Übungen zur Vorlesung Grundbegriffe der Theoretischen Informatik

technische universität dortmund

Thomas Schwentick

Gaetano Geck, Lutz Oettershagen,
Christopher Spinrath, Marco Wilhelm

Lehrstuhl 1
Logik in der Informatik

SoSe 2018 | **Übungsblatt 10** | **19.06.2018**

Aufgabe 10.1 [Kodierung einer Turingmaschine] **2 Punkte**

Die Turingmaschine $M = (\{s, p\}, \{\triangleright, \sqcup, 0, 1\}, \delta, s)$ über dem Alphabet $\Sigma = \{0, 1\}$ ist ein binärer Dekrementierer: Die Binärzahl w^R wird um 1 dekrementiert, falls $\mathrm{Str2N}(w^R) > 0$ gilt. Die Transitionen $(q, \sigma, q', \sigma', d)$ aus δ sind dabei wie folgt definiert:

q	σ	q'	σ'	d
s	$\triangleright$	s	$\triangleright$	$\rightarrow$
s	0	s	0	$\rightarrow$
s	1	p	0	$\leftarrow$
s	$\sqcup$	h	$\sqcup$	$\downarrow$
p	0	p	1	$\leftarrow$
p	$\triangleright$	h	$\triangleright$	$\downarrow$

a) Wie werden die Symbole und Zustände von M gemäß den im Abschnitt „Universelle Turingmaschinen“ von Kapitel 16 genannten Konventionen kodiert, wenn M Eingabe für eine universelle Turingmaschine ist? Verwenden Sie die folgende Nummerierung der Zustände, Richtungen und Zeichen und ergänzen Sie die Tabellen:

x	num(x)	enc(x)
s	1	
h	4	
p	5	
$\leftarrow$	1	
$\downarrow$	2	
$\rightarrow$	3	

x	num(x)	enc(x)
$\triangleright$	1	
$\sqcup$	2	
0	3	
1	4	

(1 Punkt)

b) Geben Sie die Kodierungen $\mathrm{enc}(s, 1)$ und $\mathrm{enc}(p, 0)$ an. **(1 Punkt)**

Lösung:

a) Wir kodieren x für $x \in (Q \cup \{h\}) \cup \{\leftarrow, \downarrow, \rightarrow\} \cup \Gamma$ gemäß $\mathrm{enc}(x) = 0^{\mathrm{num}(x)}1$:

x	s	h	p	$\leftarrow$	$\downarrow$	$\rightarrow$	$\triangleright$	$\sqcup$	0	1
enc(x)	01	00001	000001	01	001	0001	01	001	0001	00001

b) Wir kodieren enc$(s, 1)$ und enc$(p, 0)$ für $\delta(q, \sigma) = (q', \sigma', d)$ gemäß

$$\text{enc}(q, \sigma) = 1\text{enc}(q)\text{enc}(\sigma)\text{enc}(q')\text{enc}(\sigma')\text{enc}(d)$$

(die Leerstellen dienen lediglich der Lesbarkeit und sind eigentlich nicht vorhanden):

x	enc(x)
$(s, 1)$	1 01 00001 000001 0001 01
$(p, 0)$	1 000001 0001 000001 00001 01

Aufgabe 10.2 [Abschlusseigenschaften] 4 Punkte

Es seien L_1 und L_2 Sprachen über einem Alphabet Σ und M_1 bzw. M_2 Turingmaschinen für die $L_1 = L(M_1)$ und $L_2 = L(M_2)$ gilt. Beschreiben Sie anschaulich, wie sich Turingmaschinen M_a und M_b für die folgende Sprachen L_a und L_b konstruieren lassen, so dass jeweils $L_x = L(M_x)$ gilt. Bedenken Sie, dass M_x die Sprache L_x nicht *entscheiden* muss. Es ist also möglich, dass sie für einige Eingaben nicht terminiert.

