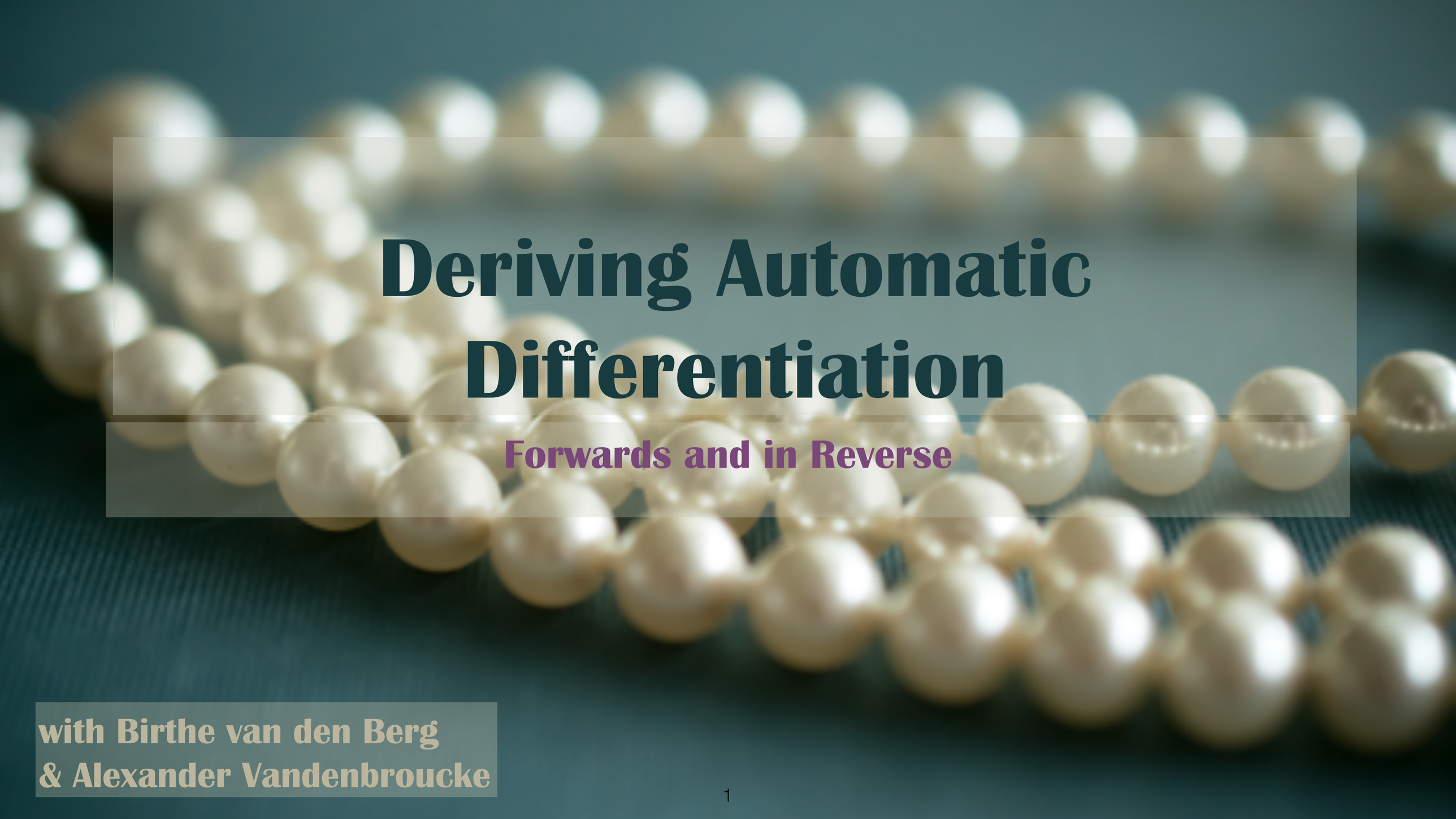
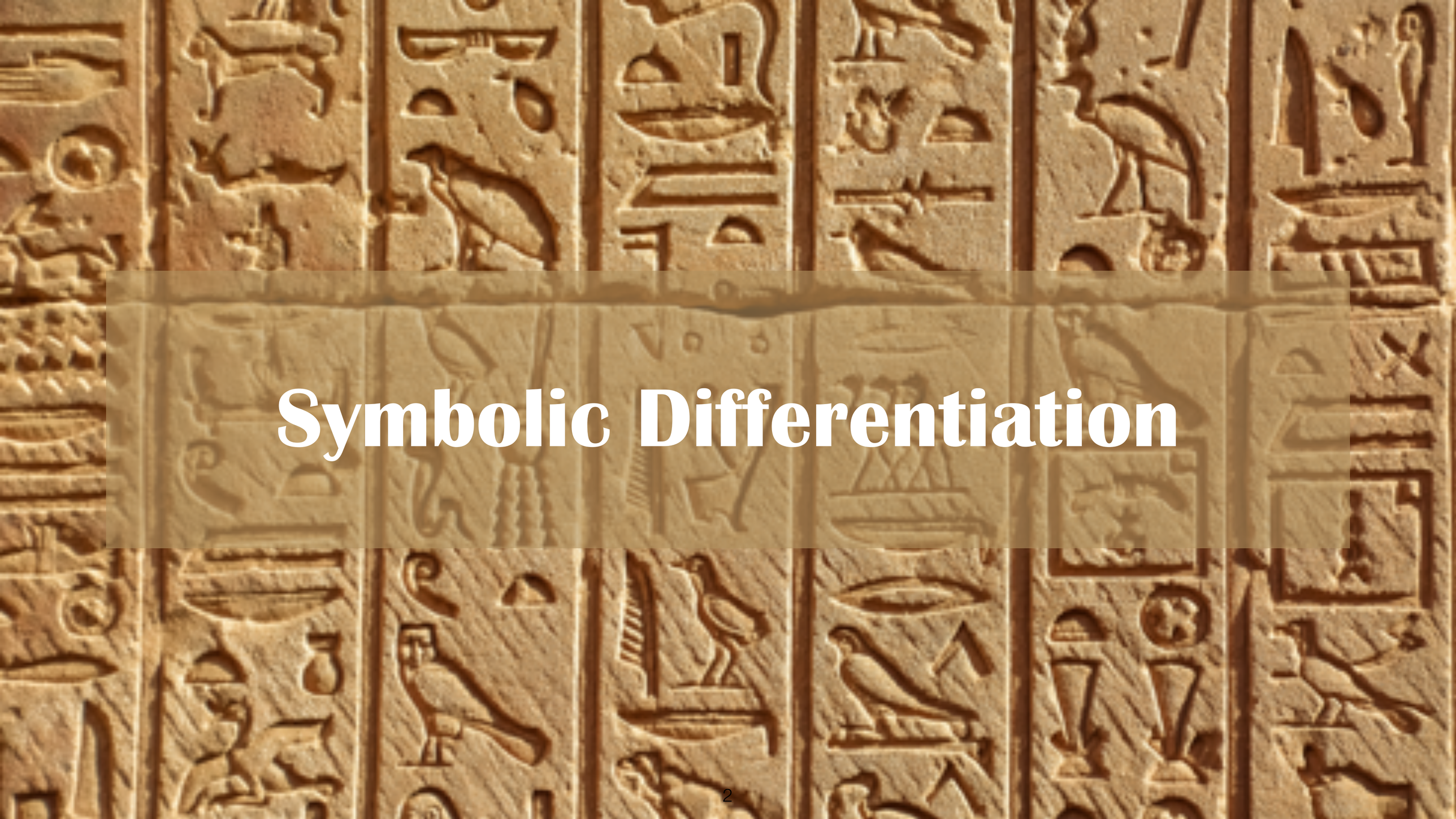


