1 Grundlagen

1.1 Universalität

Universalität. Die Funktion $U : (e, x) \to \varphi_e(x)$ ist berechenbar.

Zeitbeschränkte Universalität. Die Funktion

$$U'(e, x, t) \mapsto \begin{cases} \varphi_e(x) + 1, & *; \\ 0, & \text{sonst.} \end{cases}$$

U' ist berechenbar und total.

*falls das Programm mit Gödelnummer e auf Eingabe x nach max. t Schritten terminiert

berechenbar Eine Funktion f ist berechenbar, wenn es ein WHILE-Programm P mit $e = \text{Goed}(P)$ und $\varphi_e = f$ gibt.

1.2 Abzählbarkeit

Die Menge der totalen Funktionen $\mathbb{N} \to \{0, 1\}$ ist nicht abzählbar.

Da die Menge der WHILE-Programme abzählbar ist, gibt es also eine totale Funktion $\mathbb{N} \to \{0, 1\}$, welche nicht WHILE-berechenbar ist.

1.3 Funktionen

Partielle Funktion. Eine partielle Funktion $F : A \to B$ ordnet allen Elementen $x \in A$ ein Element $f(x) \in B$ zu oder ist für $x \in A$ undefiniert, d.h. $f(x) = \perp$.

Definitionsbereich. $dom(f) := \{x \in \mathbb{N} \mid f(x) \neq \perp\}$.

Bild. $img(f) := \{f(x) \mid x \in dom(f)\}$.

Totale Funktion. f heißt total, falls $dom(f) = A$.

Charakteristische Funktion. Die charakteristische Funktion einer Sprache $A \subseteq \mathbb{N}$ ist $\chi_A : \mathbb{N} \to \{0, 1\}$ mit

$$\forall x \in \mathbb{N} : \chi_A(x) = \begin{cases} 1, & x \in A; \\ 0, & x \notin A. \end{cases}$$

Partiell charakteristische Funktion. Die partiell charakteristische Funktion einer Sprache $A \subseteq \mathbb{N}$ ist $\hat{\chi}_A : \mathbb{N} \to \{0, 1\}$ mit

$$\forall x \in \mathbb{N} : \hat{\chi}_A(x) = \begin{cases} 1, & x \in A; \\ \perp, & x \notin A. \end{cases}$$

2 Berechenbarkeitstheorie

Eine Funktion f ist **berechenbar**, wenn es ein WHILE-Programm P mit $e = \text{göd}(P)$ und $\varphi_e = f$ gibt. Eine Sprache $A \in \mathbb{N}$ ist **entscheidbar** $\Leftrightarrow$ Sowohl A als auch $\overline{A}$ sind semi-entscheidbar.

Eine Sprache $A \subseteq \mathbb{N}$ ist **rekursiv aufzählbar** $\Leftrightarrow$ A ist semi-entscheidbar.

Eine Sprache $A \subseteq \mathbb{N}$ heißt rekursiv aufzählbar, falls sie das Bild $img(f)$ einer berechenbaren Funktion f ist. *Die berechenbare Funktion f zählt A rekursiv auf*

2.1 Berechenbare Funktionen

Min-Suche: Falls $p : \mathbb{N}^2 \to \{0, 1\}$ total und berechenbar ist, so ist die Funktion $x \mapsto \min\{y \in \mathbb{N} \mid p(x, y) \neq 0\}$ berechenbar. (Falls die Menge leer ist, so ist die Funktion für das entsprechende x undefiniert)

2.2 REC / RE / co-RE

$A \in \text{REC} \Leftrightarrow A$ ist entscheidbar. $\Leftrightarrow A \in \text{RE} \wedge A \in \text{co} - \text{RE}$ $\Leftrightarrow \chi_A$ ist berechenbar. $\Leftrightarrow \exists$ monotones, berechenbares, totoales f mit $img(f) = A$

$A \in \text{RE}$. $\Leftrightarrow A$ ist semi-entscheidbar. $\Leftrightarrow A$ ist rekursiv aufzählbar. $\Leftrightarrow \hat{\chi}_A$ ist berechenbar. $\Leftrightarrow \exists$ berechenbares f mit $dom(f) = A$ $\Leftrightarrow \exists$ berechenbares f mit $img(f) = A$.

