+1], \alpha[i+2], \alpha'[i]) \in \Delta.$$

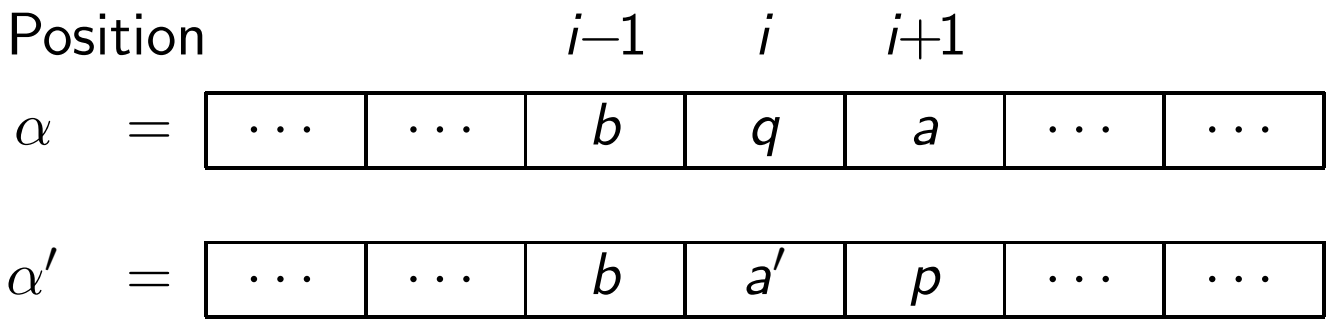
Beweis des Satzes von Cook

Beispiel:

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



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



Beweis des Satzes von Cook

Eine Rechnung von M können wir nun als Matrix beschreiben:

$$\begin{array}{lcccccccc}
\alpha_0 & = & \square & \alpha_{0,0} & \alpha_{0,1} & \cdots & \alpha_{0,p(n)} & \square & \square \\
\alpha_1 & = & \square & \alpha_{1,0} & \alpha_{1,1} & \cdots & \alpha_{1,p(n)} & \square & \square \\
 & & & & \vdots & & & & \\
\alpha_{p(n)} & = & \square & \alpha_{p(n),0} & \alpha_{p(n),1} & \cdots & \alpha_{p(n),p(n)} & \square & \square
\end{array}$$

Für jedes Tripel (a, i, t) ($a \in Q \cup \Gamma$, $-1 \leq i \leq p(n) + 2$, $0 \leq t \leq p(n)$) sei $x(a, i, t)$ eine aussagenlogische Variable.

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

Beweis des Satzes von Cook

An den Positionen -1, $p(n)+1$ und $p(n)+2$ steht immer $\square$:

$$G(n) \quad = \quad \bigwedge_{0 \leq t \leq p(n)} \Big(x(\square, -1, t) \wedge x(\square, p(n)+1, t) \wedge x(\square, p(n)+2, t) \Big)$$

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

$$X(n) \quad = \quad \bigwedge_{\substack{0 \leq t \leq p(n) \\ -1 \leq i \leq p(n)+2}} \left(\bigvee_{a \in Q \cup \Gamma} \left(x(a, i, t) \wedge \bigwedge_{b \neq a} \neg x(b, i, t) \right) \right)$$

Beweis des Satzes von Cook

Zum Zeitpunkt $t = 0$ ist die Konfiguration gleich $\square q_0 w \square^{p(n)-n+2}$:

$$S(w) = \left(x(q_0, 0, 0) \wedge \bigwedge_{i=1}^{n} x(w_i, i, 0) \wedge \bigwedge_{i=n+1}^{p(n)} x(\square, i, 0) \right)$$

Die Berechnung respektiert die lokale Relation Δ:

$$D(n) = \bigwedge_{\substack{0 \leq i \leq p(n) \\ 0 \leq t \leq p(n)-1}} \bigvee_{(a,b,c,d,e) \in \Delta} \left(\begin{array}{lll} x(a, i-1, t) & \wedge & x(b, i, t) \wedge \\ x(c, i+1, t) & \wedge & x(d, i+2, t) \wedge \\ x(e, i, t+1) & & \end{array} \right)$$

Beweis des Satzes von Cook

Sei schließlich

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

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

Es gilt also:

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

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

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

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

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

Definition: Literale, KNF

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

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

Sei KNF (bzw. DNF) die Menge der aussagenlogischen Ausdrücke in konjunktiver Normalform (bzw. disjunktiver Normalform):

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

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

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

Beispiel:

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

Beachte:

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

SAT ∩ KNF ist **NP**-vollständig

Satz 25

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

Beweis:

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

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

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

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

SAT $\cap$ KNF ist **NP**-vollständig

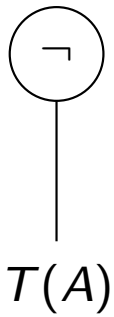
2. Beweis: Wir zeigen $\mathrm{SAT} \leq_m^{\log} \mathrm{SAT} \cap \mathrm{KNF}$.

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

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

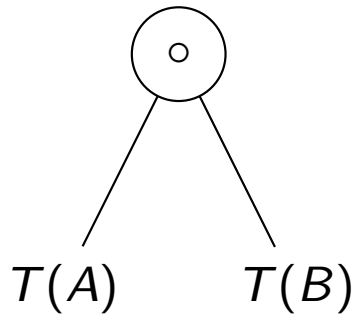
Wir können eine Formel $F \in \mathbb{A}$ als einen Baum $T(F)$ auffassen, der sich rekursiv folgendermaßen aufbauen lässt:

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



SAT $\cap$ KNF ist **NP**-vollständig

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

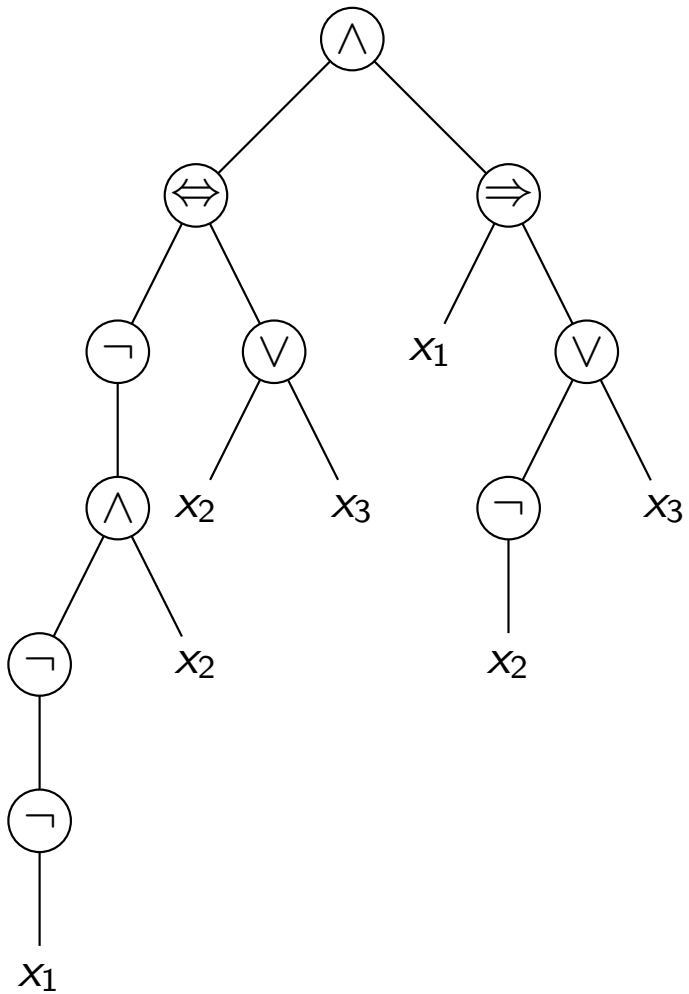


Für die Formel

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

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

SAT ∩ KNF ist **NP**-vollständig



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

SAT $\cap$ KNF ist **NP**-vollständig

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

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

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

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

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

Hier erhalten wir nach entsprechender Umformung:

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

SAT $\cap$ KNF ist **NP**-vollständig

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

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

Lemma

1. $f'(F)$ ist stets erfüllbar.
2. Sei σ eine erfüllende Belegung von $f'(F)$ und σ' die Restriktion von σ auf $V(F)$. Dann gilt: $\sigma'(F) = \sigma(v(F))$.
3. Sei $\sigma' : V(F) \to \{0,1\}$ eine beliebige Belegung. Dann existiert eine erfüllende Belegung $\sigma : V(f'(F)) \to \{0,1\}$ von $f'(F)$ mit $\forall x \in V(F) : \sigma'(x) = \sigma(x)$.

Nach (2) muss für die erfüllende Belegung in (3) $\sigma(v(F)) = \sigma'(F)$ gelten. Der Beweis von (2) und (3) mittels struktureller Induktion sei dem Leser zur Übung überlassen. (1) folgt unmittelbar aus (3).

SAT $\cap$ KNF ist **NP**-vollständig

Intuition

Den Baum $T(F)$ kann man sich als Schaltnetz vorstellen, wobei die Blätter die Eingänge bilden.

Die Wurzel $v(F)$ stellt den Ausgang dar und die inneren Knoten entsprechen Logikgattern.

Geben wir nun eine bestimmte Belegung σ in dieses Netz ein, so wird diese ebenenweise von unten nach oben propagiert und wir erhalten am Ausgang $\sigma(F)$.

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

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

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

SAT $\cap$ KNF ist **NP**-vollständig

Beweis der Behauptung:

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

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

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

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

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

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

3-SAT ist **NP**-vollständig

Definition: 3-SAT

Sei 3-KNF die Menge der Formeln in konjunktiver Form mit genau drei Literalen je Klausel:

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

3-SAT sei die Teilmenge der davon erfüllbaren Formeln:

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

Satz 26

3-SAT ist **NP**-vollständig.

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

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

Sei F eine KNF-Formel. Wir unterscheiden drei Fälle:

3-SAT ist **NP**-vollständig

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

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

3-SAT ist **NP**-vollständig

Beachte: c' kann auch geschrieben werden als

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

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

(A) Sei σ eine erfüllende Belegung für c.

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

Erweitere σ zu einer erfüllenden Belegung von c' durch:

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

3-SAT ist **NP**-vollständig

(B) Sei σ' eine erfüllende Belegung für c'.

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

$\Longrightarrow \sigma'(v(\tilde{x}_3)) = 1$ (da $\sigma'(\tilde{x}_1 \vee \tilde{x}_2 \vee v(\tilde{x}_3)) = 1$)

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

$\Longrightarrow \sigma'(\overline{v(\tilde{x}_{k-1})} \vee \tilde{x}_{k-1} \vee \tilde{x}_k)) = 0$ Widerspruch! □

Integer Programming

Es sei

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

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

Satz 27

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

Beweis:

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

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

Integer Programming

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

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

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

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

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

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

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

Also: S lösbar genau dann, wenn F erfüllbar.

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

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

Bemerkungen:

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

Exkurs: Binärcodierung von Primzahlen

Es sei

$$\mathrm{PRIM} = \{w \in 1\{0,1\}^* \mid w \text{ ist Binärcodierung einer Primzahl}\}$$

die Menge der Binärcodierungen von Primzahlen.

Im August 2002 bewiesen Agrawal, Kayal und Saxena $\mathrm{PRIM} \in \mathbf{P}$.

Wir zeigen hier den folgenden schwächeren Satz:

Satz 28

$\mathrm{PRIM} \in \mathbf{NP} \cap \mathbf{CoNP}$.

Beweis:

(1) $\mathrm{PRIM} \in \mathbf{CoNP}$ ist einfach:

Bei Eingabe $w \in 1\{0,1\}^*$ raten wir $u, v \in 1\{0,1\}^+$ mit $|u|, |v| \leq |w|$ und überprüfen, ob $w = u \cdot v$ gilt.

Beachte: Aus $u, v \in 1\{0,1\}^*$ berechnet man das Produkt $u \cdot v$ in Polynomialzeit mit der Schulmultiplikationsmethode.

$\mathrm{PRIM} \in \mathbf{NP}$

(2) $\mathrm{PRIM} \in \mathbf{NP}$: Wir benötigen den kleinen Satz von Fermat:

Kleiner Satz von Fermat (ohne Beweis)

Eine Zahl $p \in \mathbb{N}$ ist eine Primzahl genau dann, wenn eine Zahl $x \in \{0, \ldots, p-1\}$ existiert mit:

$$x^{p-1} \equiv 1 \bmod p \text{ und } \forall i \in \{1, \ldots, p-2\} : x^i \not\equiv 1 \bmod p$$

Eine nichtdeterministischer Algorithmus (ist nicht der endgültige Algorithmus), der $p \in \mathrm{PRIM}$ überprüft:

1. Akzeptiere, falls $p = 2$ oder $p = 3$, sonst weiter mit (2).
2. Rate eine Primfaktorzerlegung für $p - 1 = p_1 \cdot \ldots \cdot p_k$, $k \geq 2$.
3. Überprüfe dabei, ob $p_j \geq 2$ und rekursiv, ob $bin(p_j) \in \mathrm{PRIM}$, für jedes $j \in \{1, \ldots, k\}$.
4. Rate nichtdeterministisch ein $x \in \{0, \ldots, p-1\}$ und überprüfe, ob $x^{p-1} \equiv 1 \bmod p$ gilt.
5. Verifiziere für alle $j \in \{1, \ldots, k\}$ die Ungleichung $x^{\frac{p-1}{p_j}} \not\equiv 1 \bmod p$.

PRIM $\in$ NP

(A) Wenn der Algorithmus die Eingabe p akzeptiert, dann ist $p \in \mathrm{PRIM}$.

Angenommen der Algorithmus akzeptiert die Eingabe $p > 3$.

Sei $x \in \{0, \ldots, p-1\}$ der in (4) geratene Wert. $\quad\Longrightarrow\quad$ $x^{p-1} \equiv 1 \bmod p$.

Angenommen es gilt $x^i \equiv 1 \bmod p$ für ein $i \in \{1, \ldots, p-2\}$.

Sei m der größte gemeinsame Teiler von $p-1$ und i.

Wegen $i \leq p-2$ gilt $m < p-1$, d.h. m ist echter Teiler von $p-1$

Euklid $\quad\Longrightarrow\quad$ $\exists \alpha, \beta \in \mathbb{Z} : m = \alpha \cdot (p-1) + \beta \cdot i$.

$\Longrightarrow\quad$ $x^m = x^{\alpha \cdot (p-1) + \beta \cdot i} = (x^{p-1})^\alpha \cdot (x^i)^\beta \equiv 1 \bmod p$

Da m ein echter Teiler von $p-1$ ist, gibt es ein p_j mit:
m ist Teiler von $\frac{p-1}{p_j}$.

$\Longrightarrow\quad$ $x^{\frac{p-1}{p_j}} \equiv 1 \bmod p$ Widerspruch zu (5)

PRIM $\in$ NP

(B) Wenn $p \in \mathrm{PRIM}$ dann kann der Algorithmus bei Eingabe p akzeptieren.

Dies folgt direkt aus dem kleinen Satz von Fermat.

(C) Der Algorithmus kann so implementiert werden, dass er in Polynomialzeit arbeitet.

Betrachte für eine Primzahl p eine erfolgreiche Rechnung des Algorithmus.

Diese Rechnung kann durch einen Beweisbaum (für p) beschrieben werden, der für jeden rekursiven Aufruf einen Knoten enthält.

An den Blättern des Beweisbaums wird nur Schritt (1) ausgeführt, an den inneren Knoten des Beweisbaums werden die Schritte (1) – (5) ausgeführt.

PRIM $\in$ NP

Behauptung: Wenn p eine Primzahl ist, so gibt es einen Beweisbaum mit höchstens $2 \cdot \lfloor \log_2(p) \rfloor - 1$ vielen Knoten.

Beweis der Behauptung:

$p = 2$ oder $p = 3$: Dann gibt es einen Beweisbaum mit nur einem Knoten und $2 \cdot \lfloor \log_2(2) \rfloor - 1 = 2 \cdot \lfloor \log_2(3) \rfloor - 1 = 1$.

Sei nun $p > 3$ eine Primzahl und $p - 1 = p_1 \cdots p_k$, $k \geq 2$, eine Primfaktorzerlegung.

Der Beweisbaum für p besteht aus den Beweisbäumen für $p_1, \ldots, p_k$ und dem Wurzelknoten.
Induktion ergibt für die Anzahl der Knoten des Beweisbaums für p:

$$\begin{aligned} 1 + \sum_{i=1}^{k} (2 \cdot \lfloor \log_2(p_i) \rfloor - 1) &\leq 2 \cdot \lfloor \log_2(p_1 \cdot \ldots \cdot p_k) \rfloor - (k-1) \\ &\leq 2 \cdot \lfloor \log_2(p) \rfloor - 1, \quad \text{da } k \geq 2. \end{aligned}$$

PRIM $\in$ NP

Unser **NP**-Algorithmus, der $p \in \mathrm{PRIM}$ überprüft, geht nun wie folgt vor:

- Rate einen Beweisbaum mit höchstens $2 \cdot \lfloor \log_2(p) \rfloor - 1$ vielen Knoten.

  An jedem Knoten des Beweisbaums steht eine Zahl $q \leq p$.

  Jedes Blatt ist mit 2 oder 3 beschriftet.

  Ist ein Knoten mit q beschriftet und sind die Kinder des Knotens mit $q_1, \ldots, q_k$ beschriftet, so muss gelten: $k \geq 2$ und $q - 1 = q_1 \cdot q_2 \cdots q_k$ (kann in Polynomialzeit überprüft werden).

- Für jeden mit q beschrifteten inneren Knoten v wird noch eine Zahl $y \in \{0, \ldots, q-1\}$ geraten und überprüft, ob gilt: $y^{q-1} \equiv 1 \bmod q$ und $y^{(q-1)/q_j} \not\equiv 1 \bmod q$ für jedes mit q_j beschriftete Kind von v.

  Dies kann wieder in Polynomialzeit überprüft werden: $y^{q-1} \bmod q$ (sowie $y^{(q-1)/q_j} \bmod q$) kann durch iteriertes Quadrieren berechnet werden: Berechne nacheinander $y^1 \bmod q$, $y^2 \bmod q$, $y^4 \bmod q, \ldots$, $y^{2^i} \bmod q$, wobei 2^i die größte Zweierpotenz $\leq q - 1$ ist. $\square$

Vertex Cover ist **NP**-vollständig

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

Vertex Cover (VC) ist das folgende Problem:

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

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

Satz 29

VC ist **NP**-vollständig.

