Technische Universität München
Fakultät für Informatik
Prof. Tobias Nipkow, Ph.D.
Sascha Böhme, Lars Noschinski

Sommersemester 2011
Lösungsblatt 7
18. Juli 2011

Einführung in die Theoretische Informatik

Hinweis: *Bitte beachten Sie unbedingt die Hinweise zum Übungsablauf und zu den Aufgabentypen auf der THEO-Website (`http://theo.in.tum.de/`).*

Hausaufgabe 1 (8 Punkte)

Sei $\Sigma = \{0, 1\}$. Wie in Übung 5 bezeichnen wir mit $\overline{w}$ die Negation eines Wortes $w \in \{0, 1\}^*$, d.h. alle Nullen werden durch Einsen ersetzt und umgekehrt.

1. Geben Sie eine deterministische Turingmaschine an, die für ein Eingabewort $w \in \Sigma^*$ folgende Berechnung durchführt: Am Ende der Berechnung steht auf dem Band das Wort $w\overline{w}$ und der Kopf steht in einem Endzustand auf dem ersten Zeichen dieses Wortes.

   Kommentieren Sie ihre Konstruktion durch eine informelle Beschreibung Ihrer Lösungsidee.

2. Geben Sie nun eine Turingmaschine an, die die Sprache $L = \{w\overline{w} \mid w \in \Sigma^*\}$ akzeptiert. Kommentieren Sie wiederum ihre Konstruktion durch eine informelle Beschreibung Ihrer Lösungsidee.

Lösungsvorschlag

1. Wir formulieren die Turingmaschine so, dass sie auf auf dem am weitesten links liegenden Zeichen der Eingabe startet. Das Bandalphabet ist $\Gamma = \{0, 1, \widehat{0}, \widehat{1}\}$, wobei wir $\widehat{0}$ und $\widehat{1}$ nutzen um die Zeichen zu markieren, die nicht mehr kopiert werden müssen.

   Wir gehen wie folgt vor:

   (a) Markiere das Zeichen unter dem Cursor als kopiert; dann bewege den Kopf an das rechte Ende des Bandes und füge dort die Negation des Zeichens markiert ein.

   (b) Suche das am weitesten links stehende, nicht markierte Zeichen.

   (c) Gibt es ein solches Zeichen, springe zum ersten Schritt. Ansonsten steht jetzt $w\overline{w}$ auf dem Band, wobei jedes Zeichen markiert ist. Entferne die Markierungen.

Formal definieren wie die Turingmaschine als $M = (Q, \{0,1\}, \Gamma, \delta, q_0, \square, \{q_5\})$, wobei $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ und δ wie folgt definiert ist:

	0	1	$\widehat{0}$	$\widehat{1}$	$\square$
q_0	$(q_1, \widehat{0}, R)$	$(q_2, \widehat{1}, R)$	$(q_0, \widehat{0}, R)$	$(q_0, \widehat{1}, R)$	$(q_4, \square, L)$
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$	$(q_1, \widehat{0}, R)$	$(q_1, \widehat{1}, R)$	$(q_3, \widehat{1}, L)$
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$	$(q_2, \widehat{0}, R)$	$(q_2, \widehat{1}, R)$	$(q_3, \widehat{0}, L)$
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_3, \widehat{0}, L)$	$(q_3, \widehat{1}, L)$	$(q_0, \square, R)$
q_4	$(q_4, 0, L)$	$(q_4, 1, L)$	$(q_4, 0, L)$	$(q_4, 1, L)$	$(q_5, \square, N)$

2. Wir konstruieren eine Turingmaschine, die eine Eingabe genau dann akzeptiert, wenn diese die Form $w\overline{w}$ hat. Diesmal verwenden wir zwei unterschiedliche Markierungen, $\widehat{\ }$ und $\widetilde{\ }$. Die erste verwenden wir, um bereits verarbeitete Zeichen aus w zu markieren, die zweite für bereits verarbeitete Zeichen aus $\overline{w}$. Daher ist $\Sigma = \{0,1\}$ und $\Gamma = \Sigma \cup \{\widehat{0}, \widehat{1}, \widetilde{0}, \widetilde{1}, \square\}$. Wir gehen wiederum davon aus, dass der Kopf anfangs links auf der Eingabe steht.