$A \in \text{co} - \text{RE} \Leftrightarrow \overline{A} \in RE$.

$\text{co} - \text{REC} = \text{REC}$

2.3 Schwere und Vollständigkeit

Eine Sprache A_0 heißt **schwer** für X, falls $\forall A \in X : A \leq A_0$. (Beweis: X-schwere Sprache $\leq A_0$)

Eine Sprache A_0 heißt **vollständig** für X, wenn A_0 X-schwer ist und $A_0 \in X$.

- H ist RE schwer und vollständig
- H_e ist RE schwer und vollständig
- $\overline{H_e}$ ist co − RE schwer und vollständig
- T ist RE-schwer

Sei A RE vollstädnig/schwer, dann ist $\overline{A}$ co − RE vollständig/schwer

2.4 S-m-n

Für jedes $m, n \in \mathbb{N}$ gibt es eine berechenbare Funktion $s_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$, sodass für alle $e \in \mathbb{N}$ mit $\varphi_e : \mathbb{N}^{m+n} \to \mathbb{N}$ und alle $y \in \mathbb{N}^m, z \in \mathbb{N}^n$ gilt

$$\varphi_e(y, z) = \varphi_{s_n^m(e,y)}(z)$$

Berechnet Gödelnummer eines Programms, indem die ersten m Eingaben fest eingebaut sind.

2.5 Fixpunkte

Ein semantischer Fixpunkt einer Funktion $f : \mathbb{N} \to \mathbb{N}$ ist eine Gödelnummer e mit $\varphi_{f(e)} = \varphi_e$.

Jede berechenbare totale Funktion $f : \mathbb{N} \to \mathbb{N}$ hat einen semantischen Fixpunkt.

2.6 Satz von Rice

Indexmenge. Eine Menge $I \subseteq \mathbb{N}$ heißt Indexmenge, falls gilt: $\forall i, j \in \mathbb{N} : (i \in I \wedge \varphi_i = \varphi_j) \Rightarrow j \in I$.

Vorlage: Seien $i, j \in \mathbb{N}$ so, dass $i \in I$ und $\varphi_i = \varphi_j$. Da $i \in I$, gilt *Mengenbedingung*. Mit $\varphi_i = \varphi_j$ gilt das gleiche für j, womit $j \in I$. Also ist I eine Indesmenge.

Außerdem ist $I \neq \mathbb{N}$, da *Beispiel*. Weiterhin ist $I \neq \emptyset$, da *Beispiel*. Damit I eine nicht triviale Indexmenge und nach Satz von Rice nicht entscheidbar.

2.7 Arithmetische Hierarchie

Sei $A \subseteq \mathbb{N}$ eine Sprache. A ist arithmetisch (oder: in der arithmetischen Hierarchie), falls es eine Reihe von Quantoren $Q_1, \ldots, Q_n$ gibt und ein **total berechenbares Prädikat** P sodass für alle $x \in \mathbb{N}$, $x \in A \Leftrightarrow Q_1 y_1 \ldots Q_n y_n P(x, y_1, \ldots, y_n)$.

Prädikat. Ein Prädikat ist eine totale Abbildung $P : \mathbb{N}^k \to \{0, 1\}$; für alle $x \in \mathbb{N}^k$ wird dann $P(X)$ als "wahr" interpretiert, falls $P(x) = 1$ und sonst als "falsch".

Für $Q_1 = \exists$ ist A eine Σ_n-Sprache. Für $Q_1 = \forall$, ist A eine Π_n-Sprache.

Seien $n, m \in \mathbb{N}$, mit $m > n$. Falls $A \in \Pi_n$, dann auch $A \in \Pi_m$.

