

Algorithmen und Programmierung I

WS 2004 / 2005

Übung 5 (20 + 5 Punkte)

Ausgabe: Mo. 22.11.2004

Abgabe: Mi. 1.12.2004, 10.00

Verspätet abgegebene Lösungen werden nicht akzeptiert.

Die Abgabe erfolgt in Zweiergruppen. Die Gruppe darf nur mit ausdrücklicher Genehmigung des Tutors gewechselt werden.

Abgegeben werden müssen

- Alle Lösungen einmal ausgedruckt. **Handschriftliche Lösungen werden nicht akzeptiert.**
- Programme, Testdaten, Testergebnisse in elektronischer Form in Absprache mit den Tutorinnen und Tutoren.

Aufgabe 4.5 von Übung 4 (5 P)

Aufgabe 5.1 (5 Punkte)

Wir setzen die Aufgabe 4.3 fort. Gesucht ist jetzt ein Addierer, der zwei gleich lange n -stellige positive Binärzahlen addiert (kein Vorzeichen, keine Komplementdarstellung, also nur positive Zahlen, die, wie im Beispiel zu sehen, als Folge fester Länge von Bits dargestellt werden¹). Argumente sind Listen mit je n binären Elementen I oder O , Wert soll die binäre Summe sein, die als Liste der Länge n ausgegeben wird. Der Übertrag der höchstwertigen (am weitesten links stehenden) Stelle wird ignoriert (siehe auch Skizze am Ende des Übungsblatts). Die Signatur ist:

`addBinN :: Reg -> Reg -> Reg`. Dabei ist `Reg` (für Register) die Typ-Aliasbezeichnung:
`type Reg = [Bit]`

Hinweis: Das niederwertigste Bit steht am Ende eines Registers. Man beginnt beim Addierschaltnetz beim niederwertigsten Bit x_0 und berücksichtigt den möglichen Übertrag in die jeweils nächste Stelle und zwar bis man beim höchstwertigen, am Anfang des Registers stehenden Bit (im Beispiel b_3) angekommen ist. Einen dort auftretenden Übertrag ignorieren wir vorläufig.

$$\begin{array}{rcl} \text{Beispiel: } & x_3x_2x_1x_0 & = & O & I & I & O & & = & 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ & y_3y_2y_1y_0 & + & O & I & O & I & & = & 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ & & & & & & & & & \\ & & & & & & & & & \\ & s_3s_2s_1s_0 & & I & O & I & I & & = & 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \end{array}$$

Im Beispiel tritt nur ein Übertrag von der dritten Position von rechts b_2 in die vierte Position b_3 auf. Unsere Registerdarstellung ist eine Liste. Listen arbeitet man bekanntlich leichter von links nach rechts ab. Deshalb empfiehlt sich in dieser Aufgabe, die Registerargumente (Listen) zunächst mit `reverse` umzudrehen und später mit dem Summenregister dasselbe zu tun.

Aufgabe 5.2 (5 Punkte)

Sind die folgenden Definitionen bzw. Ausdrücke korrekt oder nicht? (Begründung). Wenn korrekt, dann bitte auch den Typ angeben. Alle vorkommenden Funktionen (`bar`, `char`, ...) werden als bekannt vorausgesetzt.

a) `foo [] = []`
`foo [x:xs] = bar x : foo xs`

b) `[(1:(2:[]))]`

c) `[1]:[2,3]`

¹ In Rechnerstrukturen wird die Komplementdarstellung behandelt, die auch die negativen ganzen Zahlen umfasst.

- d) ['a', chr 98, 'c']
- e) ['a', '98', 'c']
- f) [[], []]
- f) [['1'], "Hallo"]
- h) ([1,2,3], "Foo")
- i) map f . map g
- j) map map
- k) map . map
- l) map (map quadrat) [[1,2], [3,4,5]] -- falls korrekt Wert angeben.

m) Welche Möglichkeiten, den Typ der Funktionsverkettung (.) auszudrücken, sind korrekt?

- (1) (.) :: (a -> b) -> (b -> c) -> (a -> c)
- (2) (.) :: b -> c -> (b -> a) -> (a -> c)
- (3) (.) :: (b -> c) -> (b -> a) -> a -> c
- (4) (.) :: (b -> c) -> (a -> b) -> (a -> c)
- (5) (.) :: (c -> a) -> (b -> c) -> b -> a
- (6) (.) :: b -> c -> b -> a -> (a -> c)
- (7) (.) :: a -> c -> b -> a -> a -> c
- (8) (.) :: (a -> b) -> (c -> a) -> (c -> b)
- (9) (.) :: (c -> a) -> (a -> b) -> (c -> b)
- (10) keine ist richtig

Aufgabe 5.3 (5 Punkte)

a) Schreiben Sie eine Funktion

```
join :: [(a,b)] -> [(a,c)] -> [(a,b,c)]
```

die folgendes leistet: Zwei Listen xs , ys von Tupeln werden auf eine Liste zs von Tripeln abgebildet, die so definiert ist: $(x, y, z) \in zs \Leftrightarrow (x, y) \in xs \wedge (x, z) \in y$

Verwenden Sie die ZF-Notation.

b) Gesucht ist eine Verallgemeinerung der Funktion **wTab** aus der VL, die für eine Liste von zweistelligen Operatoren die Wahrheitstabellen ausgibt. Die Funktion hat folgende Signatur:

```
truthTab :: [(String, (Bool -> Bool -> Bool))] -> [(String, [(Bool, Bool, Bool)])]
```

Beispiel :

```
truthTabs [("and", (&&)), ("or", (||))]
```

```
[("and", [(False, False, False), (False, True, False), (True, False, False), (True, True, True)]), ("or", [(False, False, False), (False, True, True), (True, False, True), (True, True, True)])]
```

(Um ein schöneres Layout kümmern wir uns später...)

Aufgabe 5.4 (5 Punkte)

a) Unter welchen Bedingungen gilt

```
[x | x <- xs, y <- ys] == [x | y <- ys, x <- xs]
```

b) Die Filterfunktion lässt sich so definieren:

```
filter = concat . map bx
      where bx ...
```

Gesucht ist die Definition von **bx**.

c) Gesucht ist eine Funktion, die für alle natürlich Zahlen $0 < n < 1000$ die Tripel (x,y,z) liefert, für die $x^2 + y^2 = z^2$.

Schaltbild eines Addier-Schaltnetzes aus 4 Volladdierern für zwei 4-Bit-Zahlen (zu Aufgabe 5.1)