   Zunächst eine informelle Beschreibung:

   (a) Markiere das erste Zeichen mit $\widehat{\ }$ und rate nicht-deterministisch die Position des ersten Zeichens aus $\overline{w}$. Markiere dieses Zeichen mit $\widetilde{\ }$.

   (b) Bewege den Kopf auf das am weitesten links stehende, unmarkierte Zeichen und markiere es mit $\widehat{\ }$. Gibt es kein solches Zeichen, springe zu Schritt (d).
   Ansonsten vergleiche es mit dem ersten unmarkierten Zeichen nach einer $\widetilde{\ }$-Markierung. Sind die beiden Zeichen gleich, markiere das zweite mit $\widetilde{\ }$ und wiederhole diesen Schritt. Andernfalls wird das Wort nicht akzeptiert.

   (c) Wiederhole Schritt (b)

   (d) Überprüfe, ob es noch ein unmarkiertes Zeichen gibt. Wenn nein, akzeptiere die Eingabe.

   Formal definieren wie die Turingmaschine als $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, \{q_f\})$, wobei $Q = \{q_0, \ldots, q_9, q_f\}$ und die partielle Übergangsfunktion δ wie folgt definiert ist.

	0	1	$\widehat{0}$	$\widehat{1}$	$\widetilde{0}$	$\widetilde{1}$	$\square$
q_0	$(q_1, \widehat{0}, R)$	$(q_2, \widehat{1}, R)$					$(q_f, \square, N)$
q_1	$(q_1, 0, R)$	$(q_1, 1, R)$					
		$(q_3, \widetilde{1}, L)$					
q_2	$(q_2, 0, R)$	$(q_2, 1, R)$					
	$(q_3, \widetilde{0}, L)$						
q_3	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_3, \widehat{0}, L)$	$(q_3, \widehat{1}, L)$	$(q_3, \widetilde{0}, L)$	$(q_3, \widetilde{1}, L)$	$(q_4, \square, R)$
q_4	$(q_5, \widehat{0}, R)$	$(q_7, \widehat{1}, R)$	$(q_4, \widehat{0}, R)$	$(q_4, \widehat{1}, R)$	$(q_9, \widehat{0}, R)$	$(q_9, \widehat{1}, R)$	
q_5	$(q_5, 0, R)$	$(q_5, 1, R)$			$(q_6, \widetilde{0}, R)$	$(q_6, \widetilde{1}, R)$	
q_6		$(q_3, \widetilde{1}, L)$			$(q_6, \widetilde{0}, R)$	$(q_6, \widetilde{1}, R)$	
q_7	$(q_7, 0, R)$	$(q_7, 1, R)$			$(q_8, \widetilde{0}, R)$	$(q_8, \widetilde{1}, R)$	
q_8	$(q_3, \widetilde{0}, L)$				$(q_8, \widetilde{0}, R)$	$(q_8, \widetilde{1}, R)$	
q_9					$(q_9, \widetilde{0}, R)$	$(q_9, \widetilde{1}, R)$	$(q_f, \square, N)$

Die Bedeutungen der Zustände sind dabei wie folgt:

- q_1 bzw. q_2: Rate die Anfangsposition von $\overline{w}$, wenn das erste Zeichen von w 0 bzw. 1 ist.
- q_3: Bewege den Kopf ans linke Ende des beschriebenen Bandes.
- q_4: Finde das erste nicht-markierte Zeichen (in w).
- q_5, q_6 bzw. q_7, q_8: Vergleiche 0 bzw. 1 mit dem ersten, nicht-markierten Zeichen in $\overline{w}$.
- q_9: Stelle sicher, dass alle Zeichen markiert sind.