Deriving Automatic Differentiation

Forwards and in Reverse

with Birthe van den Berg
& Alexander Vandenbroucke



Symbolic Differentiation

Textbook Rules

$$\frac{\partial c}{\partial x} = 0$$

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & , x = y \\ 0 & , x \neq y \end{cases}$$

$$\frac{\partial (e_1 + e_2)}{\partial x} = \frac{\partial e_1}{\partial x} + \frac{\partial e_2}{\partial x}$$

$$\frac{\partial (e_1 \times e_2)}{\partial x} = \frac{\partial e_1}{\partial x} \times e_2 + e_1 \times \frac{\partial e_2}{\partial x}$$

Minimal Expression Language

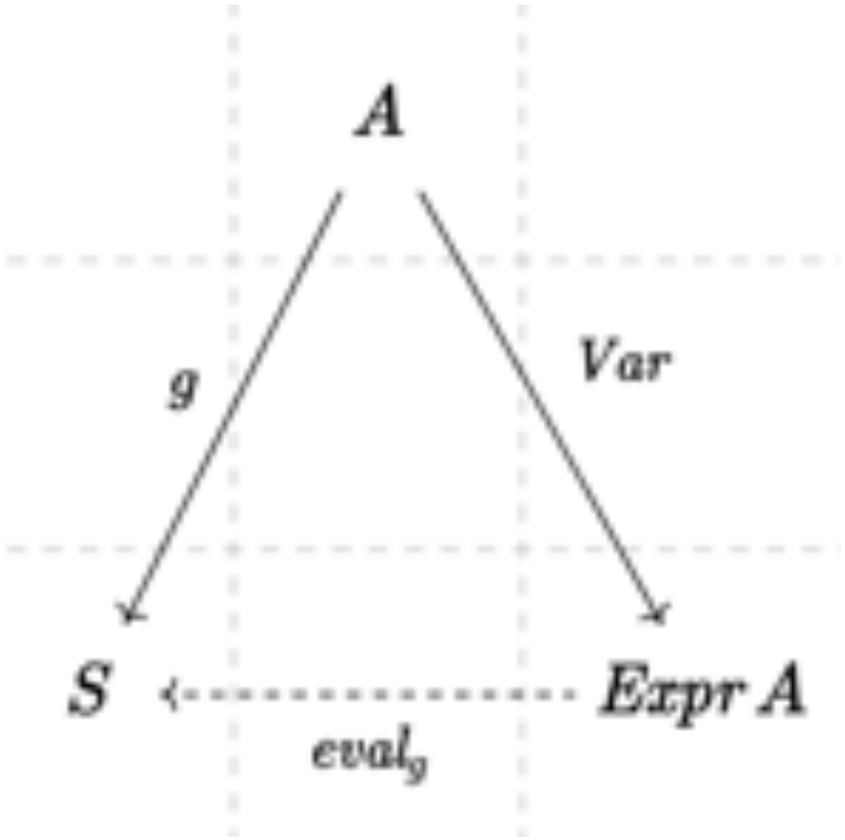
```
> class Semiring d where
>   zero  :: d
>   one   :: d
>   plus  :: d → d → d
>   times :: d → d → d

> instance Semiring Int where
>   zero  = 0
>   one   = 1
>   plus  = (+)
>   times = (*)
```

