Universität Karlsruhe

Informatik 1 WS 00/01

Institut für Telematik, Forschungsgruppe C&M
Prof. Dr. S. Abeck, R. Scholderer

Musterlösungen zum Übungsblatt 12

Lösung

1. Funktional und induktive Deutung der Fakultätsfunktion

Teilaufgabe 1:

$\tau : (N^I \rightarrow N^I) \rightarrow (N^I \rightarrow N^I)$

$$\tau[f]\,(x) = \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ x * f(x-1), \text{ falls } x > 0 \end{cases}$$

Teilaufgabe 2:

fac_0:

1. Schritt: Anwendung der Definition

$fac_0(x) = I$

fac_1:

1. Schritt: Es ist $fac_1(x) = \tau[fac_0]\,(x)$ nach Definition

$$\tau[fac_0]\,(x) = \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ x * fac_0(x-1), \text{ falls } x > 0 \end{cases}$$

2. Schritt: Einsetzen von $fac_0(x) = I$ nach Definition

$$= \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ x * I, \text{ falls } x > 0 \end{cases}$$

3. Schritt: * ist strikt

$$= \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ I, \text{ falls } x > 0 \end{cases}$$

4. Schritt: Zusammenfassung des 1. und 3. Falls

$$= \begin{cases} I, \text{ falls } x = I \lor x > 0 \\ 1, \text{ falls } x = 0 \end{cases}$$

fac$_2$:

1. Schritt: Es ist $fac_2(x) = \tau[fac_1]\,(x)$ nach Definition

$$\tau[fac_1]\,(x) = \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ x * fac_1(x-1), \text{ falls } x > 0 \end{cases}$$

2. Schritt: Einsetzen von $fac_1(x-1)$ mit den dazugehörigen Bedingungen

$$= \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ x * I \text{ falls } x > 0 \land (x-1 = I \lor x-1 > 0) \\ x * 1 \text{ falls } x > 0 \land x-1 = 0 \end{cases}$$

3. Schritt: * ist strikt

$$= \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ I, \text{ falls } x > 0 \land (x-1 = I \lor x-1 > 0) \\ x, \text{ falls } x > 0 \land x-1 = 0 \end{cases}$$

4. Schritt: Vereinfachung der Bedingungen

$$= \begin{cases} I, \text{ falls } x = I \\ 1, \text{ falls } x = 0 \\ I, \text{ falls false} \lor x > 1 \\ x, \text{ falls } x > 0 \land x = 1 \end{cases}$$

Die dritte Bedingung ergibt sich durch Anwendung des Distributivgesetzes. Der erste sich ergebende Teilterm $x>0 \land x-1=I$ führt offensichtlich auf einen Widerspruch (hier mit false gekennzeichnet), da mit $x>0$ gilt: $x = 1,2,3,...$, also $x-1 = 0,1,2,3,...$ und $0,1,2,3,... \in N$ ungleich I sind.

5. Schritt: Zusammenfassen des 1. und 3. Falls

$$= \begin{cases} I, \text{ falls } x = I \lor x > 1 \\ 1, \text{ falls } x = 0 \\ x, \text{ falls } x = 1 \end{cases}$$

6. Schritt: Substitution [1/x] beim dritten Fall

$$= \begin{cases} I, \text{ falls } x = I \lor x > 1 \\ 1, \text{ falls } x = 0 \\ 1, \text{ falls } x = 1 \end{cases}$$

7. Schritt: Zusammenfassen des 2. und 3. Falls

$$= \begin{cases} I, \text{ falls } x = I \lor x > 1 \\ 1, \text{ falls } x \leq 1 \end{cases}$$

Lösung

2. Funktional und induktive Deutung der Funktion teile1

Teilaufgabe 1:
Die Funktion realisiert im Terminierungsfall offensichtlich die Division n/k.
Sie terminiert jedoch nur genau dann, wenn der Quotient n/k ganzzahlig ist, also n mod k = 0. Dies hängt damit zusammen, dass der Basisfall nur für n=0 erreicht wird. n wird aber nur dann durch wiederholte Subtraktion von k zu 0, wenn k ein Vielfaches von n ist also n mod k = 0.
(Bemerkung: Im Gegensatz zur Funktion fac ist die Funktion teile1 nicht total. Das bedeutet, dass, sie nicht für alle Eingaben aus ihrem Definitionsbereich einen Wert ≠ I liefert.)

Aufrufbäume:
teile1(6,2)—teile1(4,2)—teile1(2,2)—teile1(0,2)
teile1(8,3)—teile1(5,3)—teile1(2,3)—teile1(-1,3)—teile1(- 4,3)—. . .
Es ist teile1(8,3) = I, da die Funktion nicht terminiert.

