Probleme

Unentscheidbare Probleme

Komplemente werden im folgenden weggelassen, da offensichtlich auch unentschiedbar.

- Diagonalsprache $D := \{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \text{ nicht}\}$

- Halteproblem $H := \{\langle M \rangle w \mid M \text{ hält auf } w\}$.
- ε-Halteproblem $H_\varepsilon := \{\langle M \rangle \mid M \text{ hält auf } \varepsilon\}$.
- Totales Halteproblem $H_{tot} := \{\langle M \rangle \mid M \text{ hält auf allen Eingaben}\}$.
- $H_{never} := \{\langle M \rangle \mid M \text{ hält auf keiner Eingabe}\}$

- PCP, MPCP und PCP mit 5 oder mehr als 7 Dominos

- Besitzt eine elementare Funktion eine elementare Stammfunktion? (Satz von Richardson)
- Dioph $:= \{\langle p \rangle \mid p \text{ multivariates Polynom über } \mathbb{Z} \text{ mit Nullstelle in } \mathbb{Z}\}$

- (VL5 Bsp) Gegeben $\langle M \rangle$:

$$L(M) = \varnothing \; ? \qquad L(M) = \Sigma^* \; ? \qquad |L(M)| < \infty \; ? \qquad |L(M)| = \infty \; ?$$

$$L(M) \text{ regulär } ? \qquad L(M) \text{ kontextfrei } ?$$

Rekursiv-aufzählbare Probleme

- H
- H_ε
- $\overline{D}$
- PCP
- Dioph
- $\{\langle M \rangle \mid w \in L(M)\}$ für festes Wort w. (Einfach M auf w simulieren)

Nicht rekursiv-aufzählbare Probleme

- $\overline{H}$
- $\overline{H_\varepsilon}$
- H_{tot} und $\overline{H_{tot}}$
- H_{never}
- D
- $\overline{\text{Dioph}}$
- $\{\langle M \rangle \mid M \text{ verwirft alle Eingaben}\}$
- $\{\langle M_1 \rangle \langle M_2 \rangle \mid L(M_1) \cap L(M_2) \neq \Sigma^*\}$ (HA 8.3)

Probleme in P

- SORTIEREN
- Graphzusammenhang
- Primzahltest
- Eulerkreis
- Minimaler Spannbaum
- Maximaler Fluss
- Maximum Matching
- ggT
- Konvexe Hülle in 2D
- LP

Probleme in NP

- SAT
- 3-SAT
- (HALF-)CLIQUE
- KANTEN-AUFSPANNEN
- INDEP-SET
- MATCHING
- VERTEX-COVER
- DOMINATING-SET
- (D-)HAM-CYCLE
- GRAPH-ISOMORPHISMUS
- COMPOSITE
- EX-COVER
- COLORING
- SUBSET-SUM
- PARTITION
- KNAPSACK
- BPP
- TSP (Δ-TSP, $\{1,2\}$-TSP)
- LP / $\{-1,0,1\}$-R.I.P.

Probleme in coNP

- UNSAT
- TAUTOLOGY
- LP
- (Komplement von $L \in \mathsf{NP}$, (JA- und NEIN-Instanzen vertauscht, gleiches Zertifikat)

Definitionen

SAT (NP-vollständig; Satz von Cook und Levin)
Eingabe: Eine Aussagenlogische Formel φ in CNF über einer Variablenmenge $X = \{x_1, \cdots, x_n\}$.
Frage: Ist φ erfüllbar?

3-SAT (NP-vollständig; SAT $\leq_p$ 3-SAT)
Eingabe: Eine Aussagenlogische Formel φ in 3-CNF über einer Variablenmenge $X = \{x_1, \cdots, x_n\}$.
Frage: Ist φ erfüllbar (ex. Variablenbelegung, sodass $\varphi \equiv 1$)?

CLIQUE (NP-vollständig; SAT $\leq_p$ CLIQUE)
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage: Enthält G eine Clique (vollständiger Teilgraph) mit $\geq k$ Knoten?

HALF-CLIQUE (NP-vollständig; CLIQUE $\leq_p$ HALF-CLIQUE)
Eingabe: Ein ungerichteter Graph $G = (V, E)$ mit $|V| = 2k, k \in \mathbb{N}$
Frage: Enthält G eine Clique mit $\geq k$ Knoten?

KANTEN-AUFSPANNEN (NP-vollständig; CLIQUE $\leq_p$ KANTEN-AUFSPANNEN)
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $r, s \in \mathbb{N}$
Frage: Existiert $R \subseteq V, |R| = r$, was $\geq s$ Kanten aufspannt?
$(e_1, e_2 \in R \implies (e_1, e_2) \in E$ aufgespannt von $R)$

INDEP-SET (NP-vollständig; CLIQUE $\leq_p$ INDEP-SET)
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage: Enthält G eine unabhängige Menge ($S \subseteq V$ pw. nicht adjazent) mit $\geq k$ Knoten?

VERTEX-COVER (NP-vollständig; INDEP-SET $\leq_p$ VERTEX-COVER)
Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$
Frage: Enthält G ein Vertex-Cover ($S \subseteq V$ berührt alle Kanten) mit $\leq k$ Knoten?

D-HAM-CYCLE (NP-vollständig; SAT $\leq_p$ D-HAM-CYCLE)
Eingabe: Ein gerichteter Graph $G = (V, A)$
Frage: Besitzt G einen gerichteten Hamiltonkreis?

HAM-CYCLE (NP-vollständig; D-HAM-CYCLE $\leq_p$ HAM-CYCLE)

Eingabe: Ein ungerichteter Graph $G = (V, E)$

Frage: Besitzt G einen Hamiltonkreis (geschl. Pfad, der jeden Knoten genau einmal enthält)?

MATCHING

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$

Frage: Enthält G ein Matching mit $\geq k$ Kanten?
(Matching $M \subseteq E$: Keine 2 Kanten in M haben gemeinsame Knoten).

DOMINATING-SET (NP-vollständig; 3-SAT $\leq_p$ DOMINATING-SET)

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$

Frage: Enthält G ein Dominating-Set mit $\leq k$ Knoten?

(Dominating-Set $D \subseteq V$: Jeder Knoten ist in D enthalten oder zu einem in D benachbart).

GRAPH-ISOMORPHISMUS

Eingabe: Zwei ungerichtete Graphen $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$.

Frage: Ist $G_1 \cong G_2$, d.h. ex Bijektion $f : V_1 \to V_2$ mit $e \in E_1 \iff (f(e_1), f(e_2)) \in E_2$?

COMPOSITE

Eingabe: $n \in \mathbb{N}$ (kodiert als Binärzahl).

Frage: Ist n keine Primzahl?

EX-COVER (NP-vollständig; nicht in VL)

Eingabe: Eine endliche Menge X und $S_1, \cdots, S_M \subseteq X$

Frage: Existiert $I \subseteq [1, m]$ sodass $(S_i)_{i \in I}$ eine Parition von X ist?

COLORING (NP-vollständig; nicht in VL)

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$

Frage: Gibt es eine Färbung $c : V \to [1, k]$ sodass $\forall e \in E : c(e_1) \neq c(e_2)$?

SUBSET-SUM (NP-vollständig; 3-SAT $\leq_p$ SUBSET-SUM)

Eingabe: $a \in \mathbb{N}^k, b \in \mathbb{N}$

Frage: Existiert $I \subseteq [1, k]$ sodass $\sum_{i \in I} a_i = b$?

PARTITION (NP-vollständig; SUBSET-SUM $\leq_p$ PARTITION)
Eingabe: $a \in \mathbb{N}^k$ mit $\sum a_i = 2A, A \in \mathbb{N}$.
Frage: Existiert $I \subseteq [1,k]$ sodass $\sum_{i \in I} a_i = A$?

KNAPSACK (NP-vollständig; SUBSET-SUM $\leq_p$ KP)

Eingabe: $w, p \in \mathbb{N}^k$ und $b \in \mathbb{N}$ (und $\gamma \in \mathbb{N}$)

Zulässige Lösung: Menge $K \subseteq [1,n]$ mit $w(K) := \sum_{i \in K} w_i \leq b$

Optimierungsziel: Maximiere $p(K) := \sum_{i \in K} p_i$

Als Entscheidungsproblem: Existiert K sodass $p(k) \geq \gamma$?

BPP (NP-vollständig; PARTITION $\leq_p$ BPP)

Eingabe: $b \in \mathbb{N}$ und $w \in [1,b]^n$ (und $\gamma \in \mathbb{N}$)

Zulässige Lösung: $k \in \mathbb{N}$ und $f : [1,n] \to [1,k]$ sodass $\forall i \in [1,k] : \sum_{j \in f^{-1}(i)} w_j \leq b$
(Zuordnung von Gewichten zu Kisten, sodass Tragkraft b der Kisten nicht überschritten wird)

Optimierungsziel: Minimiere k (= Anzahl Kisten)

Als Entscheidungsproblem: Existiert eine zulässige Lösung mit $k \leq \gamma$?

$\{1,2\}$-**TSP** (NP-vollständig, HAM-CYCLE $\leq_p$ $\{1,2\}$-TSP)
Eingabe: Städte $1, ..., n$; symm. Dist. $d(i,j) \in \{1,2\}$; $\gamma \in \mathbb{N}$
Frage: Gibt es eine Rundreise mit Länge höchstens γ?

Δ-**TSP** (NP-vollständig, folgt aus $\{1,2\}$-TSP)
Eingabe: Städte $1, ..., n$; symm. Dist. $d(i,j)$, welche Dreiecksungleichung erfüllen; $\gamma \in \mathbb{N}$
Frage: Gibt es eine Rundreise mit Länge höchstens γ?

TSP (NP-vollständig, folgt aus $\{1,2\}$-TSP)
Eingabe: $d(i,j) \in \mathbb{N}$ für $1 \leq i \neq j \leq n$ (und $\gamma \in \mathbb{N}$)
Zulässige Lösung: Permutation $\pi \in S_n$.
Optimierungsziel: Minimiere $d(\pi) := \sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1}) + d(\pi_n, \pi_1)$
Als Entscheidungsproblem: Existiert eine zulässige Lösung mit $d(\pi) \leq \gamma$?

LP

Eingabe: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n, \gamma \in \mathbb{R}$.

Optimierungsvariante Primal: Für $x \in \mathbb{R}^n_{\geq 0}$ maximiere $c^{tr}x$ sodass $Ax \leq b$.

Optimierungsvariante Dual: Für $y \in \mathbb{R}^m_{\geq 0}$ minimiere $b^{tr}y$ sodass $yA \geq c$.

Entscheidungsvariante: Existiert $x \in \mathbb{R}^n$ mit $x \geq 0 \ \wedge \ Ax \leq b \ \wedge \ c^{tr}x \geq y$?

