Daniel Leeds, 15-212 R04, September 19, 2007

```
fun transpose M =
    if null (hd M) then [] else
    (map hd M) :: transpose (map tl M)

fun nth 1 (x::L) = x
 |  nth i (x::L) = nth (i-1) L
```

For all i:

```
row i = nth i
col i = map (nth i)
```

Prove for all well-formed matrices M,

P(M) : for all i, s.t. 1<= i <= m, dim M = (m,n)

```
     row i (transpose M) = col i M
```

Prove for all well-formed matrices M,
P(M) : for all i, s.t. 1<= i <= m, dim M = (m,n)

```
row i (transpose M) = col i M
```

Proof by strong induction on m+n, s.t. dim M = (m,n)

Base Case: M = [] (m+n = 0)
there is no i, s.t. 1 <= i <= m
therefore, P([]) is vacuously true

Inductive case:
Assume P(M') ($m_{M'} + n_{M'}$)
Show P(r::M') ($m_{r::M'} + n_{r::M'} = m_{M'} + n_{M'} + 1$)

```
row i (transpose r::M')
            = row i ((map hd (r::M'))::(transpose (map tl (r::M'))))
            = nth i ((map hd (r::M'))::(transpose (map tl (r::M'))))
```

If i = 1

```
            = map hd (r::M')
            = map (nth i) (r::M')
            = col i (r::M')
```

If i>1

```
            = nth (i-1) (transpose (map tl (r::M')))
            = row (i-1) (transpose (map tl (r::M')))
            = col (i-1) (map tl (r::M'))
            = map (nth (i-1)) (map tl (r::M'))
            = map ((nth (i-1)) o tl) (r::M')
            = map ((hd o tl^(i-2)) o tl) (r::M')
            = map (hd o tl^(i-1)) (r::M')
            = map (nth i) (r::M')
            = col i (r::M')
```

Lemma: P(i): Given i >= 1,
`nth i = hd o tl^(i-1)`,
where `tl^i` signifies tl composed with itself i times, e.g., `tl^0 = fn x => x` and `tl^2 = tl o tl`)

B.C. P(1)

```
nth 1 (x::L) = x
             = (hd o tl^0) (x::L)
```

Iterative case, Assume P(i'), show P(i'+1)

```
nth i'+1 (x::L) = nth i' L
                = (hd o tl^(i'-1)) L
                = (hd o tl^(i'-1)) (tl (x::L))
                = hd o (tl^(i'-1) o tl) (x::L)
                = hd o tl^i' (x::L)
                = (hd o tl^((i'+1)-1)) (x::L)
```

```
fun zap f ([],[]) = []
 |  zap f (x::L,y::L) = f (x,y)::zap f (L,R)

fun Transpose [row] = map (fn x => [x]) row
 |  Transpose (row::rows) = zap (op ::) (row, Transpose rows)
```

Prove for all well-formed matrices M,
P(M) : `Transpose M = transpose M`

Proof by strong induction on m+n, s.t. dim M = (m,n)

Base Case: M = [] (m+n = 0)

```
transpose [] = []
             = map (fn x => [x]) []
             = Transpose []
```

Inductive case:

Assume P(M’) ($m_{M'} + n_{M'}$)

Show P(r::M’) ($m_{r::M'} + n_{r::M'} = m_{M'} + n_{M'} + 1$)

```
transpose (r::M) = map hd (r::M)::transpose (map tl (r::M’))
                 = map hd (r::M)::Transpose (map tl (r::M’))
                 = (hd r::map hd M’)::
(zap (op ::) (hd (map tl (r::M’)), Transpose (tl (map tl (r::M’))))
                 = zap (op ::) (hd r::hd (map tl (r::M’)),
map hd M’::Transpose (tl (map tl (r::M’))))
                 = zap (op ::) (hd r::tl r,
map hd M’::Transpose (tl (map tl (r::M’))))
                 = ... ,map hd M’::Transpose (map tl M’))
                 = ... ,map hd M’::transpose (map tl M’))
                 = ... ,transpose M’)
                 = zap (op ::) (r,transpose M’)
                 = zap (op ::) (r,Transpose M’)
                 = Tranpose (r::M’)
```

```
fun merge p ([],ys) = ys
  | merge p (xs,[]) = xs
  | merge p (x::xs,y::ys) =
      if p (x,y)
      then x::(merge p (xs,y::ys))
      else if p(y,x)
           then y::(merge p (x::xs,ys))
           else x::y::(merge p (xs,ys))

Let: perm(P,L) mean "P is a permutation of L"

properties:
perm(P1,L1) & perm(P2,L2) -> perm(P1@P2,L1@L2)
perm(P,L) -> perm(x::P,x::L)
perm(x::y:P1@P2,L) -> perm((x::P1)@(y::P2),L)


Let: psort(S,L) mean "S is a p-sorted permutation of L"
 and psort(S) mean "S is p-sorted"
 and x<=L, where L : 'a list mean "for every y in L, not p(x,y)"

properties:
psort(x::S) <-> psort(S) && x <= S

Prove: merge p (L1,L2) = L, where psort(L,L1@L2) if psort(L1) && psort(L2)

Induct on (length L1) * (length L2)

Base case: (length L1) * (length L2) = 0
L1 = [] or L2 = []

L1 = []
merge p ([],L2) = L2
psort (L2)              (from above)

IH: L1' = x::L1, L2' = y::L2
  merge p L1' L2' =
    merge p (x::L1) (y::L2)
    if p(x,y) then ... else ...

CASE 1: p (x,y)
    x::(merge p L1 (y::L2))
    x::L

      p (x,y) && psort(y::L2) -> y<=L2 -> x<=L2, x<=(y::L2)
      psort(x::L1) -> x<=L1
      x<=L1 && x<=(y::L2) -> x<=L1@(y::L2)
      psort(L,L1@(y::L2))
      psort(L,L1@(y::L2)) && x<=L1@(y::L2) -> psort(x::L,x::L1@(y::L2))

CASE 2: p (y,x)
    y::(merge p (x::L1) L2)
    y::L

      p (y,x) && psort(x::L1) -> x<=L1 -> y<=L1, y<=(x::L1)
      psort(y::L2) -> y<=L2
      y<=L2 && y<=(x::L1) -> y<=(x::L1)@L2
```

```
      psort(L,(x::L1)@L2)
      psort(L,(x::L1)@L2) && y<=(x::L1)@L2 -> psort(y::L,y::(x::L1)@L2)
      psort(y::L,y::(x::L1)@L2) -> psort(y::L,(x::L1)@(y::L2))

CASE 3: other
    x::y::(merge p L1 L2)
    x::y::L

      psort(x::L1) -> x<=L1
      psort(y::L2) -> y<=L2
      !p(x,y) && y<=L2 -> x<=y::L2
      !p(y,x) && x<=L1 -> y<=L1
      psort(L,L1@L2) && x<=L1 && y<=L2 && x<=y::L2 && y<=L1 ->
      psort(x::y::L,x::y::L1@L2) -> psort(x::y::L,(x::L1)@(y::L2))
```