a) $L_a = L_1 \cup L_2$ **(2 Punkte)**

b) $L_b = L_1 \circ L_2$ **(2 Punkte)**

Hinweis

M_1 und M_2 können als 1-String-Turingmaschinen angenommen werden. Für M_a und M_b dürfen Sie Mehrstring-Turingmaschinen verwenden. Sie müssen die (Mehrstring-)Turingmaschinen M_a und M_b weder formal aufschreiben, noch als Diagramm angeben. Beschreiben Sie jedoch die Arbeitsweise hinreichend verständlich und nutzen Sie dafür die Tatsache, dass Turingmaschinen beliebige andere Turingmaschinen simulieren können.

Lösung:

a) Eine 2-String-Turingmaschine M_a für die Sprache L_a arbeitet folgendermaßen:

1. **Speichere die Eingabe w**
   (Schreibe w sowohl auf String 1 als auch auf String 2.)
2. **Simuliere die Turingmaschine M_1 bei Eingabe w**
   (Verwende hierzu String 1.)

   Simuliere *gleichzeitig* die Turingmaschine M_2 bei Eingabe w
   (Verwende hierzu String 2. „Gleichzeitig“ soll dabei bedeuten, dass in einem Schritt von M_a jeweils ein Schritt von M_1 (auf String 1) und ein Schritt von M_2 (auf String 2) simuliert wird.)

   - Falls die Simulation von M_1 *oder* die von M_2 akzeptiert, lasse M_a akzeptieren.

- Falls eine der beiden Simulationen anhält und verwirft, lasse sie auf die jeweils andere warten. Hält auch die zweite und verwirft, so soll auch M_a verwerfen.

Zur Korrektheit: Falls $w \in L_1$, dann akzeptiert die Simulation von M_1 und damit auch M_a. Entsprechend akzeptiert M_a, falls $w \in L_2$. Dies zeigt $L_1 \cup L_2 \subseteq L(M_a)$. Da für Eingaben $w \notin L_1 \cup L_2$ weder M_1 noch M_2 akzeptiert (sie lehnen ab oder halten nicht), akzeptiert auch M_a nicht. Damit gilt $L(M_a) = L_1 \cup L_2 = L_a$.

Hinweis

Die Konstruktion von M_a gelingt z.B. so:
Seien $M_1 = (Q_1, \Gamma_1, \delta_1, s_1)$ und $M_2 = (Q_2, \Gamma_2, \delta_2, s_2)$. Dann definiere

$$M_a = ((Q_1 \cup \{\text{ja}, \text{nein}, h\}) \times (Q_2 \cup \{\text{ja}, \text{nein}, h\}), \Gamma_1 \cup \Gamma_2, \delta, (s_1, s_2))$$

mit der Transitionsfunktion

$$\delta((q_1, q_2), (\gamma_1, \gamma_2)) = \begin{cases} ((q_1', q_2'), (\gamma_1', \gamma_2'), (r_1', r_2')), & q_1 \in Q_1,\ q_2 \in Q_2 \\ (\text{ja}, (\gamma_1, \gamma_2), (\downarrow, \downarrow)), & q_1 = \text{ja} \text{ oder } q_2 = \text{ja} \\ ((q_1', q_2), (\gamma_1', \gamma_2), (r_1', \downarrow)), & q_1 \in Q_1,\ q_2 \in \{\text{nein}, h\} \\ ((q_1, q_2'), (\gamma_1, \gamma_2'), (\downarrow, r_2')), & q_1 \in \{\text{nein}, h\},\ q_2 \in Q_2 \\ (\text{nein}, (\gamma_1, \gamma_2), (\downarrow, \downarrow)), & \text{sonst} \end{cases}$$

