Grundzüge der Theoretischen Informatik, WS 21/22: Musterlösung zum 8. Übungsblatt

Julian Dörfler

Aufgabe A8.1 (Kodierungsfunktion) *(4 Punkte)*
Wir hatten in der Vorlesung eine Kodierungsfunktion $\text{cod} : \{0,1\}^\star \to \mathbb{N}$ wie folgt definiert:

$$\text{cod}(x) = \text{bin}^{-1}(1x) - 1$$

Zeigen Sie, dass cod bijektiv ist.

Lösung A8.1 (Kodierungsfunktion) Wir zeigen zuerst, dass cod injektiv ist: Sei $\text{cod}(x) = \text{cod}(y)$, dann ist $\text{bin}^{-1}(1x) = \text{bin}^{-1}(1y)$. Da aber $1x$ und $1y$ keine führenden Nullen haben, können diese nur die selbe Zahl darstellen, wenn $x = y$.

Nun zeigen wir, dass cod ebenfalls surjektiv ist: Sei also $n \in \mathbb{N}$ beliebig. Wir wählen $x = \text{bin}(n+1)$. Hierbei enthält x in keinem Fall eine führende 0, da $n + 1 > 0$, somit gilt immer $x_1 = 1$. Nun ist

$$\begin{aligned}\text{cod}(x_2x_3\cdots x_{|x|}) &= \text{bin}^{-1}(1x_2x_3\cdots x_{|x|}) - 1\\ &= \text{bin}^{-1}(x) - 1\\ &= n + 1 - 1\\ &= n.\end{aligned}$$

Da cod also injektiv und surjektiv ist, ist cod nach Definition bijektiv.

Aufgabe A8.2 (Berechenbarkeit) *(4 Punkte)*
Wir sagen $x \in \mathbb{N}$ ist ein Fixpunkt einer partiellen Funktion f, wenn $f(x) = x$. Die Menge aller Fixpunkte von f definieren wir als

$$\text{Fix}(f) := \{x \in \mathbb{N} \mid f(x) = x\}.$$

Entscheiden Sie für die Sprache

$$L = \{i \in \mathbb{N} \mid \text{Fix}(\varphi_i) \text{ ist endlich}\}$$

für jede der folgenden Aussagen, ob diese wahr ist und beweisen Sie Ihre Antworten.

(a) L ist eine Indexmenge.

(b) $L \in \mathsf{REC}$

(c) $L \in \mathsf{RE}$

(d) $L \in \mathsf{co\text{-}RE}$.

Lösung A8.2 (Berechenbarkeit)

(a) L ist eine Indexmenge. Sei $i \in L$ und $j \in \mathbb{N}$ mit $\varphi_i = \varphi_j$. Dann gilt aber auch $\mathrm{Fix}(\varphi_i) = \mathrm{Fix}(\varphi_j)$, insbesondere ist $\mathrm{Fix}(\varphi_j)$ ebenfalls endlich und somit $j \in L$.

(b) Es gilt $L \notin \mathsf{REC}$. Dies zeigen wir mithilfe des Satz von Rice. Dazu hatten wir in Teilaufgabe (a) schon gezeigt, dass L eine Indexmenge ist, nun müssen wir noch zeigen, dass diese auch trivial ist:

$L \neq \emptyset$: Das Programm Ω terminiert auf keiner Eingabe und hat damit insbesondere keine Fixpunkte. Es gilt also $\mathrm{Fix}(\varphi_\Omega) = \emptyset$ und somit $\Omega \in L$.

$L \neq \mathbb{N}$: Die Identitätsfunktion ist `WHILE`-berechenbar. Für ein Programm mit Gödelisierung g, dass die Identsfunktion berechnet gilt nun aber $\mathrm{Fix}(\varphi_g) = \mathbb{N}$, die Menge ist also unendlich und es gilt $g \notin L$.