Beweis:

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

(1) VC ist **NP**-schwierig:

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

Vertex Cover ist **NP**-schwierig

Sei

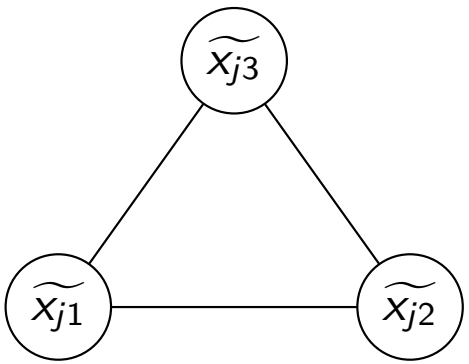
$$F = c_1 \wedge \cdots \wedge c_q$$

eine Formel in 3-KNF, wobei

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

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

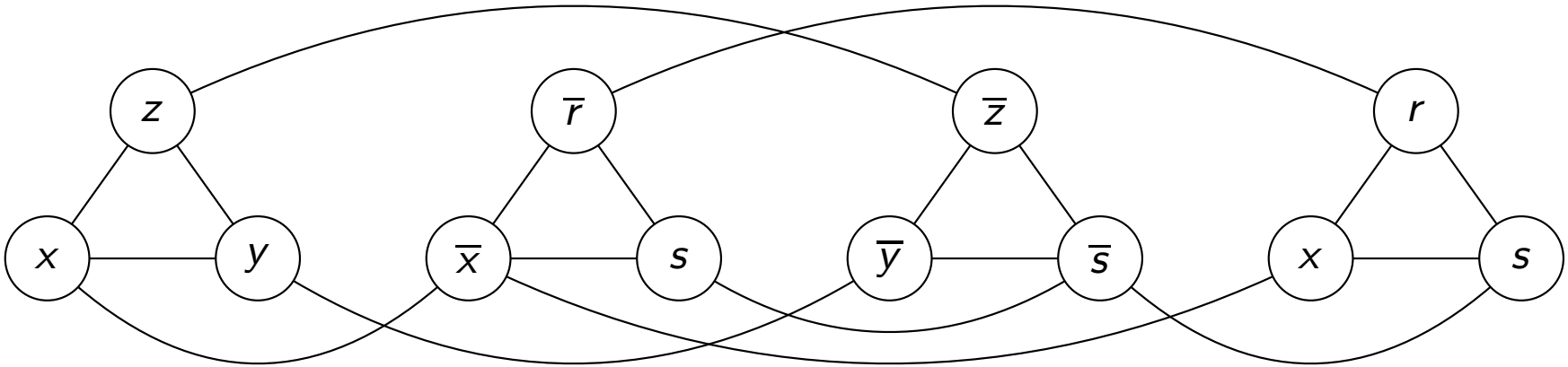
Zunächst bilden wir zu jeder Klausel $c_j = (\widetilde{x_{j1}} \vee \widetilde{x_{j2}} \vee \widetilde{x_{j3}})$ den folgenden Graphen $G(c_j)$:



Vertex Cover ist **NP**-schwierig

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

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



Vertex Cover ist **NP**-schwierig

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

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

(A) Sei σ eine erfüllende Belegung für F.

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

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

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

Vertex Cover ist **NP**-schwierig

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

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

Definiere Belegung σ durch

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

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

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

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

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

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

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

Es sei

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

Satz 30

HP und HC sind **NP**-vollständig (sogar für ungerichtete Graphen).

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

Beweis: Wir zeigen nur die NP-Vollständigkeit von HC.

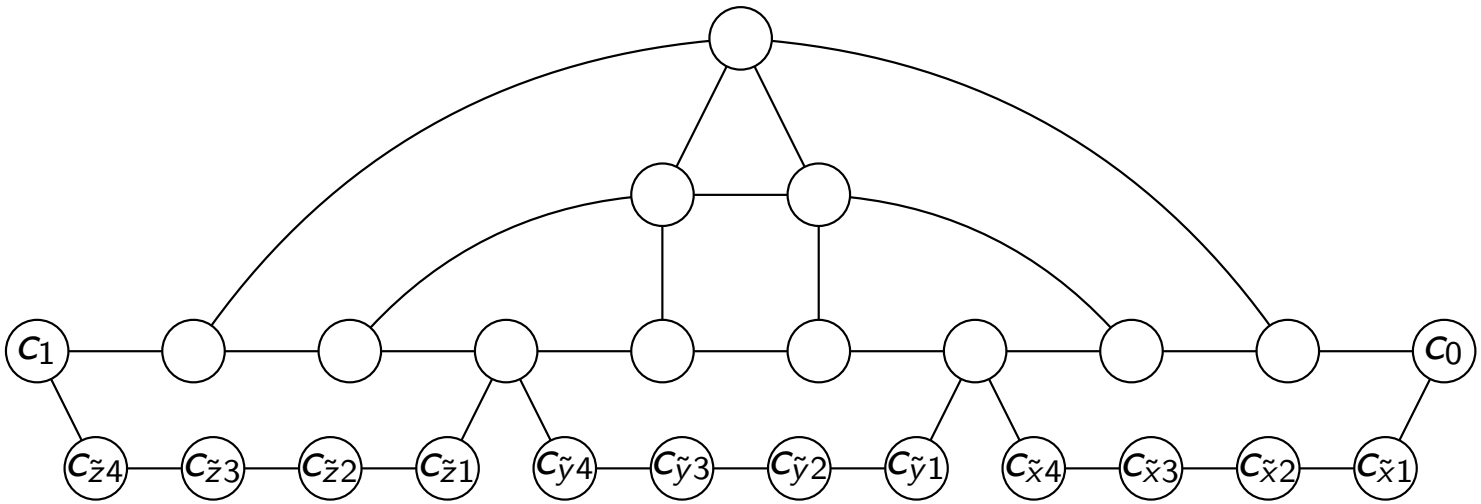
(A) $\mathrm{HC} \in \mathbf{NP}$: trivial.

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

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

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

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



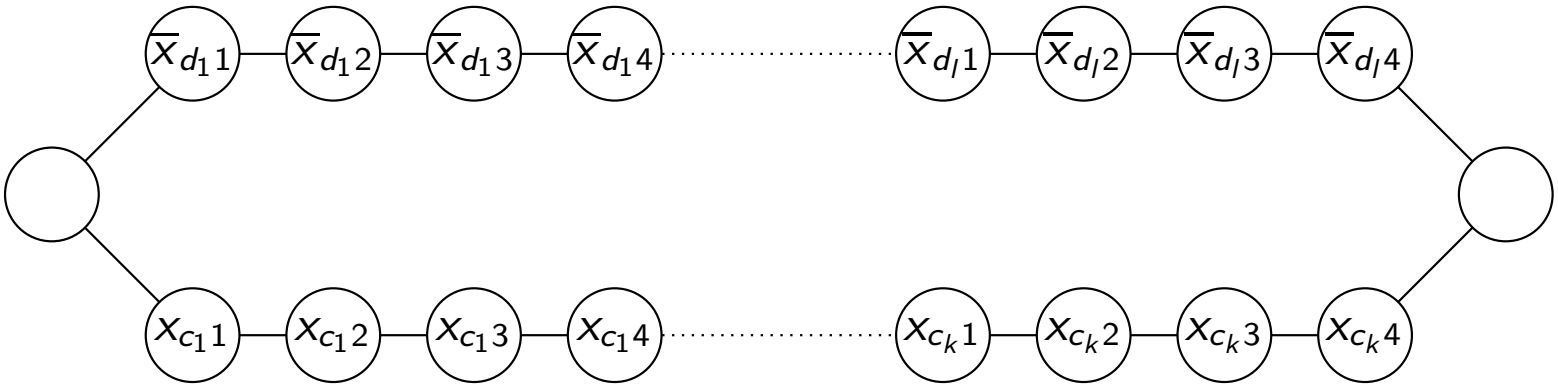
Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

Beachte:

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

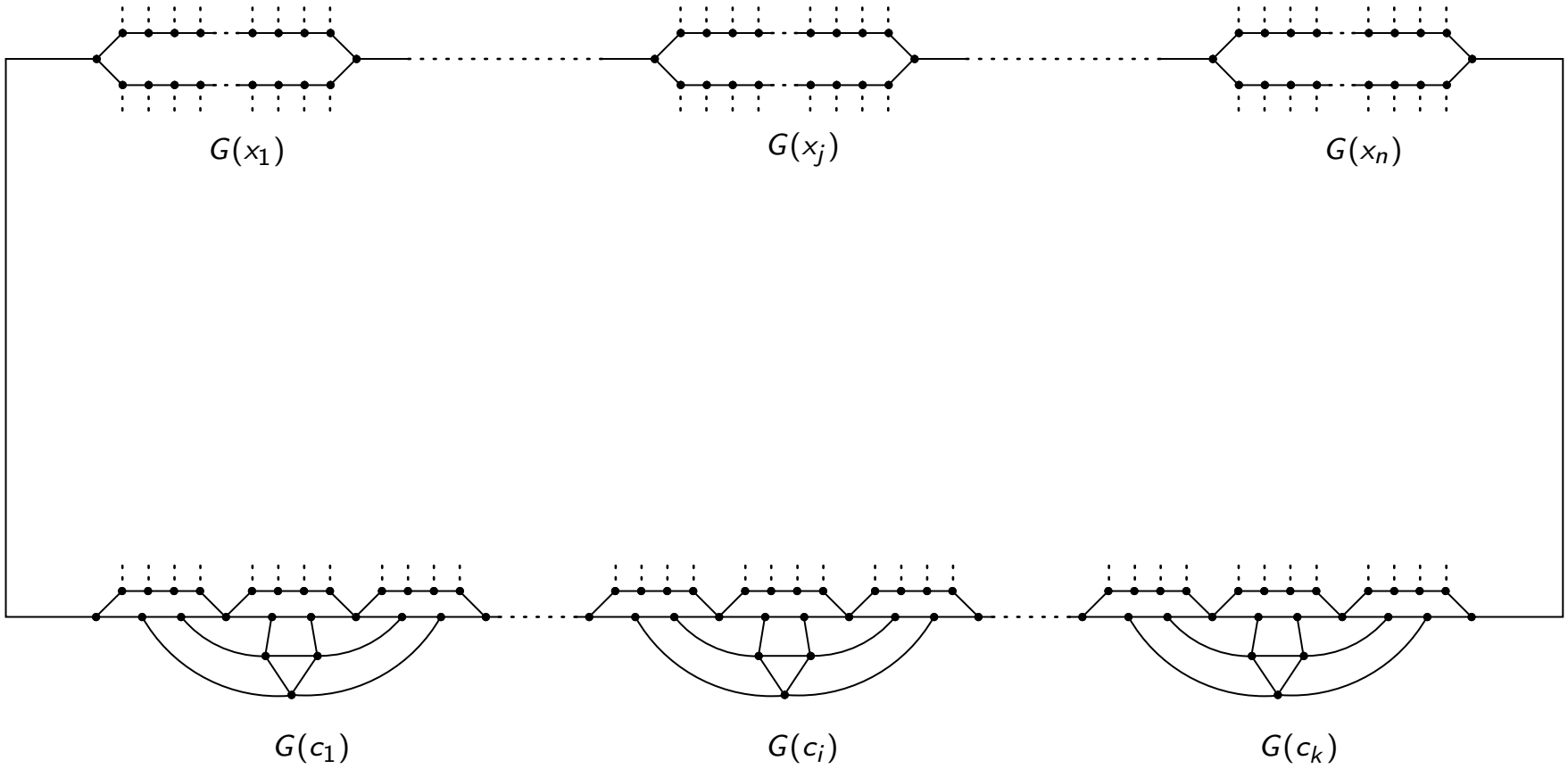
Für eine Variable x sei $\{c_1, \ldots, c_k\}$ die Menge der Klauseln mit $x \in c_i$ und $\{d_1, \ldots, d_l\}$ die Menge der Klauseln mit $\overline{x} \in d_j$.

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



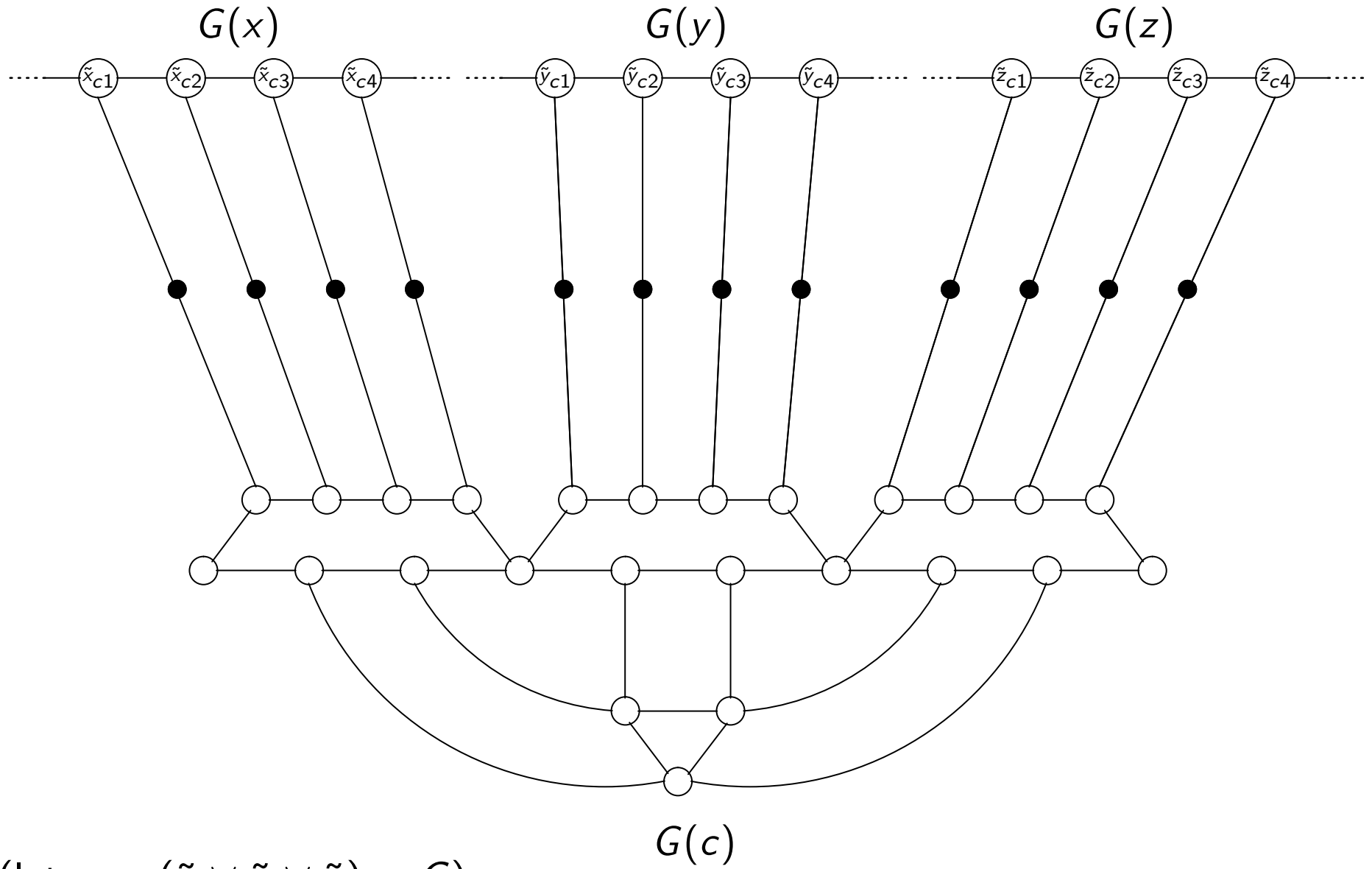
Hamilton-Kreis und Hamilton-Pfad sind **NP**-vollständig

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



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

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

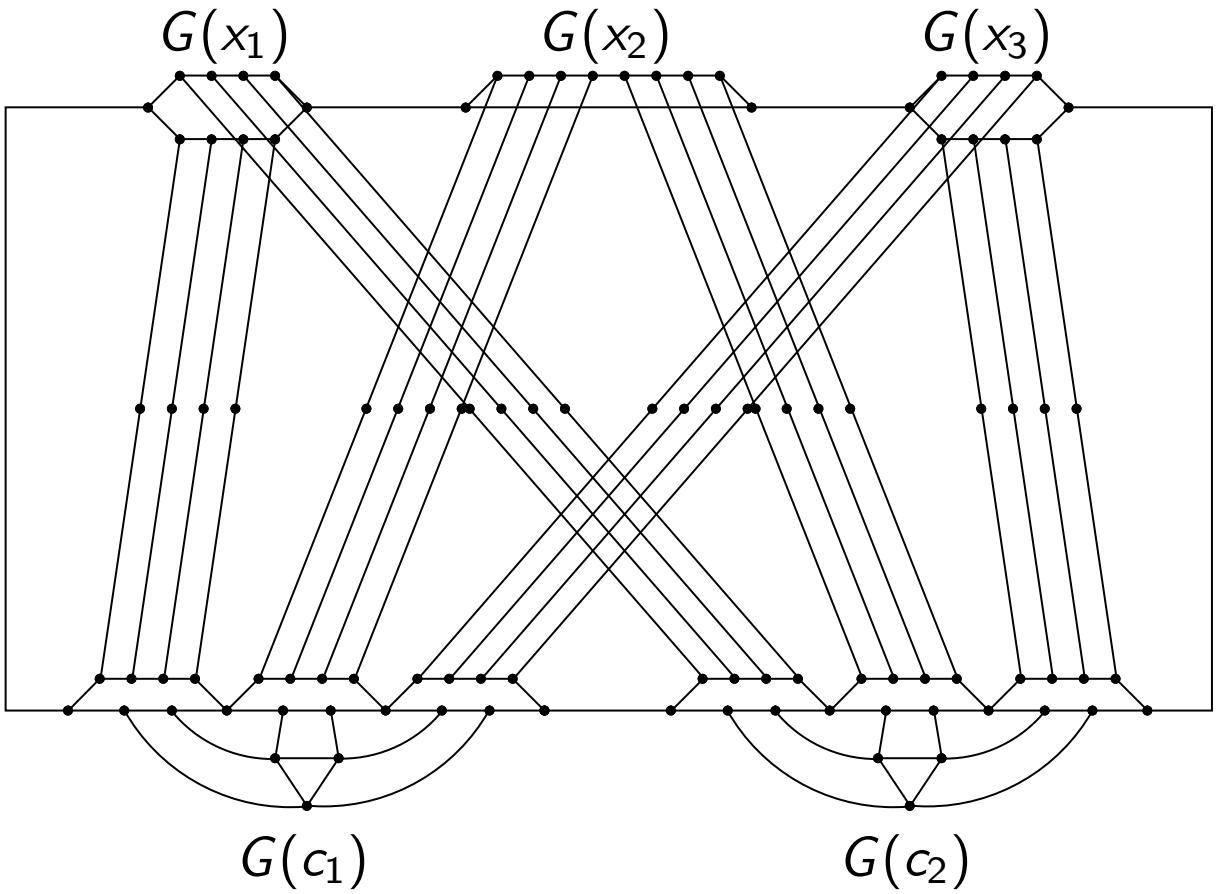


(let $c = (\tilde{x} \vee \tilde{y} \vee \tilde{z}) \in C$)

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

Beispiel: Sei $F = \underbrace{(x_1 \vee \overline{x_2} \vee \overline{x_3})}_{c_1} \wedge \underbrace{(\overline{x_1} \vee \overline{x_2} \vee x_3)}_{c_2}$.