Symbolic Representation

```
> data Expr v
>    = Var v
>    | Zero
>    | One
>    | Plus (Expr v) (Expr v)
>    | Times (Expr v) (Expr v)
```

Look, it's free!

Free (Lawless) Semiring




```
> eval :: Semiring d ⇒ (v → d) → (Expr v → d)
> eval g (Var a)          =  g a
> eval g Zero             =  zero
> eval g One              =  one
> eval g (Plus e1 e2)     =  plus (eval g e1) (eval g e2)
> eval g (Times e1 e2)    =  times (eval g e1) (eval g e2)
```

It's a fold.

```
unique semiring homomorphism g = eval g . Var
```

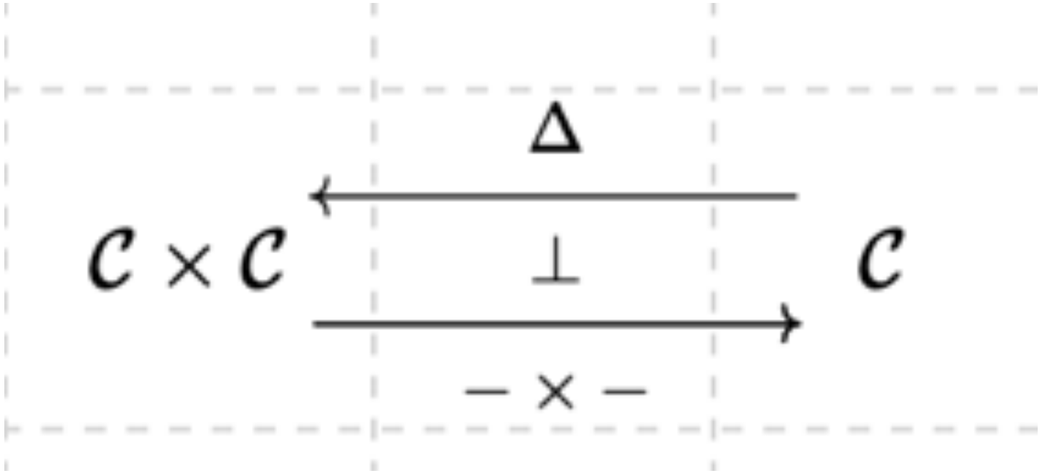
Symbolic Differentation

Is this a fold?

```
> derive :: Eq v ⇒ v → Expr v → Expr v
> derive a (Var b)         = if a == b then One else Zero
> derive a Zero            = Zero
> derive a One             = Zero
> derive a (Plus e1 e2)    = Plus (derive a e1) (derive a e2)
> derive a (Times e1 e2) = Plus (Times (derive a e1) e2)
                                (Times e1 (derive a e2))
```

Almost…

Adjoint Folds: Paramorphisms




```
> derive  :: Eq v ⇒ v → Expr v → Expr v
> derive' :: Eq v ⇒ v → Expr v → (Expr v, Expr v)
```

satisfying

```
derive x = snd . derive' x
```

Symbolic Derivation Catamorphism

Is this a homomorphism?

