



Programmierung: Musterlösung zum 10. Übungsblatt

Prof. Gert Smolka und Thorsten Brunklaus

Aufgabe 10.1: Signaturen (2+2+2)

- (a)
- ```
sig
 type set = int list
 val 'a empty : 'a list
 val 'a insert : 'a * 'a list -> 'a list
 val ''a member : ''a * ''a list -> bool
end
```
- (b)
- ```
sig
  type set
  val empty : set
  val insert : string * set -> set
  val member : string * set -> bool
end
```

(c) Auf Funktionen ist keine Gleichheit definiert.

Aufgabe 10.2: Rot-Schwarz-Bäume (2+2+3+3)

```
fun listA' E          ys = ys
  | listA' (N(_,a,x,b)) ys = listA' a (x::listA' b ys)

fun listA t = listA' t nil

fun listD' E          ys = ys
  | listD' (N(_,a,x,b)) ys = listD' b (x::listD' a ys)

fun listD t = listD' t nil

fun ordered (x::y::zs) = x<y andalso ordered(y::zs)
  | ordered _         = true

fun sb b = ordered(ListA b)

exception NotRB

fun forward (m,n) = if m=n then m else raise NotRB

fun black (N(R,_,_,_)) = raise NotRB
  | black _           = ()

fun rb E = 1
  | rb (N(B,a,_,b)) = 1 + forward(rb a, rb b)
  | rb (N(R,a,_,b)) = (black a ; black b ; forward(rb a, rb b))
```

Aufgabe 10.3: Funktor für effiziente endliche Mengen (8)

```
functor Set
  (type elem
   val compare : elem * elem -> order)
  :>
  sig
    type set
    val empty : set
    val insert : elem * set -> set
    val member : elem * set -> bool
  end
  =
  struct
    datatype color = R | B
    datatype set    = E | N of color * set * elem * set

    val empty = E

    fun member (y, E)                = false
      | member (y, N(_,a,x,b)) = case compare(y,x) of
                                   LESS    => member(y,a)
                                   | EQUAL  => true
                                   | GREATER => member(y,b)

    fun rotate (x,y,z,a,b,c,d) = N(R, N(B,a,x,b), y, N(B,c,z,d))

    fun rotateL (B, N(R,N(R,a,x,b),y,c), z, d) = rotate(x,y,z,a,b,c,d)
      | rotateL (B, N(R,a,x,N(R,b,y,c)), z, d) = rotate(x,y,z,a,b,c,d)
      | rotateL      body                      = N body

    fun rotateR (B, a, x, N(R,b,y,N(R,c,z,d))) = rotate(x,y,z,a,b,c,d)
      | rotateR (B, a, x, N(R,N(R,b,y,c),z,d)) = rotate(x,y,z,a,b,c,d)
      | rotateR      body                      = N body

    fun insert' (y, E)                = N(R,E,y,E)
      | insert' (y, N(color,a,x,b)) =
        case compare(y,x) of
          LESS    => rotateL(color, insert'(y,a), x, b)
        | EQUAL  => N(color,a,y,b)
        | GREATER => rotateR(color, a, x, insert'(y,b))

    fun insert (y,a) = case insert'(y,a) of
                          N(_,a,x,b) => N(B,a,x,b)
                        | E          => E
  end
```

