



Musterlösungen zum Übungsblatt 11

Lösung

1. Syntax der Funktions-Typvereinbarung in Gofer (1,5 Punkte)

Die Lösung soll nicht die vollständige Gofer-Syntax wiedergeben. Die vollständige Syntax findet man z.B. bei [Jon96-Appendix A].

Bevor die Syntax für Funktions-Typvereinbarungen in BNF oder als Diagramm entworfen wird, muss man die Struktur der Funktions-Typvereinbarung in Gofer analysieren. Dies geschieht anhand von einigen Beispielen.

f :: Int ® Int

Hier wird eine Funktion deklariert, die eine Integerzahl als Argument übergeben bekommt und eine Integerzahl als Ergebnis liefert.

g :: Int ® Bool ® Int

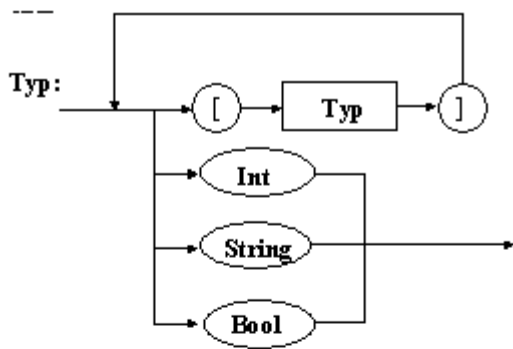
Diese Funktion bekommt eine Integerzahl und einen booleschen Wert als Argumente übergeben und gibt eine Integerzahl zurück.

h :: [String] ® [[Int]]

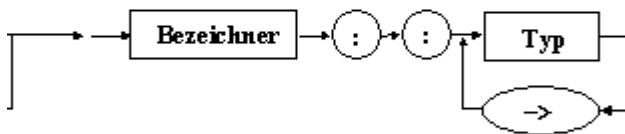
Diese Funktion bekommt eine Liste von Strings als Argument übergeben und liefert eine Liste mit Integerlisten zurück.

Die Analyse zeigt, dass zu Beginn einer Funktions-Typvereinbarung stets ein Bezeichner für einen Funktionsnamen steht, gefolgt von "::". Darauf folgen beliebig oft, jedoch wenigstens einmal unterschiedliche Typen oder Listen eines bestimmten Typs. Die einzelnen Typen werden durch \rightarrow abgetrennt. Auch Listen selbst können den Typ einer Liste sein.

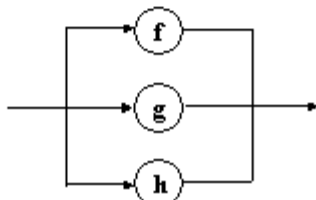
a)



Funktion:



Bezeichner:



b)

In Backus Naur Form lässt sich die wie oben beschriebene Funktions-Typvereinbarung wie folgt darstellen:

$$\begin{aligned} \langle \text{Bezeichner} \rangle &::= f \mid g \mid h \\ \langle \text{Typ} \rangle &::= \text{Int} \mid \text{Bool} \mid \text{String} \mid [\text{Typ}] \\ \langle \text{Funktion} \rangle &::= \langle \text{Bezeichner} \rangle :: \langle \text{Typ} \rangle \{ \rightarrow \langle \text{Typ} \rangle \}^* \end{aligned}$$

Lösung

2. Erste Gofeprogramme (2 Punkte)

1) Maximum

ohne Verwendung der bereitgestellten Funktionen

1. Variante

max1:: Int → Int → Int

max1 x y = if x > y then x else y

2. Variante

max2:: Int → Int → Int

max2 x y | x > y = x
| otherwise = y

3. Variante

In dieser Variante lässt sich für Datentypen, über denen eine Ordnung definiert ist (z.B. Zahlen, Character und Strings), das Maximum bestimmen

max3:: Ord a ⇒ a → a → a
max3 x y = if x > y then x else y

Veränderungen bei der Eingabe bei der 3. Variante

max3 4 5 Ergebnis: 5
max3 `a` `z` Ergebnis: z
max3 `abc` `abd` Ergebnis: abd

mit Verwendung der bereitgestellten Funktionen

4. Variante

max4:: Int → Int → Int
max4 x y = max x y

2) Fakultät

ohne Verwendung der bereitgestellten Funktionen

1. Variante

fact1:: Int → Int
fact1 x = if x == 0 then 1 else x * fact1 (x-1)

2. Variante

fact2:: Int → Int
fact2 x | x == 0 = 1
 | otherwise = x * fact1 (x-1)

3. Variante

fact3:: Int → Int
fact3 0 = 1
fact3 x = x * fact (x-1)

unter Verwendung der bereitgestellten Funktionen

4. Variante

[1..x] bezeichnet in Gofer eine Liste, die alle ganzen Zahlen von 1 bis x enthält.

fact4:: Int → Int
fact4 x = product [1..x]

3) Größter gemeinsamer Teiler

1. Variante

ggt1:: Int → Int → Int
ggt1 p q = if mod p q == 0 then q else ggt1 q (mod p q)

2. Variante

$\text{ggt2}:: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$

$\text{ggt2 } p \ q \quad | \text{ mod } p \ q == 0 \quad = q$
 $\quad | \text{ otherwise} \quad = \text{ggt2 } q \ (\text{mod } p \ q)$

4) Kleinste Zahl aus einer Liste mit 4 Elementen

1. Variante

Diese Funktion ist rekursiv definiert. Bei der Liste mit einem Element ist genau dieses Element das Kleinste. Bei einer Liste mit mehreren Elementen wird das erste Element mit dem Kleinsten der Restliste verglichen (rekursiver Abstieg in die Restliste) und je nach Ergebnis der Vergleichs, das erste Element zurückgegeben oder mit der Restliste weitergerechnet.