Dann ist $G(F)$ der folgende Graph:



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

Behauptung: $F \in \mathrm{SAT}$ genau dann, wenn $G(F)$ einen Hamilton-Kreis hat.

Angenommen σ ist eine erfüllende Belegung von F.

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

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

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

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

Anschließend endet der Weg im Startknoten.

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

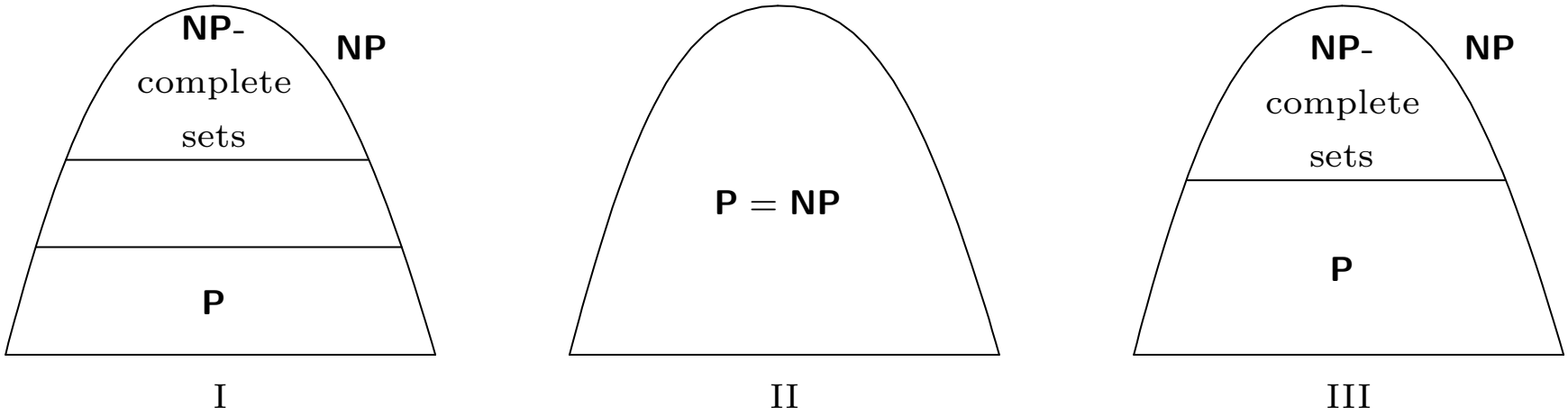
Sei C ein Hamilton-Kreis für $G(F)$.

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

Dies definiert eine erfüllende Belegung σ von F. □

Nicht **NP**-vollständige Mengen innerhalb von **NP** \ **P**

Nach unserem bisherigen Wissen wären folgende 3 Situationen möglich:



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

Satz 31 (Ladner)

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

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

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

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

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

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

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

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

Beachte: Jedes Polynom ist zeitkonstruierbar.

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

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

Wir verwenden ferner im folgenden einen beliebigen (exponentiellen) deterministischen Algorithmus zur Erkennung von SAT.

Für Mengen L und K ist

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

die symmetrische Differez.

Beachte: Ist $L \in \mathcal{C}$ für eine der hier betrachteten Komplexitätsklassen $\mathcal{C}$ und ist $|L \,\Delta\, K| < \infty$, so gilt auch $K \in \mathcal{C}$.

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

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

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

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

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

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

IA: Da $f(0) = 0$ gilt, folgt $f(0) \leq f(1)$.

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

$\rightsquigarrow$ Es gibt $i_0 < n$, $i_1 < n+1$ mit $i_0 \leq i_1$, $f(i_0) = k_0$ und $f(i_1) = k_1$.

IH $\rightsquigarrow$ $k_0 \leq k_1$.

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

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

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

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

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

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

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

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

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

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

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

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

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

Widerspruch zur Minimalität von n!

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

Falls bei der Berechnung von $f(n)$ ein Zeuge x gefunden wird, muss dieser zu $L(M_j) \,\Delta\, L_f$ gehören.

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

Also gilt $f(n) \leq 2j$ für alle n.

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

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

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

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

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

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

Man beachte nun:

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

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

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

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

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

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

Dünne Mengen und der Satz von Mahaney

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

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

existiert.

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

Satz von Mahaney, 1982

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

Beweis des Satzes von Mahaney

Es wird zunächst eine Menge SAT' definiert:

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

Hierbei ist

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

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

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

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

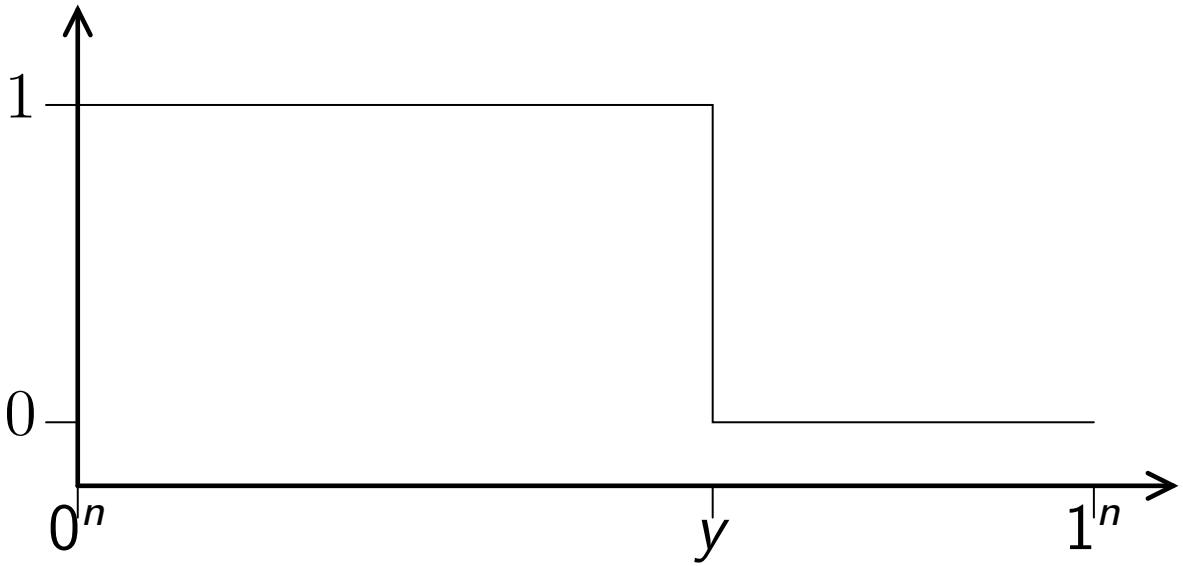
Beweis des Satzes von Mahaney

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

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

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

Der Graph von f_F ist eine Treppenfunktion:



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

Beweis des Satzes von Mahaney

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

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

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

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

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

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

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

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

Beweis des Satzes von Mahaney

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

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

Aus (6) folgt

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

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

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

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

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

Wenn dabei zwei Tupel $\langle F, y_i \rangle$ und $\langle F, y_j \rangle$ auf denselben Wert abgebildet werden, findet eine Kollisionsauflösung statt:

Beweis des Satzes von Mahaney

Kollisionsauflösung

Sei o.B.d.A. $y_i < y_j$. Dann werden alle Teilintervalle mit linken Grenzen $y_i, y_{i+1}, \ldots, y_{j-1}$ gelöscht, denn alle haben denselben Funktionswert $f_F(y_j)$.

Beachte: Dabei wird das Intervall, das die größte erfüllende Belegung y enthält (sofern überhaupt eine erfüllende Belegung existiert), nicht gelöscht: $y_i \leq y < y_j \implies H(\langle F, y_i \rangle) \neq H(\langle F, y_j \rangle)$.

Seien $z_1, z_2, \ldots, z_l$ die verbleibenden linken Grenzen.

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

Aus (7) folgt $\langle F, z_{p(n)+1} \rangle, \ldots, \langle F, z_l \rangle \notin \mathrm{SAT}'$ und die zugehörigen Intervalle mit den linken Grenzen $z_{p(n)+1}, \ldots, z_l$ können gelöscht werden, ohne das Intervall mit der größten erfüllenden Belegung y zu löschen.

Am Ende bleiben also höchsten $p(n)$ Intervalle übrig.

Beweis des Satzes von Mahaney

Jetzt wird eine Intervallhalbierung bei den verbleibenden Intervallen durchgeführt.

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

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

Beweis des Satzes von Mahaney

Die bis zu diesem Zeitpunkt verbrauchte Zeit ist polynomial beschränkt.

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

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

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

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

Vollständige Probleme für **P**

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

Satz 32

L_{cfe} ist **P**-vollständig.

Beweis:

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

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

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

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

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

Leerheit kontextfreier Sprachen ist **P**-vollständig

Wir setzen für M o.B.d.A. ähnliche Konventionen wie im Beweis des Satzes von Cook voraus:

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

Es existiert eine Funktion $\Delta : (\Gamma \cup Q)^4 \to (\Gamma \cup Q)$ existiert, so dass für alle $\alpha, \alpha' \in \Box(Q \cup \Gamma)^*\Box\Box$ mit $|\alpha| = |\alpha'|$ gilt:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \text{Conf und}$$
$$\forall i \in \{0, \ldots, p(n)\} : \Delta(\alpha[i-1], \alpha[i], \alpha[i+1], \alpha[i+2]) = \alpha'[i].$$

Leerheit kontextfreier Sprachen ist **P**-vollständig

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

$$V = \{S\} \cup \{V(a,t,j) \mid a \in \Gamma \cup Q,\ 0 \leq t \leq p(n),\ -1 \leq j \leq p(n)+2\},$$

leerem Terminalalphabet, Startsymbol S und der Regelmenge P, bestehend aus:

$$\begin{aligned}
S &\to V(q_J, t, j) \text{ für } 0 \leq t, j \leq p(n) \\
V(a, t+1, j) &\to V(b, t, j-1)V(c, t, j)V(d, t, j+1)V(e, t, j+2) \\
&\qquad \text{falls } \Delta(b, c, d, e) = a, \\
&\qquad 0 \leq t \leq p(n) - 1, 0 \leq j \leq p(n) \\
V(\square, t, j) &\to \lambda \text{ für } 0 \leq t \leq p(n), j \in \{-1, p(n)+1, p(n+2)\} \\
V(q_0, 0, 0) &\to \lambda \\
V(w_j, 0, j) &\to \lambda \text{ für } 1 \leq j \leq n \\
V(\square, 0, j) &\to \lambda \text{ für } n+1 \leq j \leq p(n)+2
\end{aligned}$$

Leerheit kontextfreier Sprachen ist **P**-vollständig

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

Sei $\alpha_0 \vdash_M \alpha_1 \vdash_M \cdots \vdash_M \alpha_{p(n)}$ die eindeutige bei $\alpha_0 = \square q_0 w_1 \cdots w_n \square^{p(n)-n+2}$ beginnende Berechnung, $\alpha_i \in \text{Conf}$.

Für $-1 \leq j \leq p(n)+2$ und $0 \leq t \leq p(n)$ sei $\alpha(t,j) = \alpha_t[j]$.

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

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

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

Die Fälle $t = 0$ und $j \in \{-1, p(n)+1, p(n)+2\}$ folgen sofort aus der Definition von $G(w)$.

Leerheit kontextfreier Sprachen ist **P**-vollständig

Falls $t \geq 1$ und $0 \leq j \leq p(n)$, so gibt es $b, c, d, e \in \Gamma \cup Q$ mit $\Delta(b, c, d, e) = a$, $\alpha(t-1, j-1) = b$, $\alpha(t-1, j) = c$, $\alpha(t-1, j+1) = d$, $\alpha(t-1, j+2) = e$.

Mit Induktion über t folgt, dass die Sprachen $L(V(b, t-1, j-1))$, $L(V(c, t-1, j))$, $L(V(d, t-1, j+1))$, $L(V(e, t-1, j+1))$ nicht leer sind.

Da $G(w)$ die Produktion

$$V(a, t, j) \to V(b, t-1, j-1)V(c, t-1, j)V(d, t-1, j+1)V(e, t-1, j+2)$$

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

Leerheit kontextfreier Sprachen ist **P**-vollständig

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

Die Fälle $t = 0$ und $j \in \{-1, p(n)+1, p(n)+2\}$ folgen sofort aus der Definition von $G(w)$.

Falls $t \geq 1$ und $0 \leq j \leq p(n)$, so muss es eine Produktion

$$V(a,t,j) \to V(b,t-1,j-1)V(c,t-1,j)V(d,t-1,j+1)V(e,t-1,j+2)$$

geben (inbesondere $\Delta(b,c,d,e) = a$), so dass $L(V(b,t-1,j-1))$, $L(V(c,t-1,j))$, $L(V(d,t-1,j+1))$ und $L(V(e,t-1,j+2))$ nicht leer sind.

Induktion $\rightsquigarrow$ $\alpha(t-1,j-1) = b$, $\alpha(t-1,j) = c$, $\alpha(t-1,j+1) = d$, $\alpha(t-1,j+2) = e$.

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

Boolesche Schaltkreise

Definition boolescher Schaltkreis

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

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

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

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

Die Knoten werden als Gatter bezeichnet.

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

Boolesche Schaltkreise

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

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

Circuit Value (CV) ist das folgende Problem:

INPUT: Ein boolescher Schaltkreis C

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

Monotone Circuit Value (MCV) ist das folgende Problem:

INPUT: Ein monotoner boolescher Schaltkreis C

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

Circuit Value ist **P**-vollständig

Satz 33

CV und MCV sind **P**-vollständig.

Beweis:

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

(ii) MCV ist **P**-schwierig:

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

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

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

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

Circuit Value ist **P**-vollständig

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

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

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

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

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

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

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

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

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

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

Circuit Value ist **P**-vollständig

Nun gilt für jedes Nichtterminal A genau einer der 4 folgenden Fälle:

1. Es gibt keine Produktion für A.
2. A steht in genau einer Produktion auf der linken Seite, und diese Produktion ist vom Typ $A \to \lambda$.
3. A steht in genau einer Produktion auf der linken Seite, und diese Produktion ist vom Typ $A \to BC$.
4. A steht in genau zwei Produktion auf der linken Seite und diese Produktionen sind beide vom Typ $A \to B$: $A \to B|C$

Die neue Grammatik erzeugt λ genau dann, wenn die alte Grammatik dies tut.

Wir bezeichnen die neue Grammatik wieder mit $G(w)$.

Circuit Value ist **P**-vollständig

Wir definieren nun einen Schaltkreis $C(w)$ wie folgt:

Jedem Nichtterminal von $G(w)$ wird nun ein Gatter in $C(w)$ zugeordnet.

Das Startsymbol von $G(w)$ ist das Ausgangsgatter von $C(w)$.

1. Ein Nichtterminal A vom Typ 1 wird ein 0-Inputgatter.
2. Ein Nichtterminal A vom Typ 2 wird ein 1-Inputgatter
3. Ein Nichtterminal A vom Typ 3 wird ein $\wedge$-Gatter mit Eingängen B und C.
4. Ein Nichtterminal A vom Typ 4 wird ein $\vee$-Gatter mit Eingängen B und C.

Circuit Value ist **P**-vollständig

Beachte: Der so erzeugte Schaltkreis $C(v)$ ist in der Tat azyklisch, denn $G(w)$ ist azyklisch.

Es gilt: $L(A) \neq \emptyset \iff$ Gatter A wertet sich in $C(w)$ zu 1 aus.

Folgerung: $L(G(w)) \neq \emptyset \iff$ Ausgangsgatter von $C(w)$ wertet sich zu 1 aus. □

Bemerkung: In einem Schaltkreis kann ein Gatter Ausgangsgrad > 1 haben. Dies scheint für die **P**-Schwierigkeit wichtig zu sein:

Die Menge der (variablen-freien) booleschen Ausdrücke kann durch folgende Grammatik definiert werden:

$$A ::= 0 \mid 1 \mid (\neg A) \mid (A \wedge A') \mid (A \vee A')$$

Boolesche Ausdrücke werden somit zu Bäumen, wenn man sie in Schaltkreise umwandelt.

Buss 1987: Die Menge der booleschen Ausdrücke, die sich zu 1 auswerten ist vollständig für die Komplexitätsklasse $\mathbf{NC}^1 \subseteq \mathbf{L}$.

Vollständige Probleme für **PSPACE**: Quantifizierte Boolesche Formeln

Quantifizierte boolesche Formeln

Die Menge M der quantifizierten booleschen Formeln ist die kleinste Menge mit:

- $x_i \in M$ für alle $i \geq 1$
- $0, 1 \in M$
- $E, F \in M, i \geq 1 \implies (\neg E), (E \wedge F), (E \vee F), \forall x_i E, \exists x_i E \in M$

Alternativ: M lässt sich durch eine eindeutige kontextfreie Grammatik über dem Terminalalphabet $\Sigma = \{x, 0, 1, (,), \neg, \wedge, \vee, \forall, \exists\}$ erzeugen.

Dabei werden Variablen durch Wörter aus $x1\{0, 1\}^*$ dargestellt.

Erfüllbarkeit von quantifizierten booleschen Formeln

Die Erfüllbarkeit von quantifizierten booleschen Formeln wird durch die Existenz einer erfüllenden Belegung definiert.

Eine Belegung ist eine Funktion $b : \{x_1, x_2, \ldots\} \to \{0, 1\}$.