(c) Es gilt $L \notin \mathsf{RE}$. Aus dem expliziten Satz von Rice aus Aufgabe P8.1 gilt mit den vorherigen Teilaufgaben $\overline{H_0} \leq L$. Da $\overline{H_0} \notin \mathsf{RE}$ gilt somit also auch $L \notin \mathsf{RE}$.

Alternativer Beweis: Wir reduzieren dafür vom Komplement $\overline{H_0}$ des speziellen Halteproblems mit der Reduktionsfunktion

$$f(i) := \text{göd}(P_i)$$

wobei P_i das folgende Programm ist:

Gegeben m, simuliere i auf i und gib danach m aus.

Für $i \in \overline{H_0}$ hält i bei Eingabe i nicht. Das Programm P_i wird also auf keiner Eingabe terminieren und kann somit auch keine Fixpunkte haben. Es gilt also $\mathrm{Fix}(\varphi_{P_i}) = \emptyset$ und $f(i) = \text{göd}(P_i) \in L$.

Für $i \notin \overline{H_0}$ hält i bei Eingabe i. Das Programm P_i hält also für jede Eingabe und berechnet die Identitätsfunktion. Somit gilt $\mathrm{Fix}(\varphi_{P_i}) = \mathbb{N}$ und $f(i) = \text{göd}(P_i) \notin L$.

Da f `WHILE`-berechenbar ist und $\overline{H_0} \notin \mathsf{RE}$, ist also auch $L_2 \notin \mathsf{RE}$.

(d) Es gilt $L \notin \mathsf{co\text{-}RE}$. Wir reduzieren dafür vom speziellen Halteproblem H_0 mit der Reduktionsfunktion

$$f(i) := \text{göd}(P_i)$$

wobei P_i das folgende Programm ist:

Gegeben m, simuliere g auf g für m Schritte. Falls diese Simulation innerhalb der m Schritte terminiert divergiere, ansonsten gib m aus.

Für $i \in H_0$ existiert eine Schrittzahl $t \in \mathbb{N}$, so dass i bei Eingabe i nach t Schritten terminiert. Dann berechnet P_i auf Eingaben $m < t$ die Identitätsfunktion und divergiert für Eingaben $m \geq t$. Es gilt also $\text{Fix}(\varphi_{P_i}) = \{0, 1, \ldots, t-1\}$, insbesondere ist die Menge der Fixpunkte also endlich. Somit gilt $f(i) = \text{göd}(P_i) \in L$.

Für $i \notin H_0$ hält i bei Eingabe i nicht. Somit berechnet P_i für alle Eingaben die Identitätsfunktion und es gilt $\text{Fix}(\varphi_{P_i}) = \mathbb{N}$ und $f(i) = \text{göd}(P_i) \notin L$.

Da f `WHILE`-berechenbar ist und $H_0 \notin$ co-RE , ist also auch $L_2 \notin$ co-RE.

Aufgabe A8.3 (RE/co-RE-Vollständigkeit) *(4 Punkte)*
Eine Sprache L is RE-vollständig, wenn $L \in$ RE und für alle $L' \in$ RE gilt, dass $L' \leq L$.
Eine Sprache L is co-RE-vollständig, wenn $L \in$ co-RE und für alle $L' \in$ co-RE gilt, dass $L' \leq L$.

(a) Zeigen Sie, dass L RE-vollständig ist genau dann, wenn $\overline{L}$ co-RE-vollständig ist.

(b) Zeigen Sie, dass H RE-vollständig ist.

(c) Zeigen Sie, dass A co-RE-vollständig ist, wobei

$$A = \{i \in \mathbb{N} \mid \forall m : \varphi_i(m) \neq m^3 \text{ oder } \varphi_i(m) \text{ ist undefiniert}\}$$

Hinweis: Konstruieren Sie für Aufgabenteil (c) eine Reduktion von $\overline{H}$ und verwenden Sie Aufgabenteil (a).

