Technische Universität München
Fakultät für Informatik
Dr. M. Schneider, Dr. M. Pizka,
Dr. A. Vilbig, Dr. R. Letz,
G. Bauer, B. Zhang

SS 2003
Lösungsvorschläge zu Blatt 6
23. Mai 2003

Übungen zur Vorlesung Einführung in die Informatik IV

Eine primitiv rekursive Funktion besteht entweder aus einer der Grundfunktionen:

$$\begin{array}{lll} succ: & \mathbb{N} \to \mathbb{N}, & succ(n) = n+1 \\ zero^{(0)}: & \to \mathbb{N}, & zero^{(0)}() = 0 \\ zero^{(1)}: & \mathbb{N} \to \mathbb{N}, & zero^{(1)}(n) = 0 \\ \pi_i^n: & \mathbb{N}^n \to \mathbb{N}, & \pi_i^n(x_1,...,x_n) = x_i; \end{array}$$

aus Komposition primitiv rekursiver Funktionen

$$\begin{array}{ll} g: & \mathbb{N}^n \to \mathbb{N} \\ h_i: & \mathbb{N}^m \to \mathbb{N}, 1 \le i \le n \end{array}$$

durch $f: \quad \mathbb{N}^m \to \mathbb{N}, \quad f = g \circ [h_1,...,h_n], \quad$ mit $f(x_1,...,x_m) = g(h_1(x_1,...,x_m),...,h_n(x_1,...,x_m))$

oder aus primitiv rekursiven Funktionen

$$\begin{array}{ll} g: & \mathbb{N}^k \to \mathbb{N} \\ h: & \mathbb{N}^{k+2} \to \mathbb{N} \end{array}$$

durch das Schema der primitiven Rekursion

$$\begin{array}{lcl} f(x_1,...,x_k,0) & = & g(x_1,...,x_k), \\ f(x_1,...,x_k,n+1) & = & h(x_1,...,x_k,n,f(x_1,...,x_k,n)), \end{array}$$

abgekürzt durch

$$f = pr(g,h).$$

Aufgabe 23 Primitiv rekursive Funktionen

Definitionsschema für einstellige Funktionen:

$$\begin{array}{ll} f(0) & = g() \\ f(succ(n)) & = h(n,\ f(n)) \end{array}$$

Definitionsschema für zweistellige Funktionen:

$$\begin{array}{ll} f(m,\ 0) & = g(m) \\ f(m,\ succ(n)) & = h(m,\ n,\ f(m,\ n)) \end{array}$$

$$\begin{array}{ll} sq(O) = 0 & = g() \\ sq(succ(n)) = sq(n) + 2 * n + 1 = add(sq(n),\ add(1\ mult(2,\ n))) & = h(n,\ sq(n)) \end{array}$$

also ist $sq = pr(g,\ h)$ mit
$g = zero^{(0)}$,
$h = add \circ [\pi_2^2,\ add \circ [one^{(2)},\ mult \circ [two^{(2)},\ \pi_1^2]]]$,
$one^{(2)} = succ \circ zero^{(1)} \circ \pi_1^2$ und
$two^{(2)} = succ \circ succ \circ zero^{(1)} \circ \pi_1^2$.

Aufgabe 24 **μ-rekursive Funktionen**

Eine μ-rekursive Definition der partiellen Funktion `psqrt`, die Quadratwurzeln natürlicher Zahlen berechnet:

$$\begin{aligned} \texttt{psqrt} &= \mu(h) \\ &\quad \text{mit} \quad h : \mathbb{N}^2 \to \mathbb{N} \text{ partiell gegeben durch:} \\ h(x,y) &= \texttt{psub}(x, y \cdot y) \quad (\texttt{psub} \text{ sei die partielle Subtraktion.}) \end{aligned}$$

Behauptung:

$$\texttt{psqrt}(x) = \begin{cases} w & \text{falls } w^2 = x \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Dabei sei w die ganzzahlige Wurzel von x, d.h. die eindeutig bestimmte natürliche Zahl mit der Eigenschaft $w^2 \leq x < (w+1)^2$.

