DEPENDABLE SYSTEMS AND SOFTWARE
Fachrichtung 6.2 — Informatik
Prof. Dr.-Ing. Holger Hermanns
Dipl.-Inform. Lijun Zhang

# Übungsblatt 5 (Programmierung I)

Lesen Sie im Skript: Kapitel 4 & 5

**Aufgabe 5.1: (Produkt)**
Schreiben Sie mit *foldl* eine Prozedur $prod : int\ list \to int$, die das Produkt der Elemente einer Liste liefert. Für die leere Liste soll 1 geliefert werden.

**Aufgabe 5.2: (foldr)**
Deklarieren Sie die Faltungsprozedur *foldr* mithilfe der Faltungsprozedur *foldl*. Verwenden Sie dabei bis auf *foldl* keine Hilfsprozedur.

**Aufgabe 5.3: (GgT)**
Sei eine Prozdeur $gcd : int * int \to int$ gegeben, die den größten gemeinsamen Teiler zweier positiver Zahlen bestimmt. Schreiben Sie mit $foldl$ eine Prozedur $gcdL : int\ list \to int$, die den größten gemeinsamen Teiler der Elemente einer nichtleeren Liste von positiven Zahlen bestimmt. Beispielsweise soll *gcdL* [15,75,20] das Ergebnis 5 liefern.

**Aufgabe 5.4: (Count)**
Schreiben Sie mithilfe von $foldl$ eine polymorphe Prozedur $count : ''a \to ''a\ list \to int$, die zählt, wie oft ein Wert in einer Liste als Element vorkommt. Beispielsweise soll $count\ 5\ [2,5,3,5] = 2$ gelten.

**Aufgabe 5.5: (Enum)**
Die Dezimaldarstellung einer natürlichen Zahl ist die Liste ihrer Ziffern. Beispielsweise hat 7856 die Dezimaldarstellung [7,8,5,6].

(a) Deklarieren Sie eine Prozedur $dec : int \to int\ list$, die die Dezimaldarstellung einer natürlichen Zahl liefert. Verwenden Sie div und mod.

(b) Deklarieren Sie mithilfe von $foldl$ eine Prozedur $num : int\ list \to int$, die zu einer Dezimaldarstellung die dargestellte Zahl liefert.

**Aufgabe 5.6: (foldl und foldr)**
Machen Sie sich klar, dass $foldr$ mithilfe von $foldl$ und Cons deklariert werden kann. Warum kann umgekehrt $foldl$ nicht mithilfe von $foldr$ und Cons deklariert werden?

**Aufgabe 5.7: (Last)**
Deklarieren Sie eine Prozedur $last : \alpha\ list \to \alpha$, die das Element in der letzten Position einer iste liefert. Wenn die Liste leer ist, soll die Ausnahme *Empty* geworfen werden.

**Aufgabe 5.8: (Take und Drop)**
Schreiben Sie zwei polymorphe Prozeduren *take* und *drop*, die gemäß $\alpha\ list * int \to \alpha\ list$ getypt sind und die folgende Spezifikation erfüllen:

(a) $take(xs,\ n)$ liefert die ersten $n$ Elemente der Liste $xs$. Falls $n < 0$ oder $|xs| < n$ gilt, soll die Ausnahme *Subscript* geworfen werden.

(b) $drop(xs,\ n)$ liefert die Liste, die man aus $xs$ erhält, wenn man die ersten $n$ Elemente weglässt. Falls $n < 0$ oder $|xs| < n$ gilt, soll die Ausnahme *Subscript* geworfen werden.

Hinweis: Verwenden Sie *orelse* und die Prozeduren *hd*, *tl* und *null*.

**Aufgabe 5.9: (Muster)**
Welche der Muster der 4 Regeln der Prozedur *test* treffen den Wert [(2,5), (7,3)]? An welche Werte werden die Variablen der Muster dabei gebunden?

```
fun test [] = 0 | test [(x,y)] = x+y
                | test [(x,5), (7,y)] = x*y
                | test (_::_::ps) = test ps
```

**Aufgabe 5.10: (Muster)**
Entscheiden Sie für jeden der Werte $(7, [1], (3, 3))$ und $([1, 2], [3, 4, 5], (11, 3))$, ob er das Muster (x, y :: _ :: z, (u,3)) trifft. Geben Sie bei einem Treffer die Bindungen für die Veriablen des Musters an.

**Aufgabe 5.11: (String)**
Schreiben Sie eine Prozedur $toInt : string \to int$, die zu einem String, der nur aus den Ziffern besteht, die durch ihn dargestellte Zahl liefert (z.B. $toInt\ ''123'' = 123$). Falls der String Zeichen enthält, die keine Ziffern sind, soll die Ausname *Domain* geworfen werden.

**Aufgabe 5.12: (sorted)**
Schreiben Sie eine Prozedur $sorted : int\ list \to bool$, die testet, ob eine Liste aufsteigend sortiert ist. Verwenden Sie dabei keine Hilfsprozedur.

**Aufgabe 5.13: (Perm)**
Schreiben Sie eine Prozedur $perm\ :\ int\ list \to int\ list \to bool$, die testet, ob zwei Listen bis auf die Anordnung ihrer Elemente gleich sind. Verwenden Sie dabei die Prozedur *isort* und die Tatsache, dass *int list* ein Typ mit Gleichheit ist.

**Aufgabe 5.14: (issort)**
Schreiben Sie eine Prozedur $issort :\ int\ list \to int\ list$, die eine Liste sortiert und dabei Mehrfachauftreten von Elementen eliminiert. Beispielsweise soll für [3,1,3,1,0] die Liste [0,1,3] geliefert werden.

**Aufgabe 5.15: (Partition)**
Deklarieren Sie eine Prozedur $partition : int \to int\ list \to int\ list * int\ list$ die zu einer Zahl $x$ und einer Liste $xs$ zwei Listen $us$ und $vs$ wie folgt liefert:

(a) $sort(us@vs) = sort\ xs$.

(b) Alle Elemente von $us$ sind echt kleiner als $x$ und alle Elemente von $vs$ sind größer gleich $x$.

Schreiben Sie *partition* mit $foldl$. Orientieren Sie sich dabei an der Prozedur *split*.

**Aufgabe 5.16: (Quicksort)**

Quicksort ist ein klassischer Sortieralgorithmus, der die zu sortierende Liste gemäß der Prozedur *partition* in zwei Teillisten zerlegt und ihre durch Rekursion berechneten Sortierungen mit Konkatenation wieder zusammenfügt. Die Essenz von Quicksort wird durch die bedingte Gleichung

$$sort(x :: xr) = (sort\ us)@[x]@(sort\ vs) \quad falls\ (us, vs) = partitition\ x\ xr$$

beschrieben. Deklarieren Sie eine Prozedur *qsort*, die Listen über int gemäß Quicksort sortiert.

**Aufgabe 5.17: (Ganzzahlige Quadratwurzel)**
Oft erspart die Verwendung höherstufiger Prozeduren die explizite Einführung von Akkumulatorargumenten. Betrachten Sie dazu die Berechnung ganzzahliger Quadratwurzeln mit der in §3.2 besprochenen Prozedur

```
fun sqrt (x:int) = first 1 (fn (k:int) => k*k>x) - 1
val sqrt : int->int
```

Deklarieren Sie eine semantisch äquivalente Prozedur, die keine höherstufigen Prozeduren verwendet.