Hausaufgabe 2 (6 Punkte)

1. Zeigen Sie, dass das if-then-else-Konstrukt in `LOOP`-Programmen wirklich nur eine syntaktische Abkürzung ist. Geben Sie dazu zu `IF` $X = 0$ `THEN` P `ELSE` Q `END` ein `LOOP`-Programm an, dass nur aus den vier primitiven Operationen besteht.

2. Zeigen Sie jetzt, dass statt einer atomaren Bedingung $X = 0$, wobei X eine Variable ist, auch beliebige Verknüpfungen solcher atomaren Bedingungen mit $\neg$ (Negation), $\wedge$ (Konjunktion) und $\vee$ (Disjunktion) verwendet werden können.

   Geben Sie dazu eine rekursive Funktion f an, die `IF` B `THEN` P `ELSE` Q `END` mit einer solchen Bedingung B als Eingabe nimmt und es in ein `LOOP`-Programm mit der gleichen Semantik umformt. In der Ausgabe dürfen $\neg$, $\wedge$ und $\vee$ nicht mehr vorkommen (wohl aber if-then-else).

   Beispiel: Die Transformationsregel für Negation lässt sich wie folgt definieren.

   $$f(\texttt{IF } \neg B \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) = f(\texttt{IF } B \texttt{ THEN } Q \texttt{ ELSE } P \texttt{ END})$$

3. Wir erweitern die Bedingungen aus dem ersten Teil, so dass zusätzlich Tests der Form $E = 0$ erlaubt sind, wobei E ein beliebiger arithmetischer Ausdruck aus Variablen, natürlichen Zahlen, $+$ und $*$ ist.

   Erweitern Sie die Funktion f, so dass sie auch solche Bedingungen auf elementare `LOOP`-Programme abbildet, d.h., arithmetische Ausdrücke kommen in der Ausgabe nicht mehr vor.

Lösungsvorschlag

1. Seien Y_1 und Y_2 zwei nicht anderweitig im Programm verwendete Variablen. Das if-then-else-Konstrukt lässt sich dann wie folgt ausdrücken:

   $$\begin{aligned}
   &Y_0 \texttt{ := } Y_0 + 1\texttt{; LOOP } X \texttt{ DO } Y_0 \texttt{ := } Y_0 - 1 \texttt{ END;}\\
   &Y_1 \texttt{ := } Y_1 + 1\texttt{; LOOP } Y_0 \texttt{ DO } Y_1 \texttt{ := } Y_1 - 1 \texttt{ END;}\\
   &\texttt{LOOP } Y_0 \texttt{ DO } P \texttt{ END;}\\
   &\texttt{LOOP } Y_1 \texttt{ DO } Q \texttt{ END;}
   \end{aligned}$$

   Dabei berechnen die ersten zwei Zeilen $Y_0 = 1 - X$ und $Y_1 = 1 - Y_0$. Daher ist $Y_0 = 1$ genau dann, wenn $Y_1 = 0$ und $Y_1 = 1$ genau dann, wenn $Y_0 = 0$, also $X \neq 0$ ist.

2. Wir definieren f wie folgt:

$$\begin{aligned} f(\texttt{IF } X = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= \texttt{IF } X = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END} \\ f(\texttt{IF } \neg B \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= f(\texttt{IF } B \texttt{ THEN } Q \texttt{ ELSE } P \texttt{ END}) \\ f(\texttt{IF } B_1 \wedge B_2 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= f(\texttt{IF } B_1 \texttt{ THEN } R \texttt{ ELSE } Q \texttt{ END}) \\ &\quad \text{mit } R = f(\texttt{IF } B_2 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) \\ f(\texttt{IF } B_1 \vee B_2 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= f(\texttt{IF } B_1 \texttt{ THEN } P \texttt{ ELSE } R \texttt{ END}) \\ &\quad \text{mit } R = f(\texttt{IF } B_2 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) \end{aligned}$$