$\text{REC} = \Sigma_0 = \Pi_0 = \Delta_1 = \Delta_0$

$\text{RE} = \Sigma_1$

$\text{co} - \text{RE} = \Pi_1$

$\varphi_e(x) = y \Leftrightarrow \exists t \in \mathbb{N}$: $U'(e, x, t) = y + 1$

$\varphi_e(x) = \perp \Leftrightarrow \forall t \in \mathbb{N}$: $U'(e, x, t) = 0$

3 Reduktionen

Seien $A \in \mathbb{N}$ und $A \in \mathbb{N}$ Sprachen und sei: $f : \mathbb{N} \to \mathbb{N}$ eine totale und berechenbare Funktion mit

$$\forall x \in \mathbb{N} : x \in A \Leftrightarrow f(x) \in B$$

A reduzierbar auf B, f ist Reduktion von A auf B

3.1 Hilfreiche Sätze

$A \leq A$ (Reflexivität)

$A \leq B \wedge B \leq C \Rightarrow A \leq C$ (Transitivität)

$A \leq B \Leftrightarrow \overline{A} \leq \overline{B}$ ($\overline{A} := \mathbb{N} \setminus A$)

$A \leq \overline{A} \Leftrightarrow A \in \text{REC}$

$A \leq B$**, dann gilt:**

Wenn $B \in REC/RE/co-RE/\Sigma_n/\Pi_n/\Delta_n$, dann ist auch $A \in REC/RE/co-RE/\Sigma_n/\Pi_n/\Delta_n$.

Wenn $A \notin REC/RE/co-RE/\Sigma_n/\Pi_n/\Delta_n$, dann $B \notin REC/RE/co-RE/\Sigma_n/\Pi_n/\Delta_n$.

Falls $L \leq \ldots$ min-Suche $\min\{\langle x, t\rangle \in \mathbb{N} \mid U'(e, x, t) \neq 0\}$ bzw. $\exists x, t : U'(e, x, t) \neq 0$

Falls $T \leq \ldots$ beschränkte Suche $\forall y \leq x : U(e, y) \neq \perp$

3.2 Template für $A \leq B$

$$h(e, t) = \begin{cases} \text{Verhalten von } B, & e \in A; \\ \text{Verhalten von } \overline{B}, & e \notin A. \end{cases}$$

Wir definieren eine Hilfsfunktion $h : \mathbb{N}^2 \to \mathbb{N}$. $\forall e, t \in \mathbb{N} : h(e, t) = \ldots$ (alternativ :=)

h ist berechenbar, weil [Abschlusseigenschafte, U',...]

Sei e_h eine Gödelnummer zu h. Diese existiert, da h berechenbar ist. Sei $f : e \mapsto s_1^1(e_h, e)$. Nun ist f total und berechenbar, da s_1^1 dies ist.

Weiterhin gilt $\forall x \in \mathbb{N} : \varphi_{f(e)}(x) = \varphi_{s_1^1(e_h,e)}(x) = \varphi_{e_h}(e, x) = h(e, x)$.

Sei $e \in \mathbb{N}$ beliebig. Nun gibt es zwei Fälle:

$e \in A$: [...] Also folgt $f(e) \in B$.

$e \notin A$: [...] Also folgt $f(e) \notin B$.

3.3 Es ist bekannt

$H \equiv H_e$; $H_e \leq T$; $\overline{H_e} \leq T$; $T \not\leq \overline{H_e}$; $\text{Even} \in \Pi_1$; Min-Suche berechenbar (siehe unten links)

4 Laufzeiten

4.1 $\mathcal{O}$-Notation

$f \in \mathcal{O}(g) \Leftrightarrow \exists c > 0 \exists n_0 \geq 0 \forall n \geq n_0 : f(n) \leq c \cdot g(n)$, also f wächst (asypmtotisch) höchstens so schnell wie g

