

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 Σ, Γ
- 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** $f : \Sigma^* \rightarrow \Gamma^*$
- Beispiel Multiplikation inklusive Trennzeichen:

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

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

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

die zu diesem Entscheidungsproblem gehörende Sprache.

Definition: Turingmaschine (TM)

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

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

Weiteres:

- Startet in q_0 , Kopf über (1. Symbol vom) Eingabewort eingerahmt von Blanks
- TM stoppt, sobald Endzustand \bar{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)

Definition: rekursive / T-berechenbare Funktionen

$f : \Sigma^* \rightarrow \Sigma^*$ 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 $\beta \neq \varepsilon$.
 α entspricht dem Wort links vom Kopf, 1. Symbol von β unter dem Kopf (evtl. B), rest rechts.

$\alpha' q' \beta'$ 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 Γ 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
- Alan Turing
- Alonzo Church
- Church-Turing-These

Definition: k -Band-TM

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

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

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 und den Kopfpositionen auf den zugeh. Bändern im Zustand.
- Werte damit δ_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 $Q = \{q_1, q_2, \dots, q_t\}, t \geq 2$
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 $0^a 10^b 10^c 10^d 10^e$.
Dann kodiert man die TM M mit s Übergängen durch:

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

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.

Person: Alan Turing (1912-1954)

- Englischer Mathematiker, Informatiker, Logiker, Philosoph
- Angesehen als Vater der theoretischen Informatik

Person: Alonzo Church (1903-1995)

- Amerikanischer Mathematiker, Logiker
- Erfinder des Lambda-Calculus

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
entschiedbare 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 $c(1), c(2), \dots$
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

$$\mathcal{O}(t(n) + n) \quad (\text{uniformes Kostenmaß})$$

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

Beweisidee: **O.E. einseitig beschränktes Band** (Positionen \mathbb{N}_0 anstatt \mathbb{Z}).

Reg. 1 für Index des Kopfes, Reg. 2 für Zustand, andere für Bandinhalt. Programm ist if-abfragen nach Zustand und gelesenen (δ Modellierung).

Problem: Collatz-Problem

Wird folgende Funktion bei wiederholter Anwendung stets bei jeder Eingabe den Wert 1 erreichen?

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

4 Unentscheidbarkeit I

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

Definition: Abzählbarkeit

Eine Menge M heißt abzählbar, wenn

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

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** $\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots$
- 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)

Bemerkungen: Entscheidungsprobleme

- Jedes Entscheidungsproblem mit binär kodierter Eingabe entspricht einem $L \subseteq \Sigma^*$
- 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:

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

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 sofort, 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

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

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

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

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

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 $\emptyset \neq \mathcal{S} \subsetneq \mathcal{R}$.
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_ε . (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 $\emptyset \neq \mathcal{S} \subsetneq \mathcal{R}$ ist L_{17} unentscheidbar.

- Der Satz von Rice **sagt nichts über Verhalten der TM aus**.
(Zustand / Anzahl Schritte / genauere Implementierungsdetails).
- Konsequenzen: Es ist nicht möglich, 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 \bar{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: Reduzierbarkeit

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 $f : \Sigma^* \rightarrow \Sigma^*$ 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

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

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 ε 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 ℓ die ersten ℓ Schritte von M bei Eingabe ε simuliert. Hält M in diesen, so geht M' in Endlosschleife, andernfalls hält M' .

7 Postsches Correspondenzproblem

- Emil Leon Post
- Postsches Correspondenzproblem (PCP)
- Einschränkungen des PCP's

Person: Emil Leon Post (1897-1954)

Problem: Postsches Correspondenzproblem (PCP)

Eine Instanz des PCP besteht aus einer endlichen Menge

$$K = \left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \dots, \begin{bmatrix} x_k \\ y_k \end{bmatrix} \right\} \quad \text{für} \quad x_1, \dots, x_k, y_1, \dots, y_k \in \Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$$

Elemente von K nennen wir Dominos. Frage:

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

Die modifizierte 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).

Aus diesen Reduktionen folgt jedoch die **rekursive Aufzählbarkeit** von MPCP und PCP.

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)
- David Hilbert
- Hilberts 10. Problem / Satz von Matiyasevich
- Satz von Davis, Robinson, ... (Ganzzahlige Polynome gleichstark wie TM's)
- Turing-Mächtigkeit
- John Conway
- Conway's Game of Life

Definition: Eine kontext-freie Grammatik (CFG) G ist ein Quadrupel (N, Σ, P, S) wobei

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

Wir definieren $L(G)$ als die Menge aller Worte über dem Terminalalphabet Σ , 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 $L(G_1) \cap L(G_2) = \emptyset$.

Beweisidee:

- Betrachte PCP-Instanz $\left\{ \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \dots, \begin{bmatrix} x_n \\ y_n \end{bmatrix} \right\}$.