3. Wir erweitern f um die folgenden Gleichungen für Ausdrücke, wobei n eine beliebige natürliche Zahl ist:

$$f(\texttt{IF } n = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) = \begin{cases} P & \text{falls } n \text{ gleich 0 ist} \\ Q & \text{sonst} \end{cases}$$

$$\begin{aligned} f(\texttt{IF } E_1 + E_2 = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= f(\texttt{IF } E_1 = 0 \wedge E_2 = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) \\ f(\texttt{IF } E_1 * E_2 = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) &= f(\texttt{IF } E_1 = 0 \vee E_2 = 0 \texttt{ THEN } P \texttt{ ELSE } Q \texttt{ END}) \end{aligned}$$

Hausaufgabe 3 (6 Punkte)

Wir bezeichnen mit TM_k solche Einband-Turingmaschinen, die jede Zelle des Bandes höchstens k-mal ändern dürfen. Dabei gelten nur Übergänge $\delta(q, x) = (q', y, X)$ mit $x \neq y$ als Änderungen einer Zelle des Bandes (mit $X \in \{N, R, L\}$).

1. Zeigen Sie, dass die Turingmaschinen TM_2 äquivalent zu herkömmlichen Turingmaschinen sind. Benutzen Sie soviel Band wie nötig.

2. Zeigen Sie, dass auch die Turingmaschinen TM_1 äquivalent zu herkömmlichen Turingmaschinen sind. Sie dürfen dabei die Resultate der ersten Teilaufgabe verwenden.

Sie müssen keine expliziten Konstruktionen angeben. Es genügen informelle, aber dennoch vollständige und genaue Beschreibungen.

Lösungsvorschlag

1. Sei A eine herkömmliche Turingmaschine und A' eine TM_2.

   Allgemein gilt die folgende Beobachtung: Zu jedem Zeitpunkt steht nur endlich viel Information auf dem Band einer Turingmaschine, und der Rest des Bandes ist mit Leerzeichen gefüllt. Wir können daher den informationstragenden Teil des Bandes durch je eine eindeutige Markierung umranden.

   Wir simulieren nun einen Schritt von A in A' wie folgt. Zunächst kopieren wir den gesamten Bandinhalt zwischen den Randmarkierungen nach rechts. Kopieren erfolgt zeichenweise, und jedes kopierte Zeichen wird markiert. Daher wird jede Zelle zweimal geändert, beim ersten Schreiben (als Ziel einer Kopie) und beim Markieren (als Quelle einer Kopie). Außerdem markieren wir uns die Position des Lese-/Schreibkopfes von A auf dem Band von A'. Beim Kopieren der Zellen, die direkt benachbart zur gespeicherten Position des Lese-/Schreibkopfes liegen, werden nur die Ergebnisse entsprechend der aktuell auszuführenden Transition von A geschrieben.

2. Analog zur Simulation einer Turingmaschine durch eine TM_2 erfolgt auch die Simulation einer Turingmaschine A durch eine TM_1 A', allerdings mit dem Unterschied, dass wir nun jede Bandzelle von A durch zwei Zellen in A' simulieren. In der ersten Zelle steht das Zeichen von A (oder die gespeicherte Position des Lese-/Schreibkopfes), und die zweite Zelle wird zur Markierung beim Kopieren benutzt, bleibt aber anfangs unbeschrieben (und deshalb darf man annehmen, dass dort das Leerzeichen steht). Da die Eingabe nicht dem Zwei-Zellen-Format entspricht, muss sie zunächst einmal in dieses Format kopiert werden, wobei die ursprünglichen Eingabezeichen zur Markierung einmal überschrieben werden.

Quiz 1

Beantworten Sie kurz die folgenden Fragen:

1. Gibt es eine Turingmaschine, die sich nie mehr als vier Schritte vom Startzustand entfernt und eine unendliche Sprache akzeptiert?

