Generic Matrix Multiplication

Sean Seefried

Revision

Dot products

Take two vectors of length n

$u = (u_1, \ldots, u_n)$ $\qquad$ $v = (v_1, \ldots, v_n)$

Dot product is

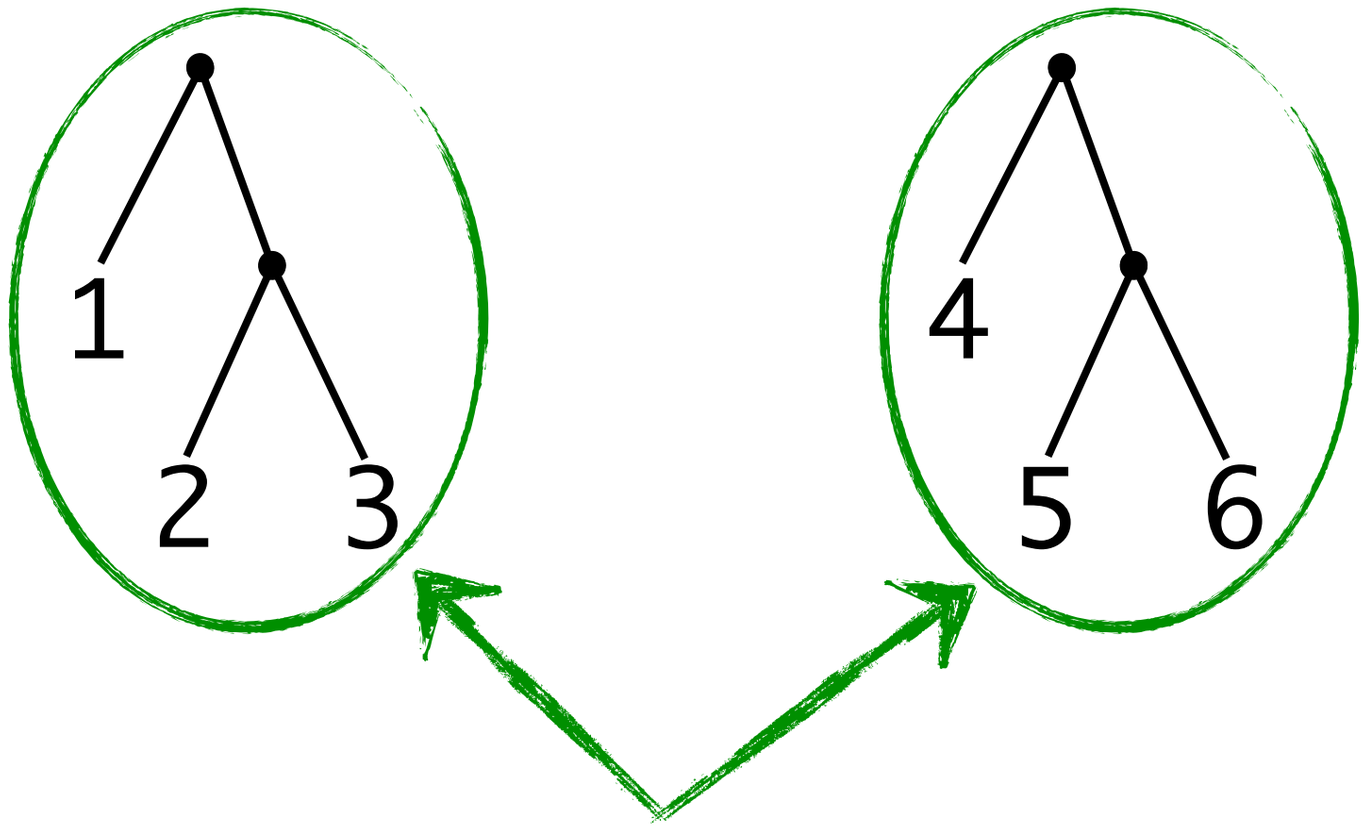
$$u \cdot v = u_1 v_1 + \ldots + u_n v_n$$