(Für weiteres siehe VL 17)

$\{-1, 0, 1\}$ **Restricted Integer Programming** (NP-vollständig; 3-SAT $\leq_p$ $\{-1, 0, 1\}$-R.I.P.)

Eingabe: $A \in \{-1, 0, 1\}^{m \times n}, b \in \{-1, 0, 1\}^{m \times 1}$

Frage: Existiert $x \in \{0, 1\}^{n \times 1}$ sodass $Ax \geq b$?

UNSAT (coNP-vollständig)

Eingabe: Eine Boolesche Formel φ in CNF über Variablenmenge $\{x_1, ..., x_n\}$.

Frage: Ist φ nicht erfüllbar?

TAUTOLOGY (coNP-vollständig)

Eingabe: Eine Boolesche Formel φ in **D**NF über Variablenmenge $\{x_1, ..., x_n\}$.

Frage: Wird φ von allen Variablenbelegungen erfüllt?

Q-SAT (PSPACE-vollständig)

Eingabe: Eine Boolesche Formel φ in CNF über Variablenmenge $\{x_1, ..., x_n, y_1, ..., y_n\}$.

Frage: $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \cdots \exists x_n \forall y_n : \varphi$?

k**-Schritt-HALTEPROBLEM** (EXPTIME-vollständig)

Eingabe: Eine DTM M, $k \in \mathbb{N}$.

Frage: Hält M auf ε nach höchstens k Schritten?

VL-Stoff

1 Turing Maschinen I

- Probleme
- Turingmaschinen
- rekursive / berechenbare Funktionen & Sprachen
- Konfigurationen
- Programmiertechniken von TM's

Definition: Probleme

- **Problem als Relation**
  $R \subseteq \Sigma^* \times \Gamma^*$ für Alphabete Σ, Γ. Dann $xRy \iff y$ ist zulässige Ausgabe zur Eingabe x.
- Bei eindeutiger Lösung **Problem als Funktion** $f : \Sigma^* \to \Gamma^*$
- **Problem als Entscheidungsproblem**: Form $f : \Sigma^* \to \{0, 1\}$
- $L := f^{-1}(1) \subseteq \Sigma^*$ ist Sprache vom durch f definiertem Entscheidungsproblem.

Definition: Turingmaschine (TM)

Eine Turingmaschine M ist gegeben durch $M = (Q, \Sigma, \Gamma, B, q_0, \overline{q}, \delta)$, wobei

- Q endliche Zustandsmenge
- Σ endliches Eingabealphabet
- $\Gamma \supsetneq \Sigma$ endliches Bandalphabet
- $B \in \Gamma \setminus \Sigma$ Leerzeichen, Blank
- $q_0 \in Q$ Anfangszustand
- $\overline{q}$ Endzustand
- $\delta : (Q \setminus \{\overline{q}\}) \times \Gamma \to Q \times \Gamma \times \{R, L, N\}$ Zustandsüberführungsfunktion

Weiteres:

- Startet in q_0, Kopf über (1. Symbol vom) Eingabewort eingerahmt von Blanks
- TM stoppt, sobald Endzustand $\overline{q}$ erreicht.
- Ausgabewort $y \in \Sigma^*$ beginnt bei Kopfposition und endet vor erstem Symbol in $\Gamma \setminus \Sigma$.

- Spezialfall Entscheidungsprobleme:
  akzeptiert $\iff$ terminiert und Ausgabe beginnt mit 1
  verwirft $\iff$ terminiert und Ausgabe beginnt nicht mit 1
- **Laufzeit** ist Anzahl von Zustandsübergängen bis zur Terminierung.
  Man Schreibt oft $t(n)$ für die maximale Laufzeit aller Eingaben der Länge n.
- **Speicherbedarf** Anzahl während Berechnung besuchter Bandzellen.
  Man Schreibt oft $s(n)$ für den maximalen Speicherbedarf aller Eingaben der Länge n.

- TM M **entscheidet** $L \subseteq \Sigma^*$ wenn $w \in L$ akzeptiert und $w \notin L$ verworfen wird (terminiert immer).

- Jede TM kann durch eine TM mit einseitig beschränktem Band (benutzt nie Positionen $p < 0$) simuliert werden. Dies hat nur konstanten Overhead. (Siehe HA 2.3)

Definition: rekursive / T-berechenbare Funktionen & Sprachen

$f : \Sigma^* \to \Sigma^*$ heißt rekursiv bzw. (T-)berechenbar,
wenn es eine TM gibt welche bei Eingabe $x \in \Sigma^*$ den Wert $f(x)$ berechnet.

Eine Sprache $L \subseteq \Sigma^*$ heißt rekursiv bzw. (T-)entscheidbar,
wenn es eine TM gibt welche stets terminiert und $w \in \Sigma^*$ akzeptiert $\iff w \in L$.

Definition: Konfiguration

Eine Konfiguration einer TM ist ein String $\alpha q \beta$, wobei $\alpha, \beta \in \Gamma^*, q \in Q$ mit $\beta \neq \varepsilon$.
Zustand ist q, auf dem Band steht $\alpha\beta$, Kopf über erstem Buchstaben von β.
Blanks ausgelassen, ausser Kopf steht auf einem.

Man schreibt $\alpha q \beta \vdash \alpha' q' \beta'$ für **direkte Nachfolgerkonfigurationen** (in einer δ-Anwendung)

Analog schreibt man für endlich viele (auch 0) Rechenschritte $\alpha q \beta \vdash^* \alpha'' q'' \beta''$.

($\vdash^*$ kann als reflexiv-transitive Hülle von $\vdash$ aufgefasst werden)

Techniken zur Programmierung TM's

- Speichere $k \in \mathbb{N}_{>0}$ Symbole im Zustandsraum:
  $$Q_{neu} := Q \times \Gamma^k$$
  Bspw sind neue Zustände für $k = 2$ dann (q_0, BB) oder $(q_1, 01)$ (wenn $0, 1 \in \Gamma$).

- k-spurige TM: TM mit zusätzlichen Vektoren in Γ^k als Symbole. Man setzt
  $$\Gamma_{neu} := \Gamma \cup \Gamma^k$$

2 Turing Maschinen II

- k-Band-TM's
- Simulation von k-Band-TM's mit 1-Band-TM's
- Gödelnummern
- Universelle TM
- Church-Turing-These

Definition: k-Band-TM

Besitzt $k \in \mathbb{N}_{>0}$ Arbeitsbänder mit unabhängigen Köpfen. Zustandsübergangsfunktion ist dann

$$\delta : (Q \setminus \{\overline{q}\}) \times \Gamma^k \to Q \times \Gamma^k \times \{R, L, N\}^k$$

Dabei ist Band 1 das Eingabe / Ausgabeband. Die anderen sind zunächst leer (Blanks).

Satz: Simulation von k-Band TM's durch 1-Band-TM's

Eine k-Band TM M mit Zeitbedarf $t(n)$ und Platzbedarf $s(n)$ kann mit einer 1-Band-TM M' in Zeitbedarf $\mathcal{O}(t^2(n))$ und Platzbedarf $\mathcal{O}(s(n))$ simuliert werden.

Also **quadratischer Zeitverlust** und **konstanter Speicherverlust**.

Bewies via $2k$ Spuren; Inhalt der Bänder und Positionen der Köpfe (markiert mit #).
Jeder Rechenschritt von M wird wie folgt durch M' simuliert:

- Kopf steht auf linkestem #, M' kennt Zustand von M.
- Laufe nach rechts und speichere alle Zeichen an den Kopfpositionen auf den zugehörigen Bändern im Zustand.
- Werte damit δ_M aus.
- Laufe zurück und verändere entsprechend Kopfpositionen / Bandinhalte.

- Nach t Schritten von M linkestes/rechtestes # höchstens $2t$ Positionen auseinanderliegen
- Simulation eines Schrittes also in $\mathcal{O}(t(n))$
- Für $t(n)$ Schritte damit $\mathcal{O}(t(n)^2)$

Definition: Gödelnummer

Die Gödelnummer einer TM M wird durch $\langle M \rangle$ bezeichnet.

- Eindeutige, **präfixfreie** Kodierung über $\{0, 1\}$.
- $\langle M \rangle$ beginnt und endet stets mit 111, enthält sonst 111 nicht.
- Man beschränkt sich auf TM's mit $Q = \{q_1, q_2, \cdots, q_t\}, t \geq 2$
  wobei q_1, q_2 Anfangs-/Endzustand sind. Ferner soll $\Gamma = \{0, 1, B\}$.
- Man kodiert den t-ten Übergang

  $$\delta(q_a, X_b) = (q_c, X_d, D_e) \quad \text{durch} \quad code(t) = 0^a 10^b 10^c 10^d 10^e$$

  Wobei $X_1 = 0, X_2 = 1, X_3 = B, D_1 = L, D_2 = N, D_3 = R$.

  Dann kodiert man die TM M mit s Übergängen durch:

  $$\langle M \rangle := 111 code(1) 11 code(2) 11 \ldots 11 code(s) 111$$

Definition: Universelle Turingmaschine

Eine general-purpose-TM. Eingabe ist ein Wort der Form $\langle M \rangle w$ für $w \in \{0, 1\}^*$.

Simulation via 3-Band TM in quadratischer Zeit möglich. Band1 ist Band von M, Band2 ist $\langle M \rangle$, Band3 aktueller Zustand von M.

Universelle 1-Band-TM mit **konstantem Zeitverlust**, wenn man $\langle M \rangle$ auf Spur2 und Zustand auf Spur3 "mit dem Kopf der TM M mitführt", und $|\langle M \rangle|$ als Konstante ansieht.

Behauptung: Church-Turing-These (1930)

Die Klasse der TM-berechenbaren Funktionen stimmt mit der Klasse der "intuitiv berechenbaren" Funktionen überein.

Daher (in dieser Vorlesung)

$$\text{berechenbare Funktion} = \text{TM-berechenbare Funktion} = \text{rekursive Funktion}$$

$$\text{entscheidbare Sprache} = \text{TM-entscheidbare Sprache} = \text{rekursive Sprache}$$

3 Registermaschinen

- Registermaschinen + Kostenmaße
- Simulation RAM durch TM
- Simulation TM durch RAM
- Collatz Problem

Definition: Registermaschine (RAM)

Besteht aus Befehlszähler, Akkumulator c(0), unbeschränkter Speicher c(1),c(2),...
Programme haben Befehlssatz:

(IND/C)LOAD, (IND)STORE, (IND/C)ADD, (IND/C)SUB, (IND/C)MULT, (IND/C)DIV

IF c(0) ? x THEN GOTO j wobei j Zeile im Programm und $? \in \{=, <, \leq, \geq, >\}$

GOTO, END

- Inhalt der Register sind Elemente von $\mathbb{N}$ (beliebig groß)
- Eingabe ebenfalls in $\mathbb{N}^*$, zu Beginn "in den ersten Registern".
- Andere Register mit 0 initialisiert.
- Befehlszähler startet mit 1. Als nächstes wird immer die Zeile, auf die der Befehlszähler verweist, ausgeführt.
- Rechnung stoppt sobald END ausgeführt wird.
- Ausgabe befindet sich dann "in den ersten Registern".