Lösung A8.3 (RE/co-RE-Vollständigkeit)

(a) Sei L RE-vollständig. Dann ist $L \in$ RE und für alle $L' \in$ RE gilt $L' \leq L$. Somit gilt also $\overline{L} \in$ co-RE und für alle $L' \in$ RE gilt $\overline{L'} \leq \overline{L}$ nach Aufgabe P7.1 des siebten Präsenzblattes. $L' \in$ RE ist nun aber äquivalent zu $\overline{L'} \in$ RE, also gilt $\overline{L} \in$ co-RE.

Da alle diese Implikationen sogar Äquivalenzen waren gilt also die Aussage.

(b) Aus Aufgabe A6.4 wissen wir, dass $L \in$ RE impliziert, dass es ein `WHILE`-Programm W gibt, mit $\text{dom}(\varphi_W) = L$. Sei nun $f(x) := \langle W, x \rangle$. Diese Reduktionsfunktion ist offensichtlich `WHILE`-berechenbar. Nun gilt

$$x \in L \Rightarrow x \in \text{dom}(\varphi_W) \Rightarrow W \text{ hält auf Eingabe } x \Rightarrow \langle W, x \rangle \in H \Rightarrow f(x) \in H$$

und

$$x \notin L \Rightarrow x \notin \text{dom}(\varphi_W) \Rightarrow W \text{ hält nicht auf Eingabe } x \Rightarrow \langle W, x \rangle \notin H \Rightarrow f(x) \notin H\,.$$

Daraus folgt, dass $L \leq H$. Da $H \in$ RE bekannt ist, folgt dass H RE-vollständig ist.

(c) Nach Aufgabenteil (a) ist $\overline{H}$ co-RE-vollständig.

Nun zeigen wir $\overline{H} \leq A$ durch die folgende offensichtlich `WHILE`-berechenbare Funktion: Gegeben $\langle g, x \rangle$, geben wir die Gödelisierung des folgenden Programms $P_{g,x}$ aus:

Gegeben m, simuliere g auf x. Danach gib m^3 aus.

Sei nun $\langle g, x\rangle \in \overline{H}$. Dann hält g auf Eingabe x nicht. Daher hält $P_{g,x}$ auf keiner Eingabe m und berechnet somit insbesondere niemals m^3, also $f(\langle g, x\rangle) = \text{göd}(P_{g,x}) \in A$.
Sei nun $\langle g, x\rangle \notin \overline{H}$. Dann hält g auf Eingabe x. Daher hält $P_{g,x}$ auf jeder Eingabe m und berechnet insbesondere immer m^3, also $f(\langle g, x\rangle) = \text{göd}(P_{g,x}) \notin A$.

Sei nun $L' \in$ co-RE beliebig. Dann ist $L' \leq \overline{H}$ und durch die Transitivität von $\leq$ auch $L' \leq A$.

Bleibt also noch zu zeigen, dass $A \in$ co-RE. Wir geben dazu einen Semi-Entscheider für $\overline{A}$ an:

Bei Eingabe g, simuliere für alle Paare $\langle m, t\rangle$ g mit Eingabe m für t Schritte. Falls eine dieser Simulationen mit Ausgabe m^3 terminiert, gebe 1 aus.

A ist also co-RE-vollständig.

Aufgabe A8.4 (Universelles `FOR`-Programm) *(4 Punkte)*
In dieser Aufgabe dürfen Sie annehmen, dass es eine Gödelisierung $\mathtt{g}_{\mathtt{FOR}}$ für `FOR`-Programme gibt[1].

(a) Zeigen Sie, dass es *kein* universelles `FOR`-Programm gibt, d.h. ein `FOR`-Programm $U_{\mathtt{FOR}}$, welches als Eingabe eine Gödelisierung $\mathtt{g}_{\mathtt{FOR}}(P)$ und m erhält und $\Phi_P(m)$ ausgibt.

