Lehrstuhl für Informatik 1
Prof. Dr. Berthold Vöcking
Alexander Skopalik

WS 2007/08
5.11.2007

# Übung zur Vorlesung
## Berechenbarkeit und Komplexität

## Lösung Blatt 3

---

**Aufgabe 3.1:** (**5**)

Zunächst der Algorithmus zur Berechnung von $\lfloor\sqrt{n}\rfloor$ in C:

```
i=1; j=1;
while (j<=n)
{
  i=i+1;
  j=i*i;
}
return i-1;
```

Es wird so lange die Zahl $i$ quadriert, bis $j = i^2 > n$ ist. Dann wird der Wert von $i$ aus der vorhergehenden Iteration ausgegeben, in der letztmalig $i \leq \sqrt{n}$ war.

Übersetzt in die RAM-Sprache sieht es dann wie folgt aus.

```
  CLOAD 1
  STORE 2 // i=1
  STORE 3 // j=1
$while
  LOAD 1
  SUB 3  // c(0)=n-j
  IF c(0) < 0 GOTO end
  CLOAD 1
  ADD 2
  STORE 2 // i=i+1
  LOAD 2
  MUL 2
  STORE 3 // j=i*i
  GOTO while
$end
  LOAD 2
  CSUB 1
  STORE 1
  END
```

Anstatt die absoluten Sprungadressen anzugeben, verwenden wir zur Vereinfachung und besseren Lesbarkeit die von Assemblersprachen bekannten Sprungmarken. Ansonsten ist dieses RAM-Programm eine genaue Übersetzung des obigen C-Programms.

**Aufgabe 3.2:** (**10**)

Sei $M$ die simulierte TM, und sei $S$ die simulierende TM.

Die Idee besteht darin, dass $S$ bei der Simulation eines Schrittes von $M$ in fester Reihenfolge jeweils über sämtliche Speicherzellen läuft, auf die $M$ in diesem Schritt bisher zugegriffen haben könnte. Sei der Kopf beim Start auf Position 0. Zum Zeitpunkt $t > 0$ kann der Kopf von $M$ nur auf den Positonen $-t, -t+1, \ldots, t-1, t$ sein.

Daher läuft $S$ bei der Simulation von Schritt $t$ zunächst nach rechts bis zur Position $t$, dann nach links bis zur Position $-t$, um dann wieder zur Position 0 zurückzukehren. Dieses Verhalten ist unabhängig von der Eingabe.

Zusätzlich muss $S$ noch die simulierte Kopfposition von $M$ speichern. Dazu erweitern wir $S$ auf zwei Spuren, so dass die erste Spur das Band von $M$, und die zweite Spur die Kopfposition von $M$ enthält. Die zweite Spur enthält an der Stelle, wo der Kopf von $M$ zum Zeitpunkt $t$ steht, eine Markierung, und ansonsten Blanks. Zu Anfang steht die Markierung an Position 0.

Wenn $S$ bei einem Simulationsschritt auf die Markierung stößt, simuliert $S$ einen Schritt von $M$ gemäß dessen Zustandsüberführungsfunktion $\delta$, mit dem Unterschied, dass die Kopfbewegung von $M$ nur simuliert wird. $S$ bewegt sich also unabhängig von der Eingabe weiter, verschiebt aber die Markierung entsprechend. Falls es dabei vorkommt, dass $S$ sich nach rechts weiterbewegt, und sich der Kopf von $M$ nach links bewegt, so verändert $S$ die Markierung zunächst nicht, sondern läuft weiter, merkt sich dies aber im Zustand. Wenn $S$ dann zurückläuft, kann die Markierung entsprechend angepasst werden. Gleiches gilt, falls $S$ nach links, aber $M$ nach rechts läuft.

Offensichtlich arbeitet $S$ oblivious und hält genau dann, wenn $M$ hält.

**Aufgabe 3.3:** (**5+5**)

(a) Wir nutzen aus, dass eine RAM in einem Register eine beliebig große natürliche Zahl abspeichern kann. Weiterhin benutzen wir Aufgabe 2.3, und brauchen daher nur eine TM mit einseitig unendlichem Band zu simulieren. Wir nehmen $\Sigma = \{0, 1\}$ und $\Gamma = \Sigma \cup \{B\}$ an.

