Technische Universität München
Institut für Informatik
PD Dr. Markus Holzer
Dipl.-Inf. Michael Tautschnig

WS 2007/2008
Übungsblatt 3
Mittwoch, 7. November 2007

Übung zur Vorlesung Automaten, Formale Sprachen und Berechenbarkeit

Lösungshinweise

Aufgabe 1 *Endiche Folgen*

Nach dem Satz von Schröder-Bernstein reicht es, jeweils eine injektive Abbildung $f : M \to \mathbb{N}$ und $f' : \mathbb{N} \to M$ anzugeben. Sei $\mathbb{P} = \{p_1, p_2, p_3, \ldots\}$ die Menge aller Primzahlen. Wir wählen als Abbildung f:

$$f((a_1, \ldots, a_n)) = p_1^n p_2^{a_1} p_3^{a_2} \ldots p_{n+1}^{a_n};$$

f ist injektiv, da die Primfaktorzerlegung einer natürlichen Zahl eindeutig ist. Für f' wählt man etwa $f'(n) = (n)$; f' is trivialerweise injektiv.

Aufgabe 2 *Funktionen*

Sei $l = \max_{x \in \mathbb{N}}\{f(x) \neq 0\}$; nach Voraussetzung existiert diese Zahl. Die Funktion $f(x)$ kann nun effektiv durch ein Programm berechnet werden, das für alle Werte $x > l$ Null ausgibt und die Werte für $x \leq l$ aus einer Tabelle bezieht, die Teil des Programmcodes ist.

Aufgabe 3 *Bijektionen*

(i) Eine effektiv berechenbare Bijektion ist etwa $f(x) = x$.

(ii) Sei ϕ_i eine Aufzählung aller effektiv berechenbarer Abbildungen $\mathbb{N} \to \mathbb{N}$. Wir zeigen, dass es eine Bijektion $f : \mathbb{N} \to \mathbb{N}$ gibt, die von keinem ϕ_i berechnet wird. Diese Abbildung f wird schrittweise definiert – im i-ten Schritt bestimmen wir den Wert von $f(i)$. Setze $M_0 = \mathbb{N}$. Im jeweils i-ten Schritt ($i \geq 0$), wähle das kleinste Element z_i aus M_i aus, sodass $\phi_i(i) \neq z_i$ und definiere $f(i) = z_i$; setze $M_{i+1} = M_i \backslash \{z_i\}$ und starte die nächste Iteration. Die erhaltene Funktion f ist eine Bijektion (da jedes Element aus $\mathbb{N}$ exakt ein Mal verwendet wird). Zudem ist f aufgrund der Konstruktion nicht effektiv berechenbar, da es sich an mindestens einer Stelle von jedem ϕ_i unterscheidet.

Aufgabe 4 *WHILE-Programme*

Zur Vereinfachung der Notation nehmen wir bei allen Makros implizit an, dass die Berechnungen auf "frischen Kopien" der Eingabevariablen durchgeführt werden und sich insbesondere Änderungen der Eingabevariablen innerhalb eines Makros nicht auf das aufrufende Programm auswirken. Dies ist durch entsprechendes Umkopieren der Variablen immer möglich.

Ein Additions-Makro kann durch Zählen realisiert werden:

$R = \mathbf{ADD}(X, Y)$
  $ZERO := 0$
  while $Y \neq ZERO$ **do**
    $X := succ(X)$
    $Y := pred(Y)$
  od
  $R := succ(X)$
  $R := pred(R)$

Ein Makro für die Multiplikation kann durch iterierte Addition gewonnen werden:

$R = \mathbf{MUL}(X, Y)$
  $ZERO := 0$
  $Z := 0$
  while $Y \neq ZERO$ **do**
    $Z := \mathbf{ADD}(Z, X)$
    $Y := pred(Y)$
  od
  $R := succ(Z)$
  $R := pred(R)$

Unter Verwendung von **MUL** kann in analoger Weise nun ein Makro zur Auswertung von x^y erstellt werden.

$R = \mathbf{POW}(X, Y)$
  $ZERO := 0$
  $Z := 1$
  while $Y \neq ZERO$ **do**
    $Z := \mathbf{MUL}(Z, X)$
    $Y := pred(Y)$
  od
  $R := succ(Z)$
  $R := pred(R)$

Durch wiederholte Anwenung des Makros **MUL** kann schließlich die gewünschte Funktion realisiert werden:

  $ZERO := 0$
  $Z := \mathbf{succ}(X)$
  $Z := \mathbf{pred}(Z)$
  $Y := pred(Y)$
  while $Y \neq ZERO$ **do**
    $Z := \mathbf{POW}(2, Z)$
    $Y := pred(Y)$
  od
  $X := succ(Z)$
  $X := pred(X)$