$f \in \Omega(g) \Leftrightarrow g \in \mathcal{O}(f)$, also f wächst mindestens so schnell wie g

$f \in \Theta(g) \Leftrightarrow f \in \mathcal{O}(g) \wedge f \in \Omega(g)$, also f wächst genauso schnell wie g

$f \in o(g) \Leftrightarrow \forall c > 0 \exists n_0 \geq 0 \forall n \geq n_0 : f(n) < c \cdot g(n)$, also f wächst echt langsamer als g

$f \in \omega(g) \Leftrightarrow g \in o(f)$, als f wächst echt schneller als g

4.2 Datenstrukturen

- Array: Anlegen $\mathcal{O}(n)$, Zugriff $\mathcal{O}(1)$, Element mit Rang k finden $\mathcal{O}(n)$ (QuickSelect)
- Sortieren: $\mathcal{O}(n \log n)$
- Counting Sort: $\mathcal{O}(n+k)$, falls $k \in \mathcal{O}(n) \Rightarrow \mathcal{O}(n)$
- BinarySearch: $\mathcal{O}(\log n)$
- Heap: Aufbau $\mathcal{O}(n)$, Insert / ExtractMin/Max $\mathcal{O}(\log n)$ — kann als Priority-Queue verwendet werden
- Baum traversieren: $\mathcal{O}(n)$, Suchen, Einfügen, Löschen $\mathcal{O}(\log n)$ (balanciert, AVL-Suchbäume)

5 Formalisieren

Eingabe / Ausgabe: Typ: Array A der Länge n über $\mathbb{N}/\{0,1\}/\ldots$ (0- oder 1-initialisiet), Integer b, ...

Intepretation: $A[i]$ repräsentiert/enthält/...

Hinzufügen: von "Null-Elementen", etc.

Korrektheitsbedingung: Ein Array R (Ausgabeformat) ist valide, wenn Rahmbedingungen eingehalten, Definieren von Kostenfunktionen (z.B. $w(R)$), ...

R ist nun korrekt, wenn es valide ist, und wenn für alle validen R^* gilt: $w(R) \geq w(R^*)$

Interpretation: Meist während der vorherigen Bestandteilen

6 Algorithmen

Algorithmusbeschreibung: Fließtext oder Pseudocode / Flussdiagramm mit Erklärungen

Einleitender Satz, der Lösungsidee erklärt

Implementierungsdetails können ausgelassen werden, sollten aber eindeutig ableitbar sein ("Wir iterieren über alle Elemente des Arrays und machen für jedes [...]")

Korrektheitsbeweis: R ist valide, da Algorithmuseigenschaft,...

Wir zeigen die Korrektheit über ...

Laufzeitanalyse: Worst-Case-Laufzeit in Landau-Notation

6.1 Greedy

Metrik $w(X)$ wird maximiert / minimiert

6.1.1 Greedy Stays Ahead

Vergleiche Präfix der iterativ aufgebauten Lösung L des Algo. mit entsprechendem Präfix einer optimalen Lösung L^*.

- Zeige: Präfix von L ist bezüglich einer Metrik w' mindestens so gut wie entsprechendes Präfix von L^*.
- Zeige induktiv: L ist mindestens so gut wie L^* bezüglich w'
- Für $w' = w$ ist Optimalität damit gezeigt (ansonsten zusätzliches Argument dafür)

6.1.2 Greedy Stays Ahead Beispiel

Die Optimalität zeigen wir per Greedy-Stays-Ahead: Sei also S^* eine optimale Lösung. Wir zeigen induktiv für alle $i \leq |B|$, dass $B[i] \geq B^*[i]$. Anfangs gilt das sowieso, da beide mit 0 beginnen müssen. Gelte die Aussage für ein beliebiges i und betrachte $i+1$. Angenommen, es wäre $B^*[i+1] > B[i+1]$. Ist $B^*[i+1] \leq B[i]+d$, dann hätte unser Algorithmus den Wert ebenfalls zur Wahl gehabt und ausgewählt. Andernfalls ist $B^*[i+1] > B[i]+d \geq B^*[i]+d$, womit B^* nicht valide wäre. Also ist die Annahme falsch und es gilt $B[i+1] \geq B^*[i+1]$. Per Induktion gilt die obige Aussage also für alle i insbesondere wissen wir $B[|B|1] \geq B^*[|B|1]$, also kann B^* nicht weniger Elemente enthalten und B ist optimal.