2. Welche Sprachen lassen sich mit Turingmaschinen, die ihren Kopf immer nur nach rechts bewegen, erkennen?

3. Vergleichen Sie die beiden folgenden `WHILE`-Programme: `WHILE` $b_0 \wedge b_1$ `DO` P `END` und `WHILE` b_0 `DO` P `END; WHILE` $b_0 \wedge b_1$ `DO` P `END`. Zeigen diese Programme das gleiche Verhalten?

Lösungsvorschlag

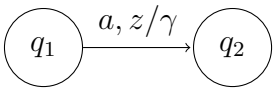
1. Ja, die Turingmaschine kann einfach unabhängig von der Eingabe direkt in einen akzeptierenden Zustand wechseln.

2. Die regulären Sprachen. Da der Kopf sich nur nach rechts bewegt, liest eine solche Turingmaschine nur einmal die Eingabe, gegebenenfalls gefolgt von (unendlich) vielen Blanks. Nachdem die Eingabe gelesen wurde, hängt es nur noch von dem aktuellen Zustand der TM ab, ob irgendwann ein Endzustand erreicht werden wird, nicht mehr vom Band.

3. Nein. Ist b_0 immer wahr, aber b_1 immer falsch, so terminiert das erste Programm direkt, das zweite aber nie.

Tutoraufgabe 1

Ein *Queue-Automat* (kurz: QA) ist ein Tupel $A = (Q, \Sigma, \Gamma, q_0, z_0, \delta)$. Dabei sind Q (Zustandsmenge), Σ (Eingabealphabet) und Γ (Queuealphabet) endliche Mengen ähnlich wie bei einer Turingmaschine, $q_0 \in Q$ ist der Startzustand, und $z_0 \in \Gamma$ beschreibt die initiale Queue. Die Übergangsfunktion δ hat den Typ $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$. Graphisch kann eine Transition $(q_2, \gamma) \in \delta(q_1, a, z)$ eines Queue-Automaten wie folgt dargestellt werden:



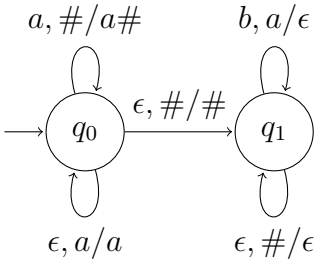
Eine *Konfiguration* eines Queue-Automaten ist ein Tripel $(q, w, \gamma) \in Q \times \Sigma^* \times \Gamma^*$. Dabei ist q der aktuelle Zustand, w das noch zu lesende Wort und γ der aktuelle Inhalt der Queue. Daraus ergibt sich die Übergangsrelation $\to_A$ zwischen Konfigurationen:

$$(q, bw, z\gamma) \to_A (q', w, \gamma\gamma') \,, \qquad \text{wenn} \qquad (q', \gamma') \in \delta(q, b, z) \,.$$

Die (mit leerer Queue) akzeptierte Sprache eines Queue-Automaten ist

$$L_\epsilon(A) = \{w \in \Sigma^* \mid \exists q' \in Q.\ (q_0, w, z_0) \to_A^* (q', \epsilon, \epsilon)\} \,.$$

1. Der Queue-Automat $A_1 = (\{q_0, q_1\}, \{a, b\}, \{a, b, \#\}, q_0, \#, \delta)$ sei wie folgt graphisch dargestellt:



   Geben Sie eine *akzeptierende* Konfigurationsfolge für das Eingabewort $aabb$ an.

2. Geben Sie einen Queue-Automaten A_2 an mit $L(A_2) = \{ww^R \mid w \in \{a, b\}^*\}$.

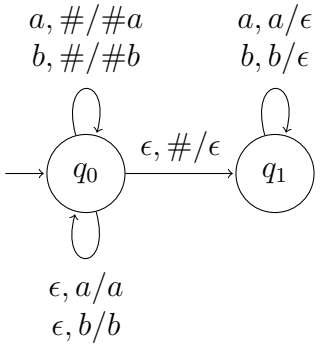
3. Zeigen Sie, dass jede (deterministische) Turingmaschine durch einen Queue-Automaten simuliert werden kann.

