RWTH Aachen **WS 2012/13**
Lehrgebiet Theoretische Informatik Übungsblatt 7
Reidl–Ries–Rossmanith–Sanchez–Tönnis 26.11.2012

Übung zur Vorlesung Berechenbarkeit und Komplexität

Aufgabe T15

Entwickeln Sie ein WHILE-Programm, daß den Wert 2^{x_1} berechnet. Analysieren Sie die Laufzeit ihres Programmes im uniformen und im logarithmischen Kostenmaß.

Lösungsvorschlag.
Unter der Annahme, daß alle Variablen die nicht zur Eingabe gehören mit 0 initialisiert sind, berechnet das folgende Programm 2^{x_1}:

Eingabe: $x_1 \in \mathbb{N}$

```
x2 := x2 + 1;
WHILE x1 ≠ 0 DO
    x3 := x2 + 0;
    WHILE x3 ≠ 0 DO
        x2 := x2 + 1;
        x3 := x3 − 1
    END;
    x1 := x1 − 1
END;
x0 := x2 + 0
```

Ausgabe: x_0

Das aktuelle Zwischenergebnis steht immer in x_2 während x_1 und x_3 als Schleifenvariablen genutzt werden.

Im uniformen Kostenmaß beträgt die Laufzeit $O(2^n)$. Im logarithmischen Kostenmaß kommt ein Faktor $n = \log(2^n)$ hinzu, da die Additionen $x = x + 1$ hier $\log(x)$ Zeit benötigen.

Aufgabe T16

Die Ackermannfunktion $A : \mathbb{N}^2 \to \mathbb{N}$ wurde in der Vorlesung folgendermaßen definert:

$$\begin{array}{lcll} A(0,m) & = & m+1 & \text{für } m \geq 0 \\ A(n+1,0) & = & A(n,1) & \text{für } n \geq 0 \\ A(n+1,m+1) & = & A(n,A(n+1,m)) & \text{für } n,m \geq 0 \end{array}$$

a) Zeigen Sie, daß die Ackermannfunktion für alle Parameter $n, m \in \mathbb{N}$ terminiert.

b) Beweisen Sie durch Induktion nach n folgende Aussage:

$$A(n,m) \leq A(n+1,m-1) \text{ für alle } m \geq 0$$

Hinweis: Nutzen Sie die Monotonie der Ackermannfunktion in beiden Parametern aus.

Lösungsvorschlag.

a) In jedem Schritt wird entweder m verringert oder m erhöht und n verringert. Jedes Mal wenn m Null erreicht, wird n verringert, also muss auch n irgendwann Null erreichen. Man beachte allerdings daß bei Verringerung von n keine obere Schranke für das Wachstum von m in den Funktionsaufrufen gibt.

b) Induktionsanfang: Nach Definition gilt $A(n+1,0) := A(n,1)$ also insbesondere $A(n,1) \leq A(n+1,0)$.

   Induktionsschritt: Sei die Behauptung jetzt schon für $n \geq 1$ bewiesen.

   Aufgrund der Definition $A(0,m) := m+1$ und der Monotonie im ersten Parameter gilt $m+1 \leq A(n,m)$ für alle $m \geq 0$.

   Damit gilt $A(n,m+1) \leq A(n,A(n,m)) \leq A(n,A(n+1,m-1)) = A(n+1,m)$, was zu zeigen war. Dabei geht in die erste Ungleichung die Monotonie im zweiten Parameter ein, in die zweite die Induktionsvoraussetzung, und in die letzte Gleichung die Definition.

Aufgabe T17

Sind WHILE-Programme immer noch Turing-mächtig, wenn die Zuweisungen $x_i := x_j + c$ nur noch für $c \in \{-1, 0\}$ erlaubt sind?

Lösungsvorschlag.

