Benotete Hausaufgaben zur Vorlesung **WiSe 2021/2022**
Berechenbarkeit und Komplexität **TU Berlin**
Niedermeier/Kellerhals/Kunz/Zschoche **13.12.2021**

Hausaufgabenblatt

Hinweise:

- Die Hausaufgabe kann bis zum **14.01.2022, 10 Uhr** auf ISIS hochgeladen werden.
- Die Hausaufgabe muss in Dreiergruppen bearbeitet werden.
- Bitte verwenden Sie die LaTeX-Vorlage auf ISIS für Ihre Abgabe.
- Plagiate werden nicht toleriert und werden scharf geahndet.
- Es können bis zu **25 Portfoliopunkte** erreicht werden.
- Alle Antworten sind zu begründen. Antworten ohne Begründung erhalten **0 Punkte**. Achten Sie insbesondere darauf, die Korrektheit Ihrer angegebenen Lösungen (Programme, Reduktionen etc.) zu begründen.
- Bitte versuchen Sie, insgesamt nicht mehr als fünf Seiten (im vorgegebenen Stil) zu schreiben.

Aufgabe 1. *Eigenschaften berechenbarer Funktionen* 8 P.

Für zwei totale Funktionen $f, g\colon \mathbb{N} \to \mathbb{N}$ sagen wir, dass

- f *größer* als g ist, falls $f(n) > g(n)$ für alle $n \in \mathbb{N}$,
- f *fast überall größer* als g ist, falls ein $n_0 \in \mathbb{N}$ existiert, sodass $f(n) > g(n)$ für alle $n \geq n_0$,
- f *fast gleich* g ist, falls ein $n_0 \in \mathbb{N}$ existiert, sodass $f(n) = g(n)$ für alle $n \geq n_0$ und
- f *oft gleich* g ist, falls unendlich viele $n \in \mathbb{N}$ existieren, sodass $f(n) = g(n)$.

Zeigen oder widerlegen Sie die folgenden Aussagen:

(a) Es gibt eine (nicht notwendigerweise berechenbare) totale Funktion, die größer als alle berechenbaren totalen Funktionen ist.

(b) Es gibt eine (nicht notwendigerweise berechenbare) totale Funktion, die fast überall größer als alle berechenbaren totalen Funktionen ist.

(c) Es gibt eine berechenbare totale Funktion $f\colon \mathbb{N} \to \mathbb{N}$ und eine unberechenbare totale Funktion $g\colon \mathbb{N} \to \mathbb{N}$, sodass f und g fast gleich sind.

(d) Es gibt eine berechenbare totale Funktion $f\colon \mathbb{N} \to \mathbb{N}$ und eine unberechenbare totale Funktion $g\colon \mathbb{N} \to \mathbb{N}$, sodass f und g oft gleich sind.

Lösung:

(a) Die Aussage ist falsch. Angenommen f wäre eine solche Funktion. Sei $g(n) := f(0)$ für alle $n \in \mathbb{N}$. Dann ist g berechenbar, da es eine konstante Funktion ist. Jedoch gilt $f(0) = g(0)$ und somit ist f nicht größer als g.

(b) Die Aussage ist richtig. Im Beweis von Übungsblatt 6, Aufgabe 4 wurde gezeigt, dass es nur abzählbar viele Turingmaschinen gibt. Folglich gibt es nach Church-Turing-These auch nur abzählbar viele berechenbare Funktionen. Sei $f_1, f_2, f_3, \ldots$ eine Abzählung aller berechenbaren Funktionen. Sei $g\colon \mathbb{N} \to \mathbb{N}$ mit

$$g(n) := 1 + \max\{f_k(n) \mid k \leq n\}.$$

Wir zeigen, dass g fast überall größer ist als alle berechenbaren Funktionen. Sei $f\colon \mathbb{N} \to \mathbb{N}$ eine beliebige berechenbare Funktion. Dann existiert ein $n_0 \in \mathbb{N}$, sodass $f = f_{n_0}$. Folglich ist $g(n) \geq f(n) + 1 > f(n)$ für alle $n \geq n_0$.

