Funktor zur Bildung eines Matrizenrings

$\langle$*Halbringe*$\rangle\equiv$

```
signature HALBRING =
   sig
      type t
      val zero : t
      val unit : t
      val sum : t * t -> t
      val prod : t * t -> t
   end

signature MATRIX =
   sig
      structure Ring : HALBRING
      type r = Ring.t
      type t
      val dimension : t -> int * int
      val tabulate : int * int
                     -> (int * int -> r) -> t
      val sub : t * (int * int) -> r
      val map : (r -> r) -> t -> t
      val zero : int * int -> t
      val unit : int * int -> t
      val sum : t * t -> t
      val prod : t * t -> t
   end
```

Mit `(tabulate (n,m) f)` erzeugt man aus f die Matrix

$$A = \begin{pmatrix} f(0,0) & \cdots & f(0,m-1) \\ \vdots & \ddots & \vdots \\ f(n-1,0) & \cdots & f(n-1,m-1) \end{pmatrix}$$

Matrix(R : HALBRING) : MATRIX

⟨*Matrix*⟩≡

```
functor Matrix(R : HALBRING) : MATRIX =
   struct
      structure Ring = R
      type r = Ring.t
      type t = r vector vector
      structure V = Vector  (* SML-NJ Library *)

      fun dimension (S:t) =
         let
            val rows = V.length S
            val columns = V.length (V.sub(S,0))
            val samelengths =
               V.foldr (fn (v,b) => b andalso
                              (V.length v = columns))
               true S
         in
            if samelengths then (rows,columns)
            else raise Size
         end

      fun sub (E:t,(i,j)) =
         let
            val (d1,d2) = dimension E
         in
            if 0 <= i andalso i < d1 andalso 0 <= j
               andalso j < d2
               then V.sub (V.sub (E,i), j)
            else raise Size
         end
```

```
fun tabulate (rows,cols) (f : int * int -> r) =
   V.tabulate(rows, fn i =>
              V.tabulate (cols, fn k => f(i,k))) :

fun map (f : r -> r) (A : t) =
   let
      val (n,m) = dimension (A:t)
   in
      tabulate (n,m) (fn (i,j) => f (sub (A,(i,j))))
   end

fun zero (n,m) =
   tabulate (n,m) (fn (i,k) => R.zero)
fun unit (n,m) =
   tabulate (n,m) (fn (i,k) => if i=k then R.unit
                               else R.zero)

fun sum (S:t,T:t) : t =
   let
      val (ns,ms) = dimension S
      val (nt,mt) = dimension T
      fun sumV (v:r vector,w:r vector) : r vector =
         V.tabulate (V.length v, fn i =>
                     R.sum (V.sub(v,i),V.sub(w,i)))
   in
      if (ns,ms)=(nt,mt) then
         V.tabulate (ns, fn i => sumV (V.sub (S,i),
                                      V.sub(T,i)))
      else raise Size
   end
```

Matrizenprodukt

⟨*Matrix*⟩+≡

```
    fun prod (S:t,T:t) : t =
      let
        val (ns,ms) = dimension S
        val (nt,mt) = dimension T
        fun prodV (v:r vector,w:r vector) : r =
          let
            fun sm i =
              if i=0 then R.zero
              else R.sum(sm (i-1),
                         R.prod (V.sub(v,i-1),
                                 V.sub(w,i-1)))
          in
            sm (V.length v)
          end
        fun column (T:t,j) =
          V.map (fn r => V.sub (r,j)) T
      in
        if ms=nt then
          tabulate (ns,mt)
          (fn (i,j) => prodV (V.sub (S,i),
                              column(T,j)))
        else raise Size
      end
  end
```

Matrixbeispiel

⟨*IntRing*⟩≡

```
structure IntRing =
   struct
      type t = int
      val zero = 0
      val unit = 1
      fun sum (x,y) = x + y
      fun prod (x,y) = x*y
   end

structure IntMatrix = Matrix(IntRing)

val Zero = IntMatrix.zero (5,5)
and Unit = IntMatrix.unit (5,5)

fun a (i,j) = i + j
and b (i, j) = i + ~j

val A = IntMatrix.tabulate (5,5) a
and B = IntMatrix.tabulate (5,5) b

IntMatrix.sum (A,B);
- #[#[0,0,0,0,0],
    #[2,2,2,2,2],
    #[4,4,4,4,4],
    #[6,6,6,6,6],
    #[8,8,8,8,8]] : IntMatrix.t
```