or

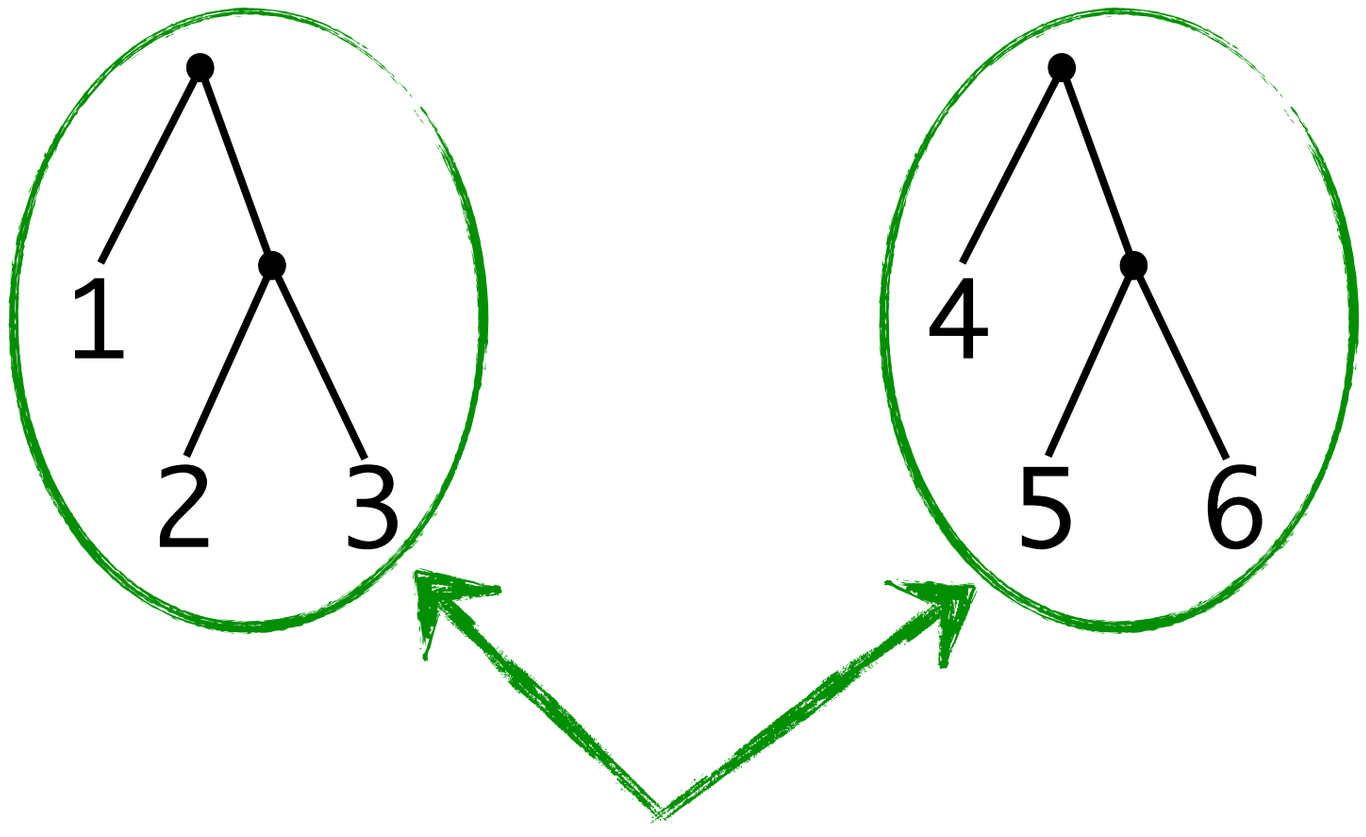
$$u \cdot v = \sum_{i=1}^{n} u_i v_i$$

What about other data types?

What about other data types?



What about other data types?



$$1*4 + 2*5 + 3*6 = 32$$

Shapes

- Encode *shap*e of data structure with GADTs
- Type system ensures that only values of same shape can be `zipWithed` together

For a data structure T

IF you can define Foldable and Applicative instances

THEN you have dot product!

