Prof. Martin Hofmann
Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik

Wintersemester 2002/03
30. Januar 2003

Lösungsvorschläge zur 11. Übung

Komplexitätstheorie

Hinweis: Das Übungsblatt wurde nachträglich noch leicht abgeändert, diese Lösungsvorschläge beziehen sich auf diese neue Version.

Lösung zu G37:

a)
$$\begin{aligned}\mathtt{f}(l) := \mathsf{match}\ l\ \mathsf{with}\ &|\ \mathsf{nil} \Rightarrow \mathsf{nil}\\ &|\ \mathsf{cons}((,d,,)h,t) \Rightarrow \mathsf{if}\ h\ \ \mathsf{then}\ \mathsf{cons}\big(d,(1,1,1),\mathtt{f}(t)\big)\\ &\qquad\qquad\qquad\qquad\quad \mathsf{else}\ \ \mathsf{cons}\big(d,(0,0,0),\mathtt{f}(t)\big)\end{aligned}$$

Durch die in LFPL gemachte Annahme, das eine einzelne Speicherzelle ($\Diamond$) immer groß genug ist, um die größte vorkommende nicht-rekursive Datenstruktur vollständig aufzunehmen, können wir Elememte flacher Typen wie bool oder bool $\otimes$ (bool $\otimes$ bool) mit "grossen if-Abfragen" kostenlos trotz linearer typsisierung kopieren.

b) Wie in a) angedeutet, kann man sich zusätzlichen Platz durch Wechsel in ein gößeres Alphabet verschaffen, ähnlich wie die Simulationen mehrerer Bänder durch mehrere Spuren auf einem Band (siehe G4). Da wir nur Akzeptor Maschinen betrachten, brauchen wir diese Kodierung hinterher auch nicht beseitigen.

Das dieses Kodieren möglich ist liegt wie in a) erklärt an der einheitlichen Zellengröße, d.h. wir können ein n-Tupel immer durch ein $n{+}1$-Tupel ersetzten. Dies bedeutet jedoch immer nur einen linearen Zuwachs an Speicherplatz.

c) *Machbar. Ein ausgearbeitete Implementierung wird belohnt!*

d) Wer rekursive Datenstrukturen wie Listen kopieren möchte, benötigt zusätzlichen Platz im Argument; deshalb hat die Eingabeliste den "aufgeblähten" Typ list(bool $\otimes$ $\Diamond$).

$$\begin{aligned}\mathtt{clone}(l)\ \ = \mathsf{match}\ l\ \mathsf{with}\ &|\ \mathsf{nil} \Rightarrow \mathsf{nil} \otimes \mathsf{nil}\\ &|\ \mathsf{cons}(d_1,h,t) \Rightarrow\\ &\quad \mathsf{match}\ h\ \mathsf{with}\ (w,d_2) \Rightarrow\\ &\quad \mathsf{match}\ \mathtt{clone}(t)\ \mathsf{with}\ (t_1,t_2) \Rightarrow\\ &\quad \mathsf{if}\ w\ \mathsf{then}\ \mathsf{cons}(d_1,1,t_1) \otimes \mathsf{cons}(d_2,1,t_2)\\ &\qquad\quad \mathsf{else}\ \ \mathsf{cons}(d_1,0,t_1) \otimes \mathsf{cons}(d_2,0,t_2)\end{aligned}$$

Lösung zu G38: Die Matrizen seien mit Trennzeichen vorgegen, d.h. $\left(\begin{smallmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{smallmatrix}\right)$ wird z.B. von dem String $1, 2, 3; 4, 5, 6;$ kodiert. Wir müssen nun eine Turing Maschine wie in H5 angeben konstruieren.

Das Produkt einer Matrizenmultiplikation können wir problemlos Schritt für Schritt auf einem Write-Once-Band ausgeben: Wir müssen uns lediglich merken welche Elemente in der Matrizen als nächsten zu Multiplizieren sind, dazu benötigen wir zwei Zeiger, welche logarithmischen Platz benötigen. Das erhaltene Produkte summieren wir in einem Register auf bis wir die jeweilige Zeile und Spalte abgearbeitet haben und es schließlich auf dem Ausgabeband ausgeben können.

Ein solches Register können wir uns erlauben, da wir wissen das wir es mit 32-bit Integers zu tun haben, das Register also nur einen von der Eingabe unabhängigen (konstanten) Platz belegt.

Lösung zu G39: Wir konstruieren Schrittweise die Menge A: Dazu initialisieren wir auf einem Arbeitsband A mit $\{x_s\}$. Jetzen prüfen wir der Reihe nach jede Regel mit allen momentan in A enthaltenen Elementen auf Anwendung und fügen gegebenfalls das Schluß-Element der Regel zu A hinzu.

Erhalten wir dabei x_t akzeptieren wir, kommt in einem Schritt jedoch keine einzige Regel zur Anwendung (überprüfbar durch ein Flag oder auch in der Zustandsmenge kodierbar), so brechen wir mit Verneinung ab; denn wenn in einem Schritt nichts neues mehr hinzukommt wird auch eine erneute Überfprüfung zu dem gleichen Schluss kommen.

Offensichtlich können maximal soviele Elemente zu A hinzukommen wie X groß ist (oder es Regeln gibt), bezeichnen wir diese Größe mit n, welche kleiner als die Eingabegröße ist. In jedem Schleifendurchlauf müssen wir jede Regel mit jedem Paar von Elementen aus A überprüfen, d.h. es sind $O\left(n^3\right)$ Schritte notwendig. Da maximal n Element hinzukommen können wir die Schleife höchstens n-mal durchlaufen, was eine Gesamtlaufzeit von $O\left(n^4\right)$ ergibt.

Wir müssen uns bei dieser Vorgehensweise die Menge A merken, sowie drei Zeiger welche sich merken welche Regel und welche Elemente aus A wir als nächtsen prüfen müssen. Damit ergibt sich ein Platzverbrauch von $O\left(n \log n\right)$.

Lösung zu H9: Eine Beispielimplementierung wird sich nach Ablauf der Abgabefrist zum Download bereit auf der Homepage der Vorlesung finden.