Lösungsvorschlag

1. Die einzig mögliche akzeptierende Folge ist:

$$(q_0, aabb, \#) \to_{A_1} (q_0, abb, a\#) \to_{A_1} (q_0, abb, \#a) \to_{A_1} (q_0, bb, aa\#) \to_{A_1}$$
$$(q_0, bb, a\#a) \to_{A_1} (q_0, bb, \#aa) \to_{A_1} (q_1, bb, aa\#) \to_{A_1} (q_1, b, a\#) \to_{A_1}$$
$$(q_1, \epsilon, \#) \to_{A_1} (q_1, \epsilon, \epsilon)$$

2. Der Queue-Automat $A_2 = (\{q_0, q_1\}, \{a, b\}, \{a, b, \#\}, q_0, \#, \delta)$ kann wie folgt graphisch dargestellt werden:



3. Sei $M = (Q, \Sigma, \Gamma, q_0, \delta, \square, F)$ die Turingmaschine, die wir durch einen Queue-Automaten A simulieren wollen. Wir nehmen an, dass $Q \cap \Gamma = \emptyset$.

   Die Konstruktionsidee ist nun, dass wir eine Konfiguration (v, q, w) der Turingmaschine als Wort $vqw\#$ in der Queue ablegen. Der QA kann die Queue rotieren, indem er ein Queue-Zeichen liest, und danach wieder ans Ende der Queue anhängt. So lässt sich die Queue flexibel manipulieren. Für jeden Schritt der Turingmaschine muss allerdings die Queue einmal komplett durchlaufen werden.

Unser Queue-Automat A hat die Zustandsmenge

$$Q' = \{\mathit{ff}, \mathit{start}\} \cup \{\mathit{find}_x \mid x \in \Gamma\} \cup \{\mathit{step}_x^q \mid x \in \Gamma, q \in Q\}$$

und die folgenden schematischen Übergänge, parametrisiert für alle $x, y \in \Gamma$ und $q \in Q$:

$$\begin{aligned}
\delta'(\mathit{start}, \epsilon, x) &\to (\mathit{find}_x, \epsilon) && \text{(zum Kopf vorlaufen, dabei}\\
\delta'(\mathit{find}_x, \epsilon, y) &\to (\mathit{find}_y, x) && \text{jeweils letztes Zeichen merken)}\\
\delta'(\mathit{find}_x, \epsilon, q) &\to (\mathit{step}_x^q, \epsilon) && \text{(zusätzlich den Zustand merken)}\\
\delta'(\mathit{step}_x^q, \epsilon, y) &\to \begin{cases} (\mathit{ff}, xq'z) & \text{falls } \delta(q,y) = (q', z, N)\\ (\mathit{ff}, xzq') & \text{falls } \delta(q,y) = (q', z, R)\\ (\mathit{ff}, q'xz) & \text{falls } \delta(q,y) = (q', z, L) \end{cases} && \text{(Schritt)}\\
\delta'(\mathit{ff}, \epsilon, x) &\to (\mathit{ff}, x) && \text{(Vorspulen bis zum \#)}\\
\delta'(\mathit{ff}, \epsilon, \#) &\to (\mathit{start}, \#) && \text{(von vorn beginnen)}\\
\delta'(\mathit{start}, \epsilon, q) &\to (\mathit{step}_\square^q, \epsilon) && \text{(Sonderfälle für den Rand)}\\
\delta'(\mathit{step}_x^q, \epsilon, \#) &\to (\mathit{start}, xq\square\#)
\end{aligned}$$

Nun gilt (hier ohne Beweis):

$$(\mathit{start}, \epsilon, vqw\#) \to_A^* (\mathit{start}, \epsilon, v'q'w'\#) \iff (v, q, w) \to_M^* (v', q', w')$$

Um dafür zu sorgen dass auch wirklich $L(A) = L(M)$ gilt, muss man den Queue-Automaten noch so erweitern, dass er am Anfang das Eingabewort in die Queue schreibt, und bei Erreichen eines Endzustands der TM die Queue komplett entleert.

