# Probleme

## Entscheidbare Probleme

- Gegeben CFG $\langle G \rangle$, ist $L(G)$ leer / endlich / $w \in L(G)$ für ein festes $w \in \Sigma^*$.

## 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 $H := \{\langle M \rangle w \mid M \text{ hält auf } w\}$.
- $\varepsilon$-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 $:= \{\langle p \rangle \mid p \text{ Polynom über } \mathbb{Z} \text{ mit Nullstelle in } \mathbb{Z}\}$

- Gegeben $\langle M \rangle$, ist $L(M) = \Sigma^*$ / leer / (un)endlich / regulär / kontext-frei?
- Gegeben CFG $\langle G \rangle$, ist $G$ eindeutig / $L(G) = \Sigma^*$ / $L(G)$ regulär?
- Gegeben CFG $\langle G \rangle$, enthält $L(G)$ ein Palindrom?
- Gegeben CFGs $\langle G_1 \rangle, \langle G_2 \rangle$, ist $L(G_1) \subseteq L(G_2)$ / $L(G_1) \cap L(G_2) = \varnothing$?

## Rekursiv-aufzählbare Probleme

- $H$
- $H_\varepsilon$
- $\overline{D}$
- Dioph

## 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

## Probleme in NP

- SAT
- 3-SAT
- CLIQUE
- INDEP-SET
- MATCHING
- VERTEX-COVER
- DOMINATING-SET
- COLORING
- HAM-CYCLE
- GI
- TSP
- COMPOSITE
- EX-COVER
- SUBSET-SUM
- PARTITION
- KNAPSACK
- BPP

## Definitionen

**SAT**

Eingabe: Eine Aussagenlogische Formel $\varphi$ in CNF über einer Variablenmenge $X = \{x_1, \cdots, x_n\}$.

Frage: Ist $\varphi$ erfüllbar (ex. Variablenbelegung, sodass $\varphi \equiv 1$)?

**3-SAT**

Eingabe: Eine Aussagenlogische Formel $\varphi$ in 3-CNF über einer Variablenmenge $X = \{x_1, \cdots, x_n\}$.
Dabei ist 3-CNF wie CNF, nur dass jede Klausul exakt 3 Literale haben muss.

Frage: Ist $\varphi$ erfüllbar (ex. Variablenbelegung, sodass $\varphi \equiv 1$)?

**CLIQUE**

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

Frage: Enthält $G$ eine Clique (vollständiger Teilgraph) mit $\geq k$ Knoten?

**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?

**MATCHING** ($\leq$ INDEP-SET)

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).

**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?

**DOMINATING-SET** ($\leq$ VERTEX-COVER)

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).

**COLORING**

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

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

**HAM-CYCLE**

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

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

**GI**

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 $e \in E_1 \iff (f(e_1), f(e_2)) \in E_2$?

**TSP**

Eingabe: Städte $1, \ldots, n$, Distanzen $d(i, j) \in \mathbb{N}$ und $\gamma \in \mathbb{N}$.

Frage: Gibt es eine Rundreise (TSP-Tour) mit Länge $\leq \gamma$?

**COMPOSITE**

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

Frage: Ist $n$ keine Primzahl?

**EX-COVER**

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?

**SUBSET-SUM**

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

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

**KNAPSACK**

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**

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 $k \leq \gamma$?

# VL-Stoff

## 1 Turing Maschinen I

- Probleme
- Turingmaschinen
- rekursive / berechenbare Funktionen
- rekursive / entscheidbare Sprachen
- Konfigurationen
- Programmiertechniken: Speicher im Zustandsraum
- Programmiertechniken: Mehrspurmaschinen
- Programmiertechniken: Weiteres

**Definition:** Probleme

- **Problem als Relation** $R \subseteq \Sigma^* \times \Gamma^*$ für Alphabete $\Sigma, \Gamma$
- Es ist $(x, y) \in R \iff y$ ist zulässige Ausgabe zur Eingabe $x$
- Beispiel Primfaktorbestimmung:

$$R := \{(x, y) \in \{0,1\}^* \times \{0,1\}^* \mid x = \text{bin}(q), y = \text{bin}(p), q, p \in \mathbb{N}, q \geq 2, p \text{ prim}, p \mid q\}$$