(c) Die Aussage ist falsch. Angenommen, es gibt solche f und g. Wir zeigen, dass g berechenbar ist, indem wir einen Algorithmus für g angeben, was einen Widerspruch zur obigen Annahme darstellt. Sei $n_0 \in \mathbb{N}$, sodass $f(n) = g(n)$ für alle $n \geq n_0$. Für $i \in \{0, \ldots, n_0 - 1\}$ sei $a_i := f(i)$. Der Algorithmus, der g berechnet arbeitet wie folgt: Auf Eingabe n prüft er, ob $n \geq n_0$. Falls $n \geq n_0$, so simuliert er den Algorithmus, der f berechnet, und gibt das entsprechende Ergebnis aus. Falls $n < n_0$, so gibt der Algorithmus a_n aus. Da nur endlich viele Werte $a_0, \ldots, a_{n_0-1}$ fest vorgegeben werden müssen, ist dieser Algorithmus endlich.

(d) Die Aussage ist richtig.

(Beweisvariante 1):

Sei χ_{H_0} die charakteristische Funktion des Halteproblems auf leerem Band H_0 und f die Funktion, die konstant auf 0 abbildet. Wir betrachten die Funktion $\chi_{H_0} \circ f'$, wobei f' definiert sein wie die Funktion f im Beweis von Übungsblatt 5, Aufgabe 4. Da f' und χ_{H_0} total sind, ist auch $\chi_{H_0} \circ f'$ total. Da die Funktion g aus dem Beweis von Übungsblatt 5, Aufgabe 4 berechenbar ist, ist $\chi_{H_0} \circ f'$ nicht berechenbar, denn ansonsten wäre $\chi_{H_0} \circ f' \circ g = \chi_{H_0}$ berechenbar und somit H_0 entscheidbar.

Es gibt unendlich viele TM, die auf leerem Band nicht halten, denn sonst wäre H_0 entscheidbar. Folglich gibt es unendlich viele n, sodass $\chi_{H_0} \circ f'(n) = 0$. Daher sind f und $\chi_{H_0} \circ f'$ oft gleich.

(Beweisvariante 2):

Nach Übungsblatt 6, Aufgabe 4 gibt es Funktionen des Typs $\mathbb{N} \to \mathbb{N}$, die nicht berechenbar ist. Sei h eine solche Funktion. Wir betrachten die folgende Funktion g.

$$g\colon \mathbb{N} \to \mathbb{N},$$
$$g(n) := \begin{cases} h(\frac{n}{2}), & \text{wenn } n \text{ gerade ist,} \\ 0, & \text{sonst.} \end{cases}$$

Die Funktion g ist total. Für alle $n \in \mathbb{N}$ gilt $h(n) = g(2 \cdot n)$. Also ist g nicht berechenbar, da ansonsten h berechenbar wäre. Sei f die Funktion, die konstant auf 0 abbildet. Es gibt unendlich viele ungerade natürliche Zahlen. Daher sind f und g oft gleich.

Aufgabe 2. *Starke Turing-Maschinen* 9 P.