Trees with shapes

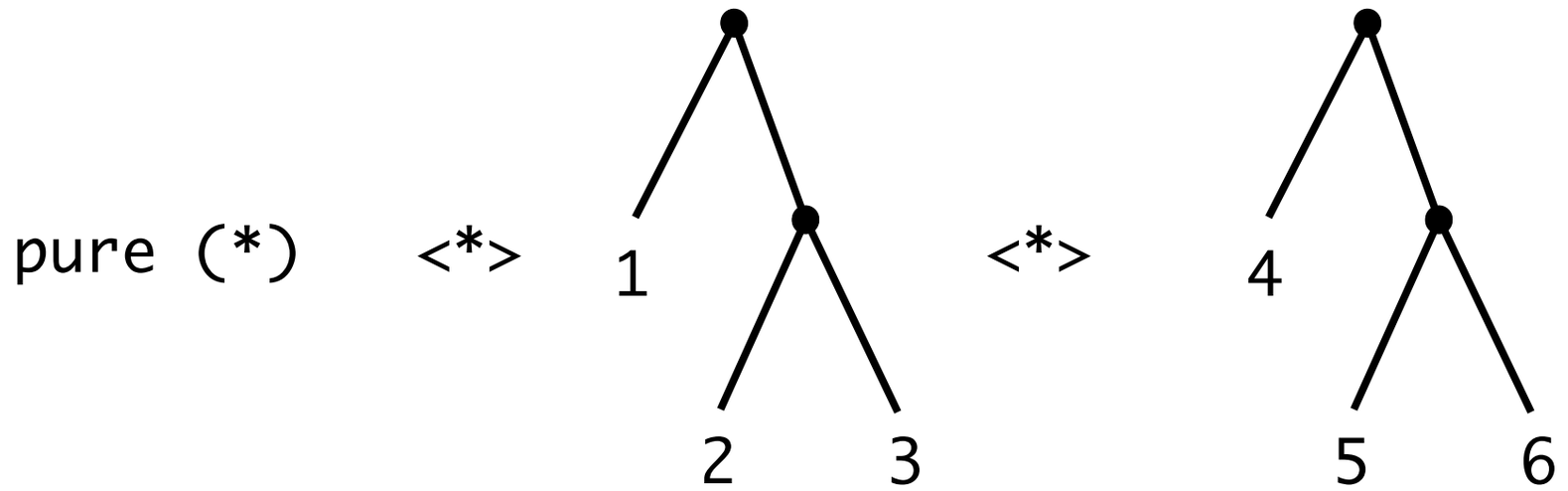
```
data Tree sh a where
  Leaf   :: a -> Tree () a
  Branch :: Tree m a -> Tree n a -> Tree (m,n) a
```

