Hausaufgaben zur Vorlesung **SoSe 2017**
Berechenbarkeit und Komplexität **TU Berlin**
Niedermeier/Froese/Molter **16.06.2017**

Hausaufgabenblatt

Hinweise:

- Die Hausaufgabe kann im Zeitraum **vom 26.06.17 bis 30.06.17** in den Tutorien abgegeben werden. Letzte Abgabemöglichkeit ist am **30.06.17 vor** der Vorlesung von **12:00 bis 12:10 Uhr** im Hörsaal.
- Die Hausaufgabe muss in Gruppen mit mindestens 2 und maximal 3 Personen bearbeitet werden. (Bei Problemen mit der Gruppenfindung, bitte bei uns melden.)
- Schreiben Sie Namen und Matrikelnummern aller Gruppenmitglieder auf die erste Seite Ihrer Abgabe. Schreiben Sie zusätzlich auf jedes abgegebene Blatt die Matrikelnummern aller Gruppenmitglieder.
- Plagiate werden nicht toleriert. Sollten zwei oder mehr Abgaben auffällig ähnliche Lösungen enthalten, so werden **alle** Abgaben mit **0 Punkten** bewertet!
- Es können bis zu **25 Portfoliopunkte** erreicht werden.
- Alle Antworten sind zu begründen (wenn nicht explizit ausgeschlossen). Antworten ohne Begründung erhalten **0 Punkte**. Achten Sie insbesondere darauf, die Korrektheit Ihrer angegebenen Lösungen (Programme, Reduktionen etc.) zu begründen.
- Sie dürfen in Ihren LOOP- und WHILE-Programmen andere Hilfskonstrukte als die angegebenen nur dann verwenden, wenn Sie diese vorher als LOOP- bzw. als WHILE-Programm definiert haben.
- Bitte bei handschriftlichen Abgaben einen *dokumentenechten* Stift verwenden.
- Wir gehen davon aus, dass die Lösungen weniger als 5 Seiten benötigen. Bitte versuchen Sie, nicht mehr als 5 Seiten (in normaler Schriftgröße mit normaler Befüllung) zu schreiben.

Aufgabe 1. *LOOP- und WHILE-Programmierung* 8 P.

Im Folgenden gelten die Definitionen für LOOP- und WHILE-Programme aus der Vorlesung. Darüberhinaus dürfen Sie in LOOP- und WHILE-Programmen folgende Hilfskonstrukte verwenden:

- $x_i := c$ (für eine konstante $c \in \mathbb{N}$)
- $x_i := x_j$
- $x_i := x_j + x_\ell$
- $x_i := x_j - x_\ell$ (modifizierte Subtraktion)
- **IF** $x_i = 0$ **THEN** P **END** (für ein LOOP- bzw. WHILE-Programm P)
- $x_i := x_j$ **MOD** x_ℓ (Modulo-Funktion, wobei x_j **MOD** $0 := 0$)

Erinnerung: Die Eingabewerte stehen zu Programmbeginn in $x_1, \ldots, x_k$. Alle anderen Variablen sind initial 0. Die Ausgabe muss am Programmende in x_0 stehen.

Gegeben seien die folgenden Funktionen

- $\text{teilbar} : \mathbb{N}^2 \to \{0,1\}$ mit $\text{teilbar}(x,y) = \begin{cases} 1, & \text{falls ein } k \in \mathbb{N} \text{ existiert, sodass } x = y \cdot k \\ 0, & \text{sonst} \end{cases}$
- $\text{kleiner} : \mathbb{N}^2 \to \{0,1\}$ mit $\text{kleiner}(x,y) = \begin{cases} 1, & \text{falls } x < y \\ 0, & \text{sonst} \end{cases}$

1) (3 P.) Die Funktion $f : \mathbb{N}^2 \to \mathbb{N}$ sei definiert als

$$f(0,y) = \text{teilbar}(y,0)$$
$$f(x+1,y) = f(x,y) + \text{teilbar}(y,x+1).$$

Was gibt der Funktionswert $f(x,y)$ an? Geben Sie ein LOOP-Programm an, das die Funktion f berechnet.