(b) Warum zeigt Ihr Beweis aus (a) nicht, dass es kein universelles `WHILE`-Programm gibt?

Lösung A8.4 (Universelles `FOR`-Programm)

(a) Wir nehmen an, es gibt $U_{\mathtt{FOR}}$ und konstruieren ein `FOR`-Programm p wie folgt: Bei Eingabe n wird $U_{\mathtt{FOR}}$ auf n und n angewendet. Danach wird das Ergebnis um 1 erhöht (es genügt, das Ergebnis nach der Simulation irgendwie zu verändern). Dann gilt:

$$\Phi_{U_{\mathtt{FOR}}}(g_{\mathtt{FOR}}(p), g_{\mathtt{FOR}}(p)) = \Phi_p(g_{\mathtt{FOR}}(p)) = \Phi_{U_{\mathtt{FOR}}}(g_{\mathtt{FOR}}(p), g_{\mathtt{FOR}}(p)) + 1\,,$$

was ein Widerspruch ist.

(b) `FOR`-Programme können nie undefiniert sein, `WHILE`-Programme schon, somit können wir das Ergebnis der Ausführung des universellen `WHILE`-Programmes danach nicht notwendigerweise verändern.

[1] Sie sollten sich aber klar machen, wie die Definition der Gödelisierungsfunktion für `WHILE`-Programme abgeändert werden kann um eine Gödelisierungsfunktion für `FOR`-Programme zu erhalten.

Aufgabe A8.5 (Halte-Orakel) *(4 Bonuspunkte)*
In dieser Aufgabe stellen wir eine Kette von Erweiterungen $\texttt{WHILE} = \texttt{WHILE}_0 \subsetneq \texttt{WHILE}_1 \subsetneq \texttt{WHILE}_2 \subsetneq \ldots$ vor, wobei jede Menge echt mächtiger ist, als die vorherige. Wir erhalten $\texttt{WHILE}_{i+1}$ aus $\texttt{WHILE}_i$, indem wir ein *Halte-Orakel* für Programme aus $\texttt{WHILE}_i$ hinzufügen[2]:

$$s = \text{“}x_o := \mathsf{halts}_i\ x_p\ x_m\text{”}\ .$$

Die Semantik von s ist wie folgt definiert: Wenn das $\texttt{WHILE}_i$-Programm $\text{göd}_i^{-1}(x_p)$ bei Eingabe x_m hält, wird 1 in x_o geschrieben, und 0 sonst. Hierbei dürfen (und sollten) Sie annehmen, dass es eine bijektive Gödelisierungsfunktion göd_i für $\texttt{WHILE}_i$-Programme gibt[3].

(a) Zeigen Sie, dass χ_{H_0} $\texttt{WHILE}_1$-berechenbar ist.

(b) Finden Sie für jedes $i \geq 1$ eine Sprache deren charakteristische Funktion nicht $\texttt{WHILE}_i$-berechenbar ist (und beweisen Sie, dass Ihre Sprache nicht $\texttt{WHILE}_i$-berechenbar ist).

(c)* Welches ist das kleinste i, so dass χ_{V_0} $\texttt{WHILE}_i$ berechenbar ist. Beweisen Sie Ihre Antwort.
Hinweis: Sie dürfen - um zu zeigen, dass χ_{V_0} nicht $\texttt{WHILE}_{i-1}$-berechenbar ist - annehmen, dass wenn V_0 $\texttt{WHILE}_{i-1}$-entscheidbar ist und $L \leq_{i-1} V_0$ eine $\texttt{WHILE}_{i-1}$-berechenbare Reduktion ist, dass dann L ebenfalls $\texttt{WHILE}_{i-1}$-entscheidbar ist.

Lösung A8.5 (Halte-Orakel)

(a) Gegeben g in x_p, führen wir $x_o := \mathsf{halts}_0\ x_p\ x_p$ aus und geben x_o zurück.