Wir zeigen für folgende (WHILE-berechenbare) Funktion, daß sie nicht durch die eingeschränkten WHILE-Programme berechenbar ist: $f : \mathbb{N} \to \mathbb{N}$, $f(n) = 1$ für alle $n \in \mathbb{N}$. Dies geht sehr einfach, in dem man per Induktion zeigt: Wenn alle Variablen zu Beginn eines WHILE-Programms mit 0 belegt sind, gilt dies auch nach der Ausführung.

Induktionsanfang: Das WHILE-Programm hat die Form $x_i := x_j + c$ für $c \in \{-1, 0\}$. Hier gilt die Aussage offensichtlich.

Induktionsschluss:

Fall 1: Das WHILE-Programm hat die Form WHILE $x_i \neq 0$ DO P END. P wird offensichtlich nicht ausgeführt, also ändert sich die Belegung der Variablen nicht.

Fall 2: Das WHILE-Programm hat die Form $P_1; P_2$. Weder P_1 noch P_2 ändert die Belegung der Variablen, also gilt auch hier die Aussage

Somit können mit den eingeschränkten WHILE-Programmen höchstens Funktionen mit $f(0) = 0$ berechnet werden.

Aufgabe H16 (8 Punkte)

Betrachten Sie die nachfolgenden Varianten des Euklidischen Algorithmus (entnommen aus Wikipedia) zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen a und b.

Variante 1:	**Variante 2:**
Eingabe: $a, b \in \mathbb{N}$	**Eingabe:** $a, b \in \mathbb{N}$
`While` $a \neq b$	`While` $b > 0$
`If` $a > b$	$r := a \bmod b$
`then` $a := a - b$	$a := b$
`else` $b := b - a$	$b := r$
`End While`	`End While`
Ausgabe: a	**Ausgabe:** a

Kommentar

Der euklidische Algorithmus berechnet den größten gemeinsamen Teiler von zwei natürlichen Zahlen a und b. Die in der Aufgabenstellung angegebenen Varianten beruhen auf dem folgenden Lemma.

Lemma 1 *Falls $b \neq 0$, dann gilt*

$$\begin{aligned} \mathrm{ggT}(a,b) &= \mathrm{ggT}(a-b,b) \\ \mathrm{ggT}(a,b) &= \mathrm{ggT}(a \bmod b, b)\ . \end{aligned}$$

Wir werden im Folgenden o.B.d.A. annehmen, daß $a \geq b$ gilt.

1. Bestimmen Sie eine möglichst scharfe untere Schranke für die Worst-Case-Laufzeit von Variante 1 im uniformen Kostenmaß.

   Lösungsvorschlag

   Betrachte den Fall $b = 1$. Wie oben angenommen gilt $a \geq b$. Da entweder $a = b$ oder $a > b$ gilt, wird in jedem Durchlauf der WHILE-Schleife wird, die Anweisung $a := a - b$ ausgeführt. Das heißt aber, daß in jeder Iteration Eins von a subtrahiert wird und somit die WHILE-Schleife $\Omega(a)$ mal ausgeführt wird. Folglich ist die Worst-Case-Laufzeit von Variante 1 durch $\Omega(a)$ nach unten beschränkt.

2. Bestimmen Sie eine möglichst scharfe obere Schranke für die Worst-Case-Laufzeit von Variante 2 im uniformen Kostenmaß.

   Lösungsvorschlag

   Variante 2 berechnet in jeder Iteration der WHILE-Schleife zunächst $r = a \bmod b$ und setzt dann $a := b$ und $b := r$. Da wie oben angenommen $a \geq b$ gilt, lässt sich der Wert von r nach der ersten Runde durch $r = a \bmod b \leq \frac{a}{2}$ abschätzen (betrachte hierzu den Fall $b = \lfloor a/2 \rfloor + 1$). Somit sind nach zwei Runden sowohl a als auch b nur noch höchstens halb so groß wie die größere der beiden Eingaben, d.h. hier a. Nach $2 \cdot k$ Runden gilt also $a, b \leq \frac{a}{2^k}$. Die Anzahl der Iterationen beträgt somit maximal $2 \cdot \log_2(a)$. Damit berechnet Variante 2 den größten gemeinsamen Teiler von a und b mit höchstens $O(\log(a))$ vielen Modulo-Operationen. Folglich ist die Worst-Case-Laufzeit von Variante 2 durch $O(\log(a))$ nach oben beschränkt.