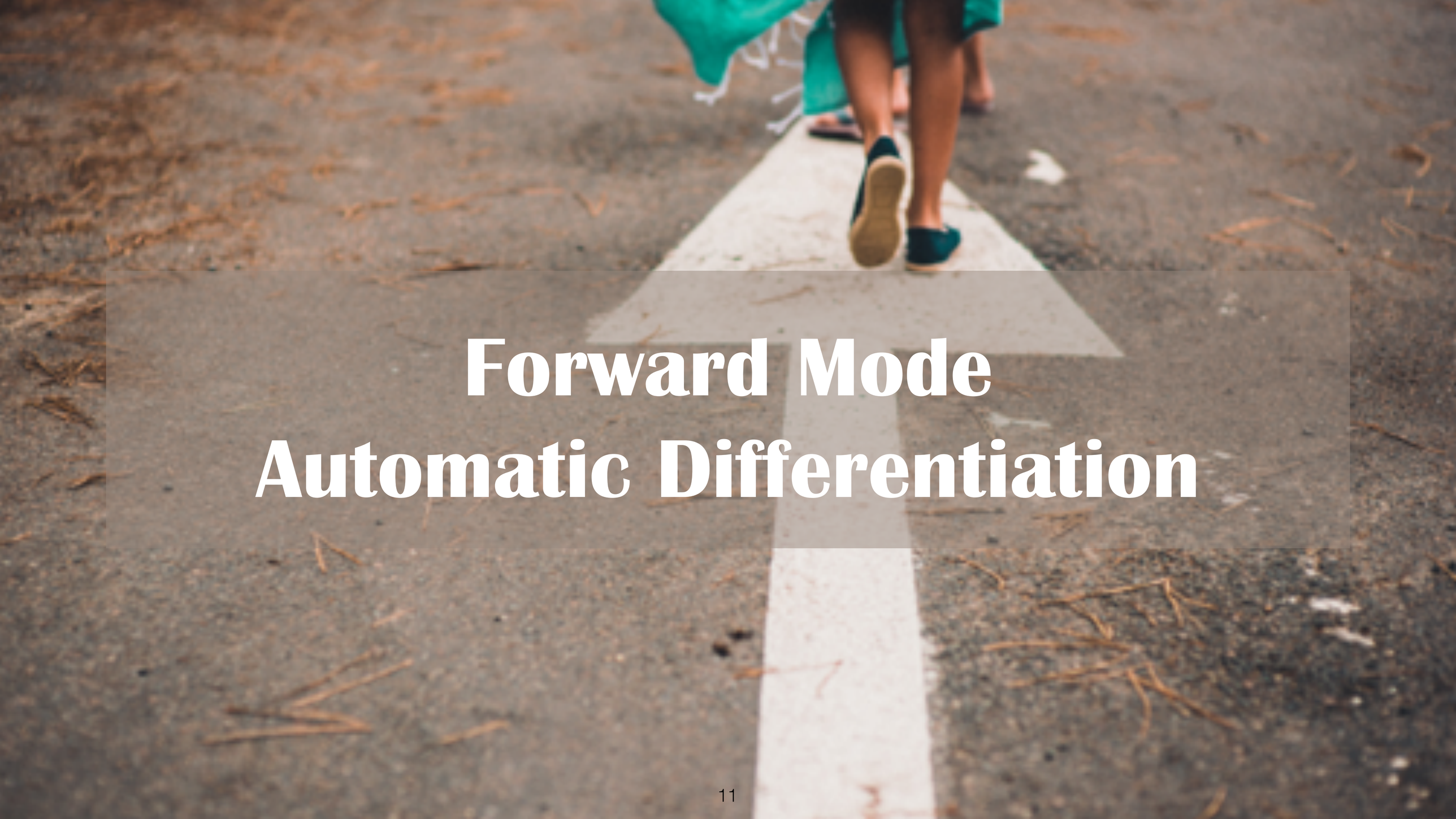
```
> derive' :: Eq v ⇒ v → Expr v → (Expr v, Expr v)
> derive' a (Var b)          = (Var b, if a == b then One else Zero)
> derive' a Zero             = (Zero, Zero)
> derive' a One              = (One, Zero)
> derive' a (Plus e1 e2)     = let (e1', de1') = derive' a e1'
>                                  (e2', de2') = derive' a e2'
>                              in (Plus e1' e2', Plus de1' de2')
> derive' a (Times e1 e2)    = let (e1', de1') = derive' a e1
>                                  (e2', de2') = derive' a e2
>                              in (Times e1' e2'
>                                  , Plus (Times de1' e2') (Times e1' de2'))
```

Yes, we can make it so.

Make it so, Number 2!

```
> data Dual d = D d d
>
> instance Semiring d ⇒ Semiring (Dual d) where
>   zero                        = D zero zero
>   one                         = D one zero
>   plus  (D f df) (D g dg)  = D (f `plus`  g) (df `plus` dg)
>   times (D f df) (D g dg)  = D (f `times` g)
>                                ((df `times`  g) `plus` (f `times` dg))
```

```
derive' x = eval g
  where
    g y = if x == y then one else zero
```



Forward Mode Automatic Differentiation

Evaluate Symbolic Derivative

Inefficient!
$O(n^2)$

```
evalDerive = (Eq v, Semiring d) ⇒ (v → d) → v → Expr v → Dual d
evalDerive gen x = eval gen . derive x
```

because of expression swell

$$\frac{\partial(e_1 \times e_2)}{\partial x} = \frac{\partial e_1}{\partial x} \times e_2 + e_1 \times \frac{\partial e_2}{\partial x}$$

Calculemus!

```
evalDerive gen x
= {- definition -}
eval gen . derive x
= {- definition -}
eval gen . sndD . eval gen'
= {- naturality of sndD -}
sndD . fmap (eval gen) . eval gen'
= {- eval fusion -}
sndD . eval (fmap gen . gen')
= {- simplify -}
 sndD . eval gen'' where
   gen'' y = D (gen y) (if x == y then one else zero)
```

$O(n^2)$

$O(n)$

Forward Mode

```
evalDerive = (Eq v, Semiring d) ⇒ (v → d) → v → Expr v → d
evalDerive gen x = eval gen . derive x
```

$O(n^2)$


```
forwardAD :: (Eq v, Semiring d) ⇒ (v → d) → v → Expr v → d
forwardAD gen x  = sndD . eval gen'' where
  gen'' y = D (gen y) (if x == y then one else zero)
```