Diese kann für eine gegebene Formel F auf die in F vorkommenden Variablen eingeschränkt werden.

Für $z \in \{0, 1\}$ und eine Belegung b sei $b[x_j \mapsto z]$ die Belegung mit

- $b[x_j \mapsto z](x_i) = b(x_i)$ für $i \neq j$ und
- $b[x_j \mapsto z](x_j) = z$.

Erfüllbarkeit von quantifizierten booleschen Formeln

Induktive Definition der Erfüllbarkeit einer Formel F bezüglich einer Belegung b:

Die Belegung b erfüllt die Formel F genau dann, wenn eine der folgenden Bedingungen zutrifft:

$F = 1$,

$F = x_j$ und $b(x_j) = 1$,

$F = (\neg E)$ und b erfüllt E nicht,

$F = (F_1 \wedge F_2)$ und b erfüllt F_1 und F_2,

$F = (F_1 \vee F_2)$ und b erfüllt F_1 oder F_2,

$F = \exists x_j E$ und $b[x_j \mapsto 0]$ oder $b[x_j \mapsto 1]$ erfüllt E,

$F = \forall x_j E$ und $b[x_j \mapsto 0]$ und $b[x_j \mapsto 1]$ erfüllen E.

Wird F von jeder Belegung erfüllt, so heißt F gültig.

Erfüllbarkeit von quantifizierten booleschen Formeln

Die Menge $\mathrm{Frei}(F)$ der freien Variablen der Formel F ist wie folgt definiert:

- $\mathrm{Frei}(0) = \mathrm{Frei}(1) = \emptyset$
- $\mathrm{Frei}(x_i) = \{x_i\}$
- $\mathrm{Frei}(\neg F) = \mathrm{Frei}(F)$
- $\mathrm{Frei}((F \wedge G)) = \mathrm{Frei}((F \vee G)) = \mathrm{Frei}(F) \cup \mathrm{Frei}(G)$
- $\mathrm{Frei}(\exists x_j F) = \mathrm{Frei}(\forall x_j F) = \mathrm{Frei}(F) \setminus \{x_j\}$

Eine Formel F mit $\mathrm{Frei}(F) = \emptyset$ nennt man geschlossen.

Beachte: Die Erfüllbarkeit einer geschlossenen Formel ist nicht von der Belegung abhängig, d. h. existiert eine erfüllende Belegung, so ist die Formel bereits gültig.

Bezeichnung: QBF ist die Menge der geschlossenen quantifizierten booleschen Formeln, die gültig sind.

QBF ist **PSPACE**-vollständig

Satz 34

QBF ist **PSPACE**-vollständig.

Beweis:

(i) $\mathrm{QBF} \in$ **PSPACE**:

Sei E eine geschlossene quantifizierte boolesche Formel, in der die Variablen $x_1, \ldots, x_n$ vorkommen.

O.b.d.A. ist E nur aus $1, x_1, \ldots, x_n, \neg, \wedge, \exists$ aufgebaut, und keine Variable x_i wird in E zweimal quantifiziert (der Algorithmus auf der nächsten Folie würde z.B. für $\exists x((\exists x : 0) \vee x)$ ein falsches Ergebnis liefern.

Der folgende rekursive deterministische Algorithmus, in dem $x_1, \ldots, x_n$ globale Variablen sind, überprüft in polynomiellen Platz, ob E gültig ist.

QBF ist **PSPACE**-vollständig

FUNCTION **check**(E)
  if $E = 1$ **then return**(1)
  elseif $E = x_i$ **then return**(x_i)
  elseif $E = (\neg F)$ **then return**(**not check**(F))
  elseif $E = (E_1 \wedge E_2)$ **then return**(**check**(E_1) **and** **check**(E_2))
  elseif $E = \exists x_i F$ **then**
    $x_i := 1$
    if check$(F) = 1$ **then**
      return(1)
    else
      $x_i := 0$
      return(**check**(F))
    endif
  endif
ENDFUNC

QBF ist **PSPACE**-vollständig

(ii) QBF ist **PSPACE**-schwierig:

Sei $L \in$ **PSPACE**. Es gilt $L = L(M)$ für eine $p(n)$-platzbeschränkte 1-Band-Turingmaschine, wobei $p(n)$ ein geeignetes Polynom sei.

Im weiteren werden Konfigurationen binär kodiert.

Ohne Einschränkung ist die Länge aller Binärkodierungen der von einer Startkonfiguration $\mathrm{Start}(w)$, $|w| = n$, aus erreichbaren Konfigurationen durch $p(n)$ und deren Anzahl durch $2^{p(n)}$ begrenzt.

Betrachte den Savitch-Ansatz (α_f ist wieder die einzige akzeptierende Konfiguration, die von $\mathrm{Start}(w)$ evtl. erreichbar ist):

$$\mathrm{Reach}(\mathrm{Start}(w), \alpha_f, p(n)) \iff w \in L$$

$$\mathrm{Reach}(\alpha, \beta, i) = \exists\gamma\big(\mathrm{Reach}(\alpha, \gamma, i-1) \wedge \mathrm{Reach}(\gamma, \beta, i-1)\big) \quad \text{für } i > 0$$

$$\mathrm{Reach}(\alpha, \beta, 0) = \alpha \vdash_M^{\leq 1} \beta$$

QBF ist **PSPACE**-vollständig

Die direkte Einsetzung würde auf eine Formel exponentieller Länge führen.

Lösung: Wir führen für die Konfigurationen Konfigurationsvariable $X, Y, U, V, \ldots$ ein und definieren für $i > 0$:

$$\mathrm{Reach}(X, Y, i) :=$$
$$\exists U \forall V \forall W \left(\begin{array}{l} \Big((V = X \wedge W = U) \vee (V = U \wedge W = Y) \Big) \\ \Longrightarrow \mathrm{Reach}(V, W, i-1) \end{array} \right)$$

1. Schritt: Berechne für Eingabe $w \in \Sigma^*$ mit $|w| = n$ durch iteriertes Anwendung obiger Rekursion, beginnend mit $\mathrm{Reach}(\mathrm{Start}(w), \alpha_f, p(n))$, eine Formel F in den Konfigurationsvariablen $X, Y, \ldots$.

In F kommen atomare Formeln der Gestalt $\mathrm{Reach}(X, Y, 0)$ und $X = Y$ sowie die Konstanten vor $\mathrm{Start}(w)$ und α_f vor.

QBF ist **PSPACE**-vollständig

2. Schritt: Wir wandeln F in eine geschlossene quantifizierte boolesche Formel um:

- Ersetze jede Konfigurationsvariable X durch einen Block von $p(n)$ booleschen Variablen $x_1, \ldots, x_{p(n)}$. Aus $\forall X$ wird somit der Block $\forall x_1 \, \forall x_2 \, \cdots \forall \, x_{p(n)}$ und entsprechend für $\exists X$.
- Die Konstanten $\mathrm{Start}(w)$ und α_f werden durch konkrete Bitstrings ersetzt.
- $X = Y$ ersetzen wir durch die Formel $\bigwedge_{i=1}^{p(n)} (x_i \Leftrightarrow y_i)$.
- Aus einer atomaren Formel $\mathrm{Reach}(X, Y, 0)$ wird wie im Beweis zum Satz von Cook eine boolesche Formel die einen 1-Schritt Übergang beschreibt.

Wir erhalten so eine geschlossene quantifizierte boolesche Formel, die genau dann erfüllbar ist, wenn $w \in L$. □

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Erinnerung: Für ein endliches Alphabet Σ ist die Menge $\mathrm{Reg}(\Sigma)$ der regulären Ausdrücke über Σ induktiv wie folgt definiert:

- $\emptyset, \varepsilon \in \mathrm{Reg}(\Sigma)$
- $\Sigma \subseteq \mathrm{Reg}(\Sigma)$
- Wenn $\alpha, \beta \in \mathrm{Reg}(\Sigma)$, dann auch $(\alpha \cup \beta), (\alpha \cdot \beta), \alpha^* \in \mathrm{Reg}(\Sigma)$

Die durch einen regulären Ausdruck α definierte Sprache $L \subseteq \Sigma^*$ ist wie folgt induktiv definiert:

- $L(\emptyset) = \emptyset$, $L(\varepsilon) = \{\lambda\}$
- $L(a) = \{a\}$ für $a \in \Sigma$
- $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$, $L(\alpha \cdot \beta) = L(\alpha)L(\beta)$, $L(\alpha^*) = L(\alpha)^*$

Seien

$$\begin{aligned} \mathrm{RegÄquiv}(\Sigma) &= \{(\alpha, \beta) \mid \alpha, \beta \in \mathrm{Reg}(\Sigma), L(\alpha) = L(\beta)\} \\ \mathrm{RegUniv}(\Sigma) &= \{\alpha \mid \alpha \in \mathrm{Reg}(\Sigma), L(\alpha) = \Sigma^*\} \end{aligned}$$

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Satz 35

RegÄquiv(Σ) und RegUniv(Σ) sind **PSPACE**-vollständig für jedes Alphabet Σ mit $|\Sigma| \geq 2$.

Beweis:

(1) RegÄquiv(Σ) $\in$ **PSPACE**.

Seien $\alpha, \beta \in \text{Reg}(\Sigma)$.

Zunächst wandeln wir α, β in nichtdeterministische endliche Automaten A, B um mit $L(A) = L(\alpha)$, $L(B) = L(\beta)$.

Dies ist in Polynomialzeit möglich.

Wir müssen in polynomiellen Platz überprüfen, ob $L(A) \subseteq L(B)$ und $L(B) \subseteq L(A)$ gilt.

Wir überprüfen nur $L(A) \subseteq L(B)$, $L(B) \subseteq L(A)$ geht analog.

Es gilt: $L(A) \subseteq L(B) \quad \Leftrightarrow \quad L(A) \cap (\Sigma^* \setminus L(B)) = \emptyset$

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Sei $A = (Q_A, \Sigma, \delta_A, q_{0,A}, F_A)$ und $B = (Q_B, \Sigma, \delta_B, q_{0,B}, F_B)$.

Ein Automat für $\Sigma^* \setminus L(B)$ ist

$$B' = (2^{Q_B}, \Sigma, \delta'_B, \{q_{0,B}\}, \{P \subseteq Q_B \mid P \cap F_B = \emptyset\})$$

wobei für alle $a \in \Sigma$, $P, R \subseteq Q_B$ gilt:

$$(P, a, R) \in \delta'_B \;\Leftrightarrow\; R = \{q \in Q_B \mid \exists p \in P : (p, a, q) \in \delta_B\}.$$

Ein Automat für $L(A) \cap (\Sigma^* \setminus L(B))$ ist dann

$$C = (Q_A \times 2^{Q_B}, \Sigma, \delta_C, (q_{0,A}, \{q_{0,B}\}), F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\})$$

wobei für alle $a \in \Sigma$, $p, r \in Q_A$, $P, R \subseteq Q_B$ gilt:

$$((p, P), a, (r, R)) \in \delta_C \;\Leftrightarrow\; (p, a, r) \in \delta_A \wedge (P, a, R) \in \delta'_B$$

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Wir müssen in polynomiellen Platz überprüfen, ob $L(C) \neq \emptyset$ gilt.

Vorsicht: Den Automaten C (oder B') können wir nicht konstruieren, er passt nicht in polynomiellen Platz!

Definiere den gerichteten Graphen

$$G = (Q_A \times 2^{Q_B}, \{((p,P),(r,R)) \mid \exists a \in \Sigma : ((p,P),a,(r,R)) \in \delta_C\}).$$

Es gilt: $L(C) \neq \emptyset$ g.d.w. in dem Graphen G ein Pfad von $(q_{0,A}, \{q_{0,B}\})$ zu einem Zustand aus $F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\}$ existiert.

Letzteres kann nichtdeterministisch in poly. Platz überprüft werden:

- Rate einen Zustand $(p,P) \in F_A \times \{P \subseteq Q_B \mid P \cap F_B = \emptyset\}$ (passt in polynomiellen Platz)
- Rate einen Pfad von $(q_{0,A}, \{q_{0,B}\})$ nach (p,P). Dabei müssen wir uns immer nur den aktuellen Knoten aus G merken (passt in polynomiellen Platz)

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

(2) RegUniv_Σ ist **PSPACE**-schwierig.

Sei $L \in$ **PSPACE** und $L(M) = L$ für eine $p(n)$-platzbeschränkte deterministische Turingmaschine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_J, q_N, \Box)$, $p(n) > n$ ein Polynom.

Sei $\Omega = Q \cup \Gamma$.

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

Konfigurationen von M werden wieder durch Wörter aus der Sprache $\text{Conf} = \{\Box uqv\Box\Box \mid q \in Q,\ uv \in \Gamma^{p(n)}\} \subseteq \Omega^{p(n)+4}$ beschrieben.

Es existiert eine Funktion $\Delta : \Omega^4 \to \Omega$ existiert, so dass für alle $\alpha, \alpha' \in \Box\Omega^*\Box\Box$ mit $|\alpha| = |\alpha'|$ gilt:

$$\alpha, \alpha' \in \text{Conf und } \alpha \vdash_M \alpha'$$
$$\Longleftrightarrow$$
$$\alpha \in \text{Conf und}$$
$$\forall i \in \{0, \ldots, p(n)\} : \Delta(\alpha[i-1], \alpha[i], \alpha[i+1], \alpha[i+2]) = \alpha'[i].$$

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Die Startkonfiguration ist $\alpha_0 := \square q_0 w_1 \cdots w_n \square^{p(n)-n+2}$.

Eine akzeptierende Berechnung (sofern sie existiert)

$$\alpha_0 \vdash_M \alpha_1 \vdash_M \alpha_2 \vdash_M \cdots \vdash_M \alpha_l \in \mathrm{Accept}_M$$

von M bei Eingabe w kodieren wir durch das Wort $\alpha_0\alpha_1\alpha_2\cdots\alpha_l \in \Omega^*$.

Wir konstruieren aus w einen regulären Ausdruck $\alpha(w)$ (mittels eines logspace transducer), so dass $L(\alpha(w))$ die Menge aller Wörter über dem Alphabet Ω, die keine akzeptierende Berechnung von M bei Eingabe w beschreiben, ist.

Also gilt: $w \notin L(M)$ genau dann, wenn $L(\alpha(w)) = \Omega^*$.

Für $C \subseteq \Omega$ identifizieren wir C mit dem regulären Ausdruck $\bigcup_{a \in C} a$.

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

Es ist $\alpha(w) = \alpha_1 \cup \alpha_2 \cup \alpha_3 \cup \alpha_4$, wobei die Ausdrücke $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ wie folgt definiert sind.

(a) Alle Wörter, die nicht die richtige Länge haben:

$$\alpha_1 = \varepsilon \cup \bigcup_{i=1}^{p(n)+3} \left(\Omega^{p(n)+4}\right)^* \Omega^i$$

(b) Alle Wörter, die nicht mit $\alpha_0 = \square q_0 w_1 \cdots w_n \square^{p(n)-n+2}$ beginnen:

$$\alpha_2 = \bigcup_{i=-1}^{p(n)+2} \Omega^{i+1} \cdot (\Omega \setminus \{\alpha_0[i]\}) \cdot \Omega^*$$

(c) Alle Wörter, die "Δ nicht respektieren":

$$\alpha_3 = \Omega^* \left(\bigcup_{u \in \Omega^4} u \cdot \Omega^{p(n)+1} \cdot (\Omega \setminus \{\Delta(u)\}) \right) \Omega^*$$

Äquivalenz regulärer Ausdrücke ist **PSPACE**-vollständig

(d) Alle Wörter, die q_J nicht enthalten:

$$\alpha_4 = (\Omega \setminus \{q_J\})^*$$

Abschließend muss das Alphabet $\Omega = Q \cup \Gamma$ noch binär kodiert werden. □

Orakel-Turingmaschinen u. relative Schwierigkeit von $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$

Eine nichtdeterministische Orakel-Turingmaschine (kurz OTM) $M^?$ ist eine nichtdeterministische Turingmaschine mit folgenden Besonderheiten:

- $M^?$ hat drei ausgezeichneten Zuständen $q_J, p_N, p_?$ sowie
- ein ausgezeichnetes Arbeitsband — das Orakelband — auf welches nur geschrieben wird.
- Das Orakelband enthält zu jedem Zeitpunkt einen String $w(o)$ über einem Alphabet Σ.
- Befindet sich $M^?$ im Zustand $q_?$ so kann $M^?$ unabhängig von den gelesenen Bandsymbolen nichtdeterministisch in den Zustand p_N oder den Zustand p_J gehen. Der Inhalt des Orakelbands wird dabei gelöscht.

Eine OTM $M^?$ ist deterministisch, falls sich $M^?$ auf allen Zuständen außer $q_?$ deterministisch verhält.

Orakel-Turingmaschinen

Zeit- und Platzschranken werden für OTMs wie für normale Turingmaschinen definiert.

Wir betrachten dafür eine OTM als eine nichtdeterministische Turingmaschine (d. h. auf allen Berechnungspfaden muss die Zeitschranke bzw. Platzschranke eingehalten werden).

Sei $A \subseteq \Sigma^*$ eine beliebige (nicht notwendigerweise entscheidbare) Menge.

Wir definieren Berechnungen einer Maschine M^A auf eine Eingabe $v \in \Gamma^*$ wie folgt:

- Auf Zuständen in $Q \setminus \{q_?\}$ verhält sich M^A wie $M^?$.
- Befindet sich $M^?$ im Zustand $q_?$, so ist der Folgezustand (unabhängig von gelesenen Bandsymbolen) p_J (bzw. p_N), falls aktuell $w(o) \in A$ (bzw. $w(o) \notin A$) gilt.

Relative Komplexitätsklassen

Wir können jetzt Komplexitätslassen wie

$$\mathbf{P}^A = \left\{ L \subseteq \Sigma^* \;\middle|\; \begin{array}{l}\text{Es gibt eine deterministische polynomial}\\ \text{zeitbeschränkte OTM } M^? \text{ mit } L = L(M^A)\end{array} \right\}$$

$$\mathbf{NP}^A = \left\{ L \subseteq \Sigma^* \;\middle|\; \begin{array}{l}\text{Es gibt eine nichtdeterministische polynomi-}\\ \text{al zeitbeschränkte OTM } M^? \text{ mit } L = L(M^A)\end{array} \right\}$$

definieren.

