1 Vectors

We now implement a simple example that shows how we can define a system of mutable vectors.

We begin by defining a 2d vector ($Vector2$) with two methods and a 3d vector ($Vector3$) with two other methods and which inherits the 2d vector:

```
type Vector2Def k = Field Float   Cons   Field Float   Cons   Method k () ()   Cons
data RecVector2 = RecVector2 (Vector2Def RecVector2)
type Vector2 = Vector2Def RecVector2
instance Recursive Vector2 where
  type Rec = RecVector2
  to = RecVector2
  from (RecVector2 v) = v
x :: Label Vector2 (Field Float)
x = labelAt Z
y :: Label Vector2 (Field Float)
y = labelAt (S Z)
norm2 :: Label Vector2 (Method RecVector2 () ())
norm2 = labelAt (S S Z)
len2 :: Label Vector2 (Method RecVector2 () Float)
len2 = labelAt (S S S Z)

mk_vector2 xv yv = Field xv   Cons   Field yv   Cons   mk_method (\this->\()->do l <-
                (this <= x) *= (/l)
                (this <= y) *= (/l))   Cons   mk_method (\this->\()->do xv <- this <= x
         yv <- this <= y
         return sqrt(xv * xv + yv * yv))

type Vector3Def k = Inherit Vector3   Cons   Field Float   Cons   Method k () ()   Co
data RecVector3 = RecVector3 (Vector3Def RecVector3)
type Vector3 = Vector3Def RecVector3
z :: Label Vector3 (Field Float)
z = labelAt (S Z)
norm3 :: Label Vector3 (Method RecVector3 () ())
norm3 = labelAt (S S Z)
len3 :: Label Vector3 (Method RecVector3 () Float)
len3 = labelAt (S S S Z)
instance Recursive Vector3 where
  type Rec = RecVector3
  to = RecVector3
  from (RecVector3 v) = v

mk_vector3 xv yv zv = Inherit (mk_vector2 xv yv)   Cons   Field z   Cons   mk_method (\
                (this <= x) *= (/l)
                (this <= y) *= (/l)
                (this <= z) *= (/l))   Cons   mk_method (\this->\()->do xv <- this <= x
         yv <- this <= y
         zv <- this <= z
         return sqrt(xv * xv + yv * yv + zv * zv))
```

Now we can use these vectors:

```
main :: h1 ~ (Vector3   Cons   Nil) => ST Nil Bool
main = mk_vector3 0.0 2.0 -1.0 >>+ (\(v :: Ref h1 Vector3) ->
       do zv <- v <= x
          let  v   = coerce v :: Ref h1 Vector2
           z v   <-  v   <= x
          (v <= norm2)()
          (v <= norm3)()
          return (v =  v  ))

res :: Bool
```

```
res = runST main Nil
```

where we expect that $res = True$. Notice how select labels that are defined for a 2d vector on an instance of a 3d vector, and how we access the same label on the value of *base* obtained through coercion on the 3d vector.