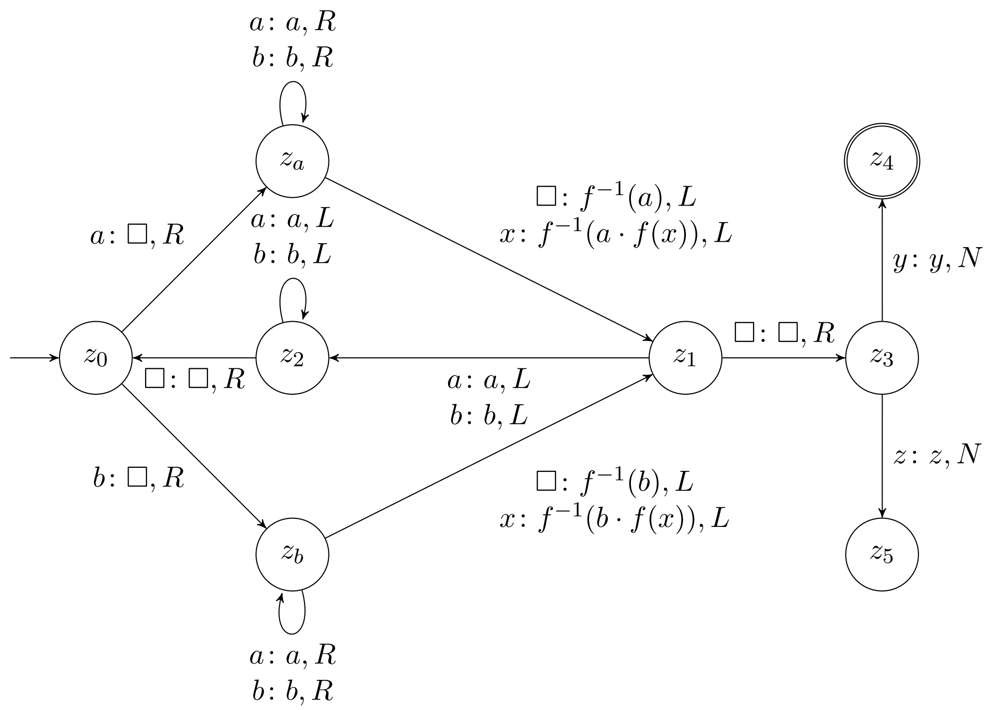
Eine *starke Turing-Maschine* ist wie eine normale deterministische Turing-Maschine ein Siebentupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$. Wie bei einer normalen Turing-Maschine ist Z eine endliche Zustandsmenge, Σ das endliche Eingabealphabet, $\delta\colon (Z \setminus E) \times \Gamma \to_p Z \times \Gamma \times \{L, R, N\}$ eine partielle Überführungsfunktion, $z_0 \in Z$ ein Startzustand, $\Box \in \Gamma \setminus \Sigma$ das Blanksymbol und $E \subseteq Z$ die Menge der Endzustände. Anders als bei einer normalen Turing-Maschine ist Γ kein endliches Bandalphabet, sondern $\Gamma = \mathbb{N} \cup \Sigma \cup \{\Box\}$. Dabei gehen wir davon aus, dass $\Sigma \cap \mathbb{N} = \emptyset$. Die Begriffe Konfiguration, Konfigurationsfolge, Berechenbarkeit einer Funktion und Entscheidbarkeit einer Sprache sind analog zu gewöhnlichen Turing-Maschinen definiert.

Beweisen Sie folgende Aussage: Für jede Sprache $L \subseteq \Sigma^*$ existiert eine starke Turing-Maschine M_L, sodass M_L auf jeder Eingabe hält und $T(M_L) = L$ gilt. Sie dürfen dabei annehmen, dass $\Sigma = \{a, b\}$.

Lösung:

Sei $f\colon \mathbb{N} \to \{a, b\}^*$ die in Übungsblatt 5, Aufgabe 4, gegebene Bijektion. Sei $L \subseteq \{a, b\}^*$ eine Sprache. Wir definieren nun eine starke Turing-Maschine $M := (Z, \{a, b\}, \mathbb{N} \cup \Sigma \cup \{\Box\}, \delta, z_0, \Box, E)$, die L akzeptiert und auf jeder Eingabe hält.

Die Idee ist, dass M die komplette Eingabe mittels f^{-1} kodiert und in eine einzelne Bandzelle schreibt. Dann kann sie in einem Schritt erkennen, ob die Eingabe in L ist. Es ist $Z := \{z_0, z_a, z_b, z_1, z_2, z_3, z_4, z_5\}$ und $E := \{z_4\}$. Die Überführungsfunktion δ ist gegeben durch:



Dabei steht x für jeden Wert in $\mathbb{N}$, y für jeden Wert in $\mathbb{N}$, sodass $f(y) \in L$, und z für jeden Wert in $\mathbb{N}$, sodass $f(z) \notin L$.

Diese Turing-Maschine akzeptiert L und hält auf jeder Eingabe. Beweisidee: Das komplette Eingabewort wird – kodiert durch f^{-1} – in eine Zelle am Ende des Wortes geschrieben. Liegt dieses Wort in L, so geht die Maschine in den Endzustand z_4. Ansonsten geht sie in z_5.

Aufgabe 3. *Eingeschränktes WHILE* 8 P.

Wir definieren eine neue Programmiersprache MinWHILE zur Berechnung von Funktionen $\mathbb{N}^k \to \mathbb{N}$.

Die Syntax von MinWHILE ist identisch zu der von WHILE mit folgenden zwei Einschränkungen:

1. Es dürfen nur Zuweisungen der Form $x_i \coloneqq x_i \pm c$ und keine der Form $x_i \coloneqq x_j \pm c$ mit $i \neq j$ verwendet werden.