Bemerkung:

1. $\mathbf{NP} \cup \mathbf{CoNP} \subseteq \mathbf{P}^{\mathrm{SAT}} \subseteq \mathbf{PSPACE}$ Es ist offen, ob „=“ gilt.
2. $A \in \mathbf{P}^A$ für alle $A \subseteq \Sigma^*$, damit kann insbesondere $\mathbf{P}^A$ unentscheidbare Sprachen enthalten.
3. Ist A entscheidbar, so ist $\mathbf{NP}^A$ eine Familie entscheidbarer Sprachen.
4. Trivialerweise gilt $\mathbf{P}^A \subseteq \mathbf{NP}^A$.

Polynomiale Turing-Reduzierbarkeit

Seien A und B Sprachen. Dann ist A polynomial Turing-reduzierbar auf B (kurz $A \leq_T^P B$), falls eine deterministische polynomial zeitbeschränkte OTM $M^?$ mit $A = L(M^B)$ existiert.

Mit dieser Definition gilt folgendes:

- Wenn $A \leq_m^P B$ (d.h. es existiert eine Polynomialzeitreduktion von A auf B), dann $A \leq_T^P B$.
- Wenn $A \leq_T^P B$ und $B \in \mathbf{P}$, dann auch $A \in \mathbf{P}$.

Eine Welt wo $\mathbf{P} = \mathbf{NP}$

Satz 36

Es gibt ein Orakel $A \subseteq \Sigma^*$ in **PSPACE** mit $\mathbf{P}^A = \mathbf{NP}^A$.

Beweis:

Betrachte eine **PSPACE**-vollständige Sprache, etwa $A = \mathrm{QBF}$.

Dann gilt

$$\mathbf{PSPACE} \subseteq \mathbf{P}^{\mathrm{QBF}} \subseteq \mathbf{NP}^{\mathrm{QBF}} \subseteq \bigcup_{k \geq 1} \mathrm{NSPACE}(n^k) = \mathbf{PSPACE}.$$

Also gilt

$$\mathbf{PSPACE} = \mathbf{P}^{\mathrm{QBF}} = \mathbf{NP}^{\mathrm{QBF}}.$$

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Satz 37 (Baker, Gill, Solovay, 1975)

Es gibt ein entscheidbares Orakel $B \subseteq \{0,1\}^*$ mit $\mathbf{P}^B \neq \mathbf{NP}^B$.

Beweis:

Das Orakel $B \subseteq \{0,1\}^*$ wird so definiert, dass gilt:
$\forall n \geq 0 : |B \cap \{w \in \{0,1\}^* \mid |w| = n\}| \leq 1$

Für eine Sprache $B \subseteq \{0,1\}^*$ sei

$$L_B = \left\{1^n \;\middle|\; \exists w \in B \text{ mit } |w| = n\right\}.$$

Dann gilt offensichtlich: $L_B \in \mathbf{NP}^B$.

Noch zu zeigen: Es gibt eine entscheidbare Sprache B mit $L_B \notin \mathbf{P}^B$.

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Sei $M_1^?, M_2^?, \ldots$ eine effektive Aufzählung aller deterministischen Orakel-Turingmaschinen mit zusätzlicher polynomieller Zeitschranke.

Wir nehmen an, dass jedes $M_i^?$ in der Aufzählung unendlich oft vorkommt.

Dies kann z.B. dadurch erreicht werden, dass $M^?(i,j) := M_i^?$ aus einer ursprünglichen Aufzählung definiert wird und dann die Orakel-Turingmaschinen aus $\{M^?(i,j) | (i,j) \in \mathbb{N} \times \mathbb{N}\}$ aufgezählt werden.

Wir definieren rekursiv das Orakel $B = \bigcup_{i \geq 0} B_i$ durch Mengen $B_0 \subseteq B_1 \subseteq B_2 \subseteq \cdots$ mit $B_i \subseteq \{w \in \{0,1\}^* \mid |w| \leq i\}$ und eine Ausnahmemengen $X_0 \subseteq X_1 \subseteq X_2 \subseteq \cdots$:

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

- Für $i = 0$ setze $B_0 = X_0 = \emptyset$.
- Für $i > 0$ simulieren wir die Rechnung von $M_i^?$ auf Input 1^i für $i^{\lceil \log i \rceil}$ Schritte.

  Wir nehmen an, dass B_{i-1} und X_{i-1} mit $B_{i-1} \cap X_{i-1} = \emptyset$ schon definiert sind.
  - Setze $X_i = X_{i-1}$.
  - Befragt $M_i^?$ im Laufe der Rechnung ein Wort w mit $|w| < i$, so wird die Rechnung entsprechend dem Orakel B_{i-1} fortgesetzt.
  - Befragt $M_i^?$ im Laufe der Rechnung ein Wort w mit $|w| \geq i$, so setze die Rechnung in dem „nein“-Zustand fort und setze $X_i := X_i \cup \{w\}$.
  - Akzeptiert $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ die Eingabe 1^i oder kommt $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ zu keiner Entscheidung, so setze $B_i := B_{i-1}$.
  - Verwirft $M_i^?$ innerhalb der Zeitschranke $i^{\lceil \log i \rceil}$ die Eingabe 1^i, so betrachte das lexikographisch erste Wort b_i in $\{0,1\}^i \setminus X_i$. Setze dann

$$B_i = \begin{cases} B_{i-1} \cup \{b_i\} & \text{falls } b_i \text{ existiert} \\ B_{i-1} & \text{sonst} \end{cases}$$

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

Es gilt mit $B = \bigcup_{i \geq 0} B_i$ und $X = \bigcup_{i \geq 0} X_i$:

- $B_i \subseteq \{w \in \{0,1\}^* \mid |w| \leq i\}$
- B ist entscheidbar, denn für $|w| = i$ gilt: $w \in B \Leftrightarrow w \in B_i$.
- $B_i \cap X_i = \emptyset$ für alle $i \geq 0$ und damit $B \cap X = \emptyset$.

Wir zeigen jetzt $L_B \notin \mathbf{P}^B$.

Angenommen, es wäre $L_B = L(M^B)$ für eine deterministische, polynomial zeitbeschränkte OTM $M^?$.

Dann gibt es ein Polynom $p(n)$ so, dass $M^?$ $p(n)$-zeitbeschränkt ist.

Wähle jetzt i genügend groß so, dass $\forall n \geq i : p(n) \leq n^{\lceil \log i \rceil}$ und $M^? = M_i^?$ gilt.

Dies ist möglich, da $M^?$ in der Aufzählung beliebig oft vorkommt.

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

1. Fall: $1^i \in L_B = L(M_i^B)$, d.h. M_i^B akzeptiert 1^i innerhalb der Zeitschranke $p(i) \leq i^{\lceil \log i \rceil}$.

Dann gilt $B_{i-1} = B_i$ nach Konstruktion von B und damit $1^i \notin L_B$ nach Definition von L_B. Widerspruch!

2. Fall: $1^i \notin L_B = L(M_i^B)$, d.h. M_i^B verwirft 1^i innerhalb der Zeitschranke $p(i) \leq i^{\lceil \log i \rceil}$.

Ist $\{0,1\}^i \setminus X_i \neq \emptyset$, so existiert ein $b_i \in B$ mit $|b_i| = i$ und damit $1^i \in L_B$. Widerspruch!

Es ist daher noch zu zeigen, dass für alle zuvor genügend groß gewählten i gilt: $\{0,1\}^i \setminus X_i \neq \emptyset$

Bei der Definition von X_i bis zum i-ten Schritt wurden maximal

$$\sum_{j=1}^{i} j^{\lceil \log j \rceil} \leq i \cdot i^{\lceil \log i \rceil} \leq 2^{\log i + \lceil \log i \rceil^2}$$

Orakelanfragen gestellt.

Eine Welt wo $\mathbf{P} \neq \mathbf{NP}$

X_i enthält also maximal $2^{\log i + \lceil \log i \rceil^2}$ Wörter der Länge $\leq i$.

Für alle zuvor genügend groß gewählten i gilt nun

$$2^{\log i + \lceil \log i \rceil^2} < 2^i,$$

weshalb $\{0,1\}^i \setminus X_i$ nicht leer ist. □

Man kann sogar zeigen, dass die Menge aller Orakel $B \subseteq \{0,1\}^*$ mit $\mathbf{P}^B = \mathbf{NP}^B$ Maß 0 innerhalb der Menge aller Sprachen über $\{0,1\}$ hat (Bennett, Gill, 1981).

Monotone Schaltkreise und der Satz von Razborov

Sei $C = (C_n)_{n \geq 0}$ eine Familie von booleschen Schaltkreisen, wobei C_n genau n Eingabegatter $x_1, \ldots, x_n$ hat.

Dann berechnet C_n eine boolesche Funktion $f_n : \{0,1\}^n \to \{0,1\}$ und C definiert die Sprache $L(C) = \bigcup_{n \geq 0} \{w \in \{0,1\}^n \mid f_n(w) = 1\}$.

Die Familie $C = (C_n)_{n \geq 0}$ ist polynomiell, falls es ein Polynom $p(n)$ gibt, so dass C_n nur höchstens $p(n)$ viele Gatter hat.

Aus der Konstruktion im Satz von Cook (bzw. der **P**-Vollständigkeit von Circuit Value) folgt sehr einfach, dass es zu jeder Sprache $L \in \mathbf{P}$ eine polynomielle Familie $C = (C_n)_{n \geq 0}$ von Schaltkreisen gibt mit $L = L(C)$.

Eine Strategie zum Beweis von $\mathbf{P} \neq \mathbf{NP}$ könnte also sein:
Zeige für eine **NP**-vollständige Sprache L, dass es keine polynomielle Familie $C = (C_n)_{n \geq 0}$ von Schaltkreisen gibt mit $L = L(C)$.

Beachte: Es gibt nicht-entscheidbare Sprachen mit polynomiellen Schaltkreisen.

Monotone Schaltkreise und der Satz von Razborov

Es ist jedoch leider extrem schwierig für ein konkretes Problem (Sprache) eine exponentielle Schranke für die Größe von Schaltkreisen zu zeigen.

Razborov gelang dies für das **NP**-vollständige Problem CLIQUE unter der Einschränkung auf monotone Schaltkreise.

Eine boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt monoton, falls $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ gilt, sofern $x_i \leq y_i$ für alle $1 \leq i \leq n$.

D.h., durch einen Wechsel von 0 nach 1 in der Eingabe kann die Ausgabe nicht auf 0 zurückfallen.

Ein Schaltkreis heißt monoton, falls er keine NOT-Gatter besitzt.

Ein monotoner Schaltkreis berechnet eine monotone Funktion.

Umgekehrt gibt es zu jeder monotonen Funktion $f : \{0,1\}^n \to \{0,1\}$ (mit $n \geq 1$) einen monotonen Schaltkreis mit höchstens $2^n + 2^{n-1} - 2$ Gattern, der f berechnet.

Monotone Schaltkreise und der Satz von Razborov

Das folgende Problem CLIQUE ist **NP**-vollständig (Übung):

Eingabe: ungerichteter Graph $G = (V, E)$ und ein $k \geq 2$.

Frage: Gibt es in G eine Clique der Größe k, d.h. gibt es $C \subseteq V$ mit: $|C| \geq k$ und $\{u, v\} \in E$ für alle $u, v \in C$ mit $u \neq v$.

Ein ungerichteter Graph $G = (\{1, \ldots, n\}, E)$ wird im folgenden kodiert durch Bits $g_{i,j} \in \{0, 1\}$ mit: $g_{i,j} = 1 \iff \{i, j\} \in E$.

Sei $\mathrm{cl}_{n,k}$ die folgende monotone boolesche Funktion:

$\mathrm{cl}_{n,k}((g_{i,j})_{1 \leq i < j \leq n}) = 1 \iff$ der durch $(g_{i,j})_{1 \leq i < j \leq n}$ definierte Graph hat eine Clique der Größe k

Satz (Razborov, 1985)

Es gibt eine Konstante $c_0 > 0$, sodass für jedes genügend große n und $k = \sqrt[4]{n}$ jeder monotone Schaltkreis für $\mathrm{cl}_{n,k}$ mindestens $r = n^{c_0 \cdot \sqrt[8]{n}}$ viele Gatter besitzt.

Sonnenblumen

Für den Beweis des Satzes von Razborov benötigen wir eine Eigenschaft von Sonnenblumen.

Definition

Eine Sonnenblume der Größe $p \geq 2$ mit Kern Z ist eine Menge $\{X_1, \ldots, X_p\}$ von p Mengen, für die gilt: $X_i \cap X_j = Z$ für alle $1 \leq i < j \leq p$. Die Mengen $X_i \setminus Z$ bezeichnen wir als Blütenblätter.

Lemma (Erdős, Rado)

Sei $p \geq 2$, $\ell \geq 1$ und $\mathcal{F} = \{X_1, \ldots, X_m\}$ eine Menge von m verschiedenen Mengen mit $m > (p-1)^\ell \cdot \ell!$ und $|X_i| \leq \ell$ für alle $1 \leq i \leq m$. Dann enthält $\mathcal{F}$ eine Sonnenblume der Größe p.

Sonnenblumen

Beweis: Induktion nach ℓ.

IA: $\ell = 1$.

$\rightsquigarrow$ $\mathcal{F}$ enthält mindestens p paarweise disjunkte Mengen der Kardinalität ≤ 1.

Diese bilden eine Sonnenblume der Größe p mit Kern $Z = \emptyset$.

IS: $\ell > 1$.

Sei $\mathcal{D} \subseteq \mathcal{F}$ eine maximale Teilmenge paarweise disjunkter Mengen.

Fall 1. $|\mathcal{D}| \geq p$.

Wähle analog zu $\ell = 1$ den Kern $Z = \emptyset$.

Sonnenblumen

Fall 2. $|\mathcal{D}| \leq p - 1$.

Dann gilt $|D| \leq (p-1) \cdot \ell$ für $D = \bigcup\{X \mid X \in \mathcal{D}\}$.

Da $\mathcal{D}$ maximal ist, gilt $\forall X \in \mathcal{F} \setminus \mathcal{D} : X \cap D \neq \emptyset$.

Aus $|\mathcal{F} \setminus \mathcal{D}| \geq m - (p-1)$ und $|D| \leq (p-1)\ell$ folgt, dass es ein $d \in D$ und mindestens $(m-(p-1))/((p-1)\ell) > (p-1)^{\ell-1} \cdot (\ell-1)! - 1/\ell$ viele Mengen $X \in \mathcal{F} \setminus \mathcal{D}$ mit $d \in X$ gibt.

Für eine weitere Menge $D_0 \in \mathcal{D}$ gilt $d \in D_0$.

Also gibt es mehr als $(p-1)^{\ell-1} \cdot (\ell-1)!$ Mengen $X \in \mathcal{F}$ mit $d \in X$.

Sei $\mathcal{F}' = \{X \setminus \{d\} \mid d \in X \in \mathcal{F}\}$.

Nach IH enthält $\mathcal{F}'$ eine Sonnenblume $\{X_1 \setminus \{d\}, \ldots, X_p \setminus \{d\}\}$ der Größe p mit Kern Z'.

Damit ist $\{X_1, \ldots, X_p\}$ eine Sonnenblume der Größe p in $\mathcal{F}$ mit Kern $Z = Z' \cup \{d\}$. □

Beweis des Satzes von Razborov

Annahme

Für jede Konstante $c_0 > 0$ gibt es für unendlich viele n einen monotonen Schaltkreis $C_{n,k}$ für die Funktion $\mathrm{cl}_{n,k}$ (mit $k = \sqrt[4]{n}$), welcher weniger als $r = n^{c_0 \cdot \sqrt[8]{n}}$ viele Gatter besitzt.

Sei $c_0 = 1/5$ im folgenden.

Wir werden für große n die obige Annahme zum Widerspruch führen.

Sei $C_{n,k}$ ein Schaltkreis für $\mathrm{cl}_{n,k}$, so wie in der Annahme gefordert.

Wir legen zunächst einige Parameter fest:

$$k := \sqrt[4]{n} \qquad p := \ell \log n$$
$$\ell := \sqrt[8]{n} \qquad M := (p-1)^{\ell} \cdot \ell!.$$

Der Logarithmus wird dabei zur Basis 2 gebildet und die Werte auf- oder abgerundet (die Rechnung ist genügend robust).

Beweis des Satzes von Razborov

Wir beschriften nun die Gatter von $C_{n,k}$ mit Mengen $\{X_1, \ldots, X_m\}$, wobei $X_i \subseteq \{1, \ldots, n\}$.

Sei $\mathcal{B}_g$ die Beschriftung für ein Gatter g.

Dann definieren wir für einen Graphen $G = (\{1, \ldots, n\}, E)$ den Wert $\mathcal{B}_g(G) \in \{0, 1\}$ durch:

$$\mathcal{B}_g(G) = 1 \quad \Longleftrightarrow \quad \exists X \in \mathcal{B}_g : \binom{X}{2} \subseteq E$$

Hierbei ist $\binom{X}{2} = \{\{u, v\} \mid u, v \in X, u \neq v\}$.

D.h.: $\mathcal{B}_g(G) = 1 \iff \mathcal{B}_g$ enthält eine Clique von G.

Beweis des Satzes von Razborov

Eine Beschriftung $\mathcal{B}$ ist zulässig, falls gilt:
$|\mathcal{B}| \leq M$ und $|X| \leq \ell$ für alle $X \in \mathcal{B}$.

Die folgende reduce-Prozedur wandelt eine beliebige Beschriftung $\mathcal{B}$ in eine zulässige Beschriftung um:

1. Ersetze $\mathcal{B}$ durch $\mathcal{B}' = \{X \in \mathcal{B} \mid |X| \leq \ell\}$.
2. Solange noch $|\mathcal{B}'| > M$ gilt, tue folgendes:

   Suche eine beliebige Sonnenblume $\{X_1, \ldots, X_p\} \subseteq \mathcal{B}'$ mit Kern Z (existiert auf Grund des Lemmas von Erdős und Rado) und ersetze $\{X_1, \ldots, X_p\}$ durch Z.

   Diesen Vorgang bezeichnen wir als Pflücken der Sonnenblume.

Die zulässige Beschriftung, die wir so erhalten, nennen wir reduce($\mathcal{B}$).

Beweis des Satzes von Razborov

Nun definieren wir eine Beschriftung der Gatter von $C_{n,k}$ induktiv:

1. Für ein Eingabegatter g_{ij} setzen wir $\mathcal{B}_{g_{ij}} = \{\{i, j\}\}$.

   Dies ist eine zulässige Beschriftung, falls n genügend groß ist.

2. Sei g ein Gatter mit den Eingangsgattern f und h.

   Induktiv seien f mit $\mathcal{B}_f$ und h mit $\mathcal{B}_h$ jeweils zulässig beschriftet.