Beweis:

Zu zeigen ist

- Im Fall $w^2 = x$ ist $h(x,y) = 0$ für $y = w$ und $h(x,y) > 0$ für $y < w$.
- Im Fall $w^2 < x$ ist $h(x,y) > 0$ oder $h(x,y)$ undefiniert.

Sei $w^2 = x$. Dann ist x eine Quadratzahl und es gilt

$$\begin{aligned} &\forall y : \\ &(y = w \;\Rightarrow\; \texttt{psub}(x, y \cdot y) = 0) \;\wedge \\ &(y < w \;\Rightarrow\; \texttt{psub}(x, y \cdot y) > 0) \end{aligned}$$

Sei $w^2 < x < (w+1)^2$. Dann ist x keine Quadratzahl und es gilt

$$\begin{aligned} &\forall y : \\ &(y \leq w \;\Rightarrow\; \texttt{psub}(x, y \cdot y) > 0) \;\wedge \\ &(w < y \;\Rightarrow\; \texttt{psub}(x, y \cdot y) \text{ undefiniert,} \\ &\qquad\qquad \text{da } x < (w+1)^2 \leq y \cdot y) \end{aligned}$$

Aufgabe 25 **Äquivalenz von μ-Rekursion und while-Programmen auf Registermaschinen**

(a) Gegeben sei die folgende μ-rekursive Funktion zur Berechnung der partiellen Subtraktion zweier natürlicher Zahlen

$$a \dot{-} b = \mu(h)(a,b),$$

wobei die primitiv rekursiv definierte Funktion $h : \mathbf{N}^3 -> \mathbf{N}$ definiert ist durch die Gleichung

$$h(a,b,y) = \text{sub}(b+y, a) + \text{sub}(a, b+y)$$

(i) Jede primitiv rekursive Funktion lässt sich in ein while-programm umsetzen.
Die Zuweisung $s_0 := h(s_a, s_b, s_y)$ wird in das folgende Registermaschinen-Programm M umgesetzt.

$while_0(pred_0)$	$s_0 := 0;$
$while_1(pred_1)$	$s_1 := 0;$
$while_2(pred_2)$	$s_2 := 0;$
$while_{a'}(pred_{a'})$	$s_{a'} := 0;$
$while_{b'}(pred_{b'})$	$s_{b'} := 0;$
$while_{y'}(pred_{y'})$	$s_{y'} := 0;$
$while_b(succ_0; succ_1; succ_{b'}; pred_b)$	$(s_0, s_1, s_{b'}, s_b) := (s_b, s_b, s_b, 0);$
$while_y(succ_0; succ_1; succ_{y'}; pred_y)$	$(s_0, s_1, s_{y'}, s_y) := (s_0 + s_y, s_1 + s_y, s_y, 0);$
$while_a(pred_0; succ_{a'}; pred_a)$	$(s_0, s_{a'}, s_a) := (s_0 - s_a, s_a, 0);$
$while_{a'}(succ_2; succ_a; pred_{a'})$	$(s_2, s_a, s_{a'}) := (s_{a'}, s_{a'}, 0);$
$while_1(pred_2; pred_1)$	$(s_2, s_1) := (s_2 - s_1, 0);$
$while_2(succ_0; pred_2)$	$(s_0, s_2) := (s_0 + s_2, 0);$
$while_{b'}(succ_b; pred_{b'})$	$(s_b, s_{b'}) := (s_{b'}, 0);$
$while_{y'}(succ_y; pred_{y'})$	$(s_y, s_{y'}) := (s_{y'}, 0);$

(ii) Die μ-Rekursion wird in das folgende Registermaschinen-Programm umgesetzt.

$while_y(pred_y)$	$s_y := 0;$
M	$s_0 := h(s_a, s_b, s_y);$
$while_0(succ_y; M)$	**while** $s_0 > 0$ **do**
	$s_y := s_y + 1$
	$s_0 := h(s_a, s_b, s_y);$
	od

(b) Wir zeigen: Jedes while-Programm auf einer Registermaschine lässt sich in eine μ-rekursive Funktion umsetzen. Wir fassen ein while-Programm auf einer n-Registermaschine auf als eine Funktion $\mathbb{N}^n -> \mathbb{N}^n$, die den Registern $s = (s_1, \ldots, s_n)$ neue Werte zuordnet.