Applicative on Trees with shape

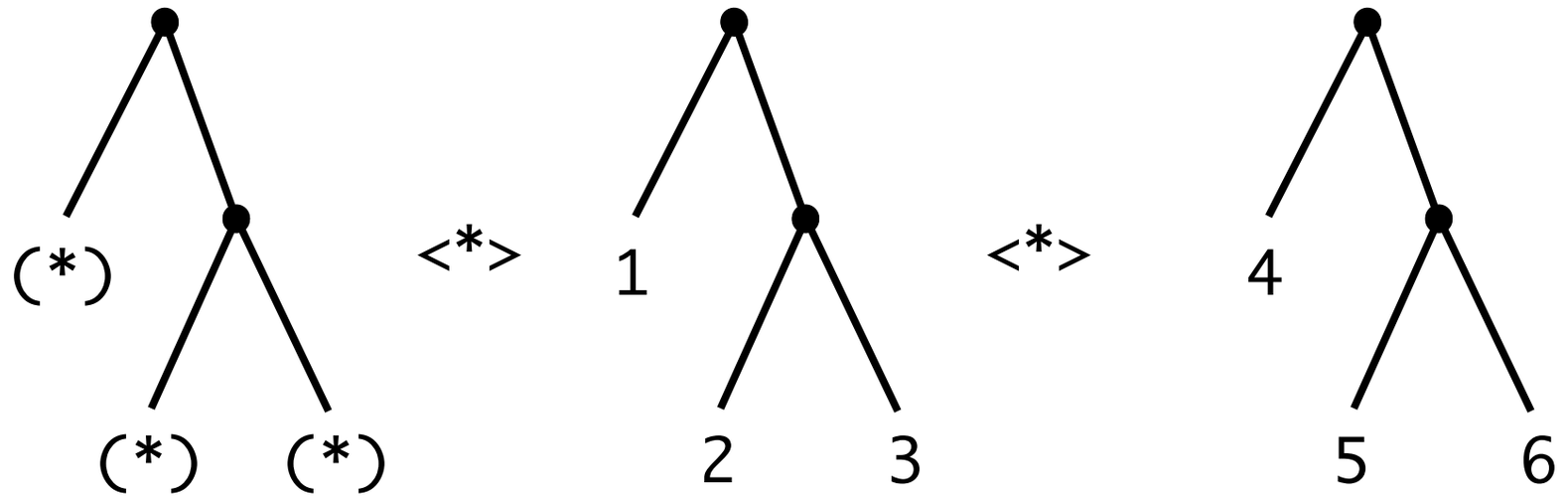
```
instance Applicative (Tree ()) where
  pure a                              = Leaf a
  Leaf fa <*> Leaf a                  = Leaf (fa a)

instance (Applicative (Tree m), Applicative (Tree n)) =>
          Applicative (Tree (m,n)) where
  pure a                                = Branch (pure a) (pure a)
  (Branch fs ft) <*> (Branch s t) = Branch (fs <*> s) (ft <*> t)
```