- Bei eindeutiger Lösung **Problem als Funktion** $f : \Sigma^* \to \Gamma^*$
- Beispiel Multiplikation inklusive Trennzeichen:

$$f : \{0,1,\#\}^* \to \{0,1,\#\}, f(\text{bin}(i_1)\#\text{bin}(i_2)) = \text{bin}(i_1 \cdot i_2)$$

- **Problem als Entscheidungsproblem**: Form $f : \Sigma^* \to \{0,1\}$
- $L := f^{-1}(1) \subseteq \Sigma^*$ ist Sprache vom durch $f$ definiertem Entscheidungsproblem.
- Beispiel Graphzusammenhang: Bestimme zur Eingabe $G = (V, E)$ ob $G$ zsmhgd.
- Wenn Graph $G$ codiert in $\Sigma$ durch $\mathit{code}(G)$, so ist

$$L := \{w \in \Sigma^* \mid \exists \text{ zusammenhängender Graph } G : w = \mathit{code}(G)\}$$

die zu diesem Entscheidungsproblem gehörende Sprache.

**Definition:** Turingmaschine (TM)

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

- $Q$ endliche Zustandsmenge
- $\Sigma$ 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 Kopfposition und endet vor erstem Symbol in $\Gamma \setminus \Sigma$

- 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
- **Speicherbedarf** Anzahl während Berechnung besuchter Bandzellen

- TM $M$ **entscheidet** $L \subset \Sigma^*$ wenn $M$ $w \in L$ akzeptiert und $w \notin L$ verwirft (hält stets)

- 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

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

**Definition:** rekursive / T-entscheidbare Sprachen

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

**Definition:** Konfiguration

Eine Konfiguration einer TM ist ein String $\alpha q \beta$, wobei $\alpha, \beta \in \Gamma^*, q \in Q$ wobei $\beta \neq \varepsilon$.
$\alpha$ entspricht dem Wort links vom Kopf, 1. Symbol von $\beta$ unter dem Kopf (evtl. $B$), Rest von $\beta$ rechts vom Kopf. Blanks werden dabei ausgelassen (außer es steht unter dem Kopf).

$\alpha' q' \beta'$ ist **direkte Nacholgerkonfiguration** von $\alpha q \beta$,
wenn sie in einem Rechenschritt aus $\alpha q \beta$ entsteht. Man schreibt $\alpha q \beta \vdash \alpha' q' \beta'$.

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

**Techniken zur Programmierung TM's (1):** Speicher im Zustandsraum

Zu $k \in \mathbb{N}_{>0}$ können wir $k$ Symbole des Bandalphabets $\Gamma$ im Zustand abspeichern,
indem wir den Zustandsraum um den Faktor $|\Gamma|^k$ vergrößern, d.h.

$$Q_{neu} := Q \times \Gamma^k$$

Bspw sind neue Zustände für $k = 2$ dann $(q_0, BB)$ oder $(q_1, 01)$ (wenn $0, 1 \in \Gamma$).

**Techniken zur Programmierung TM's (2):** Mehrspurmaschinen

$k$-spurige TM: TM mit zusätzlich $k$-Vektoren als Symbole für $k \in \mathbb{N}$. Man schreibt

$$\Gamma_{neu} := \Gamma \cup \Gamma^k$$

**Techniken zur Programmierung TM's (3):** Weiteres

- Variablen: pro Variable eine Spur
- Arrays: ebenfalls in einer Spur möglich
- Unterprogramme: eine Spur als Prozedurstack benutzen

# 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 $\delta_M$ aus
- Laufe zurück und verändere entsprechend Kopfpositionen / Bandinhalte

- Nach $t$ Schritten von $M$ können #'s 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 mit $code(t)$ in der Form $0^a10^b10^c10^d10^e$ (Siehe Folien).
  Dann kodiert man die TM $M$ mit $s$ Übergängen durch:

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

**Definition:** Universelle Turingmaschine

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

Simulation via 3-Spur TM in **konstanter Zeit** möglich: Gödelnr auf Spur 2, Zustand auf 3.

