TI-2
Mitschriften

Paul Glaser

May 30, 2023

Contents

Chapter 1

Turing-Maschinen

Definition 1.1: Deterministische Turingmaschine

Deterministische Turingmaschine: 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Q nichtleere endliche Zustandsmenge
- Σ endliches Eingabealphabet
- $\Gamma \supseteq \Sigma$ endliches Bandalphabet
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ (partielle) Überführungsfunktion
- $B \in \Gamma \backslash \Sigma$ Leersymbol des Bands
- $F \subseteq Q$ Menge von akzeptierenden (End-)Zuständen

NTM analog mit mengenwertigem $\delta : Q \times \Gamma \to \mathcal{P}_e(Q \times \Gamma \times \{L, R\})$

Konvention

Konvention: $\delta(q, X)$ undefiniert für Endzustände $q \in F$ - Turingmaschine hält an, wenn $\delta(q, X)$ undefiniert ist.

Konfiguration

Konfiguration $\hat{=}$ Zustand + Bandinhalt + Kopfposition

- Formal dargestellt als Tripel $K = (u, q, v) \in \Gamma^* \times Q \times \Gamma^+$ $\cdot u, v$: String links/rechts vom Kopf q Zustand
- Nur der bereits 'besuchte' Teil des Bandes wird betrachtet Blanks am Anfang von u oder am Ende von v entfallen, wo möglich

Akzeptierende Sprachen

$$L(M) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. (\epsilon, q_0, w) \vdash^* (u, p, v)\}$$

Zeit- und Platzbedarf

Rechenzeit $t_M(w)$

Anzahl der Konfigurationsuberg ¨ ange ¨ bis M bei Eingabe w anhalt

Speicherbedarf $s_M(w)$

Anzahl der Bandzellen, die M wahrend der Berechnung aufsucht

Komplexität: Bedarf relatov zur Größe

$$\boldsymbol{T}_M(\boldsymbol{n}) = \max\{t_M(w) || w\ | = n\}$$
$$\boldsymbol{S}_M(\boldsymbol{n}) = \max\{s_M(w) || w\ | = n\}$$

Definition 1.2: Die berechnete Funktion einer Turingmaschine

- Anfangskonfiguration: $\alpha(w) := (\epsilon, q_0, w)$
- Terminierung: $M \downarrow_K := \exists u, v, q, X.K = (u, q, Xv) \wedge \delta(q, X)$ undefiniert
- Rechenzeit: $t_M(w) := \max\left\{j \mid \alpha(w) \vdash^j K \wedge M \downarrow_K\right\}$
- Undefiniert falls dieses Maximum nicht existiert, d.h. wenn M nicht auf w hält
- Ausgabefunktion: für $K = (u, q, v)$ ist $\omega(K) := v|_\Sigma$ (längster Präfix von v, der zu Σ^* gehört)
- Ausgabe beginnt unter dem Kopf bis ein Symbol nicht aus Σ erreicht wird
- Berechnete Funktion: $f_M(w) := \left\{\omega(K) \mid \exists i \in \mathbb{N}.\alpha(w) \vdash^i K \wedge M \downarrow_K\right\}$
- $f_M(w)$ ist undefiniert, wenn diese Menge leer ist, also wenn $t_M(w)$ undefiniert ist Für DTMs ist $f_M(w) = \omega(K)$ für das eindeutig bestimmte K mit $\alpha(w) \vdash^{t_M(w)} K$

Chapter 2

Rekursive Funktionen

2.1 Primitiv-rekursive Funktionen

Definition 2.1.1: Berechenbare Grundfunktionen

- Nachfolgerfunktion: von einer Zahl zur nachsten weiterzählen s
- Projektion: aus einer Gruppe von Werten einen herauswahlen pr_k^n
- Konstante: unabhangig von der Eingabe eine feste Zahl ausgeben c_k^n

In der primitven Rekursion gibt es diese 3 Grundfunktionen und 2 möglichkeiten diese miteinander zu verbinden um neue Funktionen zu definieren.

- Komposition: Verkettung von Funktionen
- Rekursion: Programm ruft sich bei Ausfuhrung selbst auf

Note:-

Mit diesen wenigen Bausteinen lassen sich bereits fast alle Funktionen berechnen.