Let's see `liftA2 (*)` on Trees

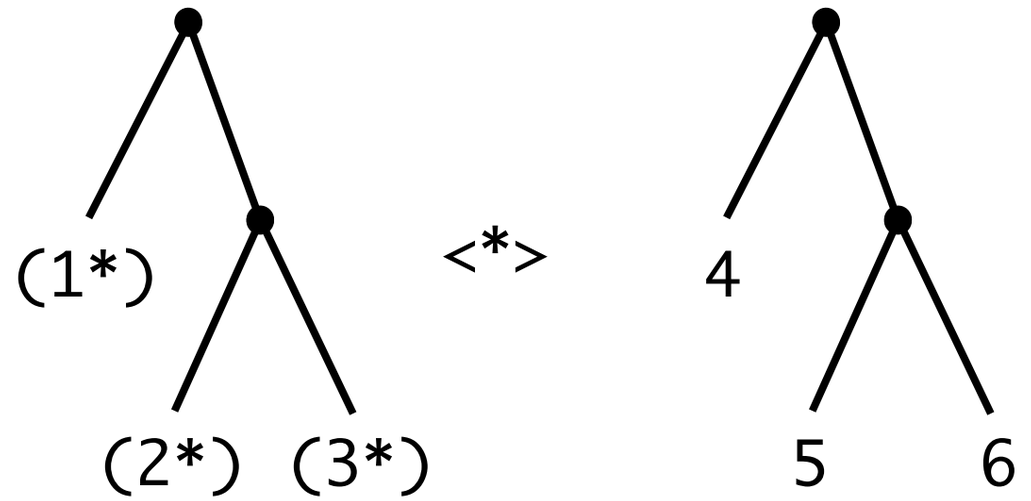
`liftA2 (*)` on Trees



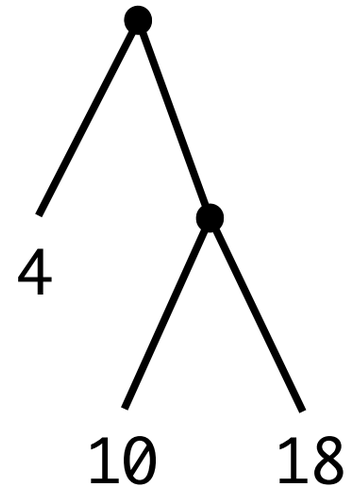
`liftA2 (*)` on Trees



`liftA2 (*)` on Trees

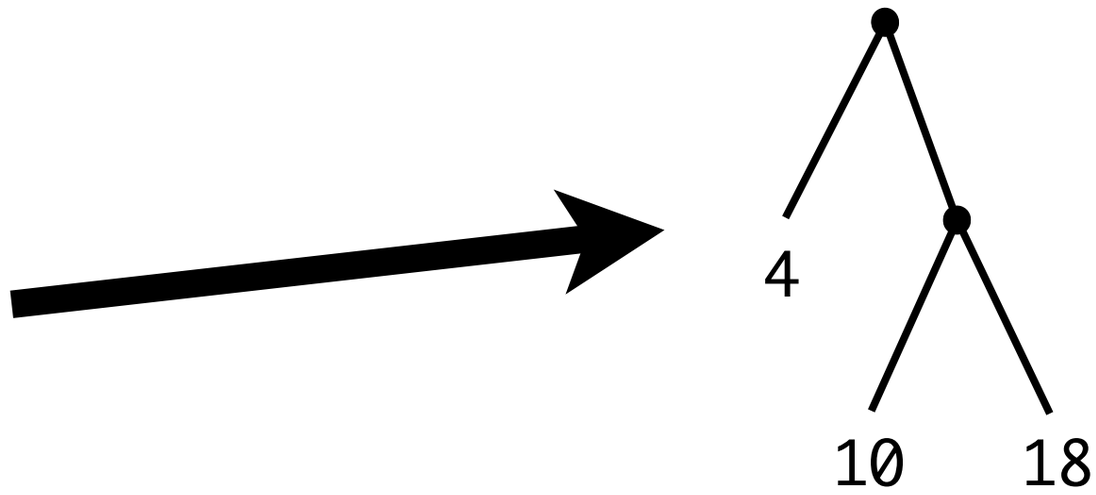


`liftA2 (*)` on Trees



`liftA2 (*)` on Trees

Now just `fold (+)` over this to get dot product



`liftA2 (*)` on Trees

Now just `fold (+)` over this to get dot product

32

Foldable on Trees with shape

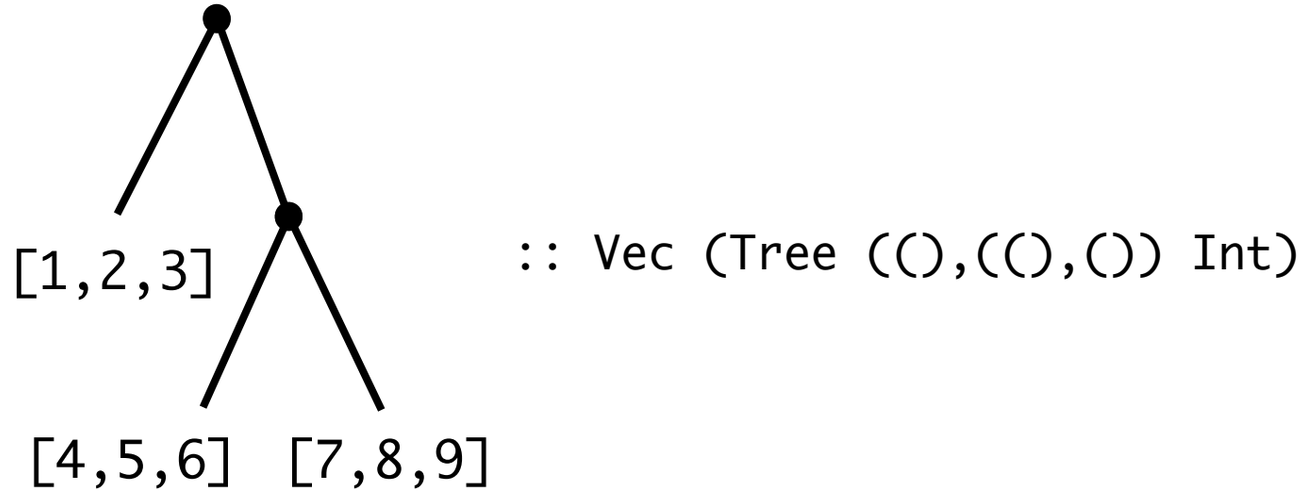
```
instance Foldable (Tree sh) where
  -- foldMap :: (Foldable f, Monoid m) => (a -> m) -> f a -> m
  foldMap f (Leaf a)     = f a
  foldMap f (Branch s t) = foldMap f s `mappend` foldMap f t

  -- fold :: (Foldable f, Monoid m) => f m -> m
```