Teilaufgabe 2:
Das Funktional lautet:
$\tau : (Z^{\perp} \times N^{\perp} \rightarrow Z^{\perp}) \rightarrow (Z^{\perp} \times N^{\perp} \rightarrow Z^{\perp})$

$$\tau[f]\,(n, k) = \begin{cases} \perp \text{ falls } n = \perp \vee k = \perp \vee k = 0 \\ 0 \text{ falls } n = 0 \wedge k > 0 \\ 1 + f(n - k, k) \text{ sonst} \end{cases}$$

Teilaufgabe 3:
Induktionsanfang: $\text{teile1}_0(n, k) = \perp$
Berechnung einiger Folgeglieder, um eine Induktionsannahme zu erhalten: (Hier wurden einige Zwischenschritte weggelassen, die jedoch anzugeben wären, wenn in einer Aufgabenstellung die Angabe des Lösungswegs gefordert ist.)

$$\text{teile1}_1(n, k) = \begin{cases} \perp, n = k \vee k = \perp \vee k = 0 \\ 0, n = 0 \wedge k > 0 \\ 1 + \text{teile1}_0(n - k, k) \text{ sonst} \end{cases}$$

(teile1$_0$*(n - k, k)* $= \perp$, + ist strikt)

$$= \ldots = \begin{cases} \perp, n = k \vee k = \perp \vee k = 0 \\ 0, n = 0 \wedge k > 0 \\ \perp \text{ sonst} \end{cases}$$

(zusammenfassen des 1. und 3. Falls)

$$= \ldots = \begin{cases} 0, n = 0 \wedge k > 0 \\ \\ \perp \text{ sonst} \end{cases}$$

$$\text{teile1}_2(n, k) = \begin{cases} \bot, \text{ falls } n = \bot \vee k = \bot \vee k = 0 \\ 0, \text{ falls } n = 0 \wedge k > 0 \\ 1 + \text{teile1}_1(n - k, k) \text{ sonst} \end{cases}$$

(+ ist strikt)

$$= \ldots = \begin{cases} \bot, n = k \vee k = \bot \vee k = 0 \\ 0, n = 0 \wedge k > 0 \\ 1+0, n - k = 0 \wedge k > 0 \\ \bot \text{ sonst} \end{cases}$$

(Zusammenfassen & Vereinfachen)

$$= \ldots = \begin{cases} 0, n = 0 \wedge k > 0 \\ 1, n = 1 * k \wedge k > 0 \\ \bot \text{ sonst} \end{cases}$$

$$\text{teile1}_3(n, k) = \begin{cases} \bot, \text{ falls } n = \bot \vee k = \bot \vee k = 0 \\ 0, \text{ falls } n = 0 \wedge k > 0 \\ 1 + \text{teile1}_2(n - k, k) \text{ sonst} \end{cases}$$

(+ ist strikt)

$$= \ldots = \begin{cases} \bot, n = k \vee k = \bot \vee k = 0 \\ 0, n = 0 \wedge k > 0 \\ 1+0, n - k = 0 \wedge k > 0 \\ 1+1, n - k = k \wedge k > 0 \\ \bot, \text{ sonst} \end{cases}$$

(Zusammenfassen & Vereinfachen)

$$= \ldots = \begin{cases} 0, n = 0 \wedge k > 0 \\ 1, n = 1 * k \wedge k > 0 \\ 2, n = 2 * k \wedge k > 0 \\ \bot, \text{ sonst} \end{cases}$$

® Annahme für teile1_i: (Dies ist die <u>Induktionsannahme</u> für festes i Î N)

$$\text{teile1}_i(n, k) = \begin{cases} m, n = m * k \wedge 0 \le m \le i-1 \wedge k > 0 \\ \\ \bot, \text{ sonst} \end{cases}$$

<u>Induktionsschluß:</u> (n ® n+1)teile1_{i+1} ergibt sich durch Einsetzen von teile1_i in τ.

$$\text{teile1}_{i+1}(n, k) = \begin{cases} \bot, n = \bot \vee k = \bot \vee k = 0 \\ 0, n = 0 \wedge k > 0 \\ 1 + \text{teile1}_i(n - k, k) \text{ sonst} \end{cases}$$

$$\begin{cases} \bot, n = \bot \vee k = \bot \vee k = 0 \\ 0, n = 0 \wedge k > 0 \end{cases}$$

$$=^{(I.V.)} \begin{cases} 1+ m, n - k = m * k \wedge 0 \leq m \leq i\text{-}1 \wedge k > 0 \\ 1+\perp \text{ sonst} \end{cases}$$

(Zusammenfassen & Vereinfachen)

$$= \begin{cases} 0, n = 0 \wedge k > 0 \\ 1+ m, n = (1+m) * k \wedge 0 \leq m \leq i\text{-}1 \wedge k > 0 \\ \perp \text{ sonst} \end{cases}$$

(setze 1+m:= = m´: Indexverschiebung)

$$= \begin{cases} 0, n = 0 \wedge k > 0 \\ m´, n = m´ * k \wedge 1 \leq m´ \leq (i+1)\text{-}1 \wedge k > 0 \\ \perp \text{ sonst} \end{cases}$$