Aufgabe 10.4: Funktor für effiziente endliche Funktionen (6)

```

functor Map
  (type key
   val compare : key * key -> order)
  :>
  sig
    type 'a map
    val empty : 'a map
    val insert : key * 'a * 'a map -> 'a map
    val lookup : key * 'a map -> 'a option
  end
  =
  struct
    datatype color = R | B
    datatype 'a map = E | N of color * 'a map * (key * 'a) * 'a map

    val empty = E

    fun compare' ((k,_),(k',_)) = compare(k,k')

    fun lookup (k, E) = NONE
      | lookup (k, N(_,a,x,b)) = case compare(k,#1x) of
          LESS => lookup(k,a)
        | EQUAL => SOME(#2x)
        | GREATER => lookup(k,b)

    fun rotate (x,y,z,a,b,c,d) = N(R, N(B,a,x,b), y, N(B,c,z,d))

    fun rotateL (B, N(R,N(R,a,x,b),y,c), z, d) = rotate(x,y,z,a,b,c,d)
      | rotateL (B, N(R,a,x,N(R,b,y,c)), z, d) = rotate(x,y,z,a,b,c,d)
      | rotateL body = N body

    fun rotateR (B, a, x, N(R,b,y,N(R,c,z,d))) = rotate(x,y,z,a,b,c,d)
      | rotateR (B, a, x, N(R,N(R,b,y,c),z,d)) = rotate(x,y,z,a,b,c,d)
      | rotateR body = N body

    fun insert' (y, E) = N(R,E,y,E)
      | insert' (y, N(color,a,x,b)) =
        case compare'(y,x) of
          LESS => rotateL(color, insert'(y,a), x, b)
        | EQUAL => N(color,a,y,b)
        | GREATER => rotateR(color, a, x, insert'(y,b))

    fun insert (k,e,a) = case insert'((k,e),a) of
          N(_,a,x,b) => N(B,a,x,b)
        | E => E
  end
end

```

Aufgabe 10.5: Große Natürliche Zahlen (3+3+2+0+4+4+4)

```
signature Bignat =
  sig
    eqtype bignat

    val fromString : string -> bignat
    val toString   : bignat -> string
    val add        : bignat * bignat -> bignat
    val sub        : bignat * bignat -> bignat          (* Domain *)
    val less       : bignat * bignat -> bool
    val mul        : bignat * bignat -> bignat
    val divmod     : bignat * bignat -> bignat * bignat (* Div *)
  end
```

```

structure Bignat :> Bignat =
  struct
    type bignat = int list (* least significant digit first *)

    val B = 10000 (* fromString expects B = 10000 *)

    fun adj (a,ar) = if null ar andalso a=0 then [] else a::ar

    fun fromInt n = if n<1 then [] else n mod B :: fromInt(n div B)

    fun add'([], br, 0) = br
      | add'([], br, c) = add'([c],br,0)
      | add'(ar, [], 0) = ar
      | add'(ar, [], c) = add'(ar,[c],0)
      | add'(a::ar, b::br, c) =
        let
          val s = a+b+c
          val (d,c') = if s < B then (s,0) else (s-B,1)
        in
          d :: add'(ar,br,c')
        end

    fun add(ar,br) = add'(ar,br,0)

    fun sub'(ar, [], 0) = ar
      | sub'([], _, _) = raise Domain
      | sub'(ar, [], c) = sub'(ar,[c],0)
      | sub'(a::ar, b::br, c) =
        let
          val s = a-b-c
          val (d,c') = if s>=0 then (s, 0) else (s+B, 1)
        in
          adj(d, sub'(ar,br,c'))
        end

    fun sub(ar,br) = sub'(ar,br,0)

    fun less (x,y) = (sub(x,y) ; false) handle Domain => true

    fun mul'([], _, c) = fromInt c
      | mul'(_, 0, c) = fromInt c
      | mul'(a::ar, b, c) = let
          val s = a*b+c
        in
          s mod B :: mul'(ar, b, s div B)
        end

    fun mul([], _ ) = []
      | mul(_, [] ) = []
      | mul(ar, b::br) = add(mul'(ar,b,0), adj(0, mul(ar, br)))
  end

```

```

fun divmod' (x,y,d) = divmod'(sub(x,y), y, d+1)
                        handle Domain => (d, x)

fun divmod (_, []) = raise Div
  | divmod (x, y) = if less(x,y) then ([],x) else
                    let
                        val (d , m ) = divmod(x,0::y)
                        val (d', m') = divmod'(m,y,0)
                    in
                        (d'::d, m')
                    end

val ord0 = ord #"0"
val ten  = fromInt 10

fun charToBignat c = fromInt(ord c - ord0)

fun fromString s = foldl (fn (c,n) => add(charToBignat c, mul(ten, n)))
                        nil (explode s)

fun pad s = case size s of
              1 => "000" ^ s
            | 2 => "00" ^ s
            | 3 => "0" ^ s
            | _ => s

fun toString' nil = "0"
  | toString' (s::ss) = s ^ foldr (fn (s,t) => pad s ^ t) "" ss

fun toString x = toString'(map Int.toString (rev x))

end

```