wobei $(q_1', \gamma_1', r_1') = \delta(q_1, \gamma_1)$ und $(q_2', \gamma_2', r_2') = \delta(q_2, \gamma_2)$ gelten sollen. Dass die Transitionsfunktion wohldefiniert ist, lässt sich schnell einsehen. Solange die Turingmaschinen M_1 und M_2 beim Lesen einer Eingabe nicht in einen Haltezustand übergehen, übernimmt die Turingmaschine M_a genau deren Verhalten (jeweils voneinander unabhängig auf String 1 bzw. String 2; Fall 1 in der Definition von δ). Sobald eine der beiden Turingmaschinen M_1 und M_2 in einen Haltezustand h oder nein übergeht, tut dies auch ihre Simulation und „wartet“ dort auf die jeweils andere (Fälle 3 und 4 in der Definition von δ). Geht eine der Turingmaschinen M_1 und M_2 in den Zustand ja über und akzeptiert somit die Eingabe, geschieht dies auch in der Simulation (Fall 1) und im darauffolgenden Schritt geht auch die Turingmaschine M_a in den Zustand ja über (Fall 2). Somit akzeptiert M_a alle Wörter sowohl in $L(M_1)$ als auch in $L(M_2)$. Nach Konstruktion kann sie auch keine weiteren Eingaben akzeptieren, da sie nur in den Haltezustand ja übergeht, sofern es auch M_1 oder M_2 getan hat, sodass tatsächlich $L_a = L_1 \cup L_2$ gilt.

Hinweis

Die naive Idee, M_a zunächst M_1 simulieren zu lassen und erst danach M_2, führt nicht zum Erfolg: Beispielsweise könnte es eine Eingabe w geben, auf der M_1 nicht anhält, die aber von M_2 akzeptiert. Dann würde die Turingmaschine M_a ebenfalls nicht anhalten und damit fälschlicherweise nicht akzeptieren.

b) Wenn $w \in L_1 \circ L_2$ gilt, gibt es $u_1 \in L_1$ und $u_2 \in L_2$ mit $w = u_1 u_2$. Die „Aufteilung“ von w in u_1 und u_2 ist dabei allerdings nicht bekannt. M_b muss dementsprechend alle möglichen „Aufteilungen“ ausprobieren. Die Menge dieser Aufteilungen ist allerdings endlich. Daher ist es möglich, bei festem n die Turingmaschinen M_1 und M_2 mit allen

möglichen Aufteilungen als Eingaben *zunächst auf n Schritte begrenzt* zu simulieren und nach und nach die Schrittzahl n zu erhöhen. Insgesamt arbeitet M_b folgendermaßen:

1. **Speichere w**
   (verwende dazu String 1)
2. **Setze $n := 1$ und $m := 0$**
   (Verwende hierzu jeweils separate Strings 2 und 3. Der Wert von m gibt an, an welcher Stelle das Wort w in u_1 und u_2 aufgetrennt wird. Zu Beginn ist also $u_1 = \varepsilon$ und $u_2 = w$.)
3. **Bestimme u_1 und u_2 zu gegebenem m**
   (z.B. durch Markierungen an w)
4. **Simuliere M_1 mit Eingabe u_1 maximal n Schritte lang**
   (verwende dazu String 4)
5. **Simuliere M_2 mit Eingabe u_2 maximal n Schritte lang**
   (verwende dazu String 5)
6. **Falls M_1 und M_2 akzeptiert haben, lasse M_b akzeptieren**
7. **Prüfe, ob $m = |w|$ gilt**
   - Falls $m = |w|$, setze $n := n + 1$ und $m := 0$
   - Falls $m < |w|$, setze $m := m + 1$
8. **Gehe zu Schritt 3**

Die Turingmaschine ist so konstruiert, dass sie erst m von 0 bis $|w|$ inkrementiert, bevor sie n inkrementiert. Damit ist sichergestellt, dass sie zu jeder Schrittzahl n alle Aufteilungen $w = u_1u_2$ betrachtet (sofern sie nicht vorher akzeptiert). Zu jedem festen n und jedem festen m simuliert sie in den Schritten 4 und 5 die Turingmaschinen M_1 und M_2 mit Eingabe u_1 bzw. u_2, aber nur n Schritte lang. Damit ist weiterhin sichergestellt, dass M_b nur endlich viele Schritte macht, ehe n inkrementiert wird.