- **Uniformes Kostenmaß:** Jeder Schritt / Befehl zählt eine Zeiteinheit
- **Logarithmisches Kostenmaß:** Die Laufzeitkosten eines Schrittes sind Maximum der Logarithmen der involvierten Zahlen. (Maximale Zahlenlänge)

Satz: Simulation von RAM durch TM

Für jede im logarithmischen Kostenmass $t(n)$-zeitbeschränkte RAM R gibt es ein Polynom q und eine $q(n + t(n))$-zeitbeschränkte TM M, welche R simuliert.

Simulation hat also **polynomiellen Overhead**. Beweisidee:

- 2-Band-TM. Band 1 für Unterprogramme, Band 2 Inhalt der benutzten Register.
- Unterprogramme für Initialisierung, Ergebnisausgabe und jede Programmzeile.
- Befehlszähler im Zustand speichern.
- Simulation von einem Schritt: Sei b Befehlszähler / Zustand.
  1. Kopiere inhalt der in Programmzeile b angesprochenen Registern auf Band 1.
  2. Führe via Unterprogramm der Zeile b entsprechende Operationen durch.
  3. Kopiere Ergebnisse zurück in die in b angegebenen Register auf Band 2.
  4. Aktualisiere b (Inkrement oder GOTO)
- Alle Unterprogramme Laufzeit polynomiell in der Länge von Band 2 (benutzte Register). Also: Eine Laufzeit polynomiell in $n + t(n)$.
- Dank Abschlusseigenschaften ist die Gesamtlaufzeit dann auch polynomiell in $n + t(n)$.

Satz: Simulation von TM durch RAM

Jede $t(n)$-zeitbeschränkte TM kann durch eine RAM simuliert werden, die Zeitbeschränkt ist durch

$$\mathcal{O}(t(n) + n) \qquad \text{(uniformes Kostenmaß)}$$
$$\mathcal{O}((t(n) + n) \cdot \log(t(n) + n)) \qquad \text{(logarithmisches Kostenmaß)}$$

Beweisidee:

- O.b.d.A. TM mit 1-seitig beschränktem Band. (Positionen $\mathbb{N}$)
- Nummeriere Zustände und Symbole zum Speichern in Registern.
- Reg 1 für Index des Kopfes, Reg 2 für Zustand, Reg $n + 3$ für Bandinhalt an Pos. $n \in \mathbb{N}$.
- Programm besteht aus if-Abfragen von Zustand und gelesenem Symbol, die dann entsprechende Register verändern (Nachahmen von δ).
- UNIFORM: Initialisierung in $\mathcal{O}(n)$, ein Schritt hat konstante Laufzeit, also $\mathcal{O}(n + t(n))$
- LOG: Kodierungslänge der Bandpositionen beschränkt durch $\mathcal{O}(\log(t(n) + n))$ also $\mathcal{O}((t(n) + n) \log(t(n) + n))$.

Problem: Collatz-Problem
Wird folgende Funktion bei wiederholter Anwendung stets bei jeder Eingabe den Wert 1 erreichen?

$$f : \mathbb{N} \to \mathbb{N}, x \mapsto \begin{cases} \frac{x}{2} & , x \text{ gerade} \\ 3x + 1 & , x \text{ ungerade} \end{cases}$$

4 Unentscheidbarkeit I

- Abzählbarkeit
- Diagonalsprache
- Unterprogrammtechnik
- Komplement und Entscheidbarkeit
- Halteproblem

Definition: Abzählbarkeit

Eine Menge M heißt abzählbar, wenn

$$M = \varnothing \quad \vee \quad \exists c : \mathbb{N} \to M \text{ surj.}$$

Wissenswertes:

- Wenn M abzählbar unendlich, gibt es eine Bijektion zwischen $\mathbb{N}$ und M.
- $\{0,1\}^*$ ist abzählbar in der **kanonischen Reihenfolge** $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$.
- Damit Menge der Gödelnummern und Menge der TM's ist abzählbar.

- i-tes Wort der kanonischen Reihenfolge über $\{0,1\}$ ist w_i.
- i-te TM der kanonischen Reihenfolge der Gödelnummern ist M_i.

- $\mathcal{P}(\mathbb{N})$ ist überabzählbar (Diagonalargument).
- $\mathbb{N}^* = \bigcup_{n \in \mathbb{N}} \mathbb{N}^n$ ist abzählbar. (Codiere binär wie Gödelnr, dann Teilmenge von $\{0,1\}^*$)
- Die Menge aller semi-entscheidbaren Sprachen ist abzählbar (da TM's abzählbar).
- Die Menge der unentscheidbaren Sprachen ist überabzählbar.

Definition: Diagonalsprache

Die Diagonalsprache ist definiert durch:

$$D := \{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w \textbf{ nicht}\}$$

Diese Sprache ist **unentscheidbar** (Diagonalargument). Beweis:

Angenommen entscheidbar, dann ex. $j \in \mathbb{N}$ mit M_j entscheidet D. Dann

$$w_j \in D \iff M_j \text{ akzeptiert } w_j \iff w_j \notin D$$

Widerspruch, also Annahme falsch, also unentscheidbar. Das Komplement

$$\overline{D} = \{w \in \{0,1\}^* \mid w = w_i \text{ und } M_i \text{ akzeptiert } w\}$$

ist ebenfalls unentscheidbar. (Entscheidbares abgeschlossen unter Komplement)

Definition: Unterprogrammtechnik

Beweistechnik zur Unentscheidbarkeit. Nehme an Problem ist entscheidbar. Benutze TM, welche dieses Problem entscheidet als Unterprogramm um ein bekanntlich unentscheidbares Problem zu entscheiden. Widerspruch, Annahme der Entscheidbarkeit muss falsch sein.

Problem: Halteproblem

Das Halteproblem H, definiert durch

$$H := \{\langle M \rangle w \mid M \text{ hält auf } w\}$$

ist nicht entscheidbar. Beweisidee:
Unterprogrammtechnik, zeige Entscheidbarkeit des Komplements der Diagonalsprache, $\overline{D}$. Zu einer Eingabe w ermitteln wir $i \in \mathbb{N}$ mit $w = w_i$ und lassen M_H auf $\langle M_i \rangle w_i$ laufen. Wenn es verwirft (also M_i nicht auf w_i hält), verwerfen wir, sonst lassen wir einfach M_i auf w_i laufen und übernehmen den Output.

5 Unentscheidbarkeit II

- ε-Halteproblem
- Partielle Funktionen
- Satz von Rice

Problem: Epsilon-Halteproblem

Das ε-Halteproblem H_ε, definiert durch

$$H_\varepsilon := \{\langle M \rangle \mid M \text{ hält auf } \varepsilon\}$$

ist nicht entscheidbar. Beweisidee:
Unterprogrammtechnik, zeige Entscheidbarkeit vom normalen Halteproblem H. Aus Eingabe $\langle M \rangle w$ konstruiere $\langle M_w \rangle$ von TM M_w, welche M auf w simuliert und die Eingabe ignoriert. Lasse dann M_{H_ε} auf $\langle M_w \rangle$ laufen.

Definition: Partielle Funktionen

TM-berechenbare Funktionen sind partielle Funktionen. TM's halten im allgemeinen nicht.

Die von einer TM M berechnete Funktion ist von der Form

$$f_M : \{0,1\}^* \to \{0,1\}^* \cup \{\perp\}$$

wobei $\perp$ für undefiniert steht, und bedeutet, dass M nicht hält. Speziell Entscheidungsprobleme:

$$f_M : \{0,1\}^* \to \{0,1,\perp\}$$

Dabei steht 0 für Verwerfen, 1 für Akzeptieren und $\perp$ für Nicht-Halten.

Satz: Satz von Rice (Henry Gordon Rice, 1920-2003)

Sei $\mathcal{R}$ die Menge der TM-berechenbaren partiellen Funktionen. Betrachte $\varnothing \neq \mathcal{S} \subsetneq \mathcal{R}$.
Dann ist

$$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ berechnet eine Funktion aus } \mathcal{S}\}$$

unentscheidbar.

- Bsp: Sei $\mathcal{S} = \{f_M \mid f_M(\varepsilon) \neq \perp\}$. Dann ist $L(\mathcal{S}) = H_\varepsilon$ unentscheidbar.

Beweisidee: Unterprogrammtechnik mit H_ε: (Syntaxchecking hier ausgelassen)
Sei u die überall undefinierte Funktion mit O.E. $u \notin \mathcal{S}$ (sonst zeige unentsch. von $\mathcal{R} \setminus \mathcal{S}$).
Wähle $f \in \mathcal{S}$ und TM N, die f berechnet. Sei $\langle M \rangle$ Eingabe für H_ε. Konstruiere M^*:

- Bei Eingabe x simuliert M^* zuerst M mit Eingabe ε.
- Danach (falls es terminiert), berechne $f(x)$ via N.

Übergebe nun M^* an $M_{L(\mathcal{S})}$, dessen Existenz nach Unterprogrammtechnik vorrausgesetzt ist, und übernehme das Ergebnis. Es folgt:

$$\langle M \rangle \in H_\varepsilon \implies M \text{ hält auf } \varepsilon \implies M^* \text{ berechnet } f \implies M^* \in L(M_{L(\mathcal{S})}) \implies \text{akzeptiert}$$

$$\langle M \rangle \notin H_\varepsilon \implies M \text{ hält nicht auf } \varepsilon \implies M^* \text{ berechnet } u \implies M^* \notin L(M_{L(\mathcal{S})}) \implies \text{verwirft}$$

Damit könnten wir also H_ε entscheiden, Widerspruch. Folglich ist $L(\mathcal{S})$ unentscheidbar. □

Anmerkungen

- Der Satz von Rice **sagt nichts über Verhalten der TM (Syntax) aus**.
  (Zustand / Anzahl Schritte / genauere Implementierungsdetails).
- Konsequenzen: Es ist unentscheidbar, ob ein gegebenes Programm in einer Turing-mächtigen Sprache eine gegebene nicht-triviale Spezifikation erfüllt.

6 Rekursive Aufzählbarkeit

- TM erkennt / Semi-Entscheidbarkeit
- Aufzähler
- Rekursive Aufzählbarkeit
- rek. aufzählbar $\iff$ semi-entscheidbar
- Abschlusseigenschaften (semi-)entscheidbarer Sprachen
- Reduktionen und Übertragungseigenschaften
- Totales Halteproblem

Definition: TM Erkennt / Semi-entscheidbar

Eine Sprache L wird von einer TM M **erkannt**, wenn M jedes Wort aus L akzeptiert und M kein Wort akzeptiert, welches nicht in L liegt.