2) (5 P.) Die Funktion $g : \mathbb{N}^2 \to \mathbb{N}$ sei definiert als $g(x,y) = \text{kleiner}(f(x,x),y)$, wobei f die Funktion aus der vorherigen Aufgabe ist.

Welche Funktion wird von $\mu(g)$ berechnet? Geben Sie hierfür den Typ an und erklären Sie, was der Funktionswert ist (hierbei soll natürlich nicht einfach nur die Definition des μ-Operators eingesetzt werden, sondern eine umgangssprachliche Beschreibung samt Begründung gegeben werden).

Schreiben Sie ein WHILE-Programm, das die Funktion $\mu(g)$ berechnet.

Hinweis: Sie können hierfür annehmen, dass P_f ein LOOP-Programm zur Berechnung von f ist. Sie dürfen die Zuweisung $x_i := P_f(x_j, x_\ell)$ verwenden (die Werte in x_j und x_ℓ sind hier die Eingaben) und davon ausgehen, dass in P_f keine Variablen vorkommen, die in Ihrem Programm verwendet werden.

Lösung 1.

1) Der Funktionswert $f(x, y)$ ist die Anzahl der Teiler von y, die kleiner gleich x sind, denn für diese gilt

$$f(x+1, y) = \begin{cases} 1 + f(x, y), & \text{falls } x+1 \text{ Teiler von } y \text{ ist} \\ f(x, y), & \text{sonst} \end{cases} = f(x, y) + \text{teilbar}(y, x+1),$$

$$\text{wobei } f(0, y) = \begin{cases} 1, \text{ für } y = 0 \\ 0, \text{ für } y > 0 \end{cases} = \text{teilbar}(y, 0).$$

Folgendes LOOP-Programm berechnet f.

IF $x_2 = 0$ **THEN** $x_0 := 1$ **END**;
LOOP x_1 **DO**
    $x_3 := x_3 + 1$;
    $x_4 := x_2$ **MOD** x_3;
    IF $x_4 = 0$ **THEN** $x_0 := x_0 + 1$ **END**
END

Falls $x_2 = 0$ gilt, wird zunächst x_0 auf 1 gesetzt (da 0 den Teiler 0 besitzt). Dann wird für alle Zahlen $x_3 = 1, \ldots, x_1$ in einer LOOP-Schleife geprüft, ob x_3 ein Teiler von x_2 ist (also ob $x_2 \bmod x_3 = 0$) und x_0 gegebenfalls um eins erhöht.

2) Nach Definition des μ-Operators ist $\mu(g)$ vom Typ $\mathbb{N} \to \mathbb{N}$. Der Funktionswert $\mu(g)(n)$ ist die kleinste Zahl m, die mindestens n Teiler der Größe höchstens m hat. Dazu beobachten wir zunächst, dass f eine totale Funktion ist, da sie rekursiv als Komposition totaler Funktionen definiert ist. Somit ist auch g als Komposition totaler Funktionen total. Nach Definition des μ-Operators ist $\mu(g)(n)$ demnach die kleinste Zahl m, sodass $g(m, n) = \text{kleiner}(f(m, m), n) = 0$. Also ist m die kleinste Zahl mit $f(m, m) \geq n$. Ein solches m existiert für jede Zahl n (z.B. hat $n!$ mindestens n Teiler).

Folgendes WHILE-Programm berechnet $\mu(g)$.

$x_2 := P_f(x_0, x_0)$;
$x_3 := x_1 - x_2$;
WHILE $x_3 \neq 0$ **DO**
    $x_0 := x_0 + 1$;
    $x_2 = P_f(x_0, x_0)$;
    $x_3 := x_1 - x_2$
END

In x_2 steht immer der aktuelle Wert $f(x_0, x_0)$ und x_3 speichert die modifizierte Differenz $x_1 - f(x_0, x_0)$. Solange x_3 nicht 0 ist (also $x_1 > f(x_0, x_0)$) wird in der WHILE-Schleife der Wert von x_0 um eins erhöht und x_2 neu berechnet. Sobald x_3 den Wert 0 annimmt, gilt also $f(x_0, x_0) \geq x_1$ und das Programm terminiert mit der Ausgabe x_0.

Aufgabe 2. *Kürzeste WHILE-Programme* 9 P.