2. Es darf keine LOOP-Schleife verwendet werden.

Die Semantik von MinWHILE ist identisch zu der von WHILE.

Zeigen Sie, dass MinWHILE Turing-mächtig ist, indem Sie beweisen, dass jedes WHILE-Programm durch ein MinWHILE-Programm simuliert werden kann.

Notation. Für ein WHILE- beziehungsweise ein MinWHILE-Programm Q sei N_Q die größte Zahl, sodass x_{N_Q} im Programm Q benutzt wird. Das bedeutet, dass Q nur die Variablen $x_0, x_1, \ldots, x_{N_Q}$ benutzt.

Lösung:

Beweis durch strukturelle Induktion.

1. Sei P ein WHILE-Programm der Form $x_i := x_j \pm c$ und sei $x_k := x_{N_P+1}$ eine in P unbenutzte Variable. Dann ist P':

   WHILE $x_i \neq 0$ **DO** $x_i := x_i - 1$ **END**;
   WHILE $x_j \neq 0$ **DO** $x_k := x_k + 1; x_j := x_j - 1$ **END**;
   WHILE $x_k \neq 0$ **DO** $x_k := x_k - 1; x_j := x_j + 1; x_i := x_i + 1$ **END**;
   $x_i := x_i \pm c$

   ein MinWhile-Programm, das P simuliert. In der ersten WHILE-Schleife wird der Wert von x_i auf 0 gesetzt. Anschließend wird der Wert von x_j in die Variable x_k kopiert und dabei x_j auf 0 gesetzt. In der letzten WHILE-Schleife wird der Wert von x_k sowohl in x_j als auch in x_i kopiert. Die letzte Ausführung erhöht bzw. verringert den Wert von x_i um die Konstante c. Zum Schluss gilt also: $x_k = 0$, x_j hat wieder seinen originalen Wert, und x_i ist gesetzt auf $x_j \pm c$.

2. Seien P_1 und P_2 zwei WHILE-Programme. Dann ist auch $P_1; P_2$ ein WHILE-Programm. Wenn P_1' und P_2' MinWHILE-Programme sind, die P_1 und P_2 simulieren, so simuliert das MinWHILE-Programm $P_1'; P_2'$ das WHILE-Programm $P_1; P_2$.

3. Sei P ein WHILE-Programm und sei P' ein MinWHILE-Programm, das P simuliert. Seien weiterhin $x_k := x_{N_{P'}} + 1$ und $x_\ell := x_{N_{P'}} + 2$ zwei in P' unbenutzte Variablen. Dann ist $\tilde{P}'$:

   WHILE $x_i \neq 0$ **DO** $x_k := x_k + 1; x_\ell := x_\ell + 1; x_i := x_i - 1$ **END**;
   WHILE $x_k \neq 0$ **DO** $x_k := x_k - 1; x_i := x_i + 1$ **END**;
   WHILE $x_\ell \neq 0$ **DO** $x_\ell := x_\ell - 1; P'$ **END**

   ein MinWHILE-Programm, das das WHILE-Programm $\tilde{P}$

   LOOP x_i **DO** P **END**

   simuliert. Betrachten wir die WHILE-Schleifen von $\tilde{P}'$. In der ersten Schleife wird der Wert von x_i nach x_k und x_ℓ kopiert und dabei x_i auf 0 gesetzt. In der zweiten Schleife wird der ursprüngliche Wert in x_i wiederhergestellt und dabei der Wert in x_k auf 0 gesetzt. In der dritten Schleife wird P' genau x_ℓ-Mal ausgeführt (beobachte, dass $x_\ell = x_i$). Da das Programm P' die Variable x_ℓ nicht benutzt, wird die Anzahl der Wiederholungen nicht durch P' beeinflusst.

4. Sei P ein WHILE-Programm und sei P' ein MinWHILE-Programm, das P simuliert. Dann ist

$$\textbf{WHILE } x_i \neq 0 \textbf{ DO } P' \textbf{ END}$$

ein MinWHILE-Programm, das das WHILE-Programm

$$\textbf{WHILE } x_i \neq 0 \textbf{ DO } P \textbf{ END}$$

simuliert.