Fakultät IV: Elektrotechnik und Informatik
Informatik 1 WS 2001/02,
Jähnichen / Flik / Lechtchinsky / Neunast / Tutoren

Aufgabenblatt 5

letzte Aktualisierung: 22. November, 13:15

Ausgabe: *23.11.2001*
Abgabe: *3./4.12.2001* *Prozent:* *100*

Themen: *Funktionen höherer Ordnung*

1. Aufgabe (30 Prozent): Funktionen als Argumente und als Resultate von Funktionen

1.1. Funktionskomposition mit Auswertung (Tut) Implementiert eine Funktion `composeApply`, die zwei Funktionen und einen Wert als Argumente erhält und die Verkettung der Funktionen auf den Wert anwendet.

Simuliert die Funktionsaufrufe `composeApply(\\ x. x * x, \\ y. y, 4)` und `composeApply(\\ x. (2 * x) + 1, succ, 5)`

1.2. Die Funktion `flip` (5 Prozent) Schreibt die Funktion `flip`, die eine Funktion `f` mit zwei Parametern als Argument erhält und eine Funktion zurückliefert, die `f` mit vertauschten Parametern aufruft. Die Parameter und der Rückgabewert sollen vom Typ `nat` sein.

1.3. Die Funktion `twice` (10 Prozent) Schreibt die Funktion `twice`, die eine Funktion `f` mit einem Parameter als Argument erhält und eine Funktion zurückliefert, die `f` zweimal auf ihren Parameter anwendet. Der Parameter soll vom Typ `nat` sein.

1.4. Funktionsverschiebung (15 Prozent) Deklariert und definiert folgenden Funktionen in einer eigenen Struktur:

1. Deklariert und definiert eine Funktion `shiftX`, die eine Funktion `h` entlang der x-Achse um `deltaX` verschiebt.
2. Deklariert und definiert eine Funktion `mirrorY`, die eine Funktion `h` an der y-Achse spiegelt.
3. Deklariert und definiert eine Funktion `myCos` (Kosinus-Funktion) mit Hilfe bereits definierter Funktionen.
4. Deklariert und definiert eine Funktion `myCot` (Kotangens-Funktion) mit Hilfe bereits definierter Funktionen.

Hinweis: Benutzt nicht die Funktionen `cos` und `cot` aus der Struktur `Real`.

2. Aufgabe (20 Prozent): Funktionalitäten

2.1. **(Tut)** Macht Euch mit den Klammerregeln von Typausdrücken in OPAL vertraut.
Aus: P. Pepper: Funktionale Programmierung. 1999. S. 94:

- Die Tupelbildung '$\times$' bindet stärker als die Funktionsbildung '$\rightarrow$'.
- Die Tupelbildung '$\times$' ist assoziativ.
- Die Funktionsbildung '$\rightarrow$' bindet nach rechts, d.h., $A \rightarrow B \rightarrow C$ ist das Gleiche wie $A \rightarrow (B \rightarrow C)$.

2.2. **(Tut)** Gegeben sind folgende Funktionalitäten. Mit Ausnahme von `f` stammen alle Funktionen aus der Struktur `Nat`. Welche Funktionalität paßt zu folgender Funktionsanwendung `f(<,=)(+)(3)(4)`?

```
1. FUN f: (nat -> bool) -> (nat -> bool) -> (nat ** nat -> nat) ** nat ** nat
          -> nat

2. FUN f: (nat ** nat -> bool) ** (nat ** nat -> bool) ->
          (nat ** nat -> nat) -> nat -> nat -> nat

3. FUN f: (nat ** nat -> bool) ** (nat ** nat -> bool) ->
          (nat ** nat -> nat) ** (nat ** nat -> nat)

4. FUN f: (nat ** nat -> bool) -> (nat ** nat  -> bool) ** (nat ** nat -> nat)
           -> nat -> nat -> nat

5. FUN f: (nat ** nat -> bool) -> ((nat ** nat -> nat) ** nat ** nat) -> nat

6. FUN f: (nat ** nat -> bool) ** (nat ** nat -> nat) ->
          (nat ** nat -> nat) ** nat ** nat -> nat
```