Ein *modifiziertes* WHILE-Programm ist folgendermaßen definiert:

Für Variablen $x_0, x_1, \ldots$ sind

- $x_i := 0$
- $x_i := x_j$
- $x_i := x_j + 1$
- $x_i := x_j - 1$ (modifizierte Subtraktion von 1)
- $x_i := x_j \cdot 2$ (Multiplikation mit 2)

modifizierte WHILE-Programme. Für modifizierte WHILE-Programme P, P' sind

- $P; P'$
- **WHILE** $x_i \neq 0$ **DO** P **END**
- **IF** $x_i > x_j$ **THEN** P **END**

ebenfalls modifizierte WHILE-Programme.

Ein *erzeugendes* Programm für eine Zahl $n \in \mathbb{N}$ ist ein modifiziertes WHILE-Programm, das auf der Eingabe $x_1 = 0$ die Zahl n in x_0 ausgibt. (Gehen Sie immer davon aus, dass alle anderen Variablen initial mit 0 belegt sind.)

Als *Länge* eines Programms bezeichnen wir die Summe aus der Anzahl der Zuweisungen (:=), der Anzahl der WHILE-Schleifen und der Anzahl der IF-Abfragen.

1) (2 P.) Argumentieren Sie zunächst, dass jede WHILE-berechenbare Funktion auch von einem *modifizierten* WHILE-Programm berechnet werden kann.

2) (2 P.) Beweisen Sie, dass es keine Konstante $c \in \mathbb{N}$ gibt, sodass für jede natürliche Zahl n ein erzeugendes Programm existiert, das kürzer als c ist.

3) (5 P.) Sei $\phi : \mathbb{N} \to \mathbb{N}$ die Funktion, die für eine natürliche Zahl n die Länge eines kürzesten erzeugenden Programms für n angibt. Beweisen Sie, dass $\phi(n)$ nicht WHILE-berechenbar ist, indem Sie einen Widerspruchsbeweis führen, bei dem Sie wie folgt vorgehen:
   - Gehen Sie davon aus, dass es ein modifiziertes WHILE-Programm P_ϕ gibt, welches auf Eingabe $x_1 = n$ die Ausgabe $x_0 = \phi(n)$ berechnet. Nehmen Sie an, dass P_ϕ die Länge c_ϕ hat und dass $x_{\max}$ die Variable mit dem größten vorkommenden Index ist. Sie dürfen desweiteren annehmen, dass P_ϕ den Inhalt von x_1 nicht ändert.
   - Konstruieren Sie ein neues modifiziertes WHILE-Programm, welches eine Zahl n in x_0 ausgibt, aber kürzer ist als $\phi(n)$. Sie dürfen dabei die Zuweisung $x_0 := P_\phi(x_1)$ verwenden, was aber die Länge Ihres Programms um c_ϕ erhöht. Sie dürfen

in Ihrem Programm Zuweisungen der Art $x_i := x_j + c$ verwenden. Falls Sie keine weiteren Erklärungen hinzufügen, erhöht eine solche Zuweisung die Länge Ihres Programms um c.

Lösung 2.

1. Wir beweisen die Aussage induktiv über den Aufbau von WHILE-Programmen. Dabei müssen wir nur die beiden folgenden Fälle betrachten (alle anderen Fälle sind auch in der Definition von modifizierten WHILE-Programmen enthalten):
   - $x_i := x_j + c$ (bzw. $x_i := x_j - c$): Offensichtlich kann eine solche Zuweisung durch c Zuweisungen $x_i := x_j + 1$ (bzw. $x_i := x_j - 1$) simuliert werden.
   - **LOOP** x_i **DO** P **END**: Dass LOOP-Schleifen, durch WHILE-Schleifen simuliert werden können, wurde bereits in der Vorlesung gezeigt.

2. Wenn $\phi(n) < c$ für alle $n \in \mathbb{N}$ und eine Konstante $c \in \mathbb{N}$, so könnte jede natürliche Zahl durch ein erzeugendes Programm der Länge höchstens c berechnet werden. Es gibt aber nur endlich viele (bis auf Variablenumbenennung) äquivalente solcher Programme für jede Konstante c, da diese nur aus einer endlichen Menge von Anweisungen erzeugt werden können. Da jedes erzeugende Programm genau eine Zahl erzeugt, folgt also ein Widerspruch, da es unendlich viele natürliche Zahlen gibt.