- 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:

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

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

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

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

Person: David Hilbert (1862-1943)

Deutscher Mathematiker. Hilberts 10. Problem.

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

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

$\text{Dioph} = \{\langle p \rangle \mid p \text{ ist ein Polynom mit ganzzahligen Koeffizienten und mit einer ganzzahligen Nullstelle}\}$

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:

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

Er sagt aus, dass **ganzzahlige Polynome so berechnungsstark 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
- μ -rekursiven Funktion von Stephen Kleene
- Alle gängigen höheren Programmiersprachen (C, Java, etc.)
- Postscript, Tex, Latex
- Power-Point (wegen Animationen)

Person: John Horton Conway (1937-)

Englischer Mathematiker, Zahlentheorie, Gruppentheorie, Game-Theory, Game Of Life

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 tod.

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 Programm I

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

Definition: Die Programmiersprache LOOP

- Variablen: x_1, x_2, x_3, \dots
- 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 $\text{LOOP } x_i \text{ DO } P \text{ ENDLOOP}$ ein LOOP-Programm.

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

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)
- $x_0 := x_1 - x_2 = \max(x_1 - x_2, 0)$
- $x_0 := x_1 \text{ DIV } x_2$
- $x_0 := x_1 \text{ MOD } x_2$
- IF $x_1 = 0$ THEN P_1 ELSE P_2 ENDIF wie folgt:
 $x_2 := 1; x_3 := 0;$
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;
- IF $x_1 = c$ THEN P_1 ELSE P_2 ENDIF auch möglich.

Definition: Die Programmiersprache WHILE

- Variablen: x_1, x_2, x_3, \dots
- 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 $[P] : \mathbb{N}^k \rightarrow \mathbb{N}^k$.

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. (benötigt explizit sogar nur die äußerste While-Schleife, alles andere könnte mit LOOP simuliert werden).

10 LOOP und WHILE Programme II

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

Person: Wilhelm Ackermann (1896-1962)

Deutscher Mathematiker, Schüler von Hilbert, Ackermann-Funktion.

Person: Donald Knuth (1938-)

Amerikanischer Informatiker, Mathematiker, big-O, TeX, Up-Arrow-Notation

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: Ackermann-Funktion

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

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

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

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

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

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 $b = [P](a)$. 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.

11 Primitiv-rekursive Funktionen

- Primitive-rekursive Funktionen
- Prädikatsfunktion
- Weitere primitiv-rekursive Funktionen
- Bijektion $\mathbb{N}^2 \rightarrow \mathbb{N}$
- Äquivalenz primitiv-rekursiv und LOOP
- Stephen Kleene
- Kleenscher μ -Operator
- Klasse der μ -rekursiven Funktionen ist Turing-mächtig

Definition: Primitiv-rekursive Funktionen

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

Die Basisfunktionen sind:

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

Komposition von prim.-rek. Funktionen ist wieder prim.-rek.

Das heißt, für prim.-rek. Funktionen $g : \mathbb{N}^a \rightarrow \mathbb{N}$ und $h_1, h_2, \dots, h_a : \mathbb{N}^b \rightarrow \mathbb{N}$ ist

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

wieder prim.-rek.

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

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

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

wieder prim.-rek.

Primitiv-rekursive Funktionen sind stets berechenbar und total.

Notation: Prädikatsfunktion

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

Bemerkung: Weitere primitiv-rekursive Funktionen

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

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

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

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

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

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

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

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

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

Wobei $\Omega : \mathbb{N}^k \rightarrow \mathbb{N}$ bijektiv.

- Hintereinanderausführung via Komposition, also

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

Für LOOP-Programme Q, R .

- LOOPS durch wiederholtes Hintereinanderausführen:

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

$$g_P(x) := g_Q^{x_i}(x) = g_Q(g_Q(\dots g_Q(g_Q(x)) \dots))$$

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 $\text{succ}(x_j)$ durch $x_0 := x_j + 1$.
- Komposition durch "geeignetes Hintereinanderausführen"
- Primitive Rekursion (siehe def., gleiche Fkt im folgenden) durch "bottom-up":

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

Person: Stephen Cole Kleene (1909-1994)

Amerikanischer Student von Alonzo Church. Rekursionstheorie, Kleene-stern, regexp's.

Definition: Der Kleen'sche μ -Operator

Es gilt im folgenden $\min \emptyset = \perp$.

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

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

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

Bspw. ist zu $g : \mathbb{N}^3 \rightarrow \mathbb{N}$ 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.

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

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

Dies benutzt den μ -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. Dies kann auch \perp sein, wenn die While-Schleife nicht terminiert.

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

```

 $x_0 := 0;$ 
 $y := g(0, x_1, \dots, x_k);$ 
WHILE  $y \neq 0$  DO
     $x_0 := x_0 + 1;$ 
     $y := g(x_0, x_1, \dots, x_k);$ 
ENDLOOP;
```

12 P versus NP