Learn You a Physics for Great Good!

>>> WORK IN PROGRESS <<<

Dimensions / Testing of value-level dimensions

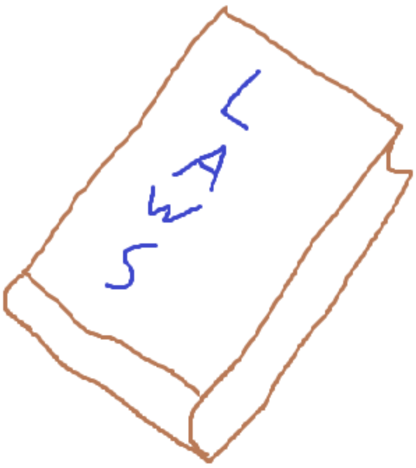
[src: Dimensions/ValueLevel/Test.lhs] Previous: Value-level dimensions Table of contents Next: Type-level dimensions

Testing of value-level dimensions

```haskell
module Dimensions.ValueLevel.Test
    ( runTests
    ) where

import Prelude hiding (length, div)
import Test.QuickCheck
import Data.List

import Dimensions.ValueLevel
```



For operations on dimensions, there are a number of laws which should hold. We will here test that the value-level dimensions obey them. One way is to use `QuickCheck`, which produces lots o’ random test cases.

Generating arbitrary dimensions

The first thing one needs in order to use `QuickCheck` is an instance of `Arbitrary` for the data type to be used in the tests. In this case it’s `Dim`.

An arbitrary example of an `Arbitrary` instance (get it?) could look like

```haskell
data IntPair = IntPair (Int, Int)


genIntPair :: Gen IntPair
genIntPair = do
  first  <- arbitrary
  second <- arbitrary
  return $ IntPair (first, second)


instance Arbitrary IntPair where
  arbitrary = genIntPair
```

Exercise Now try to implement an `Arbitrary` instance of `Dim`.

▼ **Solution**

Here's one way to do it.

```haskell
genDim :: Gen Dim
genDim = do
  le <- arbitrary
  ma <- arbitrary
  ti <- arbitrary
  cu <- arbitrary
  te <- arbitrary
  su <- arbitrary
  lu <- arbitrary
  return (Dim le ma ti cu te su lu)


instance Arbitrary Dim where
  arbitrary = genDim
```

Properties for operations on dimensions

Since dimensions are treated just like regular numbers when it comes to multiplication and division, the laws which ought to hold should be pretty clear. It's the "obvious" laws such as commutativity and so on.

The laws to test are

- Multiplication is commutative
- Multiplication is associative
- `one` is a unit for multiplication
- Multiplication and division cancel each other out
- Dividing by `one` does nothing
- Dividing by a division brings up the lowest denominator

$$\frac{x}{\frac{x}{y}} = y$$

- Multiplication by x is the same as dividing by the inverse of x.

The implementation of the first law looks like

```
-- Property: multiplication is commutative
prop_mulCommutative :: Dim -> Dim -> Bool
prop_mulCommutative d1 d2 = d1 `mul` d2 == d2 `mul` d1
```

Excercise. Implement the rest.

▼ Solution

Here's what the rest could look like.

```
-- Property: multiplication is associative
prop_mulAssociative :: Dim -> Dim -> Dim -> Bool
prop_mulAssociative d1 d2 d3 = d1 `mul` (d2 `mul` d3) ==
 (d1 `mul` d2) `mul` d3


-- Property: `one` is a unit for multiplication
prop_mulOneUnit :: Dim -> Bool
prop_mulOneUnit d = d == one `mul` d


-- Property: multiplication and division cancel each other out
prop_mulDivCancel :: Dim -> Dim -> Bool
prop_mulDivCancel d1 d2 = (d1 `mul` d2) `div` d1 == d2
```

```haskell
-- Property: dividing by `one` does noting
prop_divOne :: Dim -> Bool
prop_divOne d = d `div` one == d


-- Property: dividing by a division brings up the lowest denominator
prop_divTwice :: Dim -> Dim -> Bool
prop_divTwice d1 d2 = d1 `div` (d1 `div` d2) == d2


-- Property: multiplication same as division by inverse
prop_mulDivInv :: Dim -> Dim -> Bool
prop_mulDivInv d1 d2 = d1 `mul` d2 ==
  d1 `div` (one `div` d2)
```

We should also test the pretty-printer. But just like how that function itself is implemented isn’t interesting, neither is the code testing it. We therefore leave it out.

From this module, we export a function `runTests`. That function runs all the tests implemented here and is used with Stack.

[src: Dimensions/ValueLevel/Test.lhs] Previous: Value-level dimensions Table of contents Next: Type-level dimensions