Programmierung WS 2002 / 03:
Musterlösung zum 10. Übungsblatt

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

Aufgabe 10.1: Signaturen (4+4+4)

(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 (4+4+6+6+4)

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

fun listA t = listA' t nil
```

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

fun listD t = listD' t nil
```

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

fun sb b = ordered(listA b)
```

(d)
```
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))
```

(e) `fun redblack t = (rb t; sb t) handle NotRB => false`

Aufgabe 10.3: Funktor für effiziente endliche Mengen (10)

```
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 (10)

```
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,(k',e'),b)) = case compare(k, k') of
                                         LESS    => lookup(k,a)
                                       | EQUAL   => SOME e'
                                       | 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
```

Aufgabe 10.5: Große natürliche Zahlen ($44 = 6 + 6 + 0 + 10 + 4 + 10 + 8$**)**

```
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
  end


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

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

    fun adj (d,xs) = if null xs andalso d=0 then [] else d::xs

    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)))

    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


val one = fromString "1"
val two = fromString "2"

fun fac' n = if less(n,two) then one else mul(n, fac'(sub(n,one)))

fun fac s = toString(fac'(fromString s))

val _ = case CommandLine.arguments() of
          [s] => print (fac s ^ "\n")
        |  _  => print "Need a number\n"
```

```
- fac "1000";
> val it =
    "402387260077093773543702433923003985719374864210714632543799910429938512398
629020592044208486969404800479988610197196058631666872994808558901323829669944 59
099742450408707375991882362772718873251977950595099527612087497546249704360141 82
780946464962910563938874378864873371191810458257836478499770124766328898359557 35
432513185323958463075557409114262417474349347553428646576611667797396668820291 20
737914385371958824980812686783837455973174613608537953452422158659320192809087 82
973084313928444032812315586110369768013573042161687476096758713483120254785893 20
767169132448426236131412508780208000261683151027341827977704784635868170164365 02
415369139828126481021309276124489635992870511496497541990934222156683257208082 13
331861168115536158365469840467089756029009505376164758477284218896796462449451 60
765353408198901385442487984959953319101723355556602139450399736280750137837615 30
712776192684903435262520001588853514733161170210396817592151090778801939317811 41
945452572238655414610628921879602238389714760885062768629671466746975629112340 82
439208160153780889893964518263243671616762179168909779911903754031274622289988 00
519544441428201218736174599264295658174662830295557029902432415318161721046583 20
367869061172601587835207515162842255402651704833042261439742869330616908979684 82
590125458327168226458066526769958652682272807075781391858178889652208164348344 82
599326604336766017699961283186078838615027946595513115655203609398818061213855 86
003014356945272242063446317974605946825731037900840244324384656572450144028218 85
252470935190620929023136493273497565513958720559654228749774011413346962715422 84
586237738753823048386568897646192738381490014076731044664025989949022222176590 43
399018860185665264850617997023561938970178600408118897299183110211712298459016 41
921068884387121855646124960798722908519296819372388642614839657382291123125024 18
664935314397013742853192664987533721894069428143411852015801412334482801505139 96
942901534830776445690990731524332782882698646027898643211390835062170950025973 89
863554277196742822248757586765752344220207573630569498825087968928162753848863 39
690995982628095612145099487170124451646126037902930912088908694202851064018215 43
994571568059418727489809042547421735824010636774045957417851608292301353580818 40
096996372524230560855903700624271243416909004153690105933983835777939410970027 75
347200000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000"
```