

Übungen zu Informatik I (Lösungsvorschlag)

Aufgabe 4-1 Quadratzahltest (keine Abgabe)

Um die gesuchte Funktion definieren zu können, benötigen wir eine Hilfsfunktion, die für zwei natürliche Zahlen n und k bestimmt, ob n das Quadrat einer Zahl $i \leq k$ ist.

$qzt = \mathbf{function} (n: \mathbf{nat}, k: \mathbf{nat}) \mathbf{bool}$:

```
result true, falls n das Quadrat einer natuerlichen Zahl j ≤ k ist, false sonst
  if k = 0 then n = k
  else (n = k2 ∨ qzt(n, k - 1))
```

Mit Hilfe dieser Funktion lässt sich nun die Funktion *istquadrat* wie folgt definieren:

$istquadrat = \mathbf{function} (n: \mathbf{nat}) \mathbf{bool}$:

```
result true, falls n Quadratzahl, false andernfalls
  qzt(n, n)
```

Aufgabe 4-2 Curry und Uncurry (keine Abgabe)

a) $curry_{\mathbf{nat}}$ ist eine höherstufige Funktion, die zu $f \in \mathbf{nat} \times \mathbf{nat} \rightarrow \mathbf{nat}$ die entsprechende curried Version $curry_{\mathbf{nat}}(f)$ bestimmt. Die Funktion $curry_{\mathbf{nat}}(f)$ ist definiert durch $curry_{\mathbf{nat}}(f)(a)(b) = f(a, b)$. In Pseudocode-Notation kann $curry_{\mathbf{nat}}$ wie folgt geschrieben werden:

```
currynat = function (f: nat × nat → nat) nat → nat → nat:
  result curried Version von f
  function (x: nat) nat → nat:
    function (y: nat) nat:
      f(x, y)
```

b) Die (polymorphe) Funktion *curry* unterscheidet sich nur in den (jetzt polymorphen) Typangaben von $curry_{\mathbf{nat}}$:

```
curry = function (f: α × β → γ) α → β → γ:
  result curried Version von f
  function (x: α) β → γ:
    function (y: β) γ:
      f(x, y)
```

Beachte, das man eine curried Version nicht nur für Funktionen des Typs $f \in \alpha \times \alpha \rightarrow \beta$ bestimmen kann.

Das polymorphe *uncurry* ist einfacher, da es mit einer einzigen Funktionsabstraktion auskommt:

```

uncurry = function (f:  $\alpha \rightarrow \beta \rightarrow \gamma$ )  $\alpha \times \beta \rightarrow \gamma$ :
  result uncurried Version von f
  function (x:  $\alpha$ , y:  $\beta$ )  $\gamma$ :
    f(x)(y)

```

- c) Mit Hilfe von *uncurry* lässt sich die Funktionsauswertung als *curried* Version der (polymorphen) Identitätsfunktion darstellen:

```
id = function (x:  $\alpha$ )  $\alpha$  : x
```

```
eval = uncurry (id).
```

Beachte, das *id* polymorph ist, also insbesondere auch den Typ $(\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \beta)$ hat. Aus diesem Grunde hat *uncurry(id)* den (gewünschten) Typ $(\alpha \rightarrow \beta) \times \alpha \rightarrow \beta$.

Aufgabe 4-3

Iteration

(keine Abgabe)

- a) Die Funktion *It* lässt sich wie folgt rekursiv definieren:

```
It = function (a:  $\alpha$ , f:  $\alpha \rightarrow \alpha$ ) nat  $\rightarrow \alpha$ :
```

result *Funktion, die fuer eine natuerliche Zahl n das Ergebnis von f n-mal auf a angewandt bestimmt*

```
function (n: nat)  $\alpha$ :
```

```
  if n = 0 then a
```

```
  else f(It(a, f)(n-1))
```

- b) Man kann die Funktion $(x, n) \mapsto x^n$ mit Hilfe von *It* wie folgt definieren.

```
pot = function (x: real, n: nat) real:
```

result x^n , berechnet mit Hilfe von *It*

```
It (1.0, function(y: real) real: x*y)(n)
```

Der Term *It*(1.0, **function**(y: **real**)**real**: x*y) definiert eine Funktion vom Typ **nat** \rightarrow **real**, die einer natürlichen Zahl *n* den Wert x^n zuordnet.