Wir kodieren den Inhalt eines Bandes als Zahl im Ternärsystem, d.h. zur Basis 3. $'0'$ und $'1'$ identifizieren wir mit den Zahlen 1 und 2, das Blank $'B'$ mit der Zahl 0. Sei $\text{code}\colon \Gamma \to \{0, 1, 2\}$ diese Funktion. Wenn $w = w_1 \ldots w_n \in \Gamma^n$ der Inhalt des Bandes ist (auf den Positionen $p > n$ stehen nur Blanks), speichern wir die Zahl $t = \sum_{i=0}^{n} \text{code}(w_i) \cdot 3^i$ im Register der RAM.

Für den Inhalt eines jeden Bandes der TM benötigen wir ein Register. Die Kopfposition speichern wir in einem eigenen Register, d. h. für die Bänder $1, \ldots, k$ verwenden wir die Register $c(1), \ldots, c(2k)$.

Der Zugriff auf den Bandinhalt geschieht über DIV- und MOD-Operationen (ein MOD kann mit Hilfe von DIV und Subtraktion simuliert werden). Soll Bandposition $i$ ausgelesen werden, berechnen wir $x' = t$ DIV $3^{i-1}$, um zunächst die niedrigstwertigen $i-1$ Bits zu löschen. Das gesuchte Zeichen ist dann $\text{code}^{-1}(x)$ mit $x = x'$ MOD 3.

Alle weiteren Schritte der Simulation bleiben unverändert.

(b) Hier nutzen wir aus, dass wir nur eine RAM mit konstant vielen Registern simulieren müssen, da wir bereits wissen, dass auch RAMs mit konstant vielen Registern Turing-mächtig sind. Sei $c(k)$ das größte vom simulierten Programm benutzte Register.

(a) Der Befehl ADD $i$ bewirkt ein $c(0) = c(0) + c(i)$. Da der Akkumulator an den meisten Befehlen implizit beteiligt ist, und wir das Register $i$ nicht verändern wollen, benutzen wir zwei Hilfsregister $c(k+1) = c(0)$ und $c(k+2) = c(i)$. Etwas abstrahiert und in einer Hochsprache sieht der Befehl dann wie folgt aus.

```
c(k+1)=c(0);
c(k+2)=c(i);
while (c(k+2)>0)
{
  c(k+1)=c(k+1)+1;
  c(k+2)=c(k+2)-1;
}
c(0)=c(k+1);
```

Dabei wird davon ausgegangen, dass das ursprüngliche RAM-Programm nur die Register $c(0)$ bis $c(k)$ benutzt. Übersetzt lautet der RAM-Befehl dann:

```
  STORE k+1
  LOAD i
  STORE k+2
$while
  LOAD k+2
  IF c(0)=0 GOTO end
  LOAD k+1
  CADD 1
  STORE k+1
  LOAD k+2
  CSUB 1
  STORE k+2
  GOTO while
$end
  LOAD k+1
```

(b)
```
  LOAD i;
  IF c(0)=0 GOTO reg0
  CSUB 1
  IF c(0)=0 GOTO reg1
  CSUB 1
  IF c(0)=0 GOTO reg2
  CSUB 1
  ...
  IF c(0)=0 GOTO regk
$reg0
  LOAD 0
  GOTO end
$reg1
  LOAD 1
  GOTO end
$reg2
  ...
$regk
  LOAD k
$end
```

(c) Wir wollen testen, ob $c(0) < x$ ist, haben aber nur die Abfrage $c(0) = x$ zur Verfügung. Da wir annehmen, dass die RAM in ihren Registern ganze Zahlen abspeichert, müssen wir nacheinander für ansteigendes $y$ testen, ob $c(0) + y = x$ oder $c(0) - y = x$ ist. Dazu wird in $c(k + 1)$ der Wert $c(0) + y$, und in $c(k + 2)$ der Wert $c(0) - y$ gespeichert. Schließlich wird eine der beiden

Bedingungen erfüllt sein, so dass die Schleife abgebrochen wird, und entweder nach $j$ verzweigt wird, oder das Programm mit der nächsten Zeile fortgesetzt wird.

```
  IF c(0)=x GOTO end
  STORE k+1
  STORE k+2
$loop
  LOAD k+1
  CADD 1
  STORE k+1
  IF c(0)=x GOTO j
  LOAD k+2
  CSUB 1
  STORE k+2
  IF c(0)=x GOTO end
  GOTO loop
$end
```