Definition 2.1.2: Mathematische Definition der Grundfunktionen

Grundfunktionen:

- Nachfolgerfunktion $s : \mathbb{N} \to \mathbb{N}$ mit $s(x) = x + 1$
- Projektionsfunktionen $pr_k^n : \mathbb{N}^n \to \mathbb{N}$ $(1 \leqslant k \leqslant n)$ mit $pr_k^n(x_1, .., x_n) = x_k$
- Konstantenfunktion $c_k^n : \mathbb{N}^n \to \mathbb{N}$ $(0 \leqslant n)$ mit $c_k^n(x_1, \ldots, x_n) = k$

Operationen auf Funktionen

- Komposition $f \circ (g_1, \ldots, g_n) : \mathbb{N}^k \to \mathbb{N}$ mit $(g_1, .., g_n : \mathbb{N}^k \to \mathbb{N}, f : \mathbb{N}^n \to \mathbb{N})$
  Für $h = f \circ (g_1, .., g_n)$ gilt $h(\vec{x}) = f\left(g_1(\vec{x}), \ldots, g_n(\vec{x})\right)$
- Primitive Rekursion $\mathrm{Pr}[f, g] : \mathbb{N}^k \to \mathbb{N}$ mit $(f : \mathbb{N}^{k-1} \to \mathbb{N}, g : \mathbb{N}^{k+1} \to \mathbb{N})$
  Für $h = \mathrm{Pr}[f, g]$ gilt $h(\vec{x}, 0) = f(\vec{x})$
  und $h(\vec{x}, y + 1) = g(\vec{x}, y, h(\vec{x}, y))$

Example 2.1.1

Sei $f(x) = x + 3$, kann über Komposition realisiert werden:

$$f = s \circ s \circ s \circ pr_1^1$$

Sei $h(x, y) = x + y$, dann gilt

$$h(x, 0) = x \text{ nach Definition der primitven Rekursion } h(x, 0) = f(x) = x$$

$$h(x, y + 1) = g(x, y, h(x, y)) = 1 + h(x, y)$$

y wird rekusiv um 1 reduziert, dafür wird jedes mal 1 auf-addiert. Jetzt müssend die Funktionen f und g noch als Primitiv Rekursive Funktionen aufgeschrieben werden.

$$f = pr_1^1$$

$$g = s \circ pr_3^3 \text{ da die Funktion } g \text{ 3 Eingabewerte hat}$$

Die Funktion h ist dann

$$h = \mathrm{Pr}[f, g]$$

Chapter 3

Elementare Berechenbarkeitstheorie

3.1 Elementare Berechenbarkeitstheorie

Es gibt viele Fragen zur Berechenbarkeit:

- Welche Funktionen sind berechenbar und welche nicht?
- Welche Probleme sind (semi-)entscheidbar und welche nicht?
- Abschlusseigenschaften: wie kann man Lösungen wiederverwenden?
- Grenzen des Machbaren: was ist nicht mehr berechenbar?

Wie kann man nachweisen, dass ein Problem nicht lösbar ist?

Claim 3.1.1 Antworten hängen nicht vom Berechnungsmodell ab

Nach der Church-Turing-These

- Berechenbarkeit, (semi-)Entscheidbarkeit, (Zeit-/Platz)Komplexität sind allgemeine Konzepte
- Löse Theorie von Betrachtung konkreter Modelle
- Formuliere Grundeigenschaften (Axiome) berechenbarer Funktionen
- Beweise diese Eigenschaften mit einem Modell (Turingmaschine)
- Stütze alle Beweise für Aussagen nur noch auf diese Eigenschaften denn sie gelten für alle gleichmächtigen Berechnungsmodelle

3.2 Formuliere Theorie Berechenbarer Funktionen

Claim 3.2.1 Es reicht berechenbare Funktionen zu betrachten

(Semi-)Entscheidbarkeit einer Menge ist äquivalent zur Berechenbarkeit ihrer (partiell-)charakteristischen Funktion