Generic dot product

```
dot :: (Num a, Foldable f, Applicative f) => f a -> f a -> a
dot x y = foldSum $ liftA2 (*) x y
  where foldSum = getSum . fold . fmap Sum
```

What is a matrix?

A collection of collections



Generalising dimensions

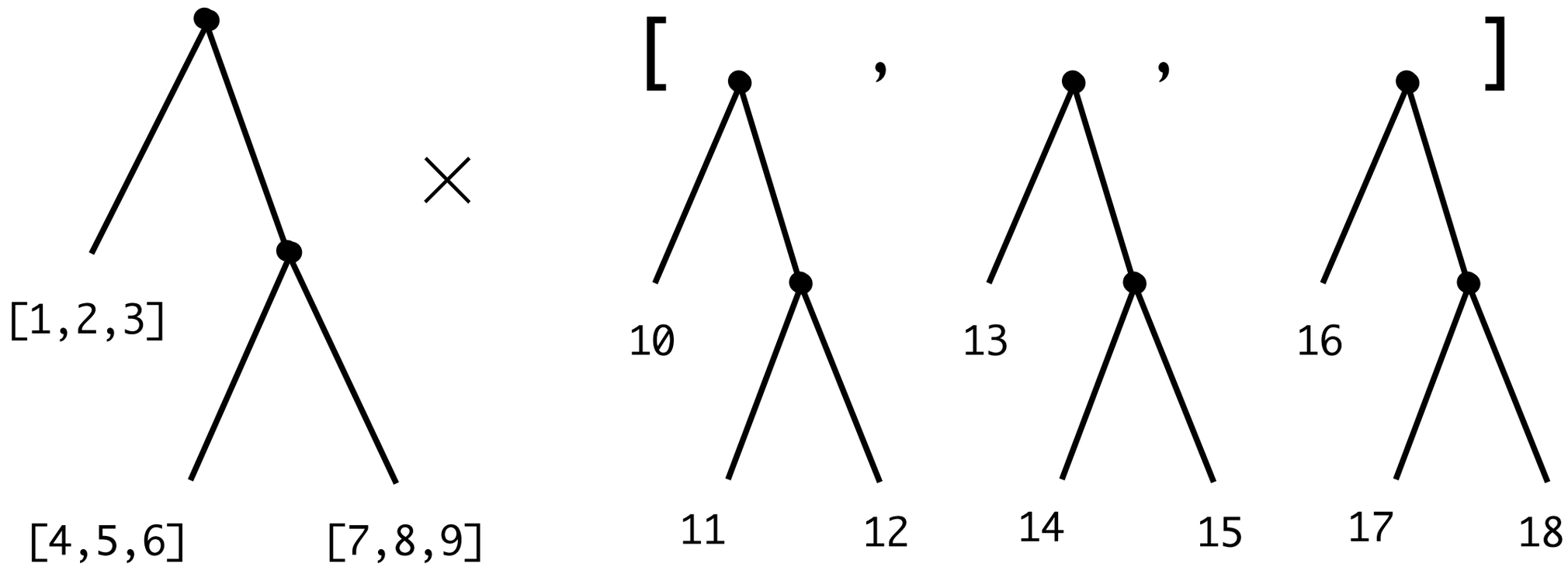
For regular matrices *dimensions* of input matrices determine dimensions of output matrix

$$m \times n \times n \times p = m \times p$$

For generic matrices *type and shape* of input matrices determine *type and shape* of output matrix