   Die Beschriftung $\mathcal{B}_g$ von g definieren wir als

$$\mathcal{B}_g = \begin{cases} \text{reduce}(\mathcal{B}_f \cup \mathcal{B}_h), & \text{falls } g = f \vee h \\ \text{reduce}(\mathcal{B}_f \cdot \mathcal{B}_h), & \text{falls } g = f \wedge h \end{cases}$$

   Dabei ist $\mathcal{B}_f \cdot \mathcal{B}_h = \{X \cup Y \mid X \in \mathcal{B}_f, Y \in \mathcal{B}_h\}$.

Am Ausgangsgatter *out* von $C_{n,k}$ erhalten wir so eine zulässige Beschriftung $\mathcal{B}_{out}$.

Beweis des Satzes von Razborov

Behauptung 1: $\mathcal{B}_{out} \neq \emptyset$, falls n genügend groß ist.

Beweis der Behauptung 1:

Für $P \subseteq \{1, \ldots, n\}$ mit $|P| = k$ definieren wir den positiven Testgraphen

$$G_P = (\{1, \ldots, n\}, \binom{P}{2}).$$

Also gilt $C_{n,k}(G_P) = 1$, da P eine Clique der Größe k ist.

Es gibt $\binom{n}{k}$ viele positive Testgraphen G_P.

Wir zeigen, dass mindestens ein $P \subseteq \{1, \ldots, n\}$ mit $|P| = k$ und $\mathcal{B}_{out}(G_P) = 1$ existiert.

Dies impliziert dann $\mathcal{B}_{out} \neq \emptyset$.

Um einen Widerspruch abzuleiten, nehmen wir an:

$$\forall P \subseteq \{1, \ldots, n\} \text{ mit } |P| = k : \mathcal{B}_{out}(G_P) = 0 \neq 1 = C_{n,k}(G_P).$$

Beweis des Satzes von Razborov

Wir stellen uns die Gatter von $C_{n,k}$ nun so aufgelistet vor, dass ein Gatter g stets nach seinen Eingangsgattern h und f kommt.

Mit $C_{n,k}(g, G_P)$ bezeichnen wir den Wert des Gatters g in $C_{n,k}$ bei Eingabe G_P.

Für jede Teilmenge $P \subseteq \{1, \ldots, n\}$ mit $|P| = k$ gibt es ein kleinstes Gatter g (bzgl. unserer Auflistung) mit

$$\mathcal{B}_g(G_P) = 0 \quad \text{und} \quad C_{n,k}(g, G_P) = 1.$$

Dieses kann kein Eingabegatter sein, da für g_{ij} gilt:

$$\mathcal{B}_{g_{ij}}(G_P) = C_{n,k}(g_{ij}, G_P).$$

Seien f und h die Eingangsgatter von g.

Also gilt: $\mathcal{B}_f(G_P) \geq C_{n,k}(f, G_P)$ und $\mathcal{B}_h(G_P) \geq C_{n,k}(h, G_P)$

Beweis des Satzes von Razborov

Angenommen, g ist ein OR-Gatter.

$\rightsquigarrow\ 1 = C_{n,k}(g, G_P) = C_{n,k}(f, G_P) \vee C_{n,k}(h, G_P)$.

Gelte o.B.d.A. $C_{n,k}(f, G_P) = 1$.

Mit $\mathcal{B}_f(G_P) \geq C_{n,k}(f, G_P) = 1$ folgt $\mathcal{B}_f(G_P) = 1$.

Sei etwa $X \in \mathcal{B}_f$ und $X \subseteq P$.

$\rightsquigarrow\ X \in \mathcal{B}_f \cup \mathcal{B}_h$

$\rightsquigarrow\ \exists X' \in \mathcal{B}_g = \mathrm{reduce}(\mathcal{B}_f \cup \mathcal{B}_h) : X' \subseteq X \subseteq P$.

$\rightsquigarrow\ \mathcal{B}_g(G_P) = 1$ — ein Widerspruch.

Also muss g ein AND-Gatter sein.

Aus $C_{n,k}(g, G_P) = 1$ folgt $C_{n,k}(f, G_P) = C_{n,k}(h, G_P) = 1$.

$\rightsquigarrow\ \mathcal{B}_f(G_P) = \mathcal{B}_h(G_P) = 1$.

Sei etwa $X \in \mathcal{B}_f$, $Y \in \mathcal{B}_h$, $X \subseteq P$ und $Y \subseteq P$.

$\rightsquigarrow\ X \cup Y \subseteq P$ und $X \cup Y \in \mathcal{B}_f \cdot \mathcal{B}_h$.

Beweis des Satzes von Razborov

Wegen $\mathcal{B}_g(G_P) = 0$ kann keine Teilmenge von $X \cup Y$ in $\mathcal{B}_g$ sein.

Also gilt $|X \cup Y| \geq \ell + 1$ (sonst wäre nach jedem Pflücken einer Sonnenblume noch eine Teilmenge von $X \cup Y$ in der aktuellen Beschriftung).

Wir haben somit gezeigt: Wenn g das kleinste Gatter mit $\mathcal{B}_g(G_P) = 0$ und $C_{n,k}(g, G_P) = 1$ ist, dann existiert $Z \in \mathcal{B}_f \cdot \mathcal{B}_h$ mit $Z \subseteq P$ und $|Z| \geq \ell + 1$.

Wir beschriften nun jedes AND-Gatter g mit allen k-elementigen Teilmengen $P \subseteq \{1, \ldots, n\}$, so dass g das kleinste Gatter mit $\mathcal{B}_g(G_P) = 0$ und $C_{n,k}(g, G_P) = 1$ ist.

Dann wird jedes Gatter nur mit $\leq M^2 \cdot \binom{n-\ell-1}{k-\ell-1}$ viele k-elementige Teilmengen P beschriftet, denn:

- $\mathcal{B}_f \cdot \mathcal{B}_h$ enthält $\leq M^2$ viele Mengen mit $\geq \ell + 1$ vielen Elementen und
- jede Teilmenge $Z \subseteq \{1, \ldots, n\}$ mit $|Z| \geq \ell + 1$ kommt in nur

$$\binom{n - |Z|}{k - |Z|} \leq \binom{n - \ell - 1}{k - \ell - 1}$$

vielen k-elememtigen Teilmengen $P \subseteq \{1, \ldots, n\}$ vor.

Beweis des Satzes von Razborov

Andererseits kommt jede k-elementige Teilmenge P als Beschriftung eines Gatters vor. Da $C_{n,k}$ weniger als r Gatter hat, folgt:

$$r \cdot M^2 \cdot \binom{n-\ell-1}{k-\ell-1} > (\text{Anzahl Gatter von } C_{n,k}) \cdot \binom{n-\ell-1}{k-\ell-1} \geq \binom{n}{k}.$$

Also gilt:

$$\begin{aligned} r &> \frac{1}{M^2} \cdot \frac{\binom{n}{k}}{\binom{n-\ell-1}{k-\ell-1}} = \frac{1}{M^2} \cdot \frac{n! \cdot (k-\ell-1)!}{k! \cdot (n-\ell-1)!} \\ &= \frac{1}{M^2} \cdot \frac{n(n-1)\cdots(n-\ell)}{k(k-1)\cdots(k-\ell)} \geq \frac{1}{M^2} \cdot \left(\frac{n-\ell}{k}\right)^\ell \\ &= \frac{1}{(p-1)^{2\ell} \cdot \ell!^2} \cdot \left(\frac{n-\ell}{k}\right)^\ell \geq \left(\frac{n-\ell}{p^2 \cdot \ell^2 \cdot k}\right)^\ell \geq n^{1/5 \cdot \ell} = n^{c_0 \sqrt[8]{n}}, \end{aligned}$$

wobei die letzte Ungleichung gilt, falls n genügend groß ist.

Dies ist jedoch ein Widerspruch, was Behauptung 1 beweist.

Beweis des Satzes von Razborov

Für die weitere Argumentation benötigen wir negative Testgraphen.

Eine Färbung $c : \{1, \ldots, n\} \to \{1, \ldots, k-1\}$ der Knotenmenge definiert den Graphen $G_c = (\{1, \ldots, n\}, E)$ mit $E = \{\{i, j\} \mid c(i) \neq c(j)\}$.

An den Eingabegattern gilt: $C_{n,k}(g_{ij}, G_c) = 1 \iff c(i) \neq c(j)$.

Der Graph G_c ist $(k-1)$-färbbar, also gilt $C_{n,k}(G_c) = 0$.

Es gibt $(k-1)^n$ Färbungen, aber sehr viel weniger negative Testgraphen.

Sei $c : \{1, \ldots, n\} \to \{1, \ldots, k-1\}$ im folgenden eine zufällig und gleichverteilt gewählte Färbung.

Für $Z \subseteq \{1, \ldots, n\}$ sei $R(Z)$ das Ereignis, dass zwei Knoten aus Z gleich gefärbt werden (repeated color):

$$R(Z) \iff \exists i, j \in Z : i \neq j \wedge c(i) = c(j)$$

Beweis des Satzes von Razborov

Für eine Beschriftung $\mathcal{B}$ gilt damit:

$$\mathcal{B}(G_c) = 1 \iff \mathcal{B} \text{ enthält eine Clique von } G_c \iff \exists Z \in \mathcal{B} : \neg R(Z).$$

Behauptung 2: Sei $\mathcal{B} \neq \emptyset$ eine zulässige Beschriftung.
Dann gilt $\Pr(\mathcal{B}(G_c) = 1) \geq 1/2$.

Beweis von Behauptung 2:

Für $i, j \in \{1, \ldots, n\}$ mit $i \neq j$ gilt $\Pr(c(i) = c(j)) = 1/(k-1)$.

Sei $Z \in \mathcal{B}$ mit $|Z| \leq \ell$. Dann gilt

$$\Pr(R(Z)) \leq \binom{|Z|}{2} \cdot \frac{1}{k-1} \leq \frac{\ell(\ell-1)}{2(k-1)} = \frac{\ell}{2(\ell+1)} \leq \frac{1}{2}.$$

Dies beweist Behauptung 2.

Beweis des Satzes von Razborov

Beachte:

- Aus Behauptung 1 ($\mathcal{B}_{out} \neq \emptyset$) und 2 folgt $\Pr(\mathcal{B}_{out}(G_c) = 1) \geq 1/2$.
- Andererseits gilt $C_{n,k}(G_c) = 0$ für alle Färbungen c.

Behauptung 3: Sei $\{Z_1, \ldots, Z_p\}$ eine Sonnenblume mit Kern Z und $|Z_i| \leq \ell$ für alle $1 \leq i \leq p$. Dann gilt

$$\Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)) \leq (1/2)^p.$$

Beweis von Behauptung 3: Wegen

$$\Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)) = \Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \mid \neg R(Z)) \cdot \Pr(\neg R(Z))$$

gilt $\Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)) \leq \Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \mid \neg R(Z))$.

Beweis des Satzes von Razborov

Beachte: Unter der Voraussetzung $\neg R(Z)$ sind die Ereignisse $R(Z_1), \ldots, R(Z_p)$ unabhängig, da die Mengen $Z_1 \setminus Z, \ldots, Z_p \setminus Z$ paarweise disjunkt sind.

Also gilt

$$\begin{aligned}
\Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \wedge \neg R(Z)) &\leq \Pr(R(Z_1) \wedge \cdots \wedge R(Z_p) \mid \neg R(Z)) \\
&= \prod_{i=1}^{p} \Pr(R(Z_i) \mid \neg R(Z)) \\
&\leq \prod_{i=1}^{p} \Pr(R(Z_i)) \\
&\leq (1/2)^p,
\end{aligned}$$

denn $\Pr(R(Z_i)) \leq 1/2$, falls $|Z_i| \leq \ell$ (siehe Beweis von Behauptung 2).

Dies beweist Behauptung 3.

Beweis des Satzes von Razborov

Wir wissen bereits, dass für mindestens die Hälfte aller Färbungen c gilt: $\mathcal{B}_{out}(G_c) = 1 \neq 0 = C_{n,k}(G_c)$.

Sei c eine Färbung mit $\mathcal{B}_{out}(G_c) = 1$ und $C_{n,k}(G_c) = 0$.

Sei g das kleinste Gatter mit $\mathcal{B}_g(G_c) = 1$ und $C_{n,k}(g, G_c) = 0$.

Dann kann g kein Eingabegatter sein.

Seien also f und h die Eingabegatter von g.

Also gilt $\mathcal{B}_f(G_c) \leq C_{n,k}(f, G_c)$ und $\mathcal{B}_h(G_c) \leq C_{n,k}(h, G_c)$.

Fall 1: g ist ein AND-Gatter.

$\rightsquigarrow\ 0 = C_{n,k}(g, G_c) = C_{n,k}(f, G_c) \wedge C_{n,k}(h, G_c)$.

Gelte o.B.d.A. $C_{n,k}(f, G_c) = 0$.

Mit $\mathcal{B}_f(G_c) \leq C_{n,k}(f, G_c)$ folgt $\mathcal{B}_f(G_c) = 0$.

Also ist keine Menge aus $\mathcal{B}_f$ eine Clique von G_c, d. h. $\forall X \in \mathcal{B}_f : R(X)$.

Also gilt für die Menge $\mathcal{B}'_g = \mathcal{B}_f \cdot \mathcal{B}_h$: $\forall Z \in \mathcal{B}'_g : R(Z)$.

Beweis des Satzes von Razborov

Fall 2: g ist ein OR-Gatter.

$\rightsquigarrow\ 0 = C_{n,k}(g, G_c) = C_{n,k}(f, G_c) \vee C_{n,k}(h, G_c).$

$\rightsquigarrow\ C_{n,k}(f, G_c) = C_{n,k}(h, G_c) = 0.$

$\rightsquigarrow\ \mathcal{B}_f(G_c) = \mathcal{B}_h(G_c) = 0$

$\rightsquigarrow\ \forall X \in \mathcal{B}_f : R(X)$ und $\forall Y \in \mathcal{B}_h : R(Y)$

Also gilt für die Menge $\mathcal{B}'_g = \mathcal{B}_f \cup \mathcal{B}_h$: $\forall Z \in \mathcal{B}'_g : R(Z)$.

In beiden Fällen gilt also $\forall Z \in \mathcal{B}'_g : R(Z)$.

Nun entsteht die Beschriftung $\mathcal{B}_g$ aus $\mathcal{B}'_g$ durch folgende 2 Schritte:

(1) Entfernen aller Mengen mit mehr als ℓ Elementen.

(2) Wiederholtes Pflücken von Sonnenblumen.

Nach Schritt (1) gilt immer noch $R(Z)$ für alle Mengen in der Beschriftung.

Beweis des Satzes von Razborov

Da am Ende aber $\mathcal{B}_g(G_c) = 1$ gilt, muss durch eine der Pflückungen eine Clique von G_c in die Beschriftung kommen.

Insgesamt gibt es am Gatter g nur maximal $M^2/(p-1)$ viele Pflückungen ($\mathcal{B}'_g$ enthält $\leq M^2$ viele Elemente und jede Pflückung reduziert die Anzahl der Mengen in der Beschriftung um $p-1$).

Betrachte die erste Pflückung bei der eine Clique in die Beschriftung von g kommt.

Angenommen die Sonnenblume $\{Z_1, \ldots, Z_p\}$ wird dabei durch den Kern Z ersetzt, d. h.:

- Der Kern Z ist eine Clique von G_c: $\neg R(Z)$
- $Z_1, \ldots, Z_p$ sind keine Cilquen von G_c: $R(Z_1), \ldots, R(Z_p)$

Nach Behauptung 3 ($\Pr(R(Z_1) \wedge \ldots \wedge R(Z_p) \wedge \neg R(Z)) \leq (1/2)^p$) kann dies nur für höchstens $(1/2)^p \cdot (k-1)^n$ viele Färbungen c gelten.

Beweis des Satzes von Razborov

Da es weniger als r Gatter in $C_{n,k}$ gibt, erhalten wir

$$\begin{aligned}\frac{1}{2}(k-1)^n &\leq \text{Anzahl der Färbungen } c \text{ mit } \mathcal{B}_{out}(G_c) = 1 \\ &< (\frac{1}{2})^p \cdot (k-1)^n \cdot \frac{M^2}{p-1} \cdot r\end{aligned}$$

d. h. $2^{p-1} < M^2 \cdot r/(p-1)$.

Dies bedeutet für genügend große n:

$$\begin{aligned}r &> \frac{p-1}{M^2} \cdot 2^{p-1} = \frac{(p-1) \cdot 2^{p-1}}{((p-1)^\ell \cdot (\ell!))^2} \geq \frac{2^p}{(p \cdot \ell)^{2\ell}} \\ &= \frac{n^\ell}{(p \cdot \ell)^{2\ell}} = \left(\frac{n}{p^2 \cdot \ell^2}\right)^\ell = \left(\frac{\sqrt{n}}{\log^2 n}\right)^\ell \geq n^{1/5\ell} = n^{c_0 \sqrt[8]{n}}.\end{aligned}$$

Dies ist erneut ein Widerspruch. Damit ist der Satz von Razborov bewiesen. □

Beweis des Satzes von Razborov

Razborovs Resultat für CLIQUE hat ernsthafte Hoffnungen auf einen Beweis für $\mathbf{P} \neq \mathbf{NP}$ erzeugt.

Insbesondere wurde die folgende Vermutung aufgestellt:

Vermutung

Jede monotone Sprache $L \subseteq \{0,1\}^*$ in $\mathbf{P}$ kann durch eine polynomielle Familie von monotonen Schaltkreisen entschieden werden.

Zusammen mit Razborovs Resultat würde dies CLIQUE $\notin \mathbf{P}$ und damit $\mathbf{P} \neq \mathbf{NP}$ beweisen.

Leider ist die obige Vermutung falsch!

MATCHING ist das Problem, ob ein gegebener bipartiter Graph ein perfektes Matching hat. Dieses Problem liegt in $\mathbf{P}$ und ist monton.

Razborov selbst konnte seine Techniken für CLIQUE erweitern und zeigen, dass MATCHING nicht durch eine polynomielle Familie von monotonen Schaltkreisen entschieden werden kann.

Randomisierte Turingmaschinen

Definition

Eine randomisierte Turingmaschine (RTM) ist eine deterministische Turingmaschine M mit einem zusätzlichen Eingabeband (das Zufallsband), welches mit einem nach rechts unendlichen String über dem Alphabet $\{0,1\}$ gefüllt ist.