(b) Sei $i \geq 1$ und

$$H_i = \{j \in \mathbb{N} \mid \text{göd}_i^{-1}(j) \text{ hält auf Eingabe } j\}$$

Der Beweis, dass χ_{H_i} nicht $\texttt{WHILE}_i$-berechenbar ist, ist - bis auf die Verwendung von göd_i statt göd - *derselbe* Beweis, der zeigt, dass χ_{H_0} nicht $\texttt{WHILE}$-berechenbar ist.

(c) Wir zeigen zuerst, dass χ_{V_0} $\texttt{WHILE}_2$-berechenbar ist:
Hierzu betrachten wir folgendes $\texttt{WHILE}_1$-Programm P:
Input: g
1: **loop** über alle möglichen $m \in \mathbb{N}$:
2: **if** $\mathsf{halts}_0\ g\ m$ **then**
3: Simuliere g mit Eingabe m.
4: Wenn diese Simulation einen Wert $\neq 0$ ausgibt, beende das Programm mit Ausgabe 1.
5: **else**
6: Beende das Programm mit Ausgabe 1.
7: **fi**

[2]Siehe Definition 7.1 (simple statements) im Vorlesungsskript.
[3]Hierbei gilt natürlich $\text{göd}_0 = \text{göd}$

8: **end loop**

Falls nun $g \in V_0$ ist, so hält g auf jeder Eingabe mit Ausgabe 0, damit divergiert P auf Eingabe g.
Falls nun $g \notin V_0$ ist, so gibt es eine Eingabe bei der g entweder divergiert oder eine Ausgabe $\neq 0$ produziert. In beiden Fällen terminiert P auf Eingabe g mit Ausgabe 1.

P (ein $\texttt{WHILE}_1$ Programm) berechnet also $\chi'_{\overline{V_0}}$.

Wir können nun aber $g \in V_0$ mit Hilfe eines $\texttt{WHILE}_2$-Programmes entscheiden, indem wir das Halte-Orakel halts_1 benutzen um zu prüfen, ob P bei Eingabe g terminiert und in diesem Fall 0 ausgeben und 1 sonst.

Wir müssen also noch zeigen, dass V_0 nicht von $\texttt{WHILE}_1$-Programmen entschieden werden kann. Dazu zeigen wir $\overline{H_1} \leq V_0$. Dies impliziert dann auch $\overline{H_1} \leq_1 V_0$ mit der selben Reduktionsfunktion.

Gegeben die Gödelisierung eines $\texttt{WHILE}_1$ Programmes g, gibt die Reduktionsfunktion f die Gödelisierung des folgenden WHILE-Programmes P_g aus:
Input: Schrittzahl t
Input: Array *guesses*
Input: Array *stepcount*
1: Simuliere t Schritte von g auf Eingabe g, wobei der i-te Halte-Orakel-Aufruf mit $guesses[i]$ beantwortet wird. Falls das Array *guesses* Länge $< i$ hat, oder $guesses[i] \notin \{0, 1\}$, so beende P_g mit Ausgabe 0. Zusätzlich speichere die Eingabe an das Halte-Orakel als $p[i]$ und $m[i]$
2: Hält g innerhalb der t Schritte nicht, so beende P_g mit Ausgabe 0.
3: **loop** über alle Anfragen $p[i]$ $m[i]$ an das Halte-Orakel, die mit 1 beantwortet wurden.
4: Falls das Array *stepcount* eine Länge $< i$ hat, so beende P_g mit Ausgabe 0.
5: Simuliere $p[i]$ auf Eingabe $m[i]$ für $stepcount[i]$ Schritte.
6: Falls $p[i]$ innerhalb dieser Simulation nicht terminiert beende P_g mit Ausgabe 0.
7: **end loop**
8: Simuliere interleaved für alle Anfragen $m[i]$ $p[i]$ an das Halte-Orakel, die mit 0 beantwortet wurden, $m[i]$ auf Eingabe $p[i]$.
9: Falls irgendeine dieser Simulationen terminiert, gib 0 aus.
10: Falls es keine Anfragen an das Halte-Orakel gab, die mit 0 beantwortet wurden, dann divergiere.