2.3. **(20 Prozent)** Ihr habt Euch in **2.2** für eine Funktionalität entschieden. Welche der folgenden Ausdrücke sind bzgl. dieser korrekt? Von welchem Typ sind diese korrekten Ausdrücke?

```
1. f                              6. f(<=,>)(*)(33)(pow(8,6))
2. f(|=,=)                        7. f(>=,div)(\\ x, y. y-x)(2+10)
3. f(even?)(odd?)                 8. f(=,>)(mod)(double(2))
4. f(>,mod)                       9. f(|=,=)(min)
5. f(<,*)(+)("2"!)(succ(0))      10. f(even?,double)(/)(half(8))
```

3. Aufgabe (20 Prozent): Currying/Uncurrying

3.1. Funktion `curry` (Tut) Definiert die Funktion `curry`, die eine Funktion in Tupelnotation in eine Funktion in Curry-Notation umwandelt.

Die umzuwandelnde Funktion hat zwei Parameter vom Typ `nat`, der Rückgabetyp ist ebenfalls `nat`.

3.2. Funktion `uncurry` (5 Prozent) Definiert die Funktion `uncurry`, die das Gegenteil von `curry` tut. Sie wandelt Funktionen von Curry-Schreibweise in Tupelschreibweise um.

3.3. Currying anwenden (15 Prozent) Implementiert die Funktion `FUN myAdd: nat ** nat -> nat`, die zwei Zahlen addiert, in Lambda- und in Gleichungsnotation.

Definiert des weiteren die Funktion `myAddCurry` unter Verwendung der Funktion `curry`. Sie soll ebenfalls zwei Zahlen addieren, aber Currying verwenden.

Definiert die Funktion `plusFive`, die unter Verwendung der Funktion `myAddCurry` 5 zu ihrem Parameter addiert.

4. Aufgabe (30 Prozent): Funktionen höherer Ordnung auf Sequenzen

4.1. `map` und `filter` (Tut) Diskutiert die zwei Funktionale `map` und `filter`.

Implementiert die Funktionen `map` und `filter`.

Implementiert die Funktion `quicksort` unter Verwendung dieser Funktionale.

Implementiert eine Funktion `toUpper`, die eine Sequenz von `char` in Großbuchstaben umwandelt.

4.2. Anwendung von Funktionalen (15 Prozent) Verwendet für die Lösung dieser Aufgabe die Funktionale `map` und `filter`.

1. Definiert eine Funktion `addFive`, die auf alle Zahlen einer Sequenz 5 addiert. Benutzt hierfür die Funktion `plusFive`.
2. Definiert eine Funktion `toEven`, die alle Zahlen einer Sequenz gerade macht. Überlegt euch ein geeignetes Verfahren.
3. Definiert eine Funktion `splitOddEven`, die alle geraden und ungeraden Zahlen einer Sequenz trennt.
4. Benutzt die Funktionen `splitOddEven` und `myAddCurry`, um in einer Funktion `raiseEvenBy10` alle geraden Zahlen einer Sequenz um 10 erhöht zurückzuliefern.
5. Definiert die Funktion `countEven`, die die geraden Zahlen einer Sequenz zählt.

4.3. Die Funktion `filteredMap` (15 Prozent) Definiert die Funktion `filteredMap`, die der vordefinierten Funktion `map` entspricht. Die übergebene Funktion wird aber nur auf die Listenelemente angewendet, für die ein gegebenes Prädikat `true` liefert.

Benutzt die Funktion `filteredMap`, um eine Funktion `raise` zu definieren, die alle Zahlen einer Sequenz, für die ein gegebenes Prädikat gilt, um einen bestimmten Betrag erhöht. Benutzt dafür `myAddCurry`.