M ist $t(n)$-zeitbeschränkt, falls M bei Eingabe x und jeder Belegung des Zufallsbandes nach höchstens $t(|x|)$ Schritten terminiert.

Wir können dann das Zufallsband auf die Länge $t(|x|)$ einschränken.

Dann akzepiert M eine Eingabe x mit der Wahrscheinlichkeit

$$\operatorname{Prob}_{r \in \{0,1\}^{t(|x|)}} [M \text{ akzeptiert bei Eingabe x und Zufallsbandbelegung } r].$$

Interaktive Beweissysteme

Definition (Goldwasser, Micali, Rackoff 1985)

Ein interaktives Beweissystem (IBS) ist ein Paar (A, B) mit:

- A (Alice) ist eine beliebige Funktion $A : \bigcup_{i \geq 0} (\{0,1\}^*)^{1+2i} \to \{0,1\}^*$.
- B (Bob) ist eine randomisierte Turingmaschine mit Ausgabe, dessen Eingabe von der Form $(x, \ldots)$ ($\ldots$ steht für eine Folge von Bitstrings) ist. Bobs Rechenzeit ist durch $q(|x|)$ für ein Polynom $q(n)$ beschränkt.
- Alice und Bob erhalten die gleiche Eingabe x und kommunizieren über ein gemeinsames Arbeitsband (Kommunikationsband) in $p(|x|)$ vielen Runden für ein Polynom $p(n)$.
- Runde i beginnt mit der Botschaft $a_i = A(x, a_1, b_1, \ldots, a_{i-1}, b_{i-1})$ von Alice an Bob, die auf das Kommunikationsband geschrieben wird.
- Danach sendet Bob $b_i = B(x, a_1, b_1, \ldots, a_{i-1}, b_{i-1}, a_i, r_1, \ldots, r_i)$ als Antwort an Alice. Hierbei ist r_i der Zufallsstring für Runde i.
- In der letzten Runde $p(|x|)$ entscheidet Bob, ob die Eingabe x akzeptiert oder abgelehnt wird.

Bemerkungen:

- Da Bob in Polynomialzeit arbeitet, sind Bobs Nachrichten b_i an Alice automatisch polynomiell in der Länge von x beschränkt. Für Alice muss dies nicht unbedingt gelten.

  Aber: Ist Bob $q(n)$-zeitbeschränkt (für ein Polynom $q(n)$), so kann Bob nur die ersten $q(|x|)$ vielen Bits jeder Nachricht a_i von Alice an Bob lesen.

  Also können wir o.B.d.A. verlangen, dass $|a_i| \leq q(|x|)$ gilt. Wir setzen dies im folgenden stets voraus.

- Bob teilt seine Zufallsbits r_i nicht Alice mit (private coins).

- Der Wahrscheinlichkeitsraum wird durch die Zufallsstrings $r_1, r_2, \ldots, r_{p(|x|)}$ erzeugt. Diese befinden sich zu Beginn auf dem Zufallsband. In Runde i hat Bob Zugriff auf das Anfangsstück $r_1, r_2, \ldots, r_i$.

Interaktive Beweissysteme und die Klasse **IP**

Definition

Ein IBS (A, B) entscheidet eine Sprache $L \subseteq \Sigma^*$, falls für alle Eingaben $x \in \Sigma^*$ gilt:

- Ist $x \in L$, so akzeptiert Bob die Eingabe mit einer Wahrscheinlichkeit $\geq (1 - 2^{-|x|})$.
- Ist $x \notin L$, so gibt es kein IBS (A', B) in dem Bob die Eingabe mit einer höheren Wahrscheinlichkeit als $2^{-|x|}$ akzeptiert.

IP ist die Menge der Sprachen, die durch ein IBS entschieden werden können:

$$\mathbf{IP} \quad = \quad \{L \subseteq \Sigma^* \mid \exists \text{ IBS } (A, B) : (A, B) \text{ entscheidet } L\}$$

Intuition zu interaktiven Beweissystemen

- Bob wird häufig auch als der Verifizierer bezeichnet, Alice als der Beweiser.
- Mittels eines Kommunikationsprotokolls soll herausgefunden werden, ob $x \in L$ oder nicht.
- Vollständigkeit: Wenn $x \in L$, dann überzeugt ein Beweiser, der sich an das Protokoll hält, den Verifizierer mit hoher Wahrscheinlichkeit $(1 - 2^{-|x|})$.
- Korrektheit: Wenn $x \notin L$, dann überzeugt jeder Beweiser (auch solche, die sich nicht an das Protokoll halten), den Verifizierer nur mit geringer Wahrscheinlichkeit $(2^{-|x|})$.

Die Klasse **IP**

Bemerkung: NP $\subseteq$ **IP**: Alice kann die erfolgreiche Rechnung einer **NP**-Maschine Bob zur Überprüfung vorlegen.

Für ein konkretes Problem aus **NP** kann Alice auch eine Lösung angeben: Für SAT übermittelt Alice beispielsweise eine erfüllende Belegung an Bob. Bob wertet die Formel unter dieser Belegung aus und akzeptiert, wenn sich als Wert der Formel „wahr“ ergibt.

Lemma

IP ist unter polynomialer Zeitreduktion abgeschlossen:

$$L \leq_m^p L' \text{ und } L' \in \mathbf{IP} \;\Rightarrow\; L \in \mathbf{IP}$$

Beweis: Gelte $L \leq_m^p L' \in \mathbf{IP}$ und sei f die Reduktion von L auf L'.

Dann berechnen sowohl Bob und Alice bei Eingabe x zunächst $f(x)$ und lassen dann das Protokoll für L' laufen. □

Beispiel: Nicht-Isomorphie von Graphen

Das Problem, ob zwei ungerichtete Graphen isomorph sind, liegt in **NP**.

Es ist aber weder bekannt, ob es in **P** liegt, noch, ob es **NP**-vollständig ist.

Es gibt allerdings Hinweise, die darauf hindeuten, dass dieses Problem nicht **NP**-schwierig ist.

Es ist nicht bekannt, ob das Problem der Nicht-Isomorphie von Graphen in **NP** liegt.

Satz (Goldreich, Micali, Wigderson 1987)

Nicht-Isomorphie von Graphen gehört zu **IP**.

Beispiel: Nicht-Isomorphie von Graphen

Beweis:

Die Eingabe besteht aus 2 Graphen $G_0 = (V_0, E_0)$ und $G_1 = (V_1, E_1)$. o.B.d.A. $V_0 = V_1 = \{1, \ldots, n\}$ (wenn $|V_0| \neq |V_1|$, dann kann Bob sofort akzeptieren).

Das Protokoll zwischen Alice und Bob arbeitet wie folgt:

Bob wählt zunächst m Permutationen $\pi_i \in \mathrm{Perm}(\{1, \ldots, n\})$ und m Bits $b_i \in \{0, 1\}$ zufällig ($1 \leq i \leq m$).

Dann übermittelt Bob an Alice in einer einzigen Runde die Liste $(\pi_1(G_{b_1}), \ldots, \pi_m(G_{b_m}))$.

Bob verlangt von Alice jetzt einen Bitstring $b'_1 b'_2 \cdots b'_m$ und akzeptiert, falls $b_i = b'_i$ für alle $i \in \{1, \ldots, m\}$.

Beispiel: Nicht-Isomorphie von Graphen

Fall 1: $G_0 \not\cong G_1$.

Dann kann Alice den Bitstring $b_1 b_2 \cdots b_m$ aus $(\pi_1(G_{b_1}), \ldots, \pi_m(G_{b_m}))$ rekonstruieren.

Also wird Bob mit Wahrscheinlichkeit 1 akzeptieren.

Fall 2: $G_0 \cong G_1$.

In diesem Fall sieht Alice lediglich eine Liste von m vielen Zufallspermutation von G_0.

Alice muss also einen von Bob zufällig gewählten und ihr nicht bekannten String der Länge m "erraten".

Die Wahrscheinlichkeit, dass ihr das gelingt, ist nur 2^{-m}, d. h. Bobs Antwort lautet (für jede Alice) mit Wahrscheinlichket $1 - 2^{-m}$ "Isomorph". □

Zero-Knowledge-Beweise

Das obige Protokoll für Nicht-Isomorphie von Graphen hat eine interessante zusätzliche Eigenschaft:

Falls $G_0 \cong G_1$, so teilt Alice Bob keinen Isomorphismus zwischen G_0 und G_1 mit. Mehr noch, Bob erhält keinerlei Information (zero-knowledge) über solch einen Isomorphismus.

Solche Beweissysteme nennt man auch Zero-Knowledge-Beweissysteme, sie spielen in der Kryptographie eine wichtige Rolle.

IP $\subseteq$ PSPACE

Satz

IP $\subseteq$ PSPACE

Beweis: Sei $L \in$ **IP**.

Fixiere eine fest gewählte randomisierte polynomiell zeitbeschränkte Turingmaschine B (Bob) und eine Eingabe $x \in \Sigma^*$ mit $|x| = n$.

Sei $p(n)$ das Polynom, dass die Anzahl der Runden in dem Protokoll bestimmt.

O.B.d.A sei $q(n)$ ein Polynom, so dass alle Nachrichten von Alice und Bob $(a_1, \ldots, a_{p(n)})$ und umgekehrt $(b_1, \ldots, b_{p(n)})$ sowie alle Zufallsstrings $r_1, \ldots, r_{p(n)}$ Länge genau $q(n)$ haben.

Eine mögliche Alice ist dann gegeben durch eine Funktion

$$A : \bigcup_{i=0}^{p(n)-1} \{0,1\}^{n+i\cdot 2q(n)} \to \{0,1\}^{q(n)}.$$

IP $\subseteq$ PSPACE

Wir konstruieren nun eine optimale Alice in polynomiellen Platz.

Beachte: Legt man eine mögliche Alice A sowie das Tupel der Zufallsstrings $r = (r_1, \ldots, r_{p(n)}) \in \{0,1\}^{p(n)q(n)}$ fest, so ist dadurch die Folge der ausgetauschten Nachrichten

$$P(A, r) = a_1 b_1 \cdots a_{p(n)} b_{p(n)}$$

(das zu A und r gehörende Protokoll) eindeutig bestimmt.

Falls Bob am Ende des Protokolls $P(A, r)$ die Eingabe x akzeptiert, so sagen wir, dass (A, r) akzeptierend ist.

Ein Protokollpräfix ist ein Tupel $P \in \{0,1\}^{i \cdot q(n)}$ mit $0 \leq i \leq 2p(n)$.

(A, r) passt zu diesem Protokollpräfix P, falls P ein Anfangsstück von $P(A, r)$ ist.

IP $\subseteq$ PSPACE

Für eine mögliche Alice A und den Protokollpräfix P definieren wir

$$f(A,P) = |\{r \in \{0,1\}^{p(n)q(n)} \mid (A,r) \text{ ist akzeptierend und passt zu } P\}|.$$

Eine mögliche Alice ist optimal bzgl. P, falls $f(A,P) = f(P)$ für das folgende Maximum:

$$f(P) = \max\{f(A,P) \mid A \text{ ist eine mögliche Alice}\}.$$

Sei λ der leere Protokollpräfix.

Der Quotient $f(\lambda)/2^{p(n)q(n)}$ ist die maximale Wahrscheinlichkeit, mit der die Eingabe x akzeptiert wird, wobei das Maximum über alle möglichen Alice gewählt wird.

Also gilt:

$$\begin{aligned} x \in L &\implies f(\lambda) \geq (1 - 2^{-n}) \cdot 2^{p(n)q(n)} \\ x \notin L &\implies f(\lambda) \leq 2^{-n} \cdot 2^{p(n)q(n)}. \end{aligned}$$

IP $\subseteq$ PSPACE

Ziel: Berechnung von $f(\lambda)$ in polynomiellen Platz.

Wir berechnen die Werte $f(P)$ rekursiv für jeden Protokollpräfix P.

Fall 1: $P = a_1 b_1 \cdots a_{p(n)} b_{p(n)}$ ein vollständiges Protokoll.

Fall 1.1: Bob lehnt am Ende des Protokolls P die Eingabe x ab.

Dann gilt $f(P) = 0$.

Fall 1.2: Bob akzeptiert am Ende des Protokolls P die Eingabe x.

Dann ist

$$f(P) = |\{(r_1, \ldots, r_{p(n)}) \mid r_1, \ldots, r_{p(n)} \in \{0,1\}^{q(n)} \text{ und } \text{für alle } 1 \leq j \leq p(n) \text{ gilt } B(x, a_1, b_1, \ldots, a_{j-1}, b_{j-1}, a_j, r_1, \ldots, r_j) = b_j\}|.$$

Fall 2: Der Protokollpräfix ist von der Form

$$P = a_1 b_1 \cdots a_{i-1} b_{i-1} a_i.$$

mit $1 \leq i \leq p(n)$.

IH: alle Werte $f(Pb_i)$ ($b_i \in \{0,1\}^{q(n)}$) sind bereits bekannt.

Dann gilt:

$$\begin{aligned} f(P) &= \max\{f(A,P) \mid A \text{ ist eine mögliche Alice}\} \\ &\overset{(a)}{=} \max\{\sum_{b_i \in \{0,1\}^{q(n)}} f(A,Pb_i) \mid A \text{ ist eine mögliche Alice}\} \\ &\overset{(b)}{=} \sum_{b_i \in \{0,1\}^{q(n)}} \max\{f(A,Pb_i) \mid A \text{ ist eine mögliche Alice}\} \\ &= \sum_{b_i \in \{0,1\}^{q(n)}} f(Pb_i) \end{aligned}$$

Beachte:

- Eine Wahl von Zufallsstrings $(r_1, \ldots, r_p)$ kann nicht gleichzeitig zu Protokollpräfixen Pb_i und Pb'_i mit $b_i \neq b'_i$ passen.
  Dies begründet (a).
- Sei A_{b_i} eine optimale Alice bezüglich Pb_i.
  Dann erhalten wir eine optimale Alice A für P wie folgt:
  Ist b_i Bob's Nachricht an Alice in Runde i, so verhält sich A im weiteren wie A_{b_i}.
  Dies begründet (b).

Fall 3: Der Protokollpräfix ist von der Form

$$P = (a_1, b_1, \ldots, a_{i-1}, b_{i-1}).$$

mit $1 \leq i \leq p(n)$.

Alice kann nun eine Antwort $a_i \in \{0,1\}^{q(n)}$ wählen.

IH: alle Werte $f(Pa_i)$ sind bereits bekannt.

Eine optimale Alice wird die Antwort a_i so wählen, dass der Wert $f(Pa_i)$ maximal wird.

Daher gilt:

$$f(P) = \max\{f(Pa_i) \mid a_i \in \{0,1\}^{q(n)}\}.$$

Dieser Ansatz führt auf einen rekursiven Algorithmus zur Berechnung von $f(\lambda)$.

IP $\subseteq$ PSPACE

Platzbedarf des rekursiven Algorithmus:

- Die Rekursionstiefe ist $2p(n)$ (polynomiell).
- An lokalen Variablen muss für jeden rekursiven Aufruf ein Protokollpräfix, ein String aus $\{0,1\}^{q(n)}$ (ein a_i bzw. b_i) sowie eine Zahl $\leq 2^{p(n)q(n)}$ (aktueller f-Wert) gespeichert werden.

  Hierfür reicht polynomieller Platz.
- Für einen terminalen Aufruf mit $P = (a_1, b_1, \ldots, a_{p(n)}, b_{p(n)})$ muss für alle Zufallsstrings $r = (r_1, \ldots, r_{p(n)})$ überprüft werden, ob r zu P passt.

  Hierfür muss für alle $1 \leq i \leq p(n)$ die Bedingung

  $$b_i = B(x, a_1, b_1, \ldots, a_i, r_i)$$

  überprüft werden. Dies ist in Polynomialzeit möglich (uns reicht die Aussage, dass dies in polynomiellen Platz möglich ist).

Der Satz von Shamir: **IP** = **PSPACE**

Satz (Shamir, 1990)

IP = **PSPACE**.

Beweis: Noch zu zeigen: **PSPACE** $\subseteq$ **IP**.

Da **IP** unter Polynomialzeitreduktionen abgeschlossen ist und QBF **PSPACE**-vollständig ist, genügt es zu zeigen: QBF $\in$ **IP**.

Wir werden sogar ein IBS konstruieren, in dem Bob seine Zufallszahlen Alice mitteilt und am Ende (mit Wahrscheinlichkeit 1) „ja“ sagt, falls $x \in$ QBF gilt.

Sei als Eingabe eine geschlossene quantifizierte Formel F über $\wedge, \vee, \forall, \exists$ und den Literalen $\tilde{x_1}, \tilde{x_2}, \ldots$ mit $\tilde{x_i} \in \{x_i, \overline{x_i}\}$ gegeben.

O.B.d.A. wird Negation nur auf Variablen und nicht auf größere Teilausdrücke angewendet.

Beweis des Satzes von Shamir (Arithmetisierung)

F kann wie folgt in einen arithmetischen Ausdruck $a(F)$ umgeformt werden: Wir ersetzen

$\overline{x}$	durch	$(1 - x)$
$\vee$	durch	$+$
$\wedge$	durch	$\cdot$ (Multiplikation)
$\forall x$	durch	$\prod_{x \in \{0,1\}}$
$\exists x$	durch	$\sum_{x \in \{0,1\}}$

Mit dieser Übersetzung gilt:

$$F \in \text{QBF} \iff a(F) > 0$$
$$F \notin \text{QBF} \iff a(F) = 0$$

Beweis des Satzes von Shamir (Arithmetisierung)

Beispiel:

Sei $F = \forall x \exists y \Big((x \vee \overline{y}) \wedge \exists z (\overline{x} \wedge z) \Big)$. Dann ist

$$
\begin{aligned}
a(F) &= \prod_{x=0,1} \Big(\sum_{y=0,1} \Big((x + (1-y)) \cdot \sum_{z=0,1} ((1-x) \cdot z) \Big) \Big) \\
&= \prod_{x=0,1} \Big(\sum_{y=0,1} \Big((x + (1-y)) \cdot (1-x) \Big) \Big) \\
&= \prod_{x=0,1} \Big((1-x^2) + (x - x^2) \Big) \\
&= 0
\end{aligned}
$$

Das bedeutet, dass die Auswertung der Formel F „falsch“ ergibt.