(i) $succ_i$, $pred_i$ und ε sind primitiv rekursiv:
$succ_i = [\pi^n_1, \ldots succ \circ \pi^n_i \ldots \pi^n_n]$
$pred_i = [\pi^n_1, \ldots pred \circ \pi^n_i \ldots \pi^n_n]$
$\varepsilon = id$

(ii) Sind M_1 und M_2 μ-rekursiv, so auch $M_1; M_2$:
$M_1; M_2 = M_2 \circ M_1$

(iii) Ist M μ-rekursiv, so auch $while_i(M)$:

Wir definieren zunächst eine μ-rekursive Funktion p , so dass $p(s,k)$ den Wert der Register nach k-maliger Ausführung des Programms M liefert. (p ist μ-rekursiv, das M μ-rekursiv ist, d.h. das Ergebnis kann auch undefiniert sein, wenn M nicht terminiert)
$p : \mathbb{N}^n \times \mathbb{N} \to \mathbb{N}^n$
$p(s, 0) = s$
$p(s, suc(n)) = M(p(s, n))$

Wir setzen: $while_i(M)(s) = p(s, k)$

wobei k die kleinste Zahl ist, so dass nach k-maliger Ausführung des Programms M das Register s_i den Wert 0 hat, falls diese Zahl existiert, ansonsten undefiniert.
$k = \mu(h)(s) = min\{y.\ \pi_i(p(s,y)) = 0\}$
mit $h(s,y) = \pi_i(p(s,y))$

Falls das while-Programm terminiert, ist die μ-rekursive Funktion definiert, ansonsten undefiniert.

Aufgabe 26 (H) Äquivalenz von primitiver Rekursion und for-Programmen auf Registermaschinen

(a) Um die in der Aufgabenstellung gegebene Behauptung zu beweisen, ist zu zeigen: **primitiv rekursiv** $\Rightarrow$ **for-berechenbar** und **primitiv rekursiv** $\Leftarrow$ **for-berechenbar**.

- „$\Rightarrow$“: Der Aufbau primitiv rekursiver Funktionen legt folgendes Induktionsschema nahe: Der Induktionsanfang erstreckt sich über die Grundfunktionen, ein Induktionsschluß ist über die Komposition und das letztgenannte Schema zu führen.

  Induktionsanfang: Die Grundfunktionen sind for-berechenbar:

  $$\begin{array}{ll} succ & \rightarrow succ_1; \\ zero & \rightarrow for_1(\varepsilon); \\ \pi_i^n & \rightarrow for_1(\varepsilon);\ ...;\ for_{i-1}(\varepsilon); \\ & \qquad for_{i+1}(\varepsilon); ...;\ for_n(\varepsilon); \end{array}$$

  Induktionsschluß: Induktionshypothese ist, dass die primitiv rekursive Funktionen g und h (bzw. h_i) for-berechenbar sind; entsprechende Programme seien im weiteren durch G, H bzw. H_i abgekürzt. Zu zeigen ist dann, dass jede Funktion die aus g und h (bzw. h_i) in einem Schritt, also durch Komposition oder durch das Schema der primitiven Rekursion aufgebaut werden kann, wiederum for-berechenbar ist.

  Komposition: Funktionen, die durch Komposition aus G und H_i berechnet werden, sind ebenfalls for-berechenbar, denn

  $$f = g \circ [h_1, ..., h_n] \rightarrow H_1;\ ...;\ H_n;\ G$$

  .

  Schema der primitiven Rekursion: Es bleibt noch der Fall zu betrachten, dass eine Funktion f, die durch das Schema der primitiven Rekursion aus for-berechenbaren Funktionen $g(x_1, ..., x_k)$ und $h(x_1, ..., x_k, y, z)$ entsteht, wiederum for-berechenbar ist. Durch das folgende Programm wird f berechnet:

G;	result := $g(x_1, ..., x_k)$;
$for_m(\varepsilon)$;	m := 0;
$pred_y$	y := y - 1;
for$_y(H, succ_m)$;	**for** i := y **to** 0 **do**
	result := $h(x_1, ..., x_k, m, result)$;
	m := m + 1;
	od

  hierbei verifiziert man durch Induktion über die Anzahl der Schleifendurchläufe, dass `result` nach $i \geq 0$ Schleifendurchläufen den Wert $f(x_1, ..., x_k, i)$

hat. Die Schleife wird im ganzen y-mal durchlaufen, so dass `result` schließlich den gesuchten Wert $f(x_1,...,x_k,y)$ annimmt. Schließlich muß das angegebene Pseudocode-Programm noch in ein Registermaschinen-Programm transformiert werden: Die beiden dort auftretenden Zuweisungen der Variablen `result` sind per Induktionsannahme RM-berechenbar; wir kürzen diese mit G und H ab.

