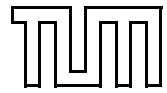
TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK



WS 11/12
Einführung in die Informatik II **Übungsblatt 7**
Univ.-Prof. Dr. Andrey Rybalchenko, M.Sc. Ruslán Ledesma Garza 13.12.2010

Dieses Blatt behandelt Kapitel 6.1 – 6.6, 15.1 – 15.3, und 15.8 aus dem Buch zur Vorlesung. Lesen Sie diese Kapitel!

Aufgabe 6.2 Deklarieren Sie eine Prozedur `scale : real → shape → shape`, die ein Objekt gemäß einem Faktor skaliert (d.h. vergrößert oder verkleinert). Beispielsweise soll `scale 0.5 (Square 3.0) = Square 1.5` gelten.

Lösungsvorschlag 6.2:

```
fun scale s (Circle r)          = Circle (s*r)
  | scale s (Square r)          = Square (s*r)
  | scale s (Triangle (a,b,c)) = Triangle (s*a, s*b, s*c)
```

Aufgabe 6.3 In §4.6.3 haben Sie gelernt, dass das Konditional eine abgeleitete Form ist. Wissen Sie noch, auf welchen Ausdruck der Kernsprache ein Konditional **if** `e1` **then** `e2` **else** `e3` reduziert?

Lösungsvorschlag 6.3:

```
(fn true ⇒ e2 | false ⇒ e3) e1
```

Aufgabe 6.5 Deklarieren Sie eine Prozedur `vars : exp → var list`, die zu einem Ausdruck eine Liste liefert, die die in dem Ausdruck vorkommenden Variablen enthält. Orientieren Sie sich an der Prozedur `subexps`.

Lösungsvorschlag 6.5:

```
fun vars e =
  case e of
      A (e1,e2) ⇒ vars e1 @ vars e2
    | M (e1,e2) ⇒ vars e1 @ vars e2
    | V s       ⇒ [s]
    | _         ⇒ []
```

Aufgabe 6.7

Deklarieren Sie eine Prozedur `check : exp → exp → bool`, die für zwei Ausdrücke e und e' testet, ob e ein Teilausdruck von e' ist.

Lösungsvorschlag 6.7:

```
fun check e e' =
  e=e' orelse
    case e of
        A (e1,e2) ⇒ check e1 e' orelse check e2 e'
      | M (e1,e2) ⇒ check e1 e' orelse check e2 e'
      | s         ⇒ false
```

Aufgabe 6.8 Schreiben Sie eine Prozedur `instantiate : env → exp → exp`, die zu einer Umgebung V und einem Ausdruck e den Ausdruck liefert, den man aus e erhält, indem man die in e vorkommenden Variablen gemäß V durch Konstanten ersetzt. Beispielsweise soll für die oben deklarierte Umgebung env und den Ausdruck `A(V "x", V "y")` der Ausdruck `A(C 5, C 3)` geliefert werden. Orientieren Sie sich an der Prozedur eval.

Lösungsvorschlag 6.8:

```
fun instantiate env (C c)      = C c
  | instantiate env (V v)      = C (env v)
  | instantiate env (A (e,e')) = A (instantiate env e, instantiate env e')
  | instantiate env (M (e,e')) = M (instantiate env e, instantiate env e')
```

Aufgabe 6.10* (Konstruktordarstellung natürlicher Zahlen) In dieser Aufgabe stellen wir die natürlichen Zahlen mit den Werten des Konstruktortyps

datatype `nat = O | S` **of** `nat`

dar: $0 \mapsto$ `O`, $1 \mapsto$ `S O`, $2 \mapsto$ `S (S O)`, $3 \mapsto$ `S (S (S O))`, und so weiter.

a) Deklarieren Sie eine Prozedur `code : int → nat`, die die Darstellung einer natürlichen Zahl liefert.

b) Deklarieren Sie eine Prozedur `decode : nat → int`, sodass `decode (code n) = n` für alle $n \in \mathbb{N}$ gilt.

c) Deklarieren Sie für `nat` kaskadierte Prozeduren `add`, `mul`, `sub` und `less`, die den Operationen $+$, $*$, $-$ und $<$ für natürliche Zahlen entsprechen. Verwenden Sie dabei keine Operationen für `int`. Werfen Sie eine Ausnahme, falls es kein sinnvolles Ergebnis gibt.