Beweis des Satzes von Shamir (Arithmetisierung)

Für die Negation $\neg F = \exists x \forall y \Big((\overline{x} \wedge y) \vee \forall z (x \vee \overline{z}) \Big)$ gilt:

$$
\begin{aligned}
a(\neg F) &= \sum_{x=0,1} \Big(\prod_{y=0,1} \Big((1-x) \cdot y + \prod_{z=0,1} (x+1-z) \Big) \Big) \\
&= \sum_{x=0,1} \prod_{y=0,1} \Big((1-x) \cdot y + x^2 + x \Big) \\
&= \sum_{x=0,1} (x^2 + x) \cdot (1 + x^2) \\
&= 4
\end{aligned}
$$

Die Auswertung der Formel $\neg F$ ergibt also „wahr“.

Beweis des Satzes von Shamir

Es stellt sich die Frage, wie groß die Zahl $a(F)$ werden kann.

Die Länge $|F|$ einer Formel definieren wir induktiv wie folgt:

$$\begin{aligned} |0| = |1| = |x| = |\overline{x}| &= 1 \\ |F \wedge G| = |F \vee G| &= |F| + |G| \\ |\forall x F| = |\exists x F| &= 1 + |F| \end{aligned}$$

Lemma

$a(F) \leq 2^{2^{|F|}}$.

Beweis:

Zunächst eliminieren wir Quantoren, indem wir jedes Vorkommen von $\forall x G$ ($\exists x G$) durch $G_{x=0} \wedge G_{x=1}$ ($G_{x=0} \vee G_{x=1}$) ersetzen.

Sei F' die resultierende Formel.

Durch Induktion über den Aufbau von F zeigt man $|F'| \leq 2^{|F|}$.

Beweis des Satzes von Shamir

Dann zeigt man durch Induktion über den Aufbau einer Quantoren-freien Formel F': $a(F') \leq 2^{|F'|}$. □

Beispiel: Sei $F = \forall x_1 \ldots \forall x_k \exists y \exists z (y \vee z)$. Dann ist

$$
\begin{aligned}
a(F) &= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} \sum_{y=0,1} \sum_{z=0,1} (y+z) \\
&= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} \sum_{y=0,1} (2y+1) \\
&= \prod_{x_1=0,1} \cdots \prod_{x_k=0,1} (4) \\
&= 4^{2^k}
\end{aligned}
$$

Beweis des Satzes von Shamir

Problem: Die Zahl $2^{2^{|F|}}$ benötigt selbst bei binärer Kodierung noch die Länge $2^{|F|}$ und kann deshalb nicht in einem IBS übermittelt werden.

Lösung: Rechnen modulo einer Primzahl.

Lemma

Für $n \geq 5$ gibt es im Intervall $[2^n, 2^{2n}]$ mindestens 2^n Primzahlen.

Beweis: Sei $\pi(n)$ die Anzahl der Primzahlen im Intervall $[2, n]$.

Mit elementaren Methoden kann man zeigen: $\pi(n) \geq (n/\log(n)) - 2$, siehe z. B. das Buch *Primality Testing in Polynomial Time* von M. Dietzfelbinger, Springer 2004.

Also gibt es im Intervall $[2^n, 2^{2n}]$ mindestens

$$(\frac{2^{2n}}{2n} - 2) - (2^n - 2) = 2^n \cdot \left(\frac{2^n}{2n} - 1\right) \geq 2^n$$

viele Primzahlen, falls $n \geq 5$. □

Beweis des Satzes von Shamir

Sei im folgenden $n = |F|$ und seien $p_1, \ldots, p_k$ die Primzahlen zwischen 2^n und 2^{2n}.

Dann gilt:

$$m = p_1 \cdot p_2 \cdot \ldots \cdot p_k \geq (2^n)^{(2^n)} = 2^{n \cdot 2^n} > 2^{2^n}$$

und damit $m > a(F)$, da $a(F) \leq 2^{2^n}$.

Damit gilt:

$$F \notin \text{ QBF} \iff a(F) \equiv 0 \bmod m \quad (\text{da } a(F) = 0)$$
$$F \in \text{ QBF} \iff \exists \text{ Primzahl } p \in [2^n, 2^{2n}] : a(F) \not\equiv 0 \bmod p$$

Die zweite Zeile folgt aus der Tatsache, dass $m = p_1 \cdot p_2 \cdot \ldots \cdot p_k > a(F)$ gilt und $a(F)$ damit nicht alle p_i als Teiler haben kann, falls $a(F) > 0$.

Beweis des Satzes von Shamir

Falls nun $F \in \text{QBF}$ gilt, dann berechnet Alice die kleinste Primzahl $p \geq 2^n$ mit der Eigenschaft $a(F) \not\equiv 0 \bmod p$.

Sie sendet den Wert p zusammen mit dem Beweis dafür, dass p eine Primzahl ist (beachte: PRIM $\in$ **NP**), an Bob.

Beachte: Die Länge der Binärcodierung von p liegt in $\mathcal{O}(n)$.

Alle weiteren Rechnungen werden von nun an modulo p durchgeführt.

Jetzt wird aus einem arithmetischen Ausdruck $a(F)$ ein Polynom in einer Variablen x berechnet (o.B.d.A. beginne die Formel F mit einem Quantor), indem in $a(F)$ das erste Zeichen $\prod$ (bzw. $\sum$) gestrichen wird.

Dieses Polynom ist $a(F'(x))$, falls $F = \forall x F'$ bzw. $F = \exists x F'$.

Problem: Der Grad dieses Polynoms kann exponentiell in n sein.

Lösung: einfach Formeln.

Beweis des Satzes von Shamir (einfache Formeln)

Definition (einfache Formel)

Eine Formel ist einfach, wenn zwischen der Quantifizierung Qx ($Q \in \{\forall, \exists\}$) und jedem Auftreten der Variablen x höchstens ein $\forall$-Quantor auftritt.

Lemma

Jede geschlossene quantifizierte Formel lässt sich in polynomieller Zeit in eine äquivalente einfache Formel umwandeln.

Beweis:

Jede Teilformel $\forall y : G(x_1, \ldots, x_k, y)$ (wobei $x_1, \ldots, x_k, y$ alle freien Variablen in G sind) wird ersetzt durch:

$$\forall y \; \exists y_1 \cdots \exists y_k : \bigwedge_{i=1}^{k} x_i \leftrightarrow y_i \wedge G(y_1, \ldots, y_k, y).$$

Beweis des Satzes von Shamir (einfache Formeln)

Ab jetzt können wir annehmen, dass F mit einem Quantor beginnt, alle Quantoren bis zum rechten Rand binden und F einfach ist (beginnt die ursprüngliche Formel mit einem Quantor, so gilt dies auch für die umgewandelte einfache Formel).

Lemma

Sei $F = QxF'(x)$ $(Q \in \{\exists, \forall\})$ eine einfache geschlossene Formel der Länge n. Dann ist $a(F'(x))$ ein Polynom vom Grad $\leq 2n$.

Beweis:

Wir ersetzen zunächst in $F'(x)$ jede Teilformel der Gestalt $\forall y : G$, in der x frei vorkommt, durch $G_{y=0} \wedge G_{y=1}$.

Dies verdoppelt die Länge der Formel nur, da alle Teilformeln der Gestalt $\forall y : G$, in denen frei x vorkommt, disjunkt zueinander liegen.

Unsere neue (äquivalente) Formel hat also die Länge $2n$.

Beweis des Satzes von Shamir (einfache Formeln)

Es genügt somit zu zeigen: Sei F' eine Formel, so dass keine Teilformel der Gestalt $\forall y : G$ mit x frei in G existiert. Dann ist $a(F')$ ein Polynom in den freien Variablen von F' mit $\text{Grad}_x(a(F')) \leq |F'|$.

Diese Aussage kann durch Induktion über den Aufbau von F' gezeigt werden.

Fall 1: F' ist eine Konstante oder eine Variable oder eine negierte Variable: klar.

Fall 2: $F' = F_1 \vee F_2$. Dann gilt:

$$\text{Grad}_x(a(F')) = \max\{\text{Grad}_x(a(F_1)), \text{Grad}_x(a(F_2))\} \leq \max\{|F_1|, |F_2|\} \leq |F'|.$$

Fall 3: $F' = F_1 \wedge F_2$. Dann gilt:

$$\text{Grad}_x(a(F')) = \text{Grad}_x(a(F_1)) + \text{Grad}_x(a(F_2)) \leq |F_1| + |F_2| = |F'|.$$

Beweis des Satzes von Shamir (einfache Formeln)

Fall 4: $F' = \exists y : F_1$, o.B.d.A $y \neq x$. Dann gilt:

$$\begin{aligned} \mathrm{Grad}_x(a(F')) &= \max\{\mathrm{Grad}_x(a(F_1)_{y=0}), \mathrm{Grad}_x(a(F_1)_{y=1})\} \\ &\leq \mathrm{Grad}_x(a(F_1)) \\ &\leq |F_1| \\ &\leq |F'|. \end{aligned}$$

Fall 5: $F' = \forall y : F_1$.

Dann kann x nicht frei in F_1 vorkommen.

Also gilt $\mathrm{Grad}_x(a(F')) = 0 \leq |F'|$. □

Beweis des Satzes von Shamir: das Protokoll

Erinnerung : Alice hat Bob bereits eine Primzahl $p \geq 2^n$ sowie eine Zertifikat dafür geschickt. Es gilt $a(F) \not\equiv 0 \bmod p$, falls $F \in \text{QBF}$.

Nun werden insgesamt m Runden gespielt, falls $m \leq n$ die Anzahl der Quantoren in F ist.

Sei $F = F_1 = Q_1 x_1 : G_1(x_1)$ und $p_1(x_1) = a(G_1(x_1)) \bmod p$
(ein Polynom in x_1 vom Grad höchstens $2n$).

Runde 1:

- Alice sendet Bob den Wert $a_1 = a(F_1) \bmod p$.
- Bob gibt "$F \notin \text{QBF}$" aus, falls $a_1 = 0$, sonst verlangt er einen Beweis für die Korrektheit von a_1.
- Diesen gibt Alice, indem sie Bob das Polynom $p_1(x_1)$ mitteilt.
- Falls $Q_1 = \forall$: Bob überprüft, ob $a_1 \equiv p_1(0) \cdot p_1(1) \bmod p$.
- Falls $Q_1 = \exists$: Bob überprüft, ob $a_1 \equiv p_1(0) + p_1(1) \bmod p$.

Beweis des Satzes von Shamir: das Protokoll

- Nun wählt Bob zufällig eine Zahl $r_1 \in \mathbb{Z}/p\mathbb{Z}$, teilt diese Zahl Alice mit (public coins) und berechnet $p_1(r_1) \pmod p$.
- Der arithmetische Ausdruck $a(G_1(x_1))_{x_1=r_1}$ lässt sich schreiben als

$$a(G_1(x_1))_{x_1=r_1} = b + c \cdot a(F_2(x_1))_{x_1=r_1},$$

  wobei F_2 der Teilausdruck von G_1 ist, der mit dem ersten Quantor von G_1 beginnt. Die Werte $b, c \in \mathbb{Z}/p\mathbb{Z}$ können von Bob berechnet werden.
- Falls $c = 0$ (falls kein weiterer Quantor mehr auftritt oder sich der Term c zu 0 auswertet): Bob überprüft, ob $p_1(r_1) = b$ gilt und akzeptiert im positiven Fall.
- Falls $c \neq 0$: Bob berechnet $a_2 = (p_1(r_1) - b) \cdot c^{-1}$.

Beweis des Satzes von Shamir: das Protokoll

Beachte: Wenn Alice in Runde 1 das korrekte Polynom $p_1(x_1) = a(G_1(x_1))$ angibt, dann muss $a(F_2(x_1))_{x_1=r_1} \equiv a_2 \pmod{p}$ gelten.

Sei $F_2(x_1) = Q_2 x_2 : G_2(x_1, x_2)$ und $p_2(x_2) = a(G_2(x_1, x_2))_{x_1=r_1} \pmod{p}$.

Runde 2:

- Bob will von Alice einen Beweis für $a(F_2(x_1))_{x_1=r_1} \equiv a_2 \pmod{p}$ sehen.
- Alice sendet hierfür Bob das Polynom $p_2(x_2)$.
- Bob überprüft wieder $a_2 \equiv p_2(0) \cdot p_2(1) \mod p$ (falls $Q_2 = \forall$) bzw. $a_2 \equiv p_2(0) + p_2(1) \mod p$ (falls $Q_2 = \exists$).
- Dann wählt Bob eine weitere Zufallszahl r_2 teilt diese Alice mit und berechnet $p_2(r_2)$.
- Sei $a(G_2(x_1, x_2))_{x_1=r_1, x_2=r_2} = b + c \cdot a(F_3(x_1, x_2))_{x_1=r_1, x_2=r_2}$, wobei F_3 der Teilausdruck von G_2 ist, der mit dem ersten Quantor von G_2 beginnt.

Beweis des Satzes von Shamir: das Protokoll

- Falls $c = 0$: Bob überprüft, ob $p_2(r_2) = b$ gilt und akzeptiert im positiven Fall.
- Falls $c \neq 0$: Bob berechnet $a_3 = (p_2(r_2) - b) \cdot c^{-1}$.
- In der 3. Runde muss dann Alice $a(F_3(x_1, x_2))_{x_1=r_1, x_2=r_2} \equiv a_3 \mod p$ beweisen.

Diese Protokoll wird für höchstens $m =$ (Anzahl der Quantoren in F) viele Runden durchgeführt.

Offensichtlich: Wenn $F \in$ QBF, dann akzeptiert Bob mit Wahrscheinlichkeit 1.

Gelte nun $F \notin$ QBF, d. h. $a(F) \equiv 0 \mod p$.

Mit welcher Wahrscheinlickkeit kann eine Alice Bob vom Gegenteil $F \in$ QBF überzeugen?

Beweis des Satzes von Shamir: Analyse des Protokolls

Angenommen, Bob gibt schließlich "$F \in \mathrm{QBF}$" aus.

Alice muss in Runde 1 einen Wert $a_1 \not\equiv a(F_1) \equiv 0 \bmod p$ an Bob schicken, sonst würde Bob "$F \notin \mathrm{QBF}$" ausgeben.

Da Bob $a_1 \equiv p_1(0) \cdot p_1(1) \bmod p$ (falls $Q_1 = \forall$) bzw $a_1 \equiv p_1(0) + p_1(1) \bmod p$ (falls $Q_1 = \exists$) prüft, muss auch das von Alice an Bob gesendete Polynom $p_1(x_1)$ falsch sein, d. h. $p_1(x_1) \neq a(G_1(x_1))$.

Dann ist $p_1(x_1) - a(G_1(x_1))$ ein Polynom vom Grad $\leq 2n$ und hat damit höchstens $2n$ Nullstellen in dem Körper $\mathbb{Z}/p\mathbb{Z}$.

Es gilt also $p_1(r_1) = a(G_1(x_1))_{x_1=r_1}$ an maximal $2n$ Stellen $r_1 \in \mathbb{Z}/p\mathbb{Z}$.

Also: $\underset{r_1 \in \mathbb{Z}/p\mathbb{Z}}{\mathrm{Prob}}[p_1(r_1) = a(G_1(x_1))_{x_1=r_1}] \leq \frac{2n}{2^n}$, da $p \geq 2^n$.

Beachte: Falls $p_1(r_1) \neq a(G_1(x_1))_{x_1=r_1}$ gilt, so gilt auch $a_2 \neq a(F_2(x_1))_{x_1=r_1}$.

Beweis des Satzes von Shamir: Analyse des Protokolls

Also gilt $\text{Prob}_{r_1 \in \mathbb{Z}/p\mathbb{Z}}[a_2 \neq a(F_2(x_1))_{x_1=r_1}] \geq 1 - \frac{2n}{2^n}$ zu Beginn von Runde 2.

Erinnerung: In Runde 2 will Alice Bob überzeugen, dass $a_2 = a(F_2(x_1))_{x_1=r_1}$ gilt.

Also muss Alice in Runde 2 Bob von einer falschen Identität überzeugen.

Diese Situation wiederholt sich ständig (höchstens $m \leq n$ mal).

Je mehr Runden gespielt werden, desto besser für Alice: Wenn Bob einmal (etwa in Runde i) eine Zufallszahl r_i mit

$$p_i(r_i) = a(G_i(x_1, \ldots, x_{i-1}, x_i))_{x_1=r_1,\ldots,x_{i-1}=r_{i-1},x_i=r_i}$$

trifft, dann ist der von ihm berechnete Wert a_{i+1} der korrekte arithmetische Wert für den verbliebenen Teilausdruck und Alice kann beruhigt fertig spielen — ohne erneute Täuschungsmanöver.

Beweis des Satzes von Shamir: Analyse des Protokolls

Es werden jedoch maximal $m \leq n$ viele Runden durchgeführt.

Die Wahrscheinlichkeit, dass Bob am Ende korrekt "$F \notin$ QBF" ausgibt, ist also mindestens

$$\left(1 - \frac{2n}{2^n}\right)^m \geq \left(1 - \frac{2n}{2^n}\right)^n \geq 1 - n \cdot \frac{2n}{2^n},$$

wegen $(1-x)^n \geq 1 - nx$ für $0 \leq x \leq 1$.

Die Wahrscheinlichkeit, dass Bob "$F \in$ QBF" ausgibt, ist also $\leq \frac{2n^2}{2^n}$.

Wird das Protokoll zweimal durchlaufen, dann sinkt diese Wahrscheinlichkeit auf

$$\left(\frac{2n^2}{2^n}\right)^2 = \frac{4n^4}{2^{2n}} < 2^{-n}$$

ab einem genügend großen n. □

Bemerkungen zu **IP** = **PSPACE**

- Die Inklusion **PSPACE** $\subseteq$ **IP** und damit **IP** = **PSPACE** gilt auch dann noch, wenn die Zufallsbits von Bob öffentlich sind, also Alice bekannt sind.
- Die Inklusion **PSPACE** $\subseteq$ **IP** gilt auch dann noch, wenn man Alice auf eine polynomiell platzbeschränkte Turingmaschine einschränkt.
- Schränkt man die Anzahl der Runden in interaktiven Beweissystemen auf eine Konstante ein, so erhält man die Komplexitätsklasse **AM** (Arthur-Merlin-Spiele). Es wird **AM** $\subsetneq$ **PSPACE** vermutet.
- Ersetzt man Alice durch 2 Beweiser, mit denen Bob kommunizieren kann, so erhält man die Komplexitätsklasse **MIP**.

  Es gilt: **MIP** = **NEXPTIME** (Babai, Fortnow, Lund 1991), wobei **NEXPTIME** $= \bigcup_{k \geq 1}$ **NTIME**(2^{n^k}).