- "Also: Die von M erkannte Sprache ist genau $L(M)$".
- Wenn eine TM existiert, die eine Sprache L erkennt, so ist L semi-entscheidbar.

Definition: Aufzähler

Ein Aufzähler für eine Sprache $L \subseteq \Sigma^*$ ist eine TM mit Drucker. Die TM wird ohne Eingabe mit leerem Band gestartet und gibt mit der Zeit alle Wörter in L aus (mögl. Wiederholungen).

Ausgegebene Wörter werden durch ein Trennzeichen $\# \notin \Sigma$ separiert.

Der Drucker druckt ausschließlich Wörter aus L.

Definition: Rekursive Aufzählbarkeit

Wenn es für eine Sprache L einen Aufzähler gibt, so wird L als rekursiv aufzählbar bezeichnet.

Satz: Äquivalenz semi-entscheidbar und rekursiv aufzählbar

Zu einer Sprache L haben wir

$$L \text{ semi-entscheidbar} \iff L \text{ rekursiv aufzählbar}$$

Beweisidee: "$\Longrightarrow$": Simuliere im n-ten Schritt die ersten n Wörter (kanonisch) für n Schritte und gebe diese auf dem Drucker aus, falls sie akzeptiert werden.

"$\Longleftarrow$": Zur Eingabe w lasse den Drucker laufen und akzeptiere sobald er w ausdruckt.

Satz: Abschlusseigenschaften der (Semi-)Entscheidbarkeit

- Entscheidbare Sprachen sind unter Komplementbildung abgeschlossen.
- (Semi-)Entscheidbare Sprachen sind unter $\cup, \cap$ abgeschlossen.
- L und $\overline{L}$ rekursiv aufzählbar $\implies$ L entscheidbar.

Definition: Reduktion

Seien L_1, L_2 Sprachen über einem Alphabet Σ. Dann heißt L_1 auf L_2 reduzierbar ($L_1 \leq L_2$) wenn es eine berechenbare Funktion $f : \Sigma^* \to \Sigma^*$ gibt mit

$$\forall w \in \Sigma^* : w \in L_1 \iff f(w) \in L_2$$

Für zwei Sprachen $L_1, L_2 \subseteq \Sigma^*$ gilt

$$(L_1 \leq L_2) \wedge (L_2 \text{ entscheidbar / rek. aufzählbar}) \implies L_1 \text{ entscheidbar / rek. aufzählbar}$$

Per Kontraposition erhält man:

$(L_1 \leq L_2) \wedge (L_1$ nicht entscheidbar / rek. aufzählbar$) \implies L_2$ nicht entscheidbar / rek. aufzählbar

ferner haben wir Transitivität (Tut 5.4)

$$(L_1 \leq L_2) \wedge (L_2 \leq L_3) \implies L_1 \leq L_3$$

und

$$L_1 \leq L_2 \iff \overline{L_1} \leq \overline{L_2}$$

Problem: Totales Halteproblem (H_{tot})

Das totale Halteproblem H_{tot} ist definiert durch

$$H_{tot} := \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\}$$

Sowohl H_{tot} als auch $\overline{H_{tot}}$ sind nicht rekursiv aufzählbar.

Beweisidee: Zeige $\overline{H_\varepsilon} \leq H_{tot}$ und $\overline{H_\varepsilon} \leq \overline{H_{tot}}$.

Ersteres: Wir bilden Müll auf ein festes Wort $w \in H_{tot}$ ab. Sonst sei $f(\langle M \rangle) = \langle M' \rangle$, wobei M' bei Eingaben der Länge ℓ die ersten ℓ Schritte von M bei Eingabe ε simuliert. Hält M in diesen, so geht M' in Endlosschleife, andernfalls hält M'.

Letzteres trivial mit Reduktion, bilde $\langle M \rangle$ auf $\langle M' \rangle$ ab, wobei M' Eingabe ignoriert und M mit Eingabe ε simuliert. Bilde ausserdem Müll auf Müll ab.

7 Postsches Correspondenzproblem

- Postsches Correspondezproblem (PCP)
- Einschränkungen des PCP's

Problem: Postsches Correspondenzproblem (PCP)

Eine Instanz des PCP besteht aus eine endlichen Menge

$$K = \{\left[\frac{x_1}{y_1}\right], \cdots, \left[\frac{x_k}{y_k}\right]\} \quad \text{für} \quad x_1, \ldots, x_k, y_1, \ldots, y_k \in \Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$$

Elemente von K nennen wir Dominos. Frage:

$$\text{Existiert} \quad I = (i_1, i_2, \ldots, i_n) \in [1, k]^n \quad \text{mit} \quad x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n} \quad ?$$

Die Modizierte Version, das MPCP verlangt nur, dass I mit $i_1 = 1$ beginnt.

Das PCP und MPCP sind **unentscheidbar**, dazu zeigt man $H \leq \text{MPCP} \leq \text{PCP}$.

Zu MPCP $\leq$ PCP:
Obere Wörter: Hinter jeden Buchstaben ein #. Beim Startstein noch ein # an den Anfang. Untere Wörter: Vor jeden Buchstaben ein #. Einen letzten Stein $\left[\frac{\$}{\#\$}\right]$ hinzufügen. Also bspw:

$$\left\{\left[\frac{ab}{a}\right], \left[\frac{c}{abc}\right], \left[\frac{a}{b}\right]\right\} \longrightarrow \left\{\left[\frac{\#a\#b\#}{\#a}\right], \left[\frac{c\#}{\#a\#b\#c}\right], \left[\frac{a\#}{\#b}\right], \left[\frac{\$}{\#\$}\right]\right\}$$

Zu $H \leq$ MPCP: Kodiere Konfigurationen und simuliere dadurch einen Lauf der TM.

Für weitere Details siehe VL.

Probleme: Einschränkungen des PCP's

- Zu Wörtern der Länge 1 ist das PCP entscheidbar.
- Wenn alle Wörter Länge 1 oder 2 haben ist das PCP unentscheidbar.

- Für 1 oder 2 Dominos ist das PCP entscheidbar
- Für 5 Dominos ist das PCP unentscheidbar
- Für 7 oder mehr Dominos ist das PCP unentscheidbar

8 Turing-Mächtigkeit

- Leerheit Schnitt zweier CFG's
- Satz von Richardson
- Hilberts 10. Problem / Satz von Matiyasevich
- Satz von David, Robinson,... (Ganzzahlige Polynome gleichstark wie TM's)
- Turing-Mächtigkeit
- Conway's Game of Life

Problem: Leerheit des Schnittes der Sprachen zweier CFG's

Es ist unentscheidbar, ob zu zwei CFG's G_1, G_2 gilt, dass $L(G_1) \cap L(G_2) = \varnothing$.

Beweisidee:

- Betrachte PCP-Instanz $\left\{ \left[\frac{x_1}{y_1}\right], \ldots, \left[\frac{x_n}{y_n}\right] \right\}$.
- Es seien $a, b, c \notin x_i, y_i \forall i \in [1, n]$.
- Konstruiere CFG's G_1, G_2 mit folgenden Regeln:

  $G_1: \quad S \mapsto x_1 S a^1 b \mid x_2 S a^2 b \mid \cdots \mid x_n S a^n b \mid c \qquad G_2: \quad S \mapsto y_1 S a^1 b \mid y_2 S a^2 b \mid \cdots \mid y_n S a^n b \mid c$

- PCP lösbar genau dann, wenn $L_1(G) \cap L_2(G) \neq \varnothing$. Bspw. wenn folgende Ableitungen gleich sind:

$$G_1 : S \overset{*}{\to} x_1 x_4 x_2 x_5 x_1 x_4 \, c \, a^4 b a^1 b a^5 b a^2 b a^4 b a^1 b$$
$$G_2 : S \overset{*}{\to} y_1 y_4 y_2 y_5 y_1 y_4 \, c \, a^4 b a^1 b a^5 b a^2 b a^4 b a^1 b$$

Satz: Satz von Richardson (1968), (Integration in geschlossener Form)

Es ist unentscheidbar, ob eine gegebene elementare Funktion eine elementare Stammfunktion besitzt.

Problem: Hilberts zehntes Problem (BuK-Formulierung) / Satz von Matiyasevich

Hilberts zehntes Problem handelt von diophantischen Gleichungen und ist beschrieben durch:

Dioph = $\{\langle p \rangle \mid p$ ist multivariates Polynom mit ganzzahligen Koeffizienten und ganzzahliger Nullstelle$\}$

Nach dem **Satz von Matiyasevich**(1970) ist **Dioph unentscheidbar**.

Jedoch ist **Dioph rekursiv aufzählbar**, was aus der Abzählbarkeit von $\mathbb{Z}^n$ folgt.

Satz: Satz von Davis, Robinson, Putnam, Matiyasevich. (Ganzzahlige Polynome und TM's)

Der Satz besagt, dass zu $X \subseteq \mathbb{Z}$ gilt:

$$X \text{ rek. aufzählbar} \iff \exists p \in \mathbb{Z}[x_1, \ldots, x_k] : X = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z}^{k-1} : p(x, y) = 0\}$$

Er sagt aus, dass **ganzzahlige Polynome so berechnugsstark wie TM's** sind.

Definition: Turing-Mächtigkeit

Ein Rechnermodell wird als Turing-mächtig bezeichnet, wenn jede TM-berechenbare Funktion auch durch dieses Rechnermodell berechnet werden kann.

RAM's sind Turing-mächtig

Satz: Mini-RAM / RAM mit eingeschränktem Befehlssatz ist Turing-mächtig

Die Mini-RAM verfügt nurnoch über **endlich viele Register** (1000?) und folgende 8 Befehle:

LOAD, STORE, CLOAD, CADD, CSUB, GOTO, IF $c(0) > 0$ THEN GOTO, END.

Die **Mini-RAM ist Turing-mächtig**.

Bemerkungen: Turing-mächtige Beispiele

- Lambda Calculus von Alonzo Church
- μ-rekursive Funktionen von Stephen Kleene
- Alle gängigen höheren Programmiersprachen (C, Java, etc.)
- Postscript, Tex, Latex
- Power-Point (wegen Animationen)

Definition: Conway's Game of Life (1970)

Conway's Game of Life ist ein zellulärer Automat, der auf einem unendlichen 2-dimensionalen Gitter arbeitet. Zu jedem Zeitpunkt ist jede Zelle entweder lebend oder tot.

In jedem Schritt passiert dann:

- Eine tote Zelle mit genau 3 lebenden Nachbarn ist im nächsten Schritt lebendig.
- Lebende Zellen mit weniger als 2 oder mehr als 3 lebenden Nachbarn sterben.
- Alle anderen Zellen bleiben unverändert.

Conway's Game of Life ist Turing-mächtig.