3. Nutzen Sie Ihre Abschätzungen, um zu zeigen, daß sich die uniformen Worst-Case-Laufzeiten beider Varianten durch einen exponentiellen Faktor unterscheiden.

Lösungsvorschlag

Wie in den Aufgabenteilen oben gezeigt, hat Variante 1 eine Worst-Case-Laufzeit von $\Omega(a)$, Variante 2 hingegen eine von $O(\log(a))$. Die Eingabelängen von a und b können durch $\log(a)$ bzw. $\log(b)$ abgeschätzt werden. Da wir angenommen haben, daß $a \geq b$ gilt, ist die Eingabelänge folglich durch $2 \cdot \log(a)$ beschränkt. Im Vergleich zur Eingabelänge benötigt Variante 2 also polynomielle und Variante 1 exponentielle Zeit.

4. Stimmt diese Aussage auch bezüglich der Laufzeiten im logarithmischen Kostenmaß?

Lösungsvorschlag

Ja, die Aussage stimmt weiterhin. Der Zeitaufwand im logarithmischen Kostenmaß zur Berechnung von einer Zuweisung, eines Vergleichs oder einer Subtraktion zweier Zahlen $a, b \in \mathbb{N}$ mit $a \geq b$ ist $O(\log(a))$. Die Berechnung von $(a \bmod b)$ kann mit Hilfe einer Division, einer Multiplikation und einer Subtraktion durchgeführt werden. Eine Multiplikation und eine Division benötigen nach der Schulmethode $O(\log^2(a))$ viele Schritte. Demnach wird der Aufwand von Variante 2 im Vergleich zum uniformen Kostenmaß also höchstens um einen polylogarithmischen Faktor vergrößert. Der exponentielle Gap bleibt also bestehen.

Aufgabe H17 (6 Punkte)

Geben Sie ein LOOP-Programm an, daß folgende Sprache akzeptiert:

$$L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$$

wobei $|w|_i$ die Anzahl der Stellen in w angibt, an denen die Ziffer i steht.

Gehen Sie davon aus, daß die Eingabe in der Variable x_1 als Binärzahl kodiert steht und das zudem die Länge der Eingabe in der Variable x_2 zu finden ist. Wenn der Wert x_1 in der Sprache L enhalten ist, soll am Ende des Programs x_0 eine Eins enthalten, ansonsten Null.

Nehmen Sie an, daß die Subtraktion von Variablen mit dem Wert 0 wiederum 0 ergibt (Variablen können also nie negative Werte enthalten). Zudem setzen wir voraus, daß die Eingabe x_1 nie das leere Wort enthält.

Lösungsvorschlag.

Eingabe: $x_1 \in \mathbb{N}$, $x_2 \in \mathbb{N}$

```
x6 := 0; // Anzahl Nullen
x7 := 0; // Anzahl Einzen
LOOP x2 DO
    x4 := 1;
    x5 := 0;
    x3 := x1;
    LOOP x3 DO
        x10 := 1;
        x11 := x4;
        LOOP x11 DO
            x10 := 0;
            x4 := 0;
            x5 := 1;
            x1 := x1 − 1;
        END
        LOOP x10 DO
            x11 := x5;
            LOOP x11 DO
                x4 := 1;
                x5 := 0;
            END
        END
    END
    LOOP x4 DO // Gerade
        x6 := x6 + 1;
    END
    LOOP x5 DO // Ungerade
        x7 := x7 + 1;
    END
END
x8 := x6;
x9 := x7;
LOOP x7 DO
    x8 := x8 − 1;
END
LOOP x6 DO
    x9 := x9 − 1;
END
x0 := 1;
LOOP x8 DO
    x0 := 0;
END
LOOP x9 DO
    x0 := 0;
END
```

Ausgabe: x_0