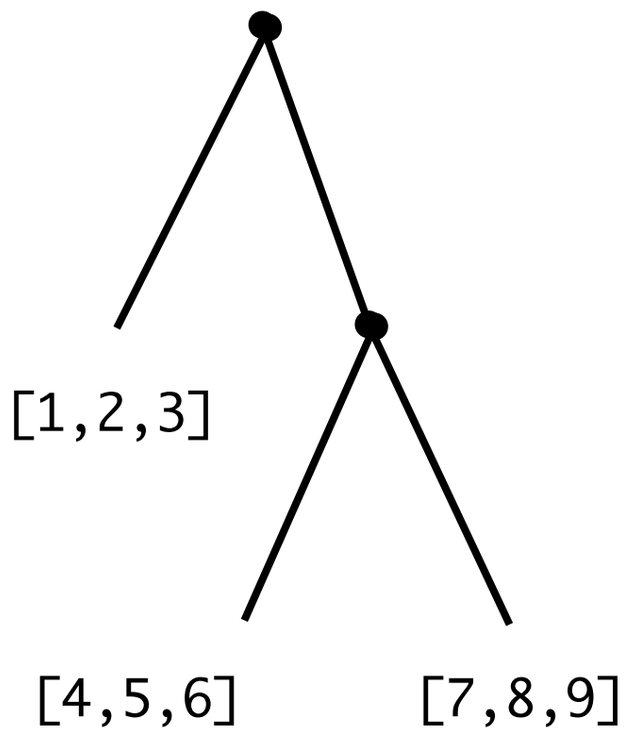
$$Tree\ s \times Vec\ n \times Vec\ n \times Tree\ t = Tree\ s \times Tree\ t$$