Tutoraufgabe 2

Wir wollen untersuchen, ob sich (ähnlich wie bei den WHILE-Programmen) auch jedes LOOP-Programm in eine „Normalform“

$$\texttt{LOOP } X \texttt{ DO } P \texttt{ END}$$

bringen lässt, so dass P keine Schleifen mehr enthält.
Wir bezeichnen mit $V(P)$ die Menge der Variablen, die in P vorkommen (diese Menge ist stets endlich). Für einen gegebenen Programmlauf bezeichnen wir mit $[x]$ den Wert der Variablen x beim Start des Programms, und mit $[x]'$ den Wert der Variablen nach Programmende.

1. Zeigen Sie: Für jedes LOOP-Programm P *ohne Schleifen* gibt es eine Konstante k, so dass gilt:
$$\max_{x \in V(P)} [x]' \leq \max_{x \in V(P)} [x] + k \; .$$

2. Zeigen Sie, dass es kein LOOP-Programm in Normalform geben kann, welches die Quadratfunktion $n \mapsto n^2$ berechnet.

Lösungsvorschlag

1. Die Beziehung zeigen wir mit Induktion über den Aufbau von P. Da wir annehmen, dass P keine Schleifen hat, müssen wir den LOOP-Fall nicht betrachten:

   - Für $P = \ x_i := x_j + c$ gilt offensichtlich $[x_i]' \leq [x_j] + c$ und, da alle anderen Variablen ihren Wert behalten, gilt auch $\max_{x \in V(P)}[x]' \leq \max_{x \in V(P)}[x] + c$. Wir wählen also $k = c$.
   - Für $P = \ x_i := x_j - c$ können wir $k = 0$ wählen, da keine Variable erhöht wird, und somit auch das Maximum nicht wachsen kann.
   - Für $P = \ P_1; P_2$ können wir per Induktionshypothese Konstanten k_1, k_2 annehmen, so dass die Variableninhalte im ersten Teilprogramm maximal um k_1 wachsen, und im zweiten Teil maximal um k_2. Somit kann man $k = k_1 + k_2$ setzen.

2. Wir nehmen an, Q sei ein LOOP-Programm in Normalform, welches die Quadratfunktion berechnet. Dann ist

   $$Q = \texttt{LOOP}\ x_0\ \texttt{DO}\ P\ \texttt{END}$$

   und P enthält keine Schleife. Wir können hierbei annehmen, dass die einzige Eingabevariable x_0 gleichzeitig die Schleifenvariable ist, denn ansonsten würde die Schleife gar nicht ausgeführt werden (da alle anderen Variablen anfangs auf Null gesetzt sind), und Q könnte sicher nicht quadrieren.

   Nach Aufgabenteil 1 gibt es ein k, so dass nach jeder Ausführung von P gilt, dass $\max_{v \in V}[v]' \leq \max_{v \in V}[v] + k$. Da P insgesamt $[x_0]$ mal durchlaufen wird, gilt für das Gesamtprogramm $\max_{v \in V}[v]' \leq \max_{v \in V}[v] + [x_0]k$. Da bei Programmbeginn alle Variablen außer x_0 auf Null gesetzt sind, haben wir $\max_{v \in V}[v] = [x_0]$ und somit $\max_{v \in V}[v]' \leq [x_0](k+1)$.

   Ruft man nun Q mit der Eingabe $k + 2$ auf, dann gilt:

   $$[x_0]' \leq \max_{v \in V}[v]' \leq (k+2)(k+1) < (k+2)^2$$

   Somit hat Q die Eingabe nicht quadriert, im Widerspruch zur Annahme.