9 LOOP und WHILE Programme I

- LOOP
- Nützliche LOOP-Programme
- WHILE
- WHILE ist Turing-mächtig
- LOOP-WHILE

Definition: Die Programmiersprache LOOP

- Variablen: $x_1, x_2, x_3, \ldots$ und x_0 zur Ausgabe. Eingabe in $x_1, \cdots, x_m$.
  Restliche Variablen mit 0 initialisiert.
- Konstanten: 0 und 1
- Symbole: :=, +, ;
- Keywords: LOOP, DO, ENDLOOP

- $x_i := x_j + c$ für $i, j \in \mathbb{N}, c \in 0, 1$ ist ein LOOP-Programm.
- Wenn P_1, P_2 LOOP-Programme sind, dann ist $P_1; P_2$ ein LOOP-Programm.
- Falls P ein LOOP-Programm ist, dann ist LOOP x_i DO P ENDLOOP ein LOOP-Programm.

Ein LOOP-Programm P berechnet eine **totale** k-stellige Funktion der Form $[P] : \mathbb{N}^k \to \mathbb{N}^k$.

LOOP-Programme sind nicht Turing-mächtig

Bemerkungen: Nützliche LOOP-Programme:

- $x_i := x_j$ via $x_i := x_j + 0$.
- $x_i := c$ für $c \in \mathbb{N}_0$ via festes $x_{zero} = 0$ und wiederholtem $x_i := x_i + 1$.
- $x_0 := x_1 + x_2$ via LOOP-Konstrukt ($x_j := x_j + 1$ einfach x_k mal)
- $x_0 := x_1 \cdot x_2$ via LOOP-Konstrukt (x_0 Anfangs 0 und dann $x_0 := x_0 + x_1$ genau x_2 mal)
- $x_0 := x_1 \dot{-} x_2 = \max(x_1 - x_2, 0)$
- $x_0 := x_1$ DIV x_2 und $x_0 := x_1$ MOD x_2
- IF $x_1 = 0$ THEN P_1 ELSE P_2 ENDIF wie folgt: (IF $x_1 = c$ auch möglich)
  $x_2 := 1; x_3 := 0;$
  LOOP x_1 DO $x_2 := 0; x_3 := 1$ ENDLOOP;
  LOOP x_2 DO P_1 ENDLOOP;
  LOOP x_3 DO P_2 ENDLOOP;

- max, min

Definition: Die Programmiersprache WHILE

- Praktisch wie LOOP. Unterschiede:
- Symbole: :=, +, ;, $\neq$
- Keywords: WHILE, DO, ENDWHILE

- Falls P ein WHILE-Programm ist, dann ist WHILE $x_i \neq 0$ DO P ENDWHILE ein WHILE-Programm.

Ein WHILE-Programm P berechnet eine (nicht unbedingt totale) k-stellige Funktion der Form $[P] : \mathbb{N}^k \to \mathbb{N}^k$.

Jedes LOOP-Programm kann durch ein WHILE-Programm simuliert werden.

Satz: While-Programme sind Turing-mächtig

Eine äußere Schleife WHILE Zustand $\neq 0$ DO, und dadrin dann mit if-Abfragen δ simuliert. Konfigurationen werden in Dezimalzahlen kodiert, wobei $\Gamma = \{1, 2, B\}$ und B durch 0 kodiert wird. Bewegung des Kopfes durch modifizieren der Konfigurationsvariablen via MOD und DIV.

Definition LOOP-WHILE (HA 7.3)

Die Programmiersprache LOOP-WHILE darf alle LOOP-keywords benutzen, und höchstens einmal eine WHILE-Schleife.

LOOP-WHILE ist Turing-mächtig, da zur Simulation der Turingmaschine nur effektiv eine WHILE-Schleife benötigt wird (die äußeste, WHILE Zustand $\neq$ Endzustand DO ...)

10 LOOP und WHILE Programme II

- Ackermann-Funktion
- Up-Arrow-Notation
- Wachstumsfunktion
- Wachstumslemma / LOOP nicht Turing-mächtig

Definition: Ackermann-Funktion

Die Ackermann-Funktion $A : \mathbb{N}^2 \to \mathbb{N}$ ist rekursiv wie folgt definiert:

$$A(0,n) = n+1 \qquad A(m+1,0) = A(m,1)$$

$$A(m+1,n+1) = A(m,A(m+1,n))$$

- $A(1,n) \equiv n+2$
- $A(2,n) \equiv 2n+3$
- $A(3,n) \equiv 2^{n+3} - 3$
- $A(4,n) \equiv \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3} - 3$

Die Ackermann-Funktion ist Turing-berechenbar und streng monoton in beiden Parametern.

Exkurs: Up-Arrow-Notation (Donald Knuth)

$$a \uparrow^m b := \begin{cases} 1 & , b = 0 \\ a \cdot b & , m = 0 \\ a^b \cdot b & , m = 1 \\ a \uparrow^{m-1} (a \uparrow^m (b-1)) & , \text{sonst} \end{cases}$$

Es gilt:

- $A(1,n) \equiv 2 + (n+3) - 3$
- $A(2,n) \equiv 2 \cdot (n+3) - 3$
- $A(3,n) \equiv 2 \uparrow (n+3) - 3$
- $A(4,n) \equiv 2 \uparrow\uparrow (n+3) - 3$
- $A(m,n) \equiv 2 \uparrow^{m-2} (n+3) - 3$

Satz: Monotonieverhalten der Ackermannfunktion

$$\forall m, n \in \mathbb{N} : A(m, n) < A(m, n + 1) \leq A(m + 1, n)$$

Definition: Wachstumsfunktion

Zu einem LOOP-Programm P und Inputs $a \in \mathbb{N}^k$ definiert man $f_P(a) := \sum_{i=1}^{k} b_i$ als die Summe der Ergebniswerte von P bei Eingabe a, also $b = [P](a)$. Die Wachstumsfunktion ist dann gegeben durch:

$$F_P(n) := \max \left\{ f_P(a) : a \in \mathbb{N}^k, \sum_{i=1}^{k} a_i \leq n \right\} = \max\{||[P](a)||_1 : a \in \mathbb{N}^k, ||a||_1 \leq n\}$$

Satz: Wachstumslemma

Sei P ein LOOP-Programm. Dann gilt:

$$\exists m_P \in \mathbb{N} : \forall n \in \mathbb{N} : F_P(n) < A(m_P, n)$$

Genauer ist

- $m_{x_i:=x_j+c} = 2$.
- Zu LOOP-Programmen P, Q ist $m_{P;Q} = \max(m_P, m_Q) + 1$.
- Zu LOOP-Programm P ist $m_{\text{LOOP } x_i \text{ DO } P \text{ ENDLOOP}} = m_P + 1$.

Es folgt, dass **LOOP-Programme nicht Turing-mächtig** sind, da die Ackermann TM-berechenbar, aber nicht LOOP-berechenbar ist.

11 Primitiv-rekursive Funktionen

- Primitiv-rekursive Funktionen
- Nützliche primitiv-rekursive Funktionen
- Bijektion $\mathbb{N}^2 \to \mathbb{N}$
- Äquivalenz primitiv-rekursiv und LOOP
- Kleenscher μ-Operator
- Klasse der μ-rekursiven Funktionen ist Turing-mächtig

Definition: Primitiv-rekursive Funktionen

Die primitiv-rekursiven Funktionen setzen sich aus Basifunktionen mittles 2 Operationen zusammen und bilden eine Unterklasse der Funktionen $\mathbb{N}^k \to \mathbb{N}$.

Die Basisfunktionen sind:

- Konstante Funktionen, also $g : \mathbb{N}^k \to \mathbb{N}, x \mapsto c$ für $c \in \mathbb{N}$.
- Projektionen, notiert $\pi_{k,i} : \mathbb{N}^k \to \mathbb{N}, x \mapsto x_i$.
- Die Nachfolgerfunktion $succ : \mathbb{N} \to \mathbb{N}, x \mapsto x + 1$.

Komposition von prim.-rek. Funktionen ist wieder prim.-rek.
Das heißt, für prim.-rek. Funktionen $g : \mathbb{N}^a \to \mathbb{N}$ und $h_1, h_2, \ldots, h_a : \mathbb{N}^b \to \mathbb{N}$ ist

$$f : \mathbb{N}^b \to \mathbb{N}, x \mapsto g(h_1(x), h_2(x), \cdots, h_a(x))$$

wieder prim.-rek.

Weiter können wir via primitiver Rekursion neue prim.-rek. Funktionen aus alten bauen:
Seien $g : \mathbb{N}^k \to \mathbb{N}, h : \mathbb{N}^{k+2} \to \mathbb{N}$ prim.-rek. Dann ist $f : \mathbb{N}^{k+1} \to \mathbb{N}$, definiert durch:

$$f(0, x_1, \cdots, x_k) := g(x_1, \cdots, x_k)$$

$$f(n+1, x_1, \cdots, x_k) := h(n, f(n, x_1, \cdots, x_k), x_1, \cdots, x_k)$$

wieder prim.-rek.

Primitiv-rekursive Funktionen sind stets berechenbar und total.

Notation: Prädikatsfunktion

Wir schreiben $[x \geq 1]$ für $f : \mathbb{N} \to \{0, 1\}, x \mapsto \begin{cases} 1 & , x \geq 1 \\ 0 & , \text{sonst} \end{cases}$. Generell $[P]$ für ein Prädikat P.

Bemerkung: Nützliche primitiv-rekursive Funktionen

- $\text{add} : \mathbb{N}^2 \to \mathbb{N}, (x, y) \mapsto x + y$
- $\text{mult} : \mathbb{N}^2 \to \mathbb{N}, (x, y) \mapsto x \cdot y$
- $\text{pred} : \mathbb{N} \to \mathbb{N}, x \mapsto \max(x - 1, 0)$
- $\text{sub} : \mathbb{N}^2 \to \mathbb{N}, (x, y) \mapsto \max(x - y, 0)$
- $[x = y], [x < y], [x \leq y] = \text{leq}$
- $\text{sgn} = [x \geq 1]$
- $\text{binom}(n, k) = \binom{n}{k}$
- $\max(x, y), \min(x, y)$
- $[x \text{ ungerade}], [x \mid y], [x \text{ prim}]$
- $x \bmod y, \quad \text{ggT}(x, y), \quad x \text{ DIV } y = \lfloor \frac{x}{y} \rfloor$

Bemerkung Bijektion $\mathbb{N}^2 \to \mathbb{N}$

Wir haben eine **primitiv-rekursive** Bijektion $\beta : \mathbb{N}^2 \to \mathbb{N}$, gegeben durch

$$\beta(x, y) := \binom{x + y + 1}{2} + x = \frac{1}{2}(x + y)(x + y + 1)$$

Aus dieser lässt sich eine primitiv-rekursive Bijektion $\langle \cdot \rangle : \mathbb{N}^k \to \mathbb{N}$ bauen, gegeben durch

$$\langle x_1, \cdots, x_k \rangle := \beta(x_1, \beta(x_2, \beta(x_3, \cdots, \beta(x_{k-1}, x_k) \cdots)))$$

Ferner gibt es primitiv-rekursive Umkehrfunktionen γ, δ mit $\forall n \in \mathbb{N} : \beta(\gamma(n), \delta(n)) = n$. Aus diesen lassen sich primitiv-rekursive Umkehrfunktionen $u_1, \cdots, u_k$ für $\langle \cdot \rangle$ bauen.