**Behauptung:** Church-Turing-These (1930)

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

Daher (in dieser Vorlesung)

berechenbare Funktion = TM-berechenbare Funktion = rekursive Funktion

entscheidbare Sprache = TM-entscheidbare Sprache = 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 des Speichers 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".

**Definition:** Kostenmaße für RAM's

**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: Verwende 2 Bänder, Unterprogramme für einzelne Zeilen des Programmes auf dem 1. Dann Registerinhalte auf dem 2.

**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.E. TM mit einseitig beschränktem Band. Reg. 1 für Index des Kopfes, Reg. 2 für Zustand, andere für Bandinhalt. Programm ist if-abfragen nach Zustand und gelesenem Symbol ($\delta$ Modellierung).

**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
- Entscheidungsprobleme
- 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 $M$.
- $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ abzählbar.
- $\{0,1\}^*$ 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 entscheidbaren Sprachen ist abzählbar (da TM's abzählbar).
- Die Menge der unentscheidbaren Sprachen ist überabzählbar.
- Die Menge aller Funktionen $f : \mathbb{N} \to \mathbb{N}$ ist überabzählbar (Nachkommastellen in $(0,1)_{\mathbb{R}}$)

**Bemerkungen:** Entscheidungsprobleme

- Jedes $L \subseteq \{0,1\}^*$ entspricht Entscheidungsproblem über binär-codiertem Alphabet.
- Dann ist $\mathcal{L} = \mathcal{P}(\{0,1\}^*)$ Menge aller Entscheidungsprobleme überabzählbar.
- Da TM's abzählbar, gibt es unentscheidbare Probleme.

**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

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

Ebenso

$$w_j \notin D \implies M_j \text{ akzeptiert } w_j \text{ nicht } \implies w_j \in D$$

Beides 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. (Allein schon, da entscheidbares abgeschl. 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.

**Bemerkungen:** Komplement und Entscheidbarkeit.

Per Unterprogrammtechnik folgt sofot, dass zu einer Sprachen $L \subseteq \{0,1\}^*$ stets gilt

$$L \text{ (un-)entscheidbar } \iff \overline{L} \text{ (un-)entscheidbar}$$

indem man Output des Unterprogramms negiert.

**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

- $\varepsilon$-Halteproblem
- Partielle Funktionen
- Satz von Rice

**Problem:** Epsilon-Halteproblem

Das $\varepsilon$-Halteproblem $H_\varepsilon$, 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 $\langle M' \rangle$ von TM $M'$, welche $M$ auf $w$ simuliert und die Eingabe ignoriert. Lasse dann $M_{H_\varepsilon}$ auf $\langle M' \rangle$ 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:

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

Dabie 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

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

nicht entscheidbar.

- Bsp 1: Sei $\mathcal{S} = \{f_M \mid f_M(\varepsilon) \neq \perp\}$. Dann ist $L(\mathcal{S}) = H_\varepsilon$ unentscheidbar.
- Bsp 2: Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* : f_M(w) \neq \perp\}$.

  $$L(\mathcal{S}) = \{\langle M \rangle \mid M \text{ hält auf jeder Eingabe}\} =: H_{tot}$$

  ist nicht entscheidbar.
- Bsp 3: Sei $\mathcal{S} = \{f_M \mid \forall w \in \{0,1\}^* : f_M(w) = 1\}$. Dann ist $L(\mathcal{S}) = \{\langle M \rangle \mid L(M) = \Sigma^*\}$ unentscheidbar.