$O(n)$

Derivatives for All

I want them all

One Partial Derivative

```
forwardAD :: (Eq v, Semiring d) ⇒ (v → d) → v → Expr v → Dual d
forwardAD gen x  = eval gen'' where
  gen'' y = D (gen y) (if x == y then one else zero)
```

O(n)

All Partial Derivatives

```
allForwardAD :: (Eq v, Semiring d) ⇒ (v → d) → Expr v → (v → Dual d)
allForwardAD gen = eval gen'' where
  gen'' y = \x → D (gen y) (if x == y then one else zero)
```

pointwise dual semiring

Evaluation shared is evaluation spared

pointwise dual semiring

```
v → Dual d
```

$$\begin{bmatrix} D\ e\ \frac{\partial e}{\partial x_1} \\ D\ e\ \frac{\partial e}{\partial x_2} \\ \vdots \\ D\ e\ \frac{\partial e}{\partial x_v} \end{bmatrix}$$

v expression values

v copies

v partial derivatives

more economic representation

```
(d, v → d)
```

$$e \begin{bmatrix} \frac{\partial e}{\partial x_1} \\ \frac{\partial e}{\partial x_2} \\ \cdots \\ \frac{\partial e}{\partial x_v} \end{bmatrix}$$

Evaluation shared is evaluation spared

more economic representation

$$(d,\ v \to d)$$

$$\left(AD'\ e_1 \begin{bmatrix} \frac{\partial e_1}{\partial x_1} \\ \frac{\partial e_1}{\partial x_2} \\ \vdots \\ \frac{\partial e_1}{\partial x_v} \end{bmatrix}\right) \otimes \left(AD'\ e_2 \begin{bmatrix} \frac{\partial e_2}{\partial x_1} \\ \frac{\partial e_2}{\partial x_2} \\ \vdots \\ \frac{\partial e_2}{\partial x_v} \end{bmatrix}\right) = AD'\ (e_1 \otimes e_2) \left(\begin{bmatrix} \frac{\partial e_1}{\partial x_1} \\ \frac{\partial e_1}{\partial x_2} \\ \vdots \\ \frac{\partial e_1}{\partial x_v} \end{bmatrix} \bullet e_2 \overline{\oplus} e_1 \bullet \begin{bmatrix} \frac{\partial e_2}{\partial x_1} \\ \frac{\partial e_2}{\partial x_2} \\ \vdots \\ \frac{\partial e_2}{\partial x_v} \end{bmatrix}\right)$$