Satz: Äquivalenz primitiv-rekursiv und LOOP

Die Menge der primitiv-rekursiven Funktionen fällt mit der Menge der LOOP-berechenbaren zusammen.

Beweisidee: LOOP in prim.-rek.: Für jedes Programm P eine Funktion $g_P : \mathbb{N} \to \mathbb{N}$, sodass

$$P[a] = b \iff g_P(\langle a \rangle) = \langle b \rangle$$

- Zuweisungen via entsprechender Veränderung des Eingabetupels, also

  $$P \equiv x_i := x_j + c \qquad \text{durch}$$
  $$g_P(x) := \langle u_0(x), \cdots, u_{i-1}(x), u_j(x) + c, u_{i+1}(x), \cdots, u_k(x) \rangle$$

- Hintereinanderausführung via Komposition, also

  $$P \equiv Q; R \qquad \text{durch} \qquad g_P(x) := g_R(g_Q(x))$$

  Für LOOP-Programme Q, R.

- LOOPS durch wiederholtes Hintereinanderausführen:

  $$P \equiv \text{LOOP } x_i \text{ DO } Q \text{ ENDLOOP} \qquad \text{durch}$$
  $$g_P(x) := h_{g_Q}(x_i, x) = \underbrace{g_Q(g_Q(\cdots g_Q(g_Q(x)) \cdots))}_{x_i}$$

  Wobei $h_{g_Q}(0, x) := x$ und $h_{g_Q}(n + 1, x) := g_Q(h_{g_Q}(n, x))$ primitiv rekursiv.

- Mit $m \leq k$ Eingabevariablen ist die Simulation dann $u_0(g_P(\langle 0, x_1, \cdots, x_m, \underbrace{0, \cdots, 0}_{k-m} \rangle))$.

Rückrichtung: prim.-rek. in LOOP:

- Konstante Funktionen durch $x_0 := c$.
- Projektion $\pi_{k,j}$ durch $x_0 := x_j$.
- Nachfolgerfunktion $succ(x_j)$ durch $x_0 := x_j + 1$.
- Komposition durch "geeignetes Hintereinanderausführen".
- Primitive Rekursion durch "bottom-up":

  $x_0 := g(x_1, \cdots, x_k)$;
  $s := 0$;
  LOOP n DO
  $x_0 := h(s, x_0, x_1, \cdots, x_k)$;
  $s := s + 1$;
  ENDLOOP;

Definition: Der Kleen'scher μ-Operator

Es gilt im folgenden $\min \varnothing = \perp$. (Weiter ist implizit vereinbart dass $f(\perp) = \perp$ für jedes f)

Es sei $g : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (partielle oder totale) Funktion. Der μ-Operator ist definiert durch:

$$\mu g : \mathbb{N}^k \to \mathbb{N} \cup \{\perp\}, x \mapsto \min\{n \in \mathbb{N} \mid g(n, x) = 0 \wedge \forall m < n : g(m, x) \neq \perp\}$$

Die resultierende Funktion gibt also die kleinste Nullstelle zu den festen letzten k Parametern, oder $\perp$ wenn diese nicht existiert.

Bspw. ist zu $g : \mathbb{N}^3 \to \mathbb{N}$ mit $g \equiv 1$ dann $\mu g \equiv \perp$

Der μ-Operator lässt sich wiederholt anwenden.

Definition: Klasse der μ-rekursiven Funktionen

Diese Klasse von (partiellen und totalen) Funktionen ist die kleinste, welche die Basisfunktionen enthält und abgeschlossen unter Komposition, primitiver-rekursion und des μ-Operators ist.

Satz: μ-rekursive Funktionen sind Turing-mächtig

Die Menge der μ-rekursiven Funktionen fällt mit der Menge der WHILE-/TM-/RAM-berechenbaren Funktionen zusammen.

Beweisidee: Es genügt zu zeigen, dass WHILE-Schleifen und μ-Operatoren sich gegenseitig simulieren können. Restliche Befehle sind analog zu LOOP-Programmen.

Man simuliert ein Programm $P \equiv$ WHILE $x_i \neq 0$ DO Q ENDWHILE durch

$$g_P(x) := h_{g_Q}(\mu(u_i \circ h_{g_Q})(x), x)$$

Wobei $h_{g_Q}(n, x) = g_Q(g_Q(\cdots g_Q(x) \cdots))$ wie oben. Man benutzt μ um das minimale n zu finden, sodass $u_i(h_{g_Q}(n, x)) = 0$, also $x_i = 0$ nach n-maligem Anwenden von Q, was eben die WHILE-Schleife darstellt.

Das WHILE-Programm zu einer Funktion, welche aus Anwendung des μ-Operators auf ein $g : \mathbb{N}^{k+1} \to \mathbb{N}$ entstanden ist, lautet wie folgt:

$x_0 := 0$;
$y := g(0, x_1, \cdots, x_k)$;
WHILE $y \neq 0$ DO
    $x_0 := x_0 + 1$;
    $y := g(x_0, x_1 \cdots, x_k)$;
ENDWHILE;

12 P versus NP

- Worst Case Laufzeit
- Polynomielle Algorithmen
- Komplexitätsklasse P
- Non-deterministische Turingmaschinen (NTM's)
- Laufzeit einer NTM
- Komplexitätsklasse NP
- Zertifikat-Charakterisierung von NP

Definition: Worst Case Laufzeit

Die Wort Case Laufzeit eines Algorithmus A sind die maximalen Laufzeitkosten auf Eingaben der Länge n bezüglich des logarithmischen Kostenmaßes der RAM. Wir schreiben dazu $t_A(n)$.

Definition: Polynomielle Algorithmen

Ein Algorithmus A heißt polynomiell (beschränkt), wenn

$$\exists \alpha \in \mathbb{N} : t_A(n) \in \mathcal{O}(n^\alpha)$$

Definition: Komplexitätsklasse P

P ist die Klasse aller Entscheidungsprobleme, für die es einen polynomiellen Algorithmus gibt.

Definition: Non-deterministische Turingmaschine (NTM)

Einziger Unterschied ist die Zustandsübergangs**relation**, für die nun

$$\delta \subseteq ((Q \setminus \{\overline{q}\}) \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

Die mögl. Rechenwege einer NTM können in einem Berechnungsbaum zsmgefasst werden. Dabei entsprechen Knoten Konfigurationen, die Wurzel der Startkonfiguration, die Kinder einer Konfiguration allen möglichen Nachfolgekonfigurationen.

Wir definieren den maximalen Verzweigungsgrad

$$\Delta := \max\{|\delta(q, a)| : q \in Q \setminus \{\overline{q}\}, a \in \Gamma\}$$

Eine NTM M akzeptiert eine Eingabe $x \in \Sigma^*$, falls es **mindestens** einen Rechenweg von M gibt, indem M die Eingabe x akzeptiert (im Sinne einer normalen TM). Dann ist in diesem Sinne:

$$L(M) := \{x \in \Sigma^* \mid M \text{ akzeptiert } x\}$$

Definition: Laufzeit einer NTM

Sei M eine NTM und $x \in \Sigma^*$ eine Eingabe. Die Laufzeit $T_M(x)$ ist gegeben durch:

- Falls $x \in L(M)$, so ist $T_M(x)$ die Länge des **kürzesten akzeptierenden** Rechenweges.
- Falls $x \notin L(M)$, so ist $T_M(x) := 0$.

Die **Worst Case Laufzeit** der NTM M auf Eingaben der Länge n ist dann

$$t_M(n) := \max\{T_M(x) \mid x \in \Sigma^n\}$$

Definition: Komplexitätsklasse NP

NP ist die Klasse alle Entscheidungsprobleme, die durch eine NTM M erkannt werden, deren Worst Case Laufzeit $t_M(n)$ polynomiell beschränkt ist.

Satz: Zertifikat Charakterisierung von NP

Eine Sprache $L \subseteq \Sigma^*$ liegt genau dann in NP, wenn es einene polynomiellen deterministischen Algorithmus V und ein Polynom p gibt, sodass:

$$\forall x \in \Sigma^* : x \in L \iff \exists y \in \{0,1\}^*, |y| \leq p(|x|) : y\#x \in L(V)$$

V heißt hier Verifizierer, das Wort $y \in \{0,1\}^*$ Zertifikat.

13 Polynomielle Reduktionen

- Lösungen zu SAT konstruieren
- Optimierungsprobleme
- EXPTIME
- NP $\subseteq$ EXPTIME
- Polynomielle Reduktionen

Satz: Lösungen zu SAT konstruieren

Wir betrachten SAT Instanzen mit n Variablen und m Klauseln. Angenommen, Algorithmus A entscheidet SAT Instanzen in $T(n, m)$ Zeit. Dann gibt es einen Algorithmus B, der zu erfüllbaren SAT Instanzen in $n \cdot T(n, m)$ eine Variablenbelegung konstruiert.

Beweisidee: Lege die Variablen einzeln fest und überprüfe ob die Formel erfüllbar bleibt.

Definition: Optimierungsprobleme

Die Eingabe eines Optimierungsproblems spezfiziert eine Menge $\mathcal{L}$ von zulässigen Lösungen zusammen mit einer Zielfunktion $f : \mathcal{L} \to \mathbb{N}$, die Kosten, Gewicht, oder Profit misst. Das Ziel ist dann, eine Optimale Lösung in $\mathcal{L}$ zu bestimmen.

Diese Probleme lassen sich oft in "sehr ähnliche" Entscheidungsprobleme umwandeln. Hierbei fügt man dem Problem eine Schranke hinzu und fragt dann, ob es eine Lösung gibt, welche dieser Schranke genügt.

Am Beispiel KP (Rucksackproblem): Angenommen $A \in \mathsf{P}$ löst das zugehörige Entscheidungsproblem

- Bestimme mit binärer Suche den optimalen Zielfunktionswert
  (min. Profit 0, max. Summe aller Profite). Dies ist immernoch polynomiell
- Bestimme beste Gegenstandswahl durch $n + 1$ Aufrufe des letzten Algoritmus:
  Teste ob optimaler Wert erreichbar wenn wir Gegenstand $k \in [1, n]$ nicht mitnehmen.

Definition: EXPTIME

EXPTIME ist die Klasse aller Entscheidungsprobleme, die durch eine DTM M entschieden werden, dessen Worst Case Laufzeit $t(n)$ durch $2^{q(n)}$ mit einem Polynom q beschränkt ist.