Claim 3.2.2 Es reicht Berechenbarkeit auf Zahlen zu betrachten

- Berechenbarkeitskonzepte auf Wörtern und Zahlen sind gleichwertig, da Zahlen als Wörter codierbar sind (binär oder anders) und andersrum
- Es ist meist leichter, mit Zahlen zu arbeiten (z.B. Rechenzeit)
- Programme und Daten sind als Zahlen codierbar

Claim 3.2.3 Es reicht einstellige Funktionen auf $\mathbb{N} \to \mathbb{N}$ zu betrachten

Funktionen auf Zahlenpaaren und -listen sind einstellig simulierbar

3.3 Berechenbare Funktionen sind aufzählbar

Claim 3.3.1 Turingmaschinen sind als Wörter codierbar

Es reicht, Turingmaschinen mit $\Gamma = \{0, 1, B\}$ und $F = \{q_1\}$ zu betrachten, sie können immer noch alles berechnen was uneingeschränlte TMs berechnen können.

Note:-

Das tatsächliche Codieren:

- Definiere $\text{code}(\delta(q, X)) \equiv qXpYD$, falls $\delta(q, X) = (p, Y, D)$ (ϵ sonst)
- Codiere die Maschine $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ durch das Wort $\text{code}(\delta(q_0, 0))\,\text{code}(\delta(q_0, 1))\,\text{code}(\delta(q_0, B)) \ldots \text{code}(\delta(q_n, B))$
- Codiere Alphabet $\{q_0, \ldots, q_n, 0, 1, B, L, R\}$ als Wörter über $\Delta = \{0, 1\}$

Claim 3.3.2 Wörter über einem Alphabet sind numerierbar

Bestimme lexikographische Ordnung der Wörter über $\Delta = \{x_1, \ldots, x_n\}$

$$\epsilon < x_1 < \ldots < x_n < x_1x_1 < x_1x_2 < \ldots < x_nx_n < x_1x_1x_1 < \ldots$$

- Zähle entsprechend durch: $w_0 := \epsilon, w_1 := x_1, ..w_n := x_n, w_{n+1} := x_1x_1, \ldots$

Claim 3.3.3 Turingmaschinen sind (bijektiv) numerierbar

Aus Claim 3.3.1 und Claim 3.3.2 lässt sich ableiten das Turingmaschinen numerierbar sind.
Zähle Wörter über Δ auf und teste, ob sie Turingmaschinen codieren:
M_i ist die Turingmaschine, deren Codierung an i-ter Stelle erscheint.
Die Nummer i wird auch die Gödelnummer von M_i genannt

3.4 Kernaxiome

Definition 3.4.1

φ_i ist die von M_i berechnete (partielle) Funktion auf $\mathbb{N}$

Φ_i ist die zugehörige Schrittzahlfunktion von M_i

Claim 3.4.1 $\varphi : \mathbb{N} \to \mathcal{R}$ is surjektiv, aber nicht bijektiv

Jede programmierbare Funktion hat einen Index, aber jede berechenbare Funktion hat unendlich viele Programme die sie realisieren.

Example 3.4.1

Bspw. Existieren unendlich viele Turingmaschinen die die Addition zweier Zahlen berechnen.
Sei M_i eine Turingmaschine welche die Addition berechnet, so können dieser unendlich viele Zustände

hinzugefügt werden, welche nichts an der berechnenden Funktion ändern. Somit gibt es unendlich viele Programme welche die Addition zweier Zahlen berechnen.

Definition 3.4.2: Axiom 1

$$\text{Für alle } i \text{ gilt domain } (\Phi_i) = \text{domain}\,(\varphi_i)$$

Per Konstruktion kann Φ_i nur definiert sein, wenn φ_i hält, wenn φ_i ist es definiert.

Definition 3.4.3: Axiom 2

$$\{\langle i, n, t\rangle \mid \Phi_i(n) = t\} \text{ ist entscheidbar}$$

Simuliere Ausführung von $\varphi_i(n)$ für maximal t Schritte

Definition 3.4.4: Axiom 3

$$u : \mathbb{N} \to \mathbb{N} \text{ mit } u\langle i, n\rangle = \varphi_i(n) \text{ ist berechenbar (UTM Theorem)}$$

-