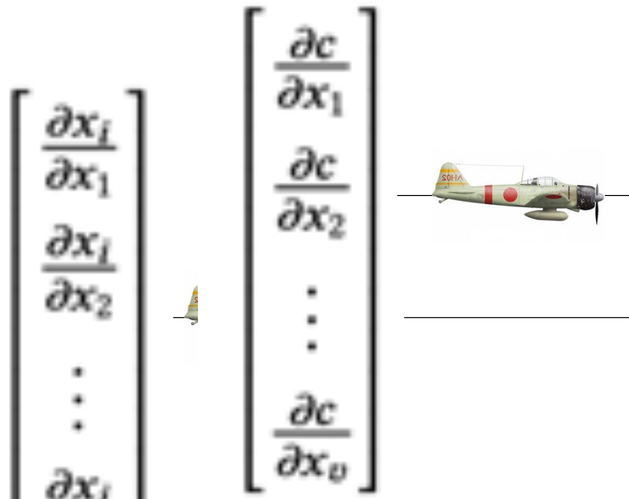
vector addition

scalar multiplication

Zeroing the zeros

sparse representation

`(d, v → d)` `(d, Map v d)`



$[]$

$[x_i \mapsto one]$

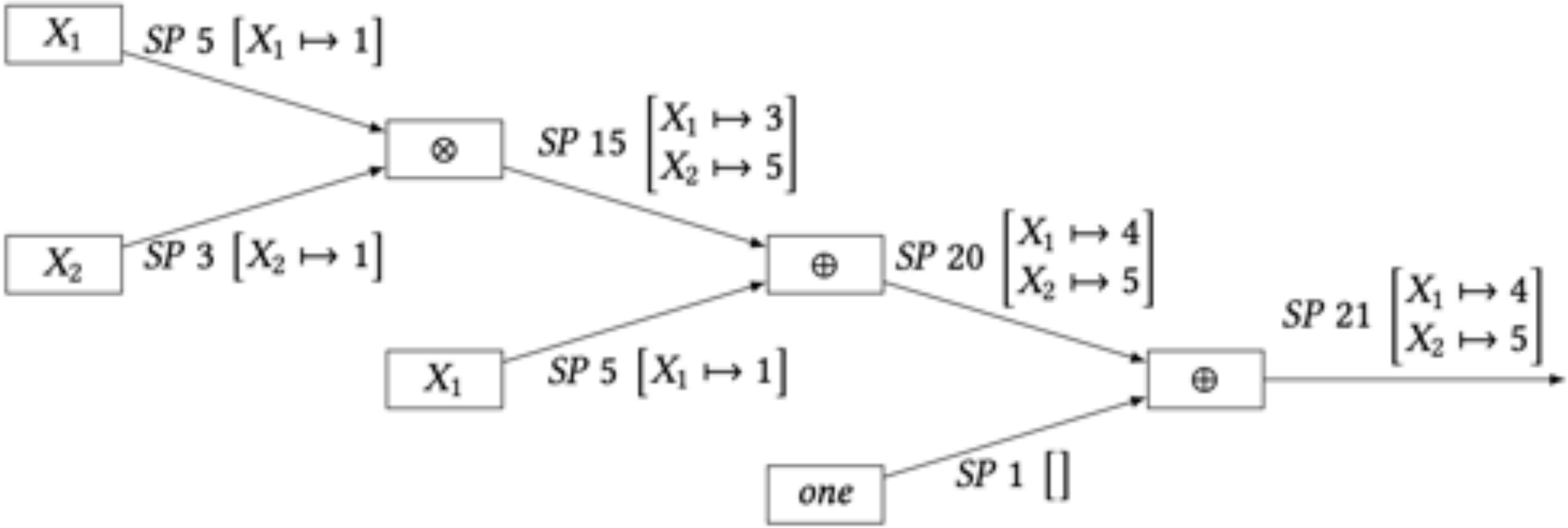
$\oplus$ preserve

$\otimes$ implicit zeros

All Derivatives, Sparsely

$$\textbf{data}\ SDual\ v\ d = SP\ \{fstUN :: d, sndUN :: Map\ v\ d\}$$

O(vn)

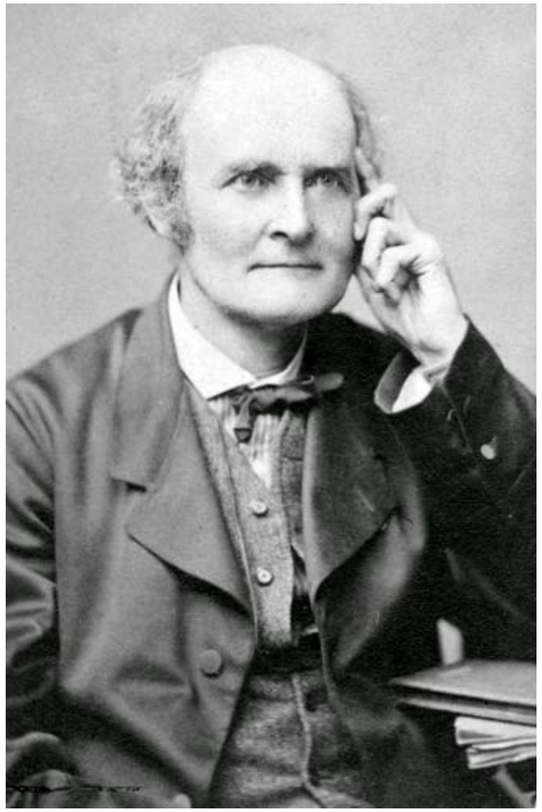


Forward in Reverse

Reverse Approach

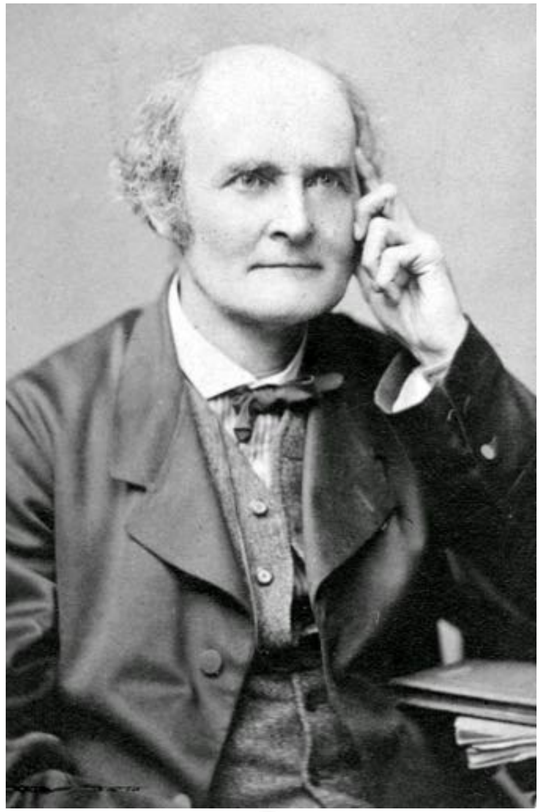
Step 1:

downwards accumulation



Step 2:

unions to upserts



Step 3:

mutation

🇬🇧-Style Cayley Representation

$$repr^L :: M \to (M \to M)$$
$$repr^L = \lambda m\ n \to n \otimes m$$

monoid of interest

$\langle M, \otimes, \epsilon \rangle$

left-nesting monoid

$\langle M \to M, flip\ (\circ), id_M \rangle$

$$abs^L :: (M \to M) \to M$$
$$abs^L = \lambda f \to f\ \epsilon$$

...for Scalar Multiplication

$$repr^{LM} = \lambda m\ n \rightarrow n \bullet m$$

vector of interest

left-nesting scalar multiplication

```
Map v d          (d → Map v d)
```

$$abs^{LM} = \lambda f \rightarrow f\ \epsilon$$

Step 1: Downwards Accumulation

upwards accumulation

```
(d, Map v d)
```

downwards accumulation

```
(d, d → Map v d)
```

left-nesting multiplication built-in

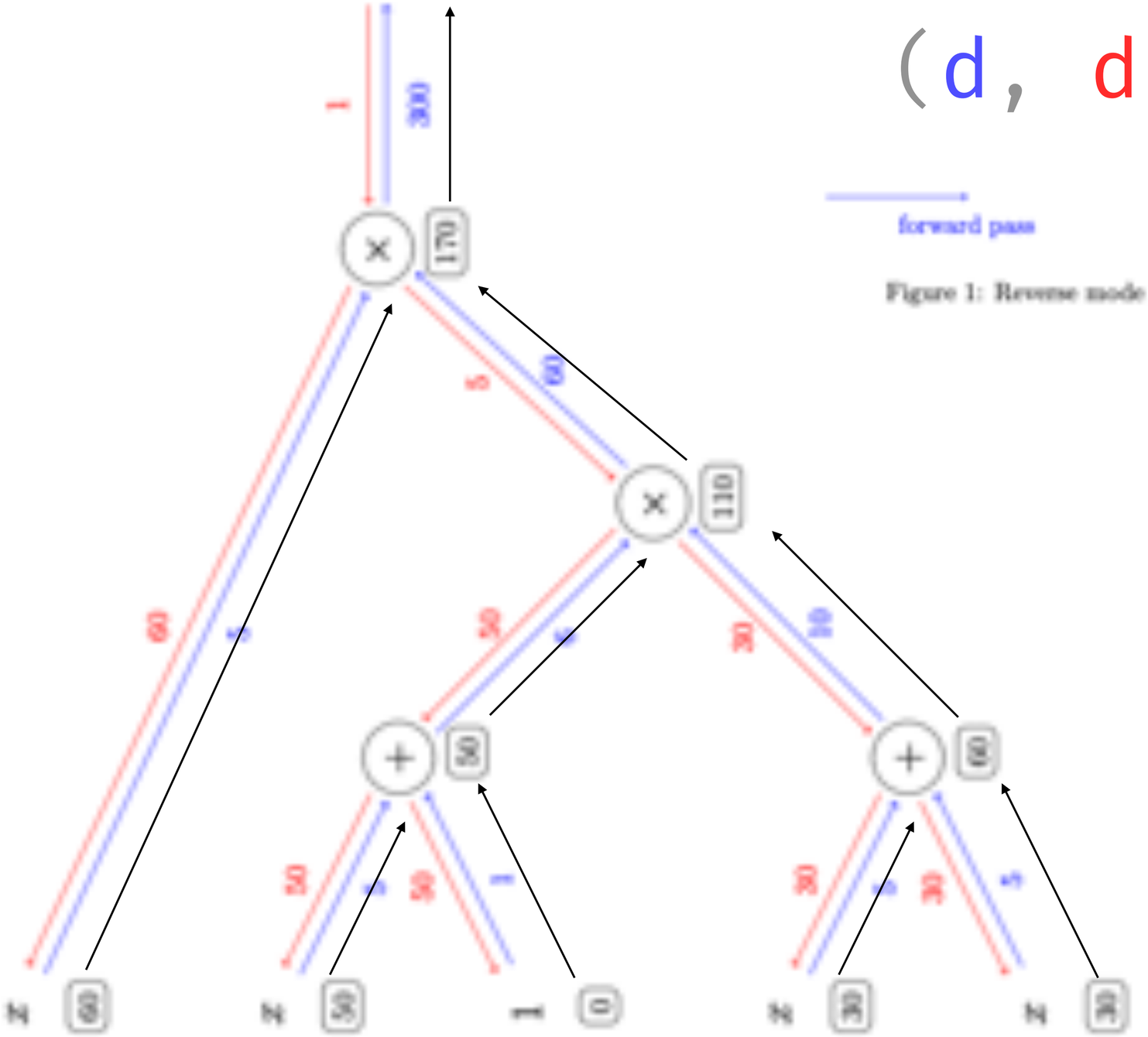
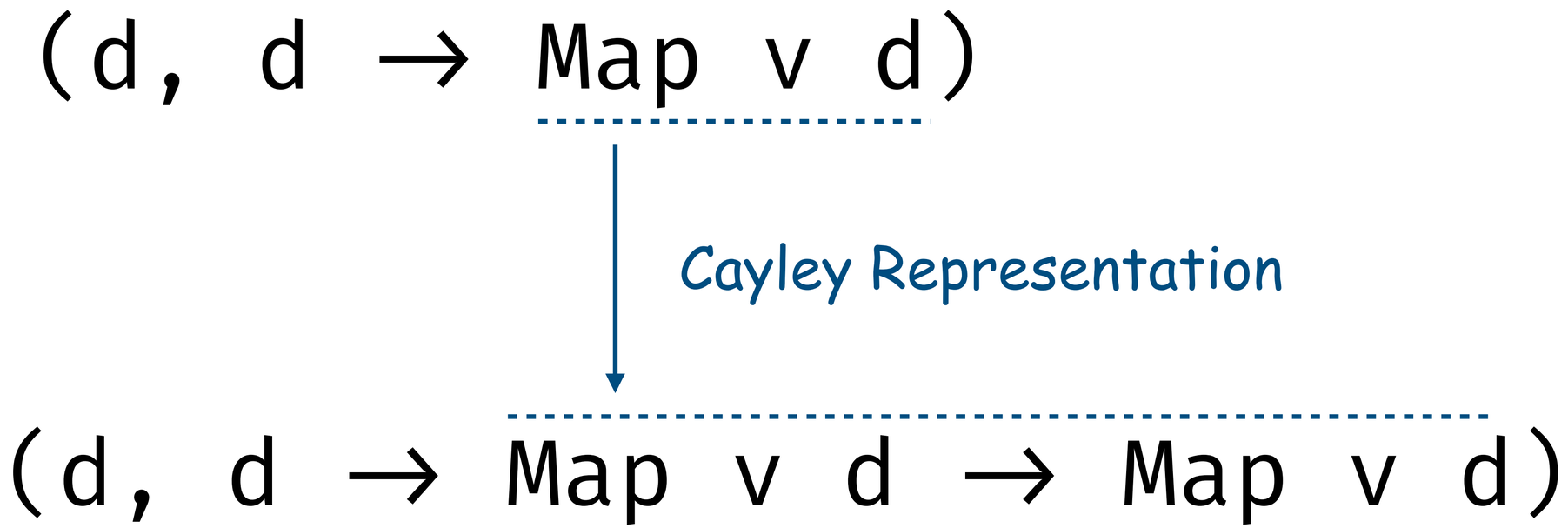