Lösungsvorschlag 6.10*:

a)
```
fun code 0 = O
  | code i = S (code (i-1))
```

b)
```
fun decode O     = 0
  | decode (S x) = 1 + decode x
```

c)
```
fun add a O     = a
  | add a (S x) = add (S a) x
```

d)
```
fun mul a O     = O
  | mul a (S x) = add a (mul a x)
```

e)
```
fun sub a     O       = a
  | sub O     _       = raise Empty
  | sub (S x) (S x') = sub x x'
```

f)
```
fun less _     O       = false
  | less O     _       = true
  | less (S x) (S x') = less x x'
```

Aufgabe 6.12

Schreiben Sie eine Prozedur `test : int → bool`, die testet, ob das Quadrat einer ganzen Zahl im darstellbaren Zahlenbereich liegt.

Lösungsvorschlag 6.12:

```
fun test i = (i*i; false) handle Overflow ⇒ true
```

Aufgabe 6.13

Führen Sie zweistellige Sequenzialisierungen (e1; e2) auf Abstraktionen und Applikationen zurück.

Lösungsvorschlag 6.13:

```
(fn a ⇒ e2) e1
```

Aufgabe 6.14

```
exception Double

fun mask compare p =
  case compare p of
      EQUAL ⇒ raise Double
    | v     ⇒ v

fun testDouble compare xs =
  ( List.sort (mask compare) xs;
    false
  ) handle Double ⇒ true
```

Schreiben Sie die Prozedur `testDouble` so um, dass die Ausnahme `Double` und die Hilfsprozedur `mask` mithilfe eines Let-Ausdrucks lokal deklariert werden.

Lösungsvorschlag 6.14:

```
fun testDouble compare xs =
  let exception Double
      fun mask compare p =
        case compare p of
            EQUAL ⇒ raise Double
          | v     ⇒ v
  in
  ( List.sort (mask compare) xs;
    false
  ) handle Double ⇒ true
  end
```

Aufgabe 6.16 Schreiben Sie eine Prozedur `append : 'a mylist → 'a mylist → 'a mylist` die zwei gemäß des Typkonstruktors mylist (**datatype** `'a mylist = Nil | Cons of 'a * 'a mylist`) dargestellte Listen konkateniert.

Lösungsvorschlag 6.16:

```
fun append a            Nil = a
  | append Nil          b   = b
  | append (Cons (h,t)) b   = Cons (h, append t b)
```

Aufgabe 15.6

Vervollständigen Sie die Deklaration `val (count, inc, reset) =` so, dass sie einen eingekapselten Zähler mit dem Anfangswert 0 und drei Prozeduren wie folgt liefert:

- `count : unit → int` liefert den Wert des Zählers.
- `inc : unit → unit` erhöht den Wert des Zählers um 1.
- `reset : unit → unit` setzt den Zähler auf 0 zurück.

Lösungsvorschlag 15.6:

```
val (count, inc, reset) =
  let
    val r = ref 0
  in
    (fn () ⇒ !r , fn () ⇒ r:= !r+1 , fn () ⇒ r:= 0 )
  end

(*usage:*)

> val count = fn : unit → int
val inc = fn : unit → unit
val reset = fn : unit → unit
val it = () : unit
> inc();
val it = () : unit
> inc();
val it = () : unit
> count();
val it = 2 : int
> reset();
val it = () : unit
> count();
val it = 0 : int
```

Aufgabe 15.21

Schreiben Sie eine Prozedur `length : 'a list → int`, die mit einer Schleife die Länge einer Liste bestimmt. Verwenden Sie die vordeklarieren Prozeduren `null`, `tl` und `not`.

Lösungsvorschlag 15.21:

```
fun length l_ =
  let
    val len = ref 0
    val l = ref l_
  in
    while not (null (!l)) do (
      l := tl (!l);
      len := !len + 1
     );
     !len
  end
```