f ist WHILE und damit auch $\texttt{WHILE}_1$ berechenbar, da f nur ein neues Programm konstruiert und somit keine Halte-Orakel benötigt. P_g selbst ist ebenfalls ein WHILE-Programm, da wir zwar g simulieren, aber den halts_0-Befehl durch WHILE-berechenbare Befehle austauschen.

Falls nun $g \in \overline{H_1}$, dann terminiert das $\texttt{WHILE}_1$-Programm $\text{göd}_1^{-1}(g)$ nicht auf Eingabe g. Falls *guesses* oder *stepcount* keine lang genugen Arrays sind, bzw *guesses*

einen Eintrag $\notin \{0,1\}$ enthält, so gibt P_g als Ausgabe 0 aus. Somit terminiert bei "korrekter" Simulation der Halte-Orakel-Aufrufe P_g in Zeile 2 mit Ausgabe 0. Ansonsten muss irgendein Orakel-Aufruf falsch beantwortet worden sein. Sei der i-te Orakel-Aufruf falsch beantwortet:

$p[i]$ hält auf Eingabe $m[i]$, aber $guesses[i] = 0$: In diesem Fall wird P_g irgendwann in Zeile 9 0 ausgeben, da die Simulation von $p[i]$ auf Eingabe $m[i]$ hält.

$p[i]$ hält auf Eingabe $m[i]$ nicht, aber $guesses[i] = 1$: In diesem Fall wird P_g in Zeile 6 0 ausgeben, da die Simulation von $p[i]$ auf Eingabe $m[i]$ nach $stepcount[i]$ Schritten nicht halten kann.

Somit gibt P_g bei jeder Eingabe 0 aus, also $f(g) = \text{göd}(P_g) \in V_0$.

Falls nun $g \notin \overline{H_1}$, so existiert eine Schrittzahl t, so dass das $\texttt{WHILE}_1$-Programm $\text{göd}_1^{-1}(g)$ auf Eingabe g nach t Schritten hält. Wärend dieser Ausführung macht g maximal t Halte-Orakel aufrufe. Sei *guesses* ein Array, dass an der i-ten Stelle 1 ist, falls der i-te Aufruf des Halte-Orakels während dieser Ausführung 1 berechnet und 0 sonst. Weiterhin sei *stepcount* so, dass dieses wenn der i-te Aufruf des Halte-Orakels $\mathsf{halts}_0\ x_p\ x_m$ war und mit 1 beantwortet wurde (x_p also auf Eingabe x_m hält), die Anzahl an Schritten enthält, so dass x_p bei Eingabe x_m nach $stepcount[i]$ Schritten terminiert.

Dann wird P_g auf Eingabe $t, guesses, stepcount$ divergieren, da die Simulation von g mit Eingabe g sich exakt so verhält, wie bei einer echten Ausführung von g auf Eingabe g, g also weiterhin nach t Schritten terminiert. Zusätzlich halten alle Programme, bei deren i-ter Halte-Orakel-Anfrage wir 1 zurückgegeben haben nach $stepcount[i]$ Schritten, während keines der Programme bei denen wir 0 zurückgegeben haben hält. Somit divergiert P_g am Ende bei der Simulation aller negativ beantworteten Halte—Orakel Anfragen. Damit gilt $f(g) = \text{göd}^{-1}(P_g) \notin V_0$.

Da H_1 nicht $\texttt{WHILE}_1$-entscheidbar ist, ist auch $\overline{H_1}$ nicht $\texttt{WHILE}_1$-entscheidbar und somit ist durch die Reduktion V_0 nicht $\texttt{WHILE}_1$-entscheidbar.