3. Es gelten alle Annahmen aus der Aufgabenstellung. Dann erzeugt folgendes modifiziertes WHILE-Programm einen Widerspruch:

   $x_{\max+1} := x_{\max+1} + 7;$
   $x_{\max+1} := x_{\max+1} + c_\phi;$
   $x_{\max+1} := x_{\max+1} \cdot 2;$
   $x_1 := x_1 + 1;$
   WHILE $x_1 \neq 0$ **DO**
   $\quad x_1 := x_1 + 1;$
   $\quad x_0 := P_\phi(x_1);$
   $\quad$ **IF** $x_0 > x_{\max+1}$ **THEN**
   $\quad\quad x_0 := x_1;$
   $\quad\quad x_1 := 0$
   $\quad$ **END**
   END

   Erklärung: Das obige Programm hat die Länge $2(7 + c_\phi)$. Dieser Wert wird zu Beginn in $x_{\max+1}$ geschrieben. Daraufhin wird in einer WHILE-Schleife für wachsende Zahlen x_1 der Wert $\phi(x_1)$ berechnet (wir nehmen hierbei an, dass in P_ϕ alle verwendeten Variablen zu Beginn mit 0 initialisiert werden) bis $\phi(x_1) > 2(7 + c_\phi)$ gilt. Diese WHILE-Schleife terminiert immer, da in Aufgabenteil 2) gezeigt wurde, dass ϕ unbeschränkt ist. Bei Programmende wird eben dieser Wert $\phi(x_1)$ in x_0 ausgegeben. Das Programm hat also Länge $2(7 + c_\phi)$ und gibt bei Eingabe $x_1 = 0$ eine Zahl aus, die größer ist als $\phi(2(7 + c_\phi))$, was ein Widerspruch zur Definition

von ϕ ist. Also existiert kein modifiziertes WHILE-Programm für ϕ und somit nach Aufgabenteil 1) auch kein (herkömmliches) WHILE-Programm.

Aufgabe 3. *(Semi-)Entscheidbar oder nicht?* 8 P.

Zeigen Sie für jede der folgenden Sprachen, ob sie entscheidbar ist oder nicht. Falls eine Sprache entscheidbar ist, beschreiben Sie hierfür die algorithmische Arbeitsweise einer deterministischen Turing-Maschine, die diese Sprache entscheidet. Um zu zeigen, dass eine Sprache unentscheidbar ist, geben Sie eine Reduktion von einem aus der Vorlesung bekannten unentscheidbaren Problem auf die jeweilige Sprache an. Begründen Sie außerdem für jede unentscheidbare Sprache, ob diese semi-entscheidbar ist oder nicht.

Hinweis: Im Folgenden sei $\mathrm{BIN}^{-1}(x) \in \mathbb{N}$ für $x \in \{0,1\}^*$ die natürliche Zahl, die durch x binärkodiert wird (wobei für das leere Wort ε gilt, dass $\mathrm{BIN}^{-1}(\varepsilon) := 0$) und M_w bezeichne die von $w \in \{0,1\}^*$ kodierte Turing-Maschine (wie in der Vorlesung definiert). Die Anzahl der *Besuche* eines Zustands während einer Berechnung entspricht der Anzahl an Konfigurationen in der Konfigurationsfolge dieser Berechnung, in denen der Zustand auftaucht.

1) (2 P.) $L_1 = \{w\#x \mid w, x \in \{0,1\}^*$ und M_w besucht auf Eingabe x jeden ihrer Nicht-Endzustände mindestens einmal$\}$

2) (2 P.) $L_2 = \{w\#x \mid w, x \in \{0,1\}^*$ und M_w hat $\mathrm{BIN}^{-1}(x)$ Zustände$\}$

3) (2 P.) $L_3 = \{w\#x \mid w, x \in \{0,1\}^*$ und M_w besucht auf Eingabe x jeden ihrer Nicht-Endzustände genau einmal, bevor sie einen Zustand zum zweiten Mal besucht oder akzeptiert$\}$

4) (2 P.) $L_4 = \{w\#x \mid w, x \in \{0,1\}^*$ und M_w besucht auf Eingabe x mindestens einen ihrer Nicht-Endzustände nicht$\}$

Lösung 3.