Beweisidee: Unterprogrammtechnik mit $H_\varepsilon$. (Siehe Folien für mehr)

Weitere Anwendungsbsp:

- Bsp 4: Sei $L_{17} = \{\langle M \rangle \mid M\text{berechnet bei Eingabe 17 Ausgabe 42}\}$. Dann ist

  $$L_{17} = L(\mathcal{S}) \text{ für } \mathcal{S} = \{f_M \mid f_M(\text{bin}(17)) = \text{bin}(42)\}$$

  Da $\varnothing \neq \mathcal{S} \subsetneq \mathcal{R}$ ist $L_{17}$ unentscheidbar.
- Der Satz von Rice **sagt nichts über Verhalten der TM 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.
- Insbesondere gilt $L$ entscheidbar $\implies$ $L$ semi-entscheidbar.
- Bsp: Das Halteproblem ist semi-entscheidbar. (Einfach simulieren).

**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

$$\exists M : M \text{ erkennt } L \iff L \text{ semi-entscheidbar} \iff L \text{ rekursiv aufzählbar}$$

**Satz:** Abschlusseigenschaften der (Semi-)Entscheidbarkeit

- (Semi-)Entscheidbare Sprachen sind unter $\cup, \cap$ abgeschlossen.
- $L$ und $\overline{L}$ rekursiv aufzählbar $\implies$ $L$ entscheidbar.
- Entscheidbare Sprachen sind unter Komplementbildung abgeschlossen.
- Semi-entscheidbare Sprachen sind **nicht** unter Komplementbildung abgeschlossen. Betrachte dazu bspw $H$ rekursiv aufzählbar, aber nicht entscheidbar.

**Definition:** Reduktion

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

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

**Satz:** Übertragungseigenschaften von Reduktionen

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

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

ebenso gilt

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

ferner haben wir

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

**Problem:** Totales Halteproblem ($H_{tot}$)

Das totale Halteproblem $H_{tot}$ ist definiert durch

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

Es sind $H_{tot}$ und $\overline{H_{tot}}$ nicht rekursiv aufzählbar.

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

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

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 $\ell$ die ersten $\ell$ Schritte von $M$ bei Eingabe $\varepsilon$ simuliert. Hält $M$ in diesen, so geht $M'$ in Endlosschleife, andernfalls hält $M'$.

# 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 $H \leq \text{MPCP} \leq \text{PCP}$. (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

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

**Definition:** Eine kontext-freie Grammatik (CFG) $G$ ist ein Quadrupel $(N, \Sigma, P, S)$ wobei

- $N$ die Menge der Non-Terminalsymbole
- $\Sigma$ das Terminalalphabet
- $P$ die Menge der Regeln der Form $A \to w, A \in N, w \in (\Sigma \cup N)^*$
- $S \in N$ das Startsymbol

Wir definieren $L(G)$ als die Menge aller Worte über dem Terminalalphabet $\Sigma$, die durch wiederholte Anwendung von Regeln in $P$ aus dem Startsymbol $S$ hergeleitet werden können.

**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:

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

- PCP lösbar genau dann, wenn $L_1(G) \cap L_2(G) \neq \varnothing$.

**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 ein Polynom mit ganzzahligen Koeffizienten und mit einer ganzzahligen 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** 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
- $\mu$-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
- LOOP-Programme
- WHILE
- WHILE ist Turing-mächtig
- LOOP-WHILE

**Definition:** Die Programmiersprache LOOP

- Variablen: $x_1, x_2, x_3, \ldots$
- 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

- Variablen: $x_1, x_2, x_3, \ldots$
- Konstanten: 0 und 1
- Symbole: :=, +, ;, $\neq$
- Keywords: WHILE, DO, ENDWHILE

- $x_i := x_j + c$ für $i, j \in \mathbb{N}, c \in 0, 1$ ist ein WHILE-Programm.
- Wenn $P_1, P_2$ WHILE-Programme sind, dann ist $P_1; P_2$ ein WHILE-Programm.
- 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 $\delta$ simuliert. (benötigt explizit sogar nur die äußerste While-Schleife, alles andere könnte mit LOOP simuliert werden).