$\text{minlist1}:: [\text{Int}] \rightarrow \text{Int}$

$\text{minlist1 } [x] \quad = x$

$\text{minlist1 } (x:xs) \quad | x < \text{minlist1 } xs \quad = x$
 $\quad | \text{ otherwise} \quad = \text{minlist1 } xs$

2. Variante

In dieser Variante lässt sich für Datentypen, über denen eine Ordnung definiert ist (z.B. Zahlen, Character und Strings), das kleinste Element bestimmen.

$\text{minlist2}:: \text{Ord } a \Rightarrow [a] \rightarrow a$

$\text{minlist2 } [x] \quad = x$

$\text{minlist2 } (x:xs) \quad | x < \text{minlist2 } xs \quad = x$
 $\quad | \text{ otherwise} \quad = \text{minlist2 } xs$

3. Variante

Durch Verwendung der Hilfsfunktion `_minlist` wird das Laufzeitverhalten verbessert. Es wird nun nur das erste mit dem zweiten Element verglichen und das größere von beiden wird gestrichen. Es erfolgt nicht mehr ein rekursiver Abstieg in die gesamte Restliste beim Vergleich.

$\text{minlist3}:: \text{Ord } a \Rightarrow [a] \rightarrow a$

$\text{minlist3 } (x:xs) \quad = \text{minlist_ } x \ xs$

$\text{minlist_}:: \text{Ord } a \Rightarrow a \rightarrow [a] \rightarrow a$

$\text{minlist_ } x \ [] \quad = x$

$\text{minlist_ } x \ (y:ys) \quad | x < y \quad = \text{minlist_ } x \ ys$
 $\quad | \text{ otherwise} \quad = \text{minlist_ } y \ ys$

Lösung

3. Gofer: Ganzzahlige Quadratwurzel (2 Punkte)

1. Variante

```
wurzel:: Int → Int
wurzel n = quad n n
quad:: Int → Int → Int
quad n i = if i*i ≤ n then i else quad n (i-1)
```

2. Variante

```
wurzel:: Int → Int
wurzel n = quad2 n 0
quad2:: Int → Int → Int
quad2 n i | (i+1)*(i+1) ≤ n = quad2 n (i+1)
          | otherwise       = i
```

3. Variante

```
wurzel:: Int → Int
wurzel n = quad3 n 0
quad3:: Int → Int → Int
quad3 n i | (i+1)*(i+1) > n = i
          | otherwise       = quad3 n (i+1)
```

Lösung

4. Gofer: Rechenstruktur Keller (2 Punkte)

```
--Deklaration eines neuen Datentyps,
--a bel. Typ der --Kellerelemente
data Keller a = Create | Push (Keller a) a
```

```
--Formulierung der Wirkungsweise der Rechenstruktur
pop(Push k x) = k
top(Push k x) = x
empty(Create) = True
empty(Push k x) = False
```

```
--Angabe der Funktionalität der Kelleroperationen
pop :: Keller a → Keller a
top :: Keller a → a
empty :: Keller a → Bool
```

```
--Beispielhafte Eingaben
f1 = top(Push (Create) 3)
f2 = top(Push (Push (Create) 3) 4)
```

```
f3 = empty(pop(Push (Push (Create) 3) 4))
f4 = Push (pop(Push (Push (Create) 3) 4)) 2
f5 = pop(Push (pop(Push (Push (Create) 3) 4)) 2)
```

Lösung

5. Gofer: Mergesort (2,5 Punkte)

Teilaufgabe 1:

```
mmerge :: [Int] → [Int] → [Int]
mmerge [] ys = ys
mmerge xs [] = xs
mmerge (x:xs) (y:ys)
  | x ≤ y = x : mmerge xs (y:ys)
  | otherwise = y : mmerge (x:xs) ys
```

Teilaufgabe 2:

Variante 1:

```
mergesort :: [Int] → [Int]

-- Basisfall
mergesort [] = []
mergesort [x] = [x]

-- Rekursionsschritt
mergesort l = mmerge(mergesort (take (div (length l) 2) l)) (mergesort (drop (div (length l) 2) l))
```

Variante 2:

```
mergesort2 :: [Int] → [Int]

--Basisfall
mergesort [] = []
mergesort [x] = [x]

-- Rekursionsschritt
mergesort l = mmerge (mergesort2 l1) (mergesort2 l2)
  where l1 = take (div n 2) l
        l2 = drop (div n 2) l
        n = length l
```
