Programmierung WS 2002 / 03:
Musterlösung zum 5. Übungsblatt

Prof. Dr. Gert Smolka, Dipl.-Inform. Thorsten Brunklaus

Aufgabe 5.1: Darstellung von Zahlen $(20 = 5 * 4)$

(a)
```
fun add (N, y)  = y
  | add(S x, y) = S(add(x, y))
```

(b)
```
fun mul (N, y)   = N
  | mul (S x, y) = add(y, mul(x, y))
```

(c)
```
fun to i = if i<0
           then raise Subscript
           else if i=0 then N
           else S (to (i - 1))

fun from N     = 0
  | from (S x) = 1 + (from x)
```

(d)
```
datatype integer = NEG of nat | NAT of nat
```

(e)
```
(*  n = NAT (to n) *)
(* -n = NEG (to (~n - 1)) *)
fun from' (NAT x) = from x
  | from' (NEG x) = (~(from x) - 1)
```

Aufgabe 5.2: B-Bäume $(12 = 3 * 4)$

(a)
```
fun bsum (L x)          = x
  | bsum (N (x, t, t')) = x + (bsum t) + (bsum t')
```

(b)
```
fun bmap f (L x)          = L (f x)
  | bmap f (N (x, t, t')) = N (f x, bmap f t, bmap f t')
```

(c)
```
fun bmember p (L x)          = p x
  | bmember p (N (x, t, t')) =
  (p x) orelse (bmember p t) orelse (bmember p t')
```

Aufgabe 5.3: L-Bäume $(12 = 3 * 4)$

(a)
```
fun lsum (T (x, ls)) = foldl (fn (x, e) => (lsum x) + e) x ls
```

(b)
```
fun lmap f (T (x, ls)) = T(f x, map (lmap f) ls)
```

(c)
```
fun lmember p (T (x, ls)) =
  foldl (fn (x, e) => e orelse (lmember p x)) (p x) ls
```

Aufgabe 5.4: Spiegeln von Bäumen $(8 = 2 * 4)$

(a)
```
fun bmirror (N (x, t, t')) = N(x, bmirror t', bmirror t)
  | bmirror  t              = t
```

(b)
```
fun lmirror (T(x,ts)) = T(x, rev(map lmirror ts))
```

Aufgabe 5.5: Schneller Test auf Doppelauftreten $(12 = 1 + 3 + 4 + 4)$

```
fun test xs =
  let
    exception Double
    fun order (m, n) = if m<n then LESS
                       else if m>n then GREATER
                            else raise Double
    val dsort = Listsort.sort order
  in
    (dsort xs; false) handle Double => true
  end
```

Aufgabe 5.6: Programmieren mit Ausnahmen und Optionen $(10 = 4 + 6)$

(a)
```
exception Found of int

fun findi p (L x)         = if (p x) then raise (Found x) else ()
  | findi p (N(x, t, t')) = (findi p (L x); findi p t; findi p t')
```

(b)
```
fun find p t =
  let
    exception Found of 'a
    fun find' p (L x)         = if (p x) then raise (Found x) else ()
      | find' p (N(x, t, t')) = (find' p (L x); find' p t; find' p t')
  in
    ((find' p t; NONE) handle (Found x) => SOME x)
  end
```

Aufgabe 5.7: Symbolisches Differenzieren $(26 = 2 + 10 + 8 + 6)$

(a)
```
val u = Add(Add(Add(Pow(X,3), Mul(Con 3, Pow(X,2))), X), Con 2)
```

(b)
```
fun derive (Con n)    = Con 0
  | derive X          = Con 1
  | derive (Add(u,v)) = Add(derive u, derive v)
  | derive (Mul(u,v)) = Add(Mul(derive u, v), Mul(u, derive v))
  | derive (Pow(u,n)) = Mul(Mul(Con n, Pow(u,n-1)), derive u)
```

(c)
```
fun simplify1 (Add(Con 0, u)) = u
  | simplify1 (Add(u, Con 0)) = u
  | simplify1 (Mul(Con 0, u)) = Con 0
  | simplify1 (Mul(u, Con 0)) = Con 0
  | simplify1 (Mul(Con 1, u)) = u
  | simplify1 (Mul(u, Con 1)) = u
  | simplify1 (Pow(u, 0))     = Con 1
  | simplify1 (Pow(u, 1))     = u
  | simplify1  u              = u
```

(d)
```
fun simplify (Add(u,v)) = simplify1(Add(simplify u, simplify v))
  | simplify (Mul(u,v)) = simplify1(Mul(simplify u, simplify v))
  | simplify (Pow(u,n)) = simplify1(Pow(simplify u, n))
  | simplify u          = u
```