1) Unentscheidbar: Wir zeigen $K \leq L_1$ mit folgender Reduktion. Für $w \in \{0,1\}^*$ sei $f(w) := \langle M \rangle \# w \in \{0,1,\#\}^*$ mit der Kodierung der Turing-Maschine M, die eine Kopie von M_w ist, mit folgenden Modifikationen:

   - Das Bandalphabet von M hat einen neuen Buchstaben a.
   - M hat einen neuen Nicht-Endzustand z^*.
   - Die Übergangsfunktion wird folgendermaßen modifiziert: sobald für einen Nicht-Endzustand von M_w ein Übergang (für gelesene 1 oder 0) nicht definiert ist oder in einen Endzustand führt, wechselt die Übergangsfunktion von M stattdessen in den Zustand z^* und schreibt ein a auf das Band. Der Kopf wird nicht bewegt. Danach besucht die TM M in beliebiger aber fester Reihenfolge nacheinander alle Nicht-Endzustände von M_w. Bei jedem Übergang wird a gelesen und wieder auf das Band geschrieben und der Kopf nicht bewegt. Wenn der letzte Zustand in der Reihenfolge besucht wurde, hält die TM M.

   Die Kodierung $\langle M \rangle$ ist berechenbar, da sie im Wesentlichen auf der (De)Kodierung von M_w beruht, welche berechenbar ist. Sonst werden nur endlich viele Modifikationen vorgenommen, die ebenfalls berechenbar sind. Also ist die Funktion f berechenbar.

Bei Eingabe $x \in \{0,1\}^*$ besucht M den Zustand z^* genau dann, wenn M_w auf x hält. Wenn z^* besucht wird, werden auch alle anderen Nicht-Endzustände besucht. Also gilt für alle $w \in \{0,1\}^*$, dass

$$\begin{aligned} w \in K &\Leftrightarrow M_w \text{ hält auf } w \\ &\Leftrightarrow M \text{ besucht } z^* \\ &\Leftrightarrow M \text{ besucht alle Nicht-Endzustände} \\ &\Leftrightarrow f(w) = \langle M \rangle \# w \in L_1. \end{aligned}$$

Die Sprache L_1 ist semi-entscheidbar, da die gegebene TM M_w auf Eingabe x simuliert werden kann. Falls $w\#x \in L_1$, so hat M_w also nach endlich vielen Schritten irgendwann alle Nicht-Endzustände besucht. Dann kann also die Simulation abgebrochen und $w\#x$ akzeptiert werden.

2) Entscheidbar: Eine DTM, die L_2 entscheidet, dekodiert zunächst das Wort w und zählt die Anzahl der Zustände von M_w in Binärkodierung und vergleicht diese mit x.

3) Entscheidbar: Eine DTM M_3, die L_3 entscheidet, arbeitet wie folgt: M_3 dekodiert das Wort w, um die TM M_w zu erhalten. Diese hat eine feste Anzahl k von Nicht-Endzuständen. Falls M_w nur Endzustände besitzt (d.h. $k = 0$), dann akzeptiert M_3. Sonst simuliert M_3 die ersten $k-1$ Berechnungsschritte von M_w auf Eingabe x.

   - Falls M_w dabei hält, ohne zu akzeptieren oder einen Zustand zweimal zu besuchen, so akzeptiert M_3.
   - Falls M_w dabei einen Zustand zwei mal besucht oder die Eingabe akzeptiert, so lehnt M_3 ab.

   Andernfalls hat M_w in diesen $k-1$ Schritten jeden Nicht-Endzustand genau einmal besucht. Daher akzeptiert M_3 in diesem Fall. Die DTM M_3 entscheidet somit immer in endlicher Zeit, ob ein gegebenes Wort in L_3 liegt.

4) Unentscheidbar: L_4 ist das Komplement von L_1, daher folgt mit derselben Reduktion wie in 1), dass $\overline{K} \leq L_4$ (siehe VL Bemerkung auf Folie 86).

   Die Sprache L_4 ist nicht semi-entscheidbar, da $\overline{L_4} = L_1$ semi-entscheidbar, aber nicht entscheidbar ist.