# Fields in Haskell

Scott N. Walck

September 13, 2024

# `Position` is a new type.

The 3 coordinate systems give us 3 ways to make a position.

```
cartesian   :: (R,R,R) -> Position   -- (x,y,z)
cylindrical :: (R,R,R) -> Position   -- (s,phi,z)
spherical   :: (R,R,R) -> Position   -- (r,theta,phi)
```

There are convenience functions if you want to construct a `Position` without commas and parentheses.

```
cart :: R -> R -> R -> Position
cyl  :: R -> R -> R -> Position
sph  :: R -> R -> R -> Position
```

## There are convenience functions if you want to construct a `Position` without commas and parentheses.

```
ghci> cyl 2 (pi/2) 4
Cart 1.2246467991473532e-16 2.0 4.0
```

- $1.2 \times 10^{-16}$ is as close as the computer can come to zero.
- Can you picture how $(s, \phi, z) = (2, \pi/2, 4)$ is the same point as $(x, y, z) = (0, 2, 4)$?

# A `Position` can be expressed in any of the 3 coordinate systems.

```
cartesianCoordinates   :: Position -> (R,R,R)
cylindricalCoordinates :: Position -> (R,R,R)
sphericalCoordinates   :: Position -> (R,R,R)
```

```
ghci> cylindricalCoordinates (cart 0 2 0)
(2.0,1.5707963267948966,0.0)
```

## A Displacement is a Vec from the source Position to the target Position.

```
displacement :: Position  -- source position
             -> Position  -- target position
             -> Vec
```

```
ghci> displacement (cart 2 3 4) (cart 4 9 16)
vec 2.0 6.0 12.0
```

A displacement can shift a source Position to a target Position.

```
shiftPosition :: Vec        -- displacement
              -> Position   -- source position
              -> Position   -- target position
```

```
ghci> shiftPosition (vec 2 6 12) (cart 2 3 4)
Cart 4.0 9.0 16.0
```

# A scalar field is a function from position to numbers.

```
type ScalarField = Position -> R
```

Examples of scalar fields

- Volume charge density
  (The number is charge density in $C/m^3$.)
- Electric potential
  (The number is electric potential in V.)
- Temperature
  (The number is temperature in K.)

# Encoding a scalar field in Haskell.

$$f(x, y, z) = x^2 + y^3 + z^4$$

```
type ScalarField = Position -> R
```

```
fa :: ScalarField
fa r = let (x,y,z) = cartesianCoordinates r
       in x**2 + y**3 + z**4
```

The local variable `r` has type `Position`.

# Catalog entry 1.2.1 is a scalar field given in cylindrical coordinates.

$$f(s, \phi, z) = s^2 z \cos \phi$$

```
type ScalarField = Position -> R

-- scalar field from Catalog 1.2.1
f :: ScalarField
f r = let (s,phi,z) = cylindricalCoordinates r
      in s**2 * z * cos phi
```

The local variable `r` has type `Position`.

# Catalog entry 1.3.4 is a scalar field given in spherical coordinates.

$$f(r, \theta, \phi) = r^2(3\cos^2\theta - 1)$$

```
type ScalarField = Position -> R
```

```
-- scalar field from Catalog 1.3.4
f134 :: ScalarField
f134 p = let (r,theta,_phi) = sphericalCoordinates p
         in r**2 * (3 * cos theta ** 2 - 1)
```

The local variable `p` has type `Position`.

# A vector field is a function from position to vectors.

```
type VectorField = Position -> Vec
```

Examples of vector fields

- Electric Field
  (The vector is electric field in N/C or V/m.)
- Magnetic Field
  (The vector is magnetic field in T.)
- Volume Current Density
  (The vector is current density in $A/m^2$.)

# Encoding a vector field in Haskell.

$$\vec{v}_a = x^2\hat{x} + 3xz^2\hat{y} - 2xz\hat{z}$$

```
type VectorField = Position -> Vec
```

```
va :: VectorField
va r = let (x,_y,z) = cartesianCoordinates r
       in x**2 *^ iHat ^+^ 3 *^ x *^ z**2 *^ jHat
                       ^-^ 2 *^ x *^ z *^ kHat
```

The local variable `r` has type `Position`.

# Vectors and vector fields

```
type VectorField = Position -> Vec

iHat :: Vec
jHat :: Vec
kHat :: Vec
xHat :: VectorField
yHat :: VectorField
zHat :: VectorField

sHat     :: VectorField
phiHat   :: VectorField
rHat     :: VectorField
thetaHat :: VectorField
```

# Comparison of unit vectors and unit vector fields

$$\vec{v}_a = x^2\hat{x} + 3xz^2\hat{y} - 2xz\hat{z}$$

Two ways of writing this vector field:

```
va :: VectorField
va r = let (x,_y,z) = cartesianCoordinates r
       in x**2 *^ iHat ^+^ 3 *^ x *^ z**2 *^ jHat
                        ^-^ 2 *^ x *^ z *^ kHat

va2 :: VectorField
va2 r = let (x,_y,z) = cartesianCoordinates r
        in x**2 *^ xHat r ^+^ 3 *^ x *^ z**2 *^ yHat r
                           ^-^ 2 *^ x *^ z *^ zHat r
```

The local variable `r` has type `Position`.

# A vector field expressed in cylindrical coordinates.

$$\vec{v} = s(2 + \sin^2\phi)\hat{s} + s\sin\phi\cos\phi\hat{\phi} + 3z\hat{z}$$

```
type VectorField = Position -> Vec

v143 :: VectorField
v143 r = let (s,phi,z) = cylindricalCoordinates r
         in s *^ (2 + sin phi **2) *^ sHat r
              ^+^ s *^ sin phi *^ cos phi *^ phiHat r
              ^+^ 3 *^ z *^ zHat r
```

The local variable `r` has type `Position`.

# A vector field expressed in spherical coordinates.

$$\vec{v} = (r\cos\theta)\hat{r} + (r\sin\theta)\hat{\theta} + (r\sin\theta\cos\phi)\hat{\phi}$$

```
type VectorField = Position -> Vec
```

```
v140 :: VectorField
v140 p = let (r,theta,phi) = sphericalCoordinates p
         in r *^ cos theta *^ rHat p
              ^+^ r *^ sin theta *^ thetaHat p
              ^+^ r *^ sin theta *^ cos phi *^ phiHat p
```

The local variable `p` has type `Position`.

# A Vec always shows in Cartesian components.

Consider the following vector field.

$$\vec{v} = s\hat{\phi}$$

```
v :: VectorField
v p = let (s,_phi,_z) = cylindricalCoordinates p
      in s *^ phiHat p
```

If we ask GHCi for the particular vector at a particular point in space, it gives us the result in Cartesian components.

```
ghci> v (cyl 2 (pi/2) 7)
vec (-2.0) 1.2246467991473532e-16 0.0
```

If we say $\vec{v}(s, \phi, z) = s\hat{\phi}$, then $\vec{v}(2, \pi/2, 7) = -2\hat{x}$. We could also say $\vec{v}(2, \pi/2, 7) = 2\hat{\phi}$, but notice that the computer is giving us the Cartesian components.

- Can you picture this in your head?