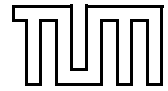
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



WS 11/12

Einführung in die Informatik II **Übungsblatt 6**
Univ.-Prof. Dr. Andrey Rybalchenko, M.Sc. Ruslán Ledesma Garza 03.12.2011

Dieses Blatt behandelt Kapitel 4.3 bis 4.6 aus dem Buch zur Vorlesung. Lesen Sie diese Kapitel!

Aufgabe 4.5 (Member) Schreiben Sie eine polymorphe Prozedur `member : 'a → 'a list → bool` die testet, ob ein Wert als Element in einer Liste vorkommt.

Lösungsvorschlag 4.5:

```
fun member x nil     = false
  | member x (y::ys) = if x=y then true else member x ys
```

Aufgabe 4.7 (Member) Schreiben Sie mithilfe von `List.exists` eine Prozedur `member : 'a → 'a list → bool`, die testet, ob ein Wert als Element in einer Liste vorkommt.

Lösungsvorschlag 4.7:

```
fun member x = List.exists (fn y ⇒ x=y)
```

Aufgabe 4.8 (Product) Schreiben Sie mit `foldl` eine Prozedur `prod:int list→int`, die das Produkt der Elemente einer Liste liefert. Für die leere Liste soll 1 geliefert werden.

Lösungsvorschlag 4.8:

```
fun prod xs = foldl (fn (x,y) ⇒ x*y) 1 xs
```

Aufgabe 4.9 (Member) Schreiben Sie mithilfe von `foldl` eine polymorphe Prozedur `member : 'a→'a list →bool`, die testet, ob ein Wert als Element in einer Liste vorkommt.

Lösungsvorschlag 4.9:

```
fun member x = foldl (fn (y,z) ⇒ z orelse x=y) false
```

Aufgabe 4.10 (Count) Schreiben Sie mithilfe von `foldl` eine polymorphe Prozedur `count: 'a→ 'a list→ int`, die zählt, wie oft ein Wert in einer Liste als Element vorkommt. Beispielsweise soll `count 5 [2, 5, 3, 5] = 2` gelten.

Lösungsvorschlag 4.10:

```
fun count x = foldl (fn (y,z) ⇒ if x=y then z+1 else z) 0
```

Aufgabe 4.11 (Dezimaldarstellung) 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→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→int`, die zu einer Dezimaldarstellung die dargestellte Zahl liefert.

Lösungsvorschlag 4.11:

```
(a)
    fun dec n =
      let fun dec_ n = if n < 10 then [n] else n mod 10 :: dec_ (n div 10)
    in List.rev (dec_ (abs n)) end;

    oder

    fun dec x = if x < 10 then [abs x] else dec (x div 10) @ [abs (x mod 10)];

(b)
    fun num xs = foldl (fn (x, s) ⇒ s * 10 + x) 0 xs;
```

Aufgabe 4.12* Deklarieren Sie die Faltungsprozedur `foldr` mithilfe der Faltungsprozedur `foldl`. Verwenden Sie zweimal `foldr`.

Lösungsvorschlag 4.12*:

```
fun foldr f xs x = foldl f x (foldl op:: [] xs);
```

Aufgabe 4.13* Deklarieren Sie die Faltungsprozedur `foldl` mit hilfe der Faltungsprozedur `foldr`. Gehen Sie wie folgt vor:

a) Deklarieren Sie `append` mit hilfe von `foldr`.

b) Deklarieren Sie `rev` mit hilfe von `foldr` und `append`.

c) Deklarieren Sie `foldl` mit hilfe von `foldr` und `rev`.

d) Deklarieren Sie `foldl` nur mit hilfe von `foldr`.