(Zusammenfassen 1. und 2. Fall)

$$= \begin{cases} m´, n = m´ * k \wedge 0 \leq m´ \leq (i+1)\text{-}1 \wedge k > 0 \\ \\ \perp \text{ sonst} \end{cases}$$

m´ = m

$$= \begin{cases} m, n = m * k \wedge 0 \leq m \leq (i+1)\text{-}1 \wedge k > 0 \\ \\ \perp \text{ sonst} \end{cases}$$

was zu zeigen war.
Die geschlossene Darstellung für die induktive Deutung teile$1_\infty(n, k)$ der rekursiven Funktionsdeklaration teile1 erhält man, indem man für m ganz N als Wertebereich zuläßt. (Es wird i = ∞ gesetzt). So ergibt sich:

$$\text{teile}1_\infty(n, k) = \begin{cases} m, \text{ falls } n = m * k \wedge m \geq 0 \wedge k > 0 \\ \\ \perp \text{ sonst} \end{cases}$$

Lösung

3. Einbettung der Funktionsdeklaration zur Funktion exp

Teilaufgabe 1:
Bei einer repetitiv rekursiven Funktonsdeklaration muss in jedem Zweig einer Fallunterscheidung ein eventueller rekursiver Aufruf als letzte Aktion erfolgen. Bei der Deklaration von exp_embed_try ist im else-Fall die letzte Aktion jedoch die Auswertung des Terms x * exp_embed_try(x, y-1, x). Diese Auswertung kann erst stattfinden, nachdem exp_embed_try(x, y-1, x) ausgewertet wurde. Der rekursive Aufruf findet also zuerst statt.
® Es liegt keine repetitive Rekursion vor.
Teilaufgabe 2:
Die repetitiv rekursive Funktionsdeklaration zu exp_embed erhält man, indem man die bei der Deklaration von exp anfallenden Zwischenergebnisse (x*exp(x, y-1)) in einer dritten Variable z zwischenspeichert. In z steht dann im Terminierungsfall das Ergebnis.

Funktionsdeklaration für exp_embed:

fct exp_embed = (**nat** *x*, **nat** *y*, **nat** *z*) **nat**:

if y $\stackrel{?}{=}$ 0
then *z*
else exp_embed (*x*, *y* - 1, *z* · *x*) fi

Für z = 1 liefert exp_embed(x,y,z) das gleiche Ergebnis wie exp(x,y).

Geschlossene Darstellung für exp_embed:
Mit der Funktion g(x,y) aus dem Hinweis ergibt sich:

exp_embed(x,y,z)= z* g(x,y)

Lösung

4. Gofer: Klammergebirge

korrekt_(t,s) verschiebt sukzessive die öffnenden Klammern aus s nach t. Wird in s eine schließende Klammer gelesen, so wird diese aus s und eine öffnende Klammer aus t (sofern vorhanden) gelöscht. korrekt_(t,s) terminiert liefert True als Ergebnis, wenn am Ende s = t = [] ist. korrekt_(t,s) liefert False als Ergebnis, wenn t leer ist, obwohl s noch schließende Klammern enthält, oder wenn s leer ist, obwohl t noch öffnende Klammern enthält.

korrekt_(t,s) verallgemeinert das Problem der Berechnung der Korrektheitseigenschaft eines Klammergebirges: korrekt_(t,s) liefert genau dann True, wenn t++s ein korrektes Klammergebirge ist, wobei t keine schließenden Klammern enthält. Für den Spezialfall t = [] überprüft korrekt_(t,s) die Korrektheit von s , löst also das gestellte Problem.

----------- Programmbeginn ---------------

```
-- rest entfernt von einer Liste den Kopf und liefert die
-- Restliste zurück.
rest :: [a] → [a]
rest [x] = []
rest (x:xs) = xs

-- korrekt s liefert true gdw. korrekt_([],s) true liefert.
korrekt:: String → Bool
korrekt s = korrekt_ ([], s)

-- überprüft die Korrektheit von t++s
korrekt_ :: (String, String) → Bool

-- Für t=s=[] ist das Ergebnis True
korrekt_ ([], []) = True

-- Ist s leer, aber t nicht, so ist das Ergebnis False
```

korrekt_ (t, []) = False

-- Öffnende Klammern von s zu t addieren
korrekt_ (t,'(': s) = korrekt_ (t++"(", s)

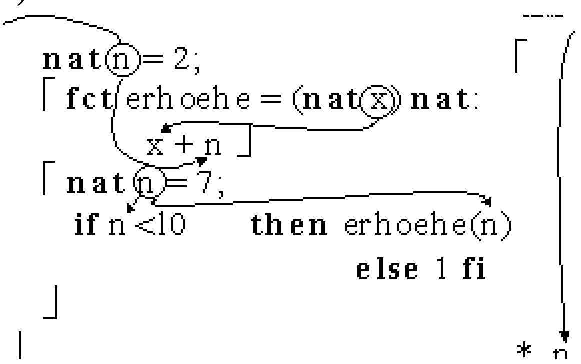
-- t leer, s enthält noch schließende Klammern: Erg.: False
korrekt_ ([], ')': s) = False

-- Öffnende Klammer in s: lösche erstes Element aus t und s
korrekt_ (t,')': s) = korrekt_ (rest t, s)
----------- Programmende ---------------

Lösung

5. Bindungen

a)



b)
Das Ergebnis des Abschnitts ist 2 * erhoehe(7) = 2 * (7+2) = 2*9 = 18.