- „⇐“: Wir zeigen: Jedes for-Programm auf der Registermaschine lässt sich in eine primitive rekursive Funktion umsetzen.

  (i) $succ_i$, $pred_i$ und ε sind primitiv rekursiv.
  $succ_i = [\pi^n_1, \ldots succ \circ \pi^n_i \ldots \pi^n_n]$
  $pred_i = [\pi^n_1, \ldots pred \circ \pi^n_i \ldots \pi^n_n]$
  $\varepsilon = id$

  (ii) Sind M_1 und M_2 primitiv rekursiv, so auch $M_1;M_2$:
  $M_1;M_2 = M_2 \circ M_1$

  (iii) Ist M primitiv rekursiv, so auch die k-malige Ausführung $p(s,k)$ von M:
  $p : \mathbb{N}^n \times \mathbb{N} \to \mathbb{N}^n$
  $p(s,0) = s$
  $p(s,suc(n)) = M(p(s,n))$

  Ist M primitiv rekursiv, so auch $for_i(M)$:
  Wir setzen: $for_i(M)(s) = p(s,i)$

(b) Die in der Aufgabenstellung gegebene For-Schleife unterscheidet sich von einer While-Schleife im wesentlichen dadurch, dass bei einer For-Schleife die Zahl der Schleifendurchläufe von Anfang an, durch die im Register i gespeicherte Zahl n auf eben dieses n festgelegt ist. Bei einer While-Schleife ist dagegen die Zahl der Durchläufe nicht fest vorgegeben, sondern hängt von dem Programm M ab. Aus dem eben durchgeführten Beweis können wir also folgern, dass bei primitiver Rekursion die Zahl der rekursiven Aufrufe abschätzbar ist. Darin unterscheiden sich primitiv rekursive Funktionen von μ-rekursiven, wo die Zahl der rekursiven Aufrufe von vornherein nicht feststeht.

Aufgabe 27 (P) Rekursionsoperator

Teilaufgabe a)

Die Fakultätsfunktion fac ist primitiv rekursiv definiert durch

$$\begin{aligned} fac(0) &= succ(zero^{(0)}()) \\ &= succ \circ zero^{(0)} \\ fac(n+1) &= (n+1) \cdot fac(n) \\ &= mult(succ(n), fac(n) \\ &= mult \circ [succ \circ \pi^2_1, \pi^2_2](n, fac(n)) \end{aligned}$$

Die zuletzt aufgeführte Gleichung erfüllt somit die für das Rekursionsschema erforderliche Form $f(n+1) = h(n, f(n))$, d.h. der Rekursionsoperator verknüpft die beiden primitv rekursiven Funktionen

$$\begin{aligned} g &= succ \circ zero^{(0)} \\ h &= mult \circ [succ \circ \pi^2_1, \pi^2_2] \end{aligned}$$

Teilaufgabe b) und c)

```
-- Constant function zero
zero0 = 0

-- Successor function
succ x = x + 1

-- Dyadic projections
pi21 (x, y) = x
pi22( x, y) = y

-- Composition operators for monadic and dyadic function g
comp  g h x = g(h x)
comp2 g h1 h2 x = g(h1 x, h2 x)

-- Auxiliary definition of multiplication
mult(x, y) = x * y

-- Primitive recursion operator for monadic function f
pr :: Int -> ((Int, Int) -> Int) -> (Int -> Int)
pr g h  =  f  where
                    f(0)      = g
                    f(n+1) = h(n, f(n))

-- Primitive recursive definition of the factorial
fac = pr (succ(zero0))
         (comp2 mult (comp succ pi21) pi22)
```