6.1.3 Austauschargument

Vergleiche Lösung L des Algorithmus mit optimaler Lösung L^*.

Option 1: Führe Annahme $L \neq L^*$ zum Widerspruch

- Unterschied zwischen L und L^*?
- Verändere L^* zu L_2^* durch Austausch
- L_2^* ist besser als L^* bezüglich w
- Widerspruch zur Optimialität von L^*

Option 2: Gleiche L^* und L ohne Wertveränderung $(w(X))$ in endlich vielen Schritten aneinander an, indem du Teiler der Lösung austauschst, ggf. über Zwischenschritte $L_2^*, L_3^*, \ldots L_z^*$

- $w(L^*) \leq w(L_2^*) \leq \cdots \leq w(L_z^*) \leq w(L)$ bei Maximierung, $\geq$ bei Minimierung
- L ist mindestens genauso gut wie L^*, also auch optimal

6.1.4 Austauschargument Beispiel

Wir zeigen die Korrektheit über ein Austauschargument: Sei R unsere Lösung und sei R^* eine optimale Lösung. Ist $R = R^*$, sind wir direkt optimal. Andernfalls gibt es eine günstigste Brücke mit Position p, die in R und R^* verschieden behandelt wird. Es ist nicht möglich, dass $R[p] = 0$ und $R^*[p] = 1$, da R diese Brücke nach Verhalten des Algorithmus auch ausgewählt hätte, wenn R^* es sich ja auch leisten konnte. Also gilt $R[p] = 1$ und $R^*[p] = 0$. Da R^* nicht schlechter als R ist, muss es eine andere Brücke an Position q geben mit $K[q] > K[p]$. In R^* kann man nun q durch p ersetzen und die Gesamt-Wirtschaft gleich lassen, ohne dass sich die Gesamtkosten erhöhen.

Derart kann man jeden Unterschied zwischen R^* und R auflösen, ohne die Gesamt-Wirtschaft der Lösung zu verändern oder die Kosten zu erhöhen, bis $R = R^*$. Da R^* optimal ist, ist also auch R optimal.

6.2 Master-Theorem (vereinfacht)

Sei für $a \geq 1, b > 1$ die Rekurrent T gegeben als

$$\forall n \in \mathbb{N} : T(n) \begin{cases} 0, & \text{falls } n < 1; \\ aT(\frac{n}{b}) + n, & \text{falls } n \geq 1 \end{cases}$$

Dann gilt für $t \in \mathbb{N}$ und $n = b^t$:

$$T(n) = (1 + o(1)) \cdot \begin{cases} \frac{b}{b-a} n, & \text{falls } a < b; \\ n \log_b n, & \text{falls } a = b; \\ \frac{a}{a-b} n^{\log_b a}, & \text{falls } a > b \end{cases}$$

6.3 DP

Beweis per Induktion: IA, IV, IS

Laufzeit: Erstellen der Datenstruktur ; Datenstruktur-Initialisierung ; Berechnungsvorschrift ; Berechnungsreihenfolge ; Ausgabe

6.4 Untere Schranken (Adversary)

Wir beschreiben einen Adversary. Ein Algorithmus kann bei diesem einen beliebigen Wert des Arrays H anfragen. Allen angefragten Knoten der untersten Ebene wird der Wert 0 zugewiesen, für alle anderen Knoten wird 1 zurückgegeben. **Der Adversary merkt sich, welche Knoten nicht angefragt wurden.**