Aufgabe 4-4

Sitzordnung

(2 Punkte)

Die gesuchte Funktion kann man durch Rekursion nach der Anzahl der Paare, also nach *n*, definieren. Zunächst ist klar, dass für 1 Paar genau zwei mögliche Reihenfolgen existieren. Man muss sich nun überlegen, wie man die Anzahl der möglichen geeigneten Sitzordnungen für *n* Paare auf die Anzahl der entsprechenden Sitzordnungen für *n* - 1 Paare zurückführen kann.

Für eine feste Reihenfolge für *n* - 1 Paare hat das *n*-te Paar (als Paar, also zunächst noch ohne Beachtung der Reihenfolge zwischen Mann und Frau) genau *n* Plätze zur Auswahl, nämlich entweder an einem Ende der Reihe (2 Möglichkeiten), oder irgendwo „in der Mitte“ (*n* - 2 Möglichkeiten). Damit sind es insgesamt *n* Möglichkeiten. Wenn man jetzt auch noch beachtet, dass das Ehepaar untereinander auch noch je zwei mögliche Reihenfolgen hat, ergibt sich insgesamt, dass es für eine feste Reihenfolge von *n* - 1 Paaren das *n*-te Ehepaar $2 \cdot n$ Möglichkeiten hat, sich zu setzen. Diese Beobachtung führt direkt zur folgenden rekursiven Funktion:

```
sitzordnung = function (n = nat) nat:
```

```
pre n  $\geq$  1
```

result Anzahl der moeglichen verschiedenen Sitzordnungen gemaess Angabe

```
if n = 1 then 2
else 2*n*sitzordnung(n-1)
```

Aufgabe 4-5

Vollkommenheitstest

(4 Punkte)

Diese Funktion lässt sich nicht direkt durch Rekursion definieren. Wir definieren also zunächst eine Hilfsfunktion, die für zwei natürliche Zahlen $n \geq 1$ und $i < n$ die Summe der Teiler $\leq i$ von n berechnet.

```
vsumme = function (n: nat, i: nat) nat
pre n ≥ 1, i < n
```

result Summe der Teiler $\leq i$ von n

```
if i=0 then 0
else if n mod i = 0 then vsumme(n, i-1) + i
else vsumme(n, i-1)
```

Mit Hilfe der Funktion `vsumme` kann man nun eine Funktion `istvollkommen`, die für eine natürliche Zahl bestimmt, ob sie eine vollkommene Zahl ist, wie folgt definieren:

```
istvollkommen = function (n: nat) bool:
pre n ≥ 1
```

result `true`, falls n vollkommen, `false` sonst

```
vsumme(n, n) = 2*n
```

Aufgabe 4-6

Nullstellenberechnung

(3 + 3 Punkte)

a) Für eine stetige Funktion f definieren wir die Funktion `nstf` wie folgt:

```
nstf = function (a: real, b: real, ε: real) real:
pre a ≤ b, ε > 0, f(a)*f(b) ≤ 0
```

result Naeherungsweise Nullstelle der Funktion f im Intervall $[a, b]$ bis auf einen Fehler ϵ

```
if b-a < ε then a
else let
  c = (a+b)/2
  in
  if f(a)*f(c) ≤ 0 then nstf(a, c, ε)
  else nstf(c, b, ε)
end
```

b) Die Funktion `nst` sei wie folgt definiert:

```
nst = function (f: real → real) (real × real × real) → real:
pre f stetig
```

result Funktion, die fuer reelle Zahlen a, b und ϵ eine naeherungsweise Nullstelle von f im Intervall $[a, b]$ bis auf einen Fehler von ϵ berechnet

```
function (a: real, b: real, ε: real) real:
pre a ≤ b, ε > 0, f(a)*f(b) ≤ 0
if b-a < ε then a
```

```
else let
   $c = (a+b)/2$ 
in
  if  $f(a)*f(c) \leq 0$  then  $nst(f)(a, c, \epsilon)$ 
  else  $nst(f)(c, b, \epsilon)$ 
end
```