Laufzeit-Beispiele: $2^{\sqrt{n}}, 2^n, 3^n, n!, n^n$

Satz: $\mathsf{NP} \subseteq \mathsf{EXPTIME}$

Beweisidee:

- Sei $L \in \mathsf{NP}$. Benutze Zertifikat-Charakterisierung und erhalte Verifizierer $V \in \mathsf{P}$.
- Nummeriere alle möglichen Zertifikate $y \in \{0,1\}^*$ mit $|y| \leq p(|x|)$ und teste jedes mit V.
- Es gibt ca. $2^{p(|x|)}$ von diesen möglichen Zertifikaten, und V ist polynomiell in $|x| + |y|$
- Damit ist die Gesamtzeit ca. $poly(|x|) \cdot 2^{p(|x|)}$

Definition: Polynomielle Reduktionen

Es seien $L_1, L_2 \subseteq \Sigma^*$. Dann ist L_1 polynomiell reduzierbar auf L_2, geschrieben $L_1 \leq_p L_2$, wenn ein polynomiell berechenbares $f : \Sigma^* \to \Sigma^*$ existiert sodass:

$$\forall x \in \Sigma^* : x \in L_1 \iff f(x) \in L_2$$

Analog zu den Entscheidbarkeitsreduktionen ergibt sich:

$$(L_2 \in \mathsf{P}) \wedge (L_1 \leq_p L_2) \implies L_1 \in \mathsf{P}$$

14 Satz von Cook und Levin

- NP-Schwierigkeit
- NP-Vollständigkeit
- SAT ist NP-vollständig (Cook & Levin)
- Kochrezept für NP-Vollständigkeitsbeweise
- 3-SAT + NP-Vollständigkeit

Definition: NP-Schwierigkeit

Ein Problem L heißt NP-schwer (NP-hard) falls gilt:

$$\forall L' \in \mathsf{NP} : L' \leq_p L$$

Es folgt sofort

$$(L \text{ NP-schwer}) \wedge (L \in \mathsf{P}) \implies \mathsf{P} = \mathsf{NP}$$

sowie

$$(L^* \text{ NP-schwer}) \wedge (L^* \leq_p L) \implies L \text{ NP-schwer}$$

Definition: NP-vollständig

Ein Problem L heißt NP-vollständig (NP-complete) falls $L \in \mathsf{NP}$ und L NP-schwer.

Die Klasse der NP-vollständigen Probleme wird mit NPC bezeichnet.

Satz: SAT ist NP-vollständig (Cook & Levin)

Beweisidee:

- Betrachte zu $L \in \mathsf{NP}$ eine NTM M mit $L(M) = L$.
- Kodiere Verhalten von M in Variablen:
  1. Variablen $Q(t, q)$, die 1 sind, gdw. M zum Zeitpunkt t in Zustand q.
  2. Variablen $H(t, j)$, die 1 sind, gdw. Kopf an Position j zum Zeitpunkt t.
  3. Variablen $B(t, j, a)$, die 1 sind, gdw zum Zeitpunkt t in Zelle j das Symbol a steht.
- Erstelle Klauseln:
  1. Zu jedem Zeitpunkt beschreiben die Variablen valide Konfiguration.
  2. Konfiguration bei Zeit $t + 1$ entsteht legal aus der zur Zeit t.
  3. Start-/Endkonfiguration sind legal.
- Geht alles in polynomiell vielen Klauseln aus jeweils polynomiell vielen Literalen.

Kochrezept: Kochrezept für NP-Vollständigkeitsbeweise

1. Man zeige $L \in \mathsf{NP}$.
2. Man wähle eine NP-vollständige Sprache L^*.
3. (Reduktionsabbildung): Man konstruiere eine Funktion f, die Instanzen von L^* auf Instanzen von L abbildet.
4. (Polynomielle Zeit): Man zeige, dass f polynomiell beschränkt ist.
5. (Korrektheit): Man beweise, dass f tatsächlich eine Reduktion ist (also $L^* \leq_p L$).

Problem: 3-SAT + NP-Vollständigkeit

3-SAT ist wie SAT, nur dass alle Formeln in 3-CNF (jede Klausel genau 3 Terme) sein müssen.

Beweisidee:

- 3-SAT $\in \mathsf{NP}$ als Spezialfall von SAT.
- Klauseln mit < 3 Termen durch Variablenwiederholung aufstocken.
  Klauseln mit > 3 Termen durch Hilfsvariablen trennen:

$$(x_1 + x_2 + x_3 + x_4) \qquad \rightarrow \qquad (x_1 + x_2 + h) + (\overline{h} + x_3 + x_4)$$

- Offensichtlich polynomiell und korrekt.

15 NP-vollständige Graphprobleme

- CLIQUE ist NP-vollständig
- INDEP-SET ist NP-vollständig
- VERTEX-COVER ist NP-vollständig
- D-HAM-CYCLE ist NP-vollständig
- HAM-CYCLE ist NP-vollständig
- TSP, Δ-TSP und $\{1,2\}$-TSP sind NP-vollständig

Satz: CLIQUE ist NP-vollständig

Beweisidee: Man reduziert SAT $\leq_p$ CLIQUE.

- Jedes Literal entspricht einem Knoten
- Knoten haben eine Kante gdw. Sie in verschied. Klauseln vorkommen und sich nicht widersprechen ($x_2, \neg x_2$ würde sich widersprechen).
- Bei m Klauseln sucht man nun eine CLIQUE von (mindestens) m Knoten in dem resultierenden Graph, die Eingabe k für CLIQUE wird also auf m gesetzt.

Satz: INDEP-SET ist NP-vollständig

Beweisidee: Zeige CLIQUE $\leq_p$ INDEP-SET via $f(V, E, k) := (V, V^2 \setminus E, k)$.

Satz: VERTEX-COVER ist NP-vollständig

Beweisidee: Zeige INDEP-SET $\leq_p$ VERTEX-COVER via $f(V, E, k) := (V, E, |V| - k)$.

Satz: D-HAM-CYCLE ist NP-vollständig

Beweisidee: Diamantengadgets, siehe VL-Folien für mehr Details.

Satz: HAM-CYCLE ist NP-vollständig

Beweisidee: Zeige D-HAM-CYCLE $\leq_p$ HAM-CYCLE, indem man jeden Knoten des gerichteten Graphen auf 3 Knoten (input, mid, output) des ungerichteten Graphen abbildet.

Satz: TSP, Δ-TSP und $\{1,2\}$-TSP sind NP-schwer.

Beweisidee: Es genügt zu zeigen, dass $\{1,2\}$-TSP NP-schwer ist.

- Zeige HAM-CYCLE $\leq_p \{1,2\}$-TSP.
- Jeder Knoten des Eingabegraphen wird zu einer "Stadt"
- Setze $\forall u, v \in \text{Städte} : d(u,v) := \begin{cases} 1 & , (u,v) \in E \\ 2 & , (u,v) \notin E \end{cases}$
- Graph hat Hamiltonkreis gdw. TSP-Instanz eine Tour mit Länge $\leq |V_G|$ hat.

16 NP-vollständige Zahlprobleme

- SUBSET-SUM ist NP-vollständig
- PARTITION ist NP-vollständig
- BPP ist NP-vollständig
- KP ist NP-vollständig
- Kodierungen
- Number
- Pseudo-polynomielle Zeit
- Stark NP-schwer
- THREE-PARTITION + stark NP-schwer

Satz: SUBSET-SUM ist NP-vollständig

Beweisidee: Man zeigt 3-SAT $\leq_p$ SUBSET-SUM.

- Codierte Dezimalzahlen mit $n+m$ Ziffern, wobei wir n Variablen und m Klauseln haben.
- Für jede Variable x_i erstelle 2 Zahlen a_i^+, a_i^-, dessen $i-te$ Ziffer 1 ist, sonst 0. Ferner hat jede der erstellten Ziffern eine 1 im Ziffernbereich $n+j, j \in [1,m]$ wenn das zugehörige Literal $(x_i, \overline{x_i})$ in Klausel c_j vorkommt. Man führt noch 2 Dummy-Zahlen d_j, d'_j für jede Klausel ein, welche nur an Ziffer $n+j$ eine 1 haben (eine Klausel kann 3 mal die gleiche Variable enthalten bspw.)
- Polynomiell da Eingabe $n+m$ Dinge (Variablen / Klauseln) hat und damit Länge $\geq n+m$. Die konstruierte Instanz benötigt $\mathcal{O}(n+m)$ Zahlen der Länge $n+m$. Die Reduktion erfolgt also in $\mathcal{O}((n+m)^2)$.

Satz: PARTITION ist NP-vollständig

Beweisidee: Man zeige SUBSET-SUM $\leq_p$ PARTITION.

- Bilde Instanz $(a_1, ..., a_n, b)$ von SUBSET-SUM auf $(a_1, ..., a_n, a_{n+1}, a_{n+2}, b)$ mit $a_{n+1} := 2S - b, a_{n+2} := S + b$, wobei ObdA $b \leq S := \sum_{i=1}^{n} a_i$. Dann ist $\sum_{i=1}^{n+2} a_i = 4S$.
- Wenn es eine Lösung zur SUBSET-SUM Instanz gibt, fügen wir $a_{n+1} = 2S - b$ hinzu und erhalten den gesuchten Summenwert $2S = 4S/2$.
- Wenn es eine Lösung zur PARTITION Instanz gibt, so sind a_{n+1}, a_{n+2} nicht in der selben Teilmenge, da $a_{n+1} + a_{n+2} > 2S$. Da die Teilmenge mit $a_{n+1} = 2S - b$ die Gesamtsumme $2S$ hat, gibt es also eine Teilmenge der $a_i, i \in [1,n]$ mit Gesamtsumme $2S - a_{n+1} = b$.

Satz: BPP ist NP-vollständig

Beweisidee: Man zeige PARTITION $\leq_p$ BPP. Die Instanz $a_1, ..., a_n$ mit $\sum_{i=1}^{n} a_i = 2A$ bildet man auf die BPP Instanz mit Gewichten $w_i = a_i$, maximaler Tragkraft $B = A$ und maximaler Anzahl an Kisten $\gamma = 2$ ab.

Satz: KP ist NP-vollständig

Beweisidee: Man zeige SUBSET-SUM $\leq_p$ KP. Die Instanz $a_1, ..., a_n, b$ von SUBSET-SUM bildet man auf die KP Instant mit Gewichten $w_i = a_i$, Profiten $p_i = a_i$, sowie Tragkraft B und minimaler Gesamtprofit $\gamma = B = b$. Die Gewichte überschreiten b nicht, was $\sum a_i \leq b$ versichert, und der Profit soll mindestens b sein, was dann $\sum a_i \geq b$ garantiert.