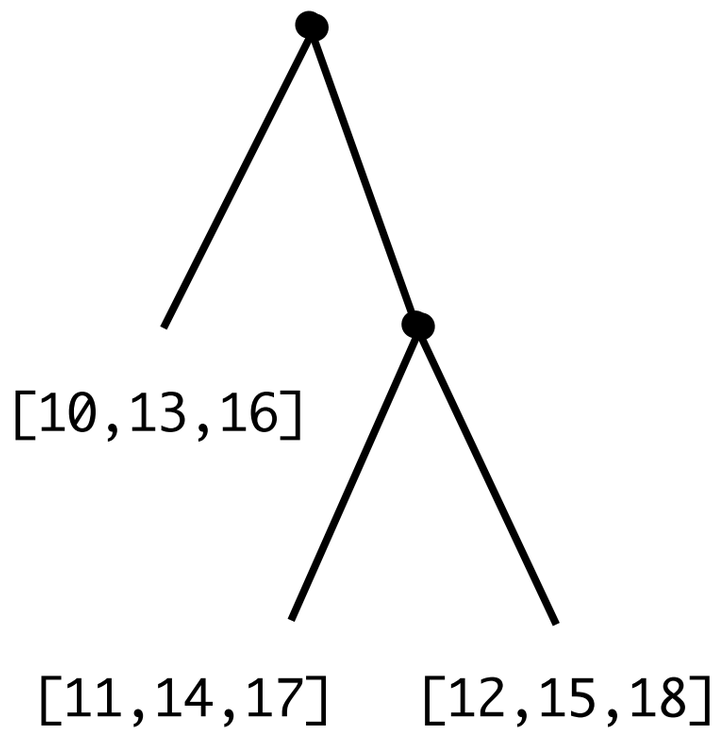
How it's done



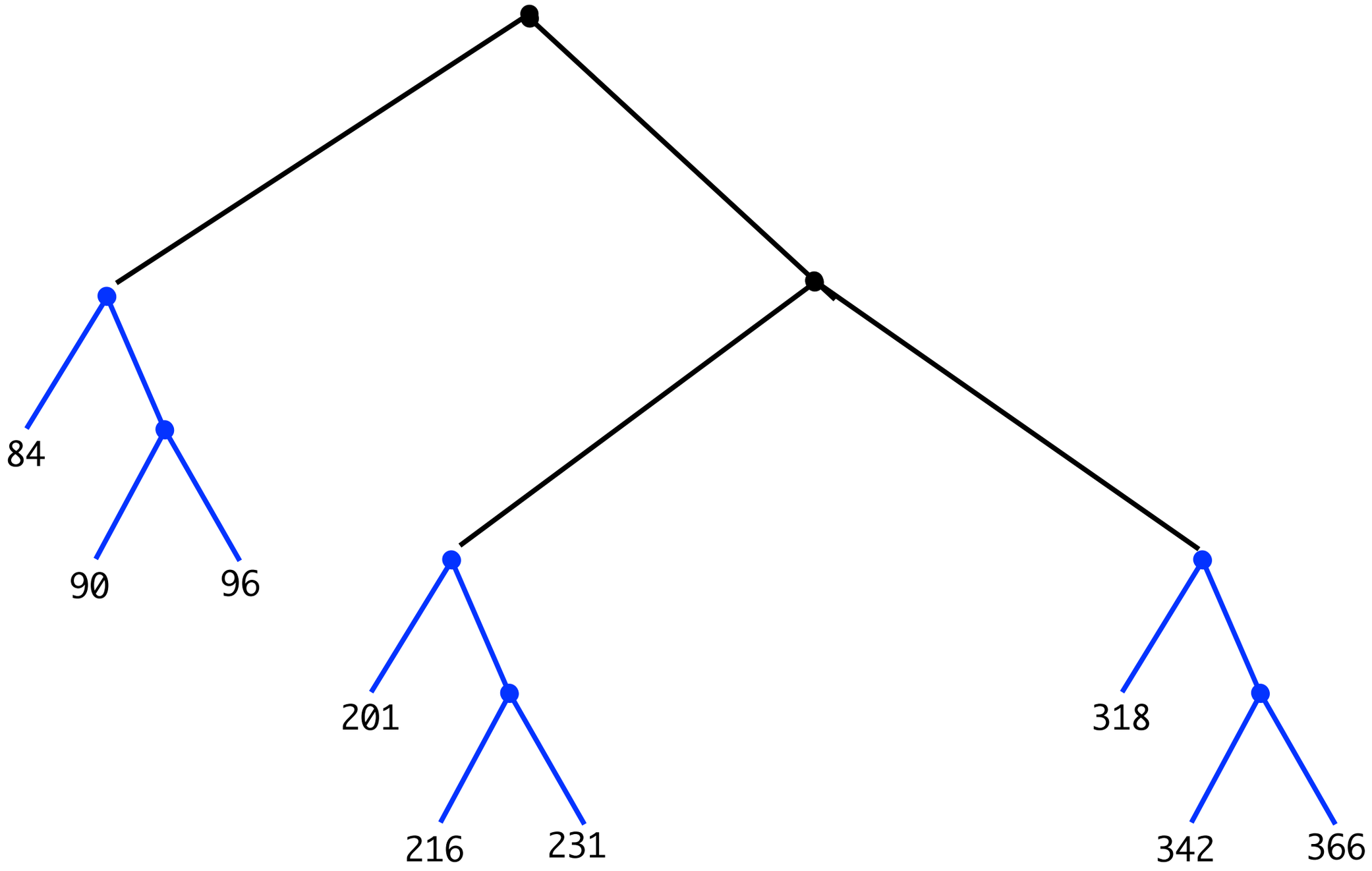
Should get a tree of trees

How it's done





How it's done



```
transpose :: (Traversable f1, Applicative f2)
          => f1 (f2 a) -> f2 (f1 a)
transpose = sequenceA

mmult :: (Num a, Applicative f1, Applicative f2, Applicative f3,
          Traversable f1, Traversable f2)
       => f1 (f2 a) -> f2 (f3 a) -> f1 (f3 a)
mmult m1 m2 = fmap (flip (fmap . dot) (transpose m2)) m1
```

DEMO

```
mmult :: f1 (f2 a) -> f2 (f3 a) -> f1 (f3 a)
```