Falls die entsprechenden $u_1 \in L_1$ und $u_2 \in L_2$ mit $w = u_1u_2$ existieren, wird M_b die Eingabe w akzeptieren: M_1 akzeptiert u_1 in diesem Fall nach endlich vielen Schritten n_1 und M_2 akzeptiert u_2 nach endlich vielen Schritten n_2. Folglich akzeptiert M_b mit $n = \max(n_1, n_2)$.

Falls es keine Aufteilung $w = u_1u_2$ gibt mit $u_1 \in L_1$ und $u_2 \in L_2$, so akzeptiert mindestens eine der beiden Turingmaschinen M_1 oder M_2 nicht. Da M_b nur dann akzeptiert, wenn sowohl M_1 als auch M_2 akzeptiert haben, akzeptiert M_b auch keine Eingaben w fälschlich.

Demnach gilt $L(M_b) = L_b$.

Hinweis

Diese Aufgabe zeigt, dass die semi-entscheidbaren Sprachen unter Vereinigung und Konkatenation abgeschlossen sind.

Hinweis

Nach Satz 13.3 gibt es zu jeder Mehrstring-Turingmaschine M eine 1-String-Turingmaschine M' mit $L(M') = L(M)$. Demnach lassen sich M_a und M_b auch als 1-String-Turingmaschinen konstruieren.

Aufgabe 10.3 [Entscheidbarkeit, Semi-Entscheidbarkeit, Unentscheidbarkeit] **7 Punkte**

Geben Sie für die folgenden Probleme an, ob sie jeweils

- entscheidbar,
- nicht entscheidbar aber semi-entscheidbar oder
- unentscheidbar

sind. Beweisen Sie Ihre Behauptungen. Gehen Sie davon aus, dass jede Turingmaschine die Zeichen 0 und 1 im Eingabe- und im Arbeitsalphabet enthält.

a) *Problem:* A
Gegeben: Turingmaschine M und $k \in \mathbb{N}_0$
Frage: Erzeugt M bei der Eingabe 0^k die Ausgabe 1? **(3 Punkte)**

b) *Problem:* B
Gegeben: Turingmaschine M
Frage: Erzeugt M bei *keiner* Eingabe die Ausgabe 1? **(4 Punkte)**

Hinweis

Laut den Konventionen im Abschnitt „Universelle Turingmaschinen“ von Kapitel 16 ist die Kodierung der Zeichen 0 und 1 eindeutig gegeben durch $\text{num}(0) = 3$ und $\text{num}(1) = 4$.

Lösung:

a) Problem A ist nicht entscheidbar aber semi-entscheidbar.

Zur Semi-Entscheidbarkeit:
Wir geben einen „Semi-Entscheidungsalgorithmus“ für A an, der für alle „Ja-Eingaben“ anhält und akzeptiert und für alle anderen Eingaben ablehnt oder nicht anhält. Dazu simulieren wir eine beliebige Turingmaschine M bei Eingabe von 0^k durch eine universelle Turingmaschine U:

1. **Initialisiere U zur Simulation von M bei Eingabe 0^k**
2. **Simuliere M bei Eingabe 0^k durch U**
   - Falls (Simulation von) M in Haltekonfiguration ist, prüfe ob die Ausgabe von M gleich 1 ist
     - Falls ja, akzeptiere
     - Falls nein, lehne ab

Für jedes Tupel $(M, k) \in A$ hält die Turingmaschine M nach Definition bei Eingabe 0^k

nach einer gewissen Anzahl $n \in \mathbb{N}_0$ von Schritten. In diesem Fall endet auch der angegebene Algorithmus nach n Simulationsschritten und akzeptiert. Für Turingmaschinen mit einer Ausgabe ungleich 1 lehnt U korrekterweise ab, da dann $(M, k) \notin A$ gilt. Für alle anderen Turingmaschinen, die bei Eingabe 0^k eine unendliche Berechnung besitzen, akzeptiert oder verwirft der Algorithmus niemals. Dies zeigt die Semi-Entscheidbarkeit.