Lösungsvorschlag 4.13*:

```
fun append x y = foldr (fn (x,xs) ⇒ x::xs) y x
fun rev x = foldr (fn (x,xs) ⇒ append xs [x]) nil x
fun foldl' f x xs = foldr f (rev xs) x
fun foldl' f x xs =
  let fun append x y = foldr (fn (x,xs) ⇒ x::xs) y x
      fun rev x = foldr (fn (x,xs) ⇒ append xs [x]) nil x
  in  foldr f x (rev xs)
end
```

Aufgabe 4.14* (Challenge) Die Rückführung von `foldl` auf `foldr` gelingt auf besonders elegante Weise, wenn man `foldr` auf einen prozeduralen Startwert anwendet.

a) Finden Sie zwei Abstraktionen e und e', die die folgende Gleichung erfüllen: `foldl f s xs = (foldr e e' xs) s`

b) Abstraktionen sollen keine Hilfsprozeduren verwenden. Machen Sie sich klar, dass `foldr e e' xs` eine zu `fn s ⇒ foldl f s xs` äquivalente Prozedur liefern muss. Daher ergibt sich e' aus dem Spezialfall `xs = nil` und e aus dem Spezialfall `xs = x :: xr`.

c) Überzeugen Sie sich davon, dass Ihre Abstraktionen die obige Gleichung auch dann erfüllen, wenn `foldl` mit `foldr` vertauscht wird.

d) Deklarieren Sie die Prozedur `foldl` gemäß der obigen Gleichung mithilfe der Prozedur `foldr`.

Lösungsvorschlag 4.14*:

```
fun foldl f x xs = foldr (fn (x,y) ⇒ fn z ⇒ y (f (x, z))) (fn x ⇒ x) xs x
```

Aufgabe 4.15 (Last) Deklarieren Sie eine Prozedur `last : 'a list → 'a` die das Element an der letzten Position einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme `Empty` geworfen werden.

Lösungsvorschlag 4.15:

```
fun last nil = raise Empty
    | last [x] = x
    | last (x :: xs) = last xs
```

Aufgabe 4.16 (Max) Schreiben Sie mit `foldl` eine Prozedur `max : int list → int`, die das größte Element einer Liste liefert. Wenn die Liste leer ist, soll die Ausnahme `Empty` geworfen werden. Verwenden Sie die vordefinierte Prozedur `Int.max : int * int → int`.

Lösungsvorschlag 4.16:

```
fun foldl f s nil = s
    | fold f s (x :: xs) = foldl f (f (x, s)) xs;

fun max nil = raise Empty
    | max (x :: xs) = foldl Int.max
x xs;
```

Aufgabe 4.17 (Take und Drop) Schreiben Sie zwei polymorphe Prozeduren `take` und `drop`, die gemäß `'a list * int → 'a 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`.

Lösungsvorschlag 4.17:

```
fun take (xs, n) =
  if n= 0 then []
  else if n < 0 orelse null xs then raise Subscript
  else hd xs :: take (tl xs, n - 1);

  fun drop (xs, n) =
    if n = 0 then xs
    else if n < 0 orelse null xs then raise Subscript
    else drop (tl xs, n - 1);
```

Aufgabe 4.18

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

Welche der Muster der vier Regeln der Prozedur `test` treffen den Wert `[(2, 5), (7, 3)]`? An welche Werte werden die Variablen der Muster dabei gebunden? Welche Zeile wird ausgeführt?

Lösungsvorschlag 4.18:

a) Zeile 3, `x:=2`, `y:=3` und Zeile 4, `ps:=[]`

b) Zeile 3.

Aufgabe 4.19 Entscheiden Sie für jeden der folgenden Werte, ob er das Muster `(x, y :: _ :: z, (u,3))` trifft. Geben Sie bei einem Treffer die Bindungen für die Variablen des Musters an.

a) `(7, [1], (3,3))`

b) `([1,2], [3,4,5], (11,3))`

Lösungsvorschlag 4.19:

a) nein

b) `x:=[1,2], y:=3, z:=[5], u:=11`

Aufgabe 4.20 Deklarieren Sie eine zu `or` äquivalente Prozedur, die mit disjunkten Regeln formuliert ist. Verwenden Sie dabei kein Konditional.

Lösungsvorschlag 4.20:

```
fun or' (false, y) = y
  | or' (true , _) = true
```