Anmerkung: Kodierungen

- Kodierungslänge einer Instanz eines Problems ist die Anzahl der Symbole in einer "vernünftigen" Beschreibung. Polynomiell große Änderungen in dieser sind für die Resultate der VL irrelevant.
- Bspw Graphen; Adjazenzlisten $\ell_1(G) = \mathcal{O}(|E| \log |V|)$ und Adjazenzmatrizen $\ell_2(G) = \mathcal{O}(|V|^2)$. Dann ist $\ell_1(G)$ polynomiell beschränkt in $\ell_2(G)$ und vice versa.
- Zahlen: $\log_a(n) = \underbrace{\log_a(b)}_{\text{konst. Faktor}} \cdot \log_b(n)$ für $a, b > 1$.
- Wert eine Zahl n hängt **exponentiell** von seiner Kodierungslänge ab.

Definition: Number

Für eine Instanz I eines Entscheidungsproblems ist $Number(I)$ der **Wert** der größten in I vorkommenden Zahl. Bspw für TSP die längste Distanz, für SUBSET-SUM $\max\{a_1, ..., a_n, b\}$ und für SAT $\max\{n, m\}$.
Der Parameter ist eher irrelevant in Problemen ohne Zahlen, wie bspw SAT.

Definition: Pseudo-polynomielle Zeit

Ein Algorithmus A löst ein Problem X in pseudo-polynomieller Zeit, falls die Laufzeit von A auf Instanzen I von X polynomiell in $|I|$ und $Number(I)$ beschränkt ist.

Anmerkung: SUBSET-SUM, PARTITION, KP sind pseudo-polynomiell lösbar

Definition: Stark NP-schwer

Ein Entscheidungsproblem X ist stark NP-schwer, wenn es ein Polynom $p : \mathbb{N} \to \mathbb{N}$ gibt, sodass X eingeschränkt auf Instanzen I mit $Number(I) \leq p(|I|)$ immernoch NP-schwer ist.
(obwohl in diesem Fall alle Zahlenwerte der Instanz I nur polynomiell in $|I|$ sind).
Stark NP-schwere Probleme sind bspw: SAT, HAM-CYCLE, TSP, BPP.

Satz: Beweismethode P = NP (Bzw. Auschlusskriterium NP-schwer und pseudopolynomiell)

Es sei X ein stark NP-schweres Entscheidungsproblem. Falls X pseudo-polynomiell lösbar ist, so gilt P = NP. Denn wir könnten dann das NP-schwere X_q polynomiell in der Eingabelänge $|I|$ lösen.

Problem: THREE-PARTITION + stark NP-schwer

Eingabe: Positive ganze Zahlen $a_1, ..., a_n, b_1, ..., b_n, c_1, ..., c_n$ mit $\sum(a_i + b_i + c_i) = nS$.

Frage: Gibt es zwei Permutationen $\pi, \sigma \in S_n$ sodass $a_{\pi(i)} + b_{\sigma(i)} + c_i = S$ für $i \in [1, n]$?

Dieses Problem ist stark NP-schwer.

17 Jenseits von P und NP

- coNP + Probleme
- Lineare Programmierung (LP)
- coNP-vollständigkeit + zsmhang NP-vollständigkeit
- Mengenbeziehungen P, NP, coNP
- GRAPH-ISOMORPHUS + Satz von Laszlo Babai
- Exponential Time Hypothesis (ETH)
- NP-intermediate, PSPACEP, NPSPACE + Probleme

Anmerkung

Viele Optimierungsprobleme, welche im echten Leben auftauchen sind NP-schwer. In vielen fällen lässt sich das Problem dennoch vernünftig bewältigen - da man sich oft auf eine gewisse Variante einschränken kann. Wichtige Strategien sind:

- Ausnutzen der Eingabestruktur durch spezielle Algorithmen.
- Parametrisierte Algorithmen
- Approximationslagorithmen und Heuristiken
- Zerteilen in geeignete Unterprobleme

Definition: coNP

Ein Entscheidungsproblem $L \subseteq \Sigma^*$ liegt in coNP, wenn für jedes Wort $x \notin L$ ein polynomiell langes Zertifikat y existiert, welches zusammen mit x in polynomieller Zeit (deterministisch) verifiziert werden kann.

Das Zertifikat symbolisiert eine kurze Widerlegung. Für NP muss man JA-Instanzen kurz beweisen, für coNP NEIN-Instanzen.

coNP Probleme:

- Non-Ham-Cycle:
  Hat ein gegebener ungerichteter Graph keinen Hamiltonkreis?
  coNP-Zertifikat: Hamiltonkreis.
- UNSAT:
  Ist eine gegebene boolesche Formel in CNF nicht erfüllbar?
  coNP-Zertifikat: erfüllende Variablenbelegung.
- TAUTOLOGY:
  Ist eine gegebene boolesche Formel in DNF eine Tautologie?
  coNP-Zertifikat: falsifizierende Variablenbelegung.

Diese 3 Probleme sind insbesondere coNP-vollständig (s.u.).

Problem: Lineare Programmierung (LP)

Eingabe: $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n, \gamma \in \mathbb{R}$.

Optimierungsvariante Primal: Für $x \in \mathbb{R}^n_{\geq 0}$ maximiere $c^{tr}x$ sodass $Ax \leq b$.

Optimierungsvariante Dual: Für $y \in \mathbb{R}^m_{\geq 0}$ minimiere $b^{tr}y$ sodass $yA \geq c$.

Entscheidungsvariante: Existiert $x \in \mathbb{R}^n$ mit $x \geq 0 \ \wedge \ Ax \leq b \ \wedge \ c^{tr}x \geq y$?

Der **Starke Dualitätssatz** besagt, dass wenn beide OptimierungsLP's zulässige Lösungen haben, diese den selben optimalen Zielfunktionswert ($c^{tr}x$ bzw. $b^{tr}y$) haben.

Es ist LP $\in$ NP mit einem Vektor x sodass $c^{tr}x \geq \gamma$ als Zertifikat für das primale LP.

Ebenso ist LP $\in$ coNP mit einem y sodass $b^{tr}y < \gamma$ als Zertifiakt für das duale LP.

Es folgt LP $\in$ NP $\cap$ coNP.

Ferner sagt ein Satz von **Leonid Genrikhovich Khachiyan** dass LP $\in$ P

Definition: coNP-vollständigkeit

Ein Entscheidungsproblem $X \in$ coNP ist coNP-vollständig, wenn $Y \leq_p X$ für jedes $Y \in$ coNP.

Satz: Zusammenhang NP- bzw. coNP-vollständig

Beim Komplement eines Entscheidungsproblems werden JA und NEIN Instanzen vertauscht. Es gilt $X \in$ NPC $\implies \overline{X} \in$ coNPC.

Satz: P $\subseteq$ NP $\cap$ coNP

Beweis: NP $\supseteq$ P = coP $\subseteq$ coNP.

Satz:

$X \in$ NPC $\cap$ coNP $\implies$ NP = coNP.
Beweis:

$$X \in \text{NPC} \quad \implies \quad \forall L \in \text{NP} : L \leq_p X \in \text{coNP} \quad \implies \quad \text{NP} \subseteq \text{coNP}$$

Analog gilt

$$X \in \text{NPC} \quad \implies \quad \forall L \in \text{NP} : L \leq_p X \quad \implies \quad \forall K \in \text{coNP} : K \leq_p \overline{X} \in \text{NP} \quad \implies \quad \text{coNP} \subseteq \text{NP}$$

Anmerkung

Viele Mathematiker denken, dass $\mathsf{NP} \cap \mathsf{coNP} = \mathsf{P}$. Es sind 3 Probleme aus $\mathsf{NP} \cap \mathsf{coNP}$ bekannt: LP, PRIMES, PARITY-GAME. Davon sind LP,PRIMES auch in P und der Status für PARITY-GAME ist ungeklärt.

Problem: GRAPH-ISOMORPHUS (GI)

Eingabe: Zwei ungerichtete Graphen G_1, G_2.

Frage: Gibt es einen Isomorphismus von G_1 nach G_2?

Das Problem liegt in NP: Man verwende den Isomorphismnus als Zertifikat.

Ungeklärt ist, ob $\text{GI} \in \mathsf{P}, \quad \text{GI} \in \mathsf{NPC}, \quad \text{GI} \in \mathsf{coNP}$?

Satz von Laszlo Babai: GI kann auf Graphen mit n Knoten in $2^{p(\log n)}$ Zeit gelöst werden, für p Polynom.

ETH: Exponential Time Hypothesis

"Es existiert eine reelle Zahl $\delta > 0$, sodass kein Algorithmus 3-SAT in Zeit $\mathcal{O}(2^{\delta n})$ löst."

Oder auch

$$\exists \delta > 0 : \forall A : A \text{ löst 3-SAT} \implies t_A(n) \notin \mathcal{O}(2^{\delta n})$$

Diese Aussage ist **unbewiesen** und impliziert $\mathsf{P} \neq \mathsf{NP}$.
Falls $\text{GI} \in \mathsf{NPC}$, so ist die Hypothese falsch.

Definition: NP-intermediate

Ein Entscheidungsproblem L heißt NP-intermediate, wenn

$$L \in \mathsf{NP} \quad \wedge \quad L \notin \mathsf{P} \quad \wedge \quad L \notin \mathsf{NPC}$$

Der **Satz von Ladner** sagt, $\mathsf{P} \neq \mathsf{NP} \implies \exists L : L$ ist NP-intermediate.

Definition: PSPACE und $\mathsf{NPSPACE}$

PSPACE ($\mathsf{NPSPACE}$) ist die Klasse aller Entscheidungsprobleme, welche von einer DTM (NTM) entschieden werden, deren Worst-Case Speicherplatzbedarf polynomiell beschränkt ist.

Satz von Savitch: $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

Ferner gilt $\mathsf{NP} \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$. Letzteres, da eine Speicherplatzbeschränkung $s(n)$ die Anzahl der Konfigurationen und damit die Laufzeit durch $2^{\mathcal{O}(s(n))}$ beschränkt.

Problem: Q-SAT

Eingabe: Eine Boolesche Formel φ in CNF über Variablenmenge $\{x_1, ..., x_n, y_1, ..., y_n\}$.

Frage: $\exists x_1 \forall y_1 \exists x_2 \forall y_2 \cdots \exists x_n \forall y_n : \varphi$?

Q-SAT ist PSPACE-vollständig

Problem: k-Schritt-HALTEPROBLEM

Eingabe: Eine DTM M, $k \in \mathbb{N}$.

Frage: Hält M auf ε nach höchstens k Schritten?

Dieses Problem ist EXPTIME-vollständig.

Bemerkung Zusammenfassung Komplexitätsklassen

Es gilt

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{NPSPACE} = \mathsf{PSPACE} \subseteq \mathsf{EXPTIME}$$

Wir wissen, dass $\mathsf{P} \neq \mathsf{EXPTIME}$. Alle anderen Verhältnisse sind unbekannt.