Die Rückgaben sind konsistent, da die Rückgabe für einen Knoten konstant definiert ist. Liefert ein Algorithmus eine Ausgabe in $o(n)$ Schritten, wurden zwei der $\lceil n/2 \rceil$ Blattknoten b_1, b_2 nicht angefragt. Gibt der Algorithmus einen Weg zurück, der nicht in b_2 endet, wird der Wert von b_2 allein auf 1 gesetzt, womit der hier endende Weg optimal wäre. Wird ein Weg zurückgegeben, der in b_2 endet, wird der Wert von b_1 allein auf 1 gesetzt, womit der hier endende Weg optimal wäre. In beiden Fällen ist die Rückgabe des Algorithmus nicht optimal.

Also kann es keinen Algorithmus geben, der in $o(n)$ immer die richtige Lösung liefert.

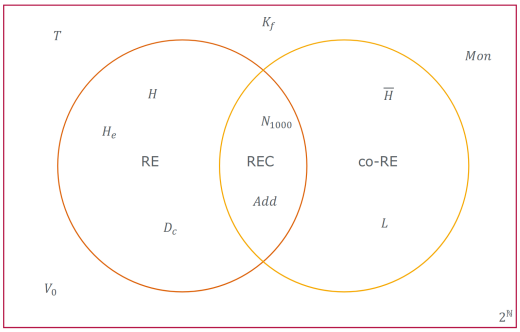
Konsistenz: Dadurch, dass wir uns bereits angefragte Stellen und unsere dazugehörige Ausgabe speichern und es nach weniger als n Anfragen noch immer mindestens eine nicht angefragte Stelle gibt, für die wir je nach Ausgabe des Lösungsalgorithmus eine Ausgabe produzieren, ist diese Ausgabe immer konsistent möglich. Unser Algorithmus reagiert also auf jede mögliche Antwort des Lösungsalgorithmus und führt diese zum Widerspruch, wenn nicht mindestens n Anfragen gestellt wurden, was die untere Schranke von $\Theta(n)$ beweist.

Wichtig bei Adversary: Zuvor angefragte Stellen werden wie zuvor beantwortet (im Zweifel explizit hinschreiben).

Eingabe: Gewicht $G \in \mathbb{N}^+$, Verkaufspreise $V \subseteq \{1,\ldots,G\} \times \mathbb{N}^+$, $|V| = m$
Ausgabe: maximale Einnahmen $E \in \mathbb{N}^+$ — Formalisierung

DP-Algorithmus

- Array A der Länge $G+1$ über $\mathbb{N}$ — Datenstruktur
- $\forall 0 \leq g \leq G$: $A[g]$ sind maximale Einnahmen für g kg Käse — Interpretation der Datenstruktur
- Gesamteinnahmen: $A[G]$ — Ausgabe / Ergebnis
- $A[0] \coloneqq 0$ — Initialisierung
- $\forall 1 \leq g \leq G: A[g] \coloneqq \max\{p_i + A[g-i] \mid (i,p_i) \in V, i \leq g\}$ — Berechnungsvorschrift
- Fülle Array von $g = 1$ bis $g = G$ — Berechnungsreihenfolge



- $T = \{e|\varphi_e \text{ ist total}\}$
- $L = \{e|\varphi_e \text{ ist nirgends definiert}\}$
- $N_{1000} = \{e|e \leq 1000\}$
- $K_f = \{e|\varphi_e = f\}$, f berechenbar
- $V_0 = \{e|\forall x \in \mathbb{N}: \varphi_e(x) = 0\}$
- $H_e = \{e|\varphi_e(e) \neq \perp\}$
- $H = \{\langle e,x\rangle|\varphi_e(x) \neq \perp\}$
- $Mon = \{e|\varphi_e \text{ ist monoton}\}$
- $D_c = \{e||dom(\varphi_e)| > c\}$
- $Add = \{\langle a,b,c\rangle|a+b=c\}$