Figure 1: Reverse mode AD of $x \times (x + 1) \times (x + x)$, for $x = 5$

Step 2: Unions to Upserts



Step 2: Unions to Upserts

```
(d, d → Map v d → Map v d)
```

$$(SC\ f\ df) \oplus (SC\ g\ dg) = SC\ (f \oplus g)\ (\lambda n\ m \to dg\ n\ (df\ n\ m))$$
$$(SC\ f\ df) \otimes (SC\ g\ dg) = SC\ (f \otimes g)\ (\lambda n\ m \to df\ (n \otimes g)\ (dg\ (n \otimes f)\ m))$$

O(1)

O(n*log v)

$$reverseAD' :: (Ord\ v, Semiring\ d) \Rightarrow (v \to d) \to Expr\ v \to Map\ v\ d$$
$$reverseAD'\ gen\ e = snd^{SC}\ (eval\ gen'\ e)\ one\ \emptyset\ \textbf{where}$$
$$gen'\ y = SC\ (gen\ y)\ (\lambda n\ m \to insertWith\ (\oplus)\ y\ n\ m)$$

O(log v)

Step 3: Mutation

```
(d, d → Map v d → Map v d)
```

Mutable Array Updates

```
(d, d → IO ())
```

O(n + v)

actual processing

array allocation

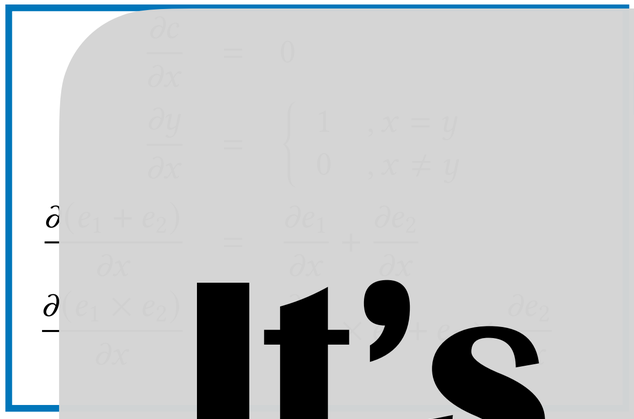
Summary

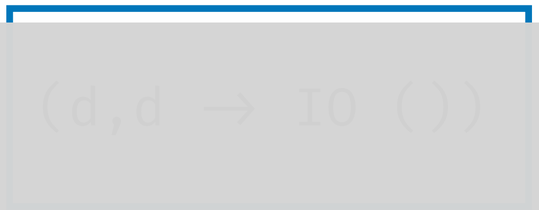
Summary

Textbook Rules

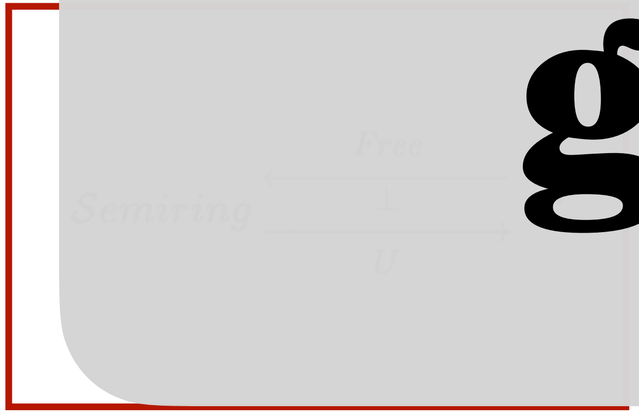
Dual Numbers

Reverse Dual Numbers





basic algebra/category theory



It's just attribute grammars!