Da es Fälle gibt, in denen der Semi-Entscheidungsalgorithmus niemals akzeptiert oder verwirft, zeigt er *nicht* die Entscheidbarkeit von A. Tatsächlich ist A sogar unentscheidbar.

Zur Unentscheidbarkeit:
Diese folgt aus der Reduzierbarkeit vom unentscheidbaren Problem TM-E-HALT auf A, d.h. wir zeigen TM-E-HALT $\leq A$. Dazu geben wir eine Reduktionsfunktion f an, die eine Turingmaschine M' auf ein zu Problem A konformes Tupel (M, k) aus Turingmaschine M und natürlicher Zahl $k \in \mathbb{N}_0$ abbildet und die Reduktionseigenschaft

$$M' \in \text{TM-E-HALT} \Leftrightarrow f(M') = (M_{M'}, k) \in A$$

erfüllt.

Konstruktion der Reduktionsfunktion
Die Reduktionsfunktion f sei derart, dass sie eine Turingmaschine M' auf $(M_{M'}, 0)$ abbildet, wobei $M_{M'}$ eine Turingmaschine ist, welche

- M' bei Eingabe von ε $(= 0^0)$ simuliert
- und, wenn M' anhält, 1 ausgibt.

Offenbar ist die geschilderte Funktion total und berechenbar (die Kodierung der Transitionsfunktion muss entsprechend erweitert werden).

Zeige: $M' \in \text{TM-E-HALT} \Rightarrow f(M') = (M_{M'}, 0) \in A$
Es sei $M' \in$ TM-E-HALT, d.h. M' hält bei Eingabe ε. Daraus folgt, dass $M_{M'}$ bei Eingabe $0^0 = \varepsilon$ ebenfalls anhält und die Ausgabe 1 erzeugt. Folglich gilt $(M_{M'}, 0) \in A$.

Zeige per Kontraposition: $M' \in \text{TM-E-HALT} \Leftarrow f(M') = (M_{M'}, 0) \in A$
Es gelte $M' \notin$ TM-E-HALT, d.h. M' hält bei Eingabe von ε nicht. Folglich hält $M_{M'}$ ebenfalls nicht. Daraus folgt $f(M') = (M_{M'}, 0) \notin A$.

b) Problem B ist weder entscheidbar noch semi-entscheidbar.

Wir zeigen zunächst, dass B unentscheidbar ist, das Komplementproblem $\overline{B}$ jedoch semi-entscheidbar ist. Daraus schlussfolgern wir mit Lemma 16.2, dass B nicht semi-entscheidbar sein kann.

Zur Unentscheidbarkeit:
Wir zeigen mit Hilfe des Satzes von Rice (Satz 15.1), dass das Problem B nicht entscheidbar ist. Dazu beobachten wir, dass B von der Form

Gegeben: Turingmaschine M
Frage: Ist $f_M \in \{f \in \mathcal{R} \mid \text{für alle } x \in \Sigma^* \text{ gilt } f(x) \neq 1\}$?

ist und damit für $S = \{f \in \mathcal{R} \mid \text{für alle } x \in \Sigma^* \text{ gilt } f(x) \neq 1\}$, was offenbar eine Menge berechenbarer partieller Funktionen ist, $B = \text{TM-FUNC}(S)$ gilt.