**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:

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

**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:

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

**Satz:** Wachstumslemma

Sei $P$ ein LOOP-Programm. Dann gilt:

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

Beweisidee: Zeige via (algebraische) Induktion, dass:

- $m_{x_i := x_j + c} = 2$.
- 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. (wächst zu krass)

# 11 Primitiv-rekursive Funktionen

- Primitiv-rekursive Funktionen
- Prädikatsfunktion
- Weitere primitiv-rekursive Funktionen
- Bijektion $\mathbb{N}^2 \to \mathbb{N}$
- Äquivalenz primitiv-rekursiv und LOOP
- Kleenscher $\mu$-Operator
- Klasse der $\mu$-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:** Weitere 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

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

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

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

**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.:

- Zuweisungen via entsprechender Veränderung des Eingabetupels, also

  $$P = [x_i := x_j + c] \qquad \text{durch}$$

  $$g_P(x) := \Omega(\pi_{k,1}(x), \cdots, \pi_{k,i-1}(x), \pi_{k,j}(x) + c, \pi_{k,i+1}(x), \cdots, \pi_{k,k}(x))$$

  Wobei $\Omega : \mathbb{N}^k \to \mathbb{N}$ bijektiv.
- Hintereinanderausführung via Komposition, also

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

  Für LOOP-Programme $Q, R$.
- LOOPS durch wiederholtes Hintereinanderausführen:

  $$P = [\text{LOOP } x_i \text{ DO } Q \text{ ENDLOOP}] \qquad \text{durch}$$

  $$g_P(x) := g_Q^{x_i}(x) = \underbrace{g_Q(g_Q(\cdots g_Q(g_Q(x)) \cdots))}_{x_i}$$

Rückrichtung: prim.-rek. in LOOP: (Erinnerung: Ausgabe bei LOOP-Programm ist $x_0$)

- 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 (siehe def., gleiche Fkt im folgenden) 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'sche $\mu$-Operator

Es gilt im folgenden $\min \varnothing = \bot$.

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

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

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

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

Der $\mu$-Operator lässt sich wiederholt anwenden.

**Definition:** Klasse der $\mu$-rekursiven Funktionen

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

**Satz:** $\mu$-rekursive Funktionen sind Turing-mächtig

Die Menge der $\mu$-rekursiven Funktionen fällt mit der Menge der WHILE-/TM-/RAM-berechenbaren Funktionen zusammen.

Beweisidee: Es genügt zu zeigen, dass WHILE-Schleifen und $\mu$-Operatoren sich gegenseitig simulieren können (da restliche Befehle wie bei LOOP sind und schon von primitiv-rekursiven Funktionen abgedeckt werden).

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

$$g_P(x_1, \cdots, x_k) := g_Q^{\mu\pi_{k,i}(g_Q^n(x))}(x) = \underbrace{g_Q(g_Q(g_Q(\cdots g_Q(x)\cdots)))}_{\mu\pi_{k,i}(\underbrace{g_Q(\cdots g_Q(x)\cdots)}_{n})}$$

Dies benutzt den $\mu$-Operator, um das kleinste $n \in \mathbb{N}$ zu finden, sodass nach $\pi_{k,i}(g_Q^n(x)) = 0$ gilt, also die $i$-te Variable nach $n$-maligem Ausführen von $g_Q$ gleich 0 ist (was ja gerade die Abbruchbedingung der While-Schleife ist). Dies kann auch $\perp$ sein, wenn die While-Schleife nicht terminiert.

Das WHILE-Programm zu einer Funktion, welche aus Anwendung des $\mu$-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
- CLIQUE
- 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, derne Worst Case Laufzeit $t_M(n)$ polynomiell beschränkt ist.

**Problem:** CLIQUE

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und $k \in \mathbb{N}$.
Frage: Enthält $G$ eine Clique (vollständigen Teilgraph) mit $\geq k$ Knoten?

Es gilt CLIQUE $\in$ NP. Beweisidee: Eine NTM $M$ mit $L(M) =$ CLIQUE geht wie folgt vor:

- Syntaxcheck
- Rate non-deterministisch einen 0-1-String $y$ der Länge $|V|$.
- M akzeptiert, falls der String $y$ mindestens $k$ Einsen enthält und die Knotenmenge $C := \{i \in V \mid y_i = 1\}$ eine Clique bildet.

**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:

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

$V$ 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 $A \in$ 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**