Offensichtlich gilt weiter $S \neq \emptyset$, da etwa die berechenbare Funktion f_ε, die jede Eingabe auf das leere Wort abbildet, in S enthalten ist. Andererseits gilt auch $S \neq \mathcal{R}$, da etwa die

berechenbare Funktion f_1, die jede Eingabe auf das Wort 1 abbildet, nicht in S enthalten ist. Mit dem Satz von Rice folgt daher die Unentscheidbarkeit von B.

Zur Nicht-Semi-Entscheidbarkeit:
Wir zeigen, dass das Komplementproblem $\overline{B}$ semi-entscheidbar ist. Wäre dann auch B semi-entscheidbar, so würde mit Lemma 16.2 folgen, dass B sogar entscheidbar ist. Von B wissen wir aber schon, dass es unentscheidbar ist. Es folgt schließlich die Nicht-Semi-Entscheidbarkeit von B.

Das Komplementproblem $\overline{B}$ lautet:

Gegeben: Turingmaschine M
Frage: Erzeugt M bei *irgendeiner* Eingabe die Ausgabe 1?

Folgender „Semi-Entscheidungsalgorithmus“ zeigt, dass das Komplementproblem $\overline{B}$ semi-entscheidbar ist:

1. **Setze $n := 0$**
2. **Für jedes $w \in \Gamma^*$ mit $|w| \leq n$:** (hiervon gibt es endlich viele)
   a) **Initialisiere universelle Turingmaschine U zur Simulation von M bei Eingabe w**
   b) **Simuliere M bei Eingabe von w für n Schritte**
      - Falls (Simulation von) M in Haltekonfiguration ist, prüfe ob die Ausgabe von M gleich 1 ist
        – Falls ja, akzeptiere
3. **Setze $n := n + 1$ und gehe zu Schritt 2**

Gibt es eine Eingabe, die bei der simulierten Turingmaschine zum Halten mit Ausgabe 1 führt, so wird sie auf diese Weise gefunden und U akzeptiert. Andernfalls hält U nicht an. Damit ist $\overline{B}$ semi-entscheidbar.

Aufgabe 10.4 [Primitiv Rekursive Funktionen] **2 Punkte**

Zeigen Sie: Die Funktion $\text{even}(x) : \mathbb{N}_0 \to \mathbb{N}_0$ mit

$$\text{even}(x) = \begin{cases} 1, & \text{x ist gerade} \\ 0, & \text{x ist ungerade} \end{cases}$$

ist primitiv rekursiv.

Lösung:

Wir beobachten, dass $\text{even}(x)$ alternierend die Werte 1 und 0 annimmt. Für die aus der Vorlesung bekannte primitiv rekursive Funktion $\tau : \mathbb{N}_0 \to \mathbb{N}_0$ mit $\tau(0) = 1$ und $\tau(x + 1) = 0$ gilt insbesondere $\tau(0) = 1$ und $\tau(1) = 0$. Damit kann $\text{even}(x)$ primitiv rekursiv definiert werden durch: $\text{even}(0) = 1$ (da konstante Funktionen primitiv rekursiv sind) und

$$\text{even}(x + 1) = h(\text{even}(x), x)$$

wobei die Hilfsfunktion $h : \mathbb{N}_0^2 \to \mathbb{N}_0$ gegeben ist durch

$$h(x_1, x_2) = \tau(\pi_1^2(x_1, x_2))$$

und als Komposition der primitiv rekursiven Funktionen τ und π_1^2 (Projektion auf die erste Komponente) wieder primitiv rekursiv ist. In Kurzschreibweise gilt also

$$\text{even}(x + 1) = \tau(\pi_1^2(\text{even}(x), x)).$$

Hinweis

Die Projektion π_1^2 ist hier von Nöten um die Definition der primitiven Rekursion einzuhalten. Die Definition von even über $\text{even}(0) = 1$ und $\text{even}(x+1) = \tau(\text{even}(x))$ würde zwar ebenfalls korrekt sein, aber nicht zeigen, dass even primitiv rekursiv ist.