Dual-Numbers Reverse AD, Efficiently

Tom J. Smeding, Matthijs I. L. Vákár

2022-06-29

Utrecht University

Overview

Naive beautiful code transformation

Overview

Naive beautiful code transformation

Overview

Naive beautiful code transformation

Semantics-preserving optimisations

Overview

Naive beautiful code transformation

Semantics-preserving optimisations

Efficient reverse AD on Haskell98

Overview

Naive beautiful code transformation

Semantics-preserving optimisations

Efficient reverse AD on Haskell98 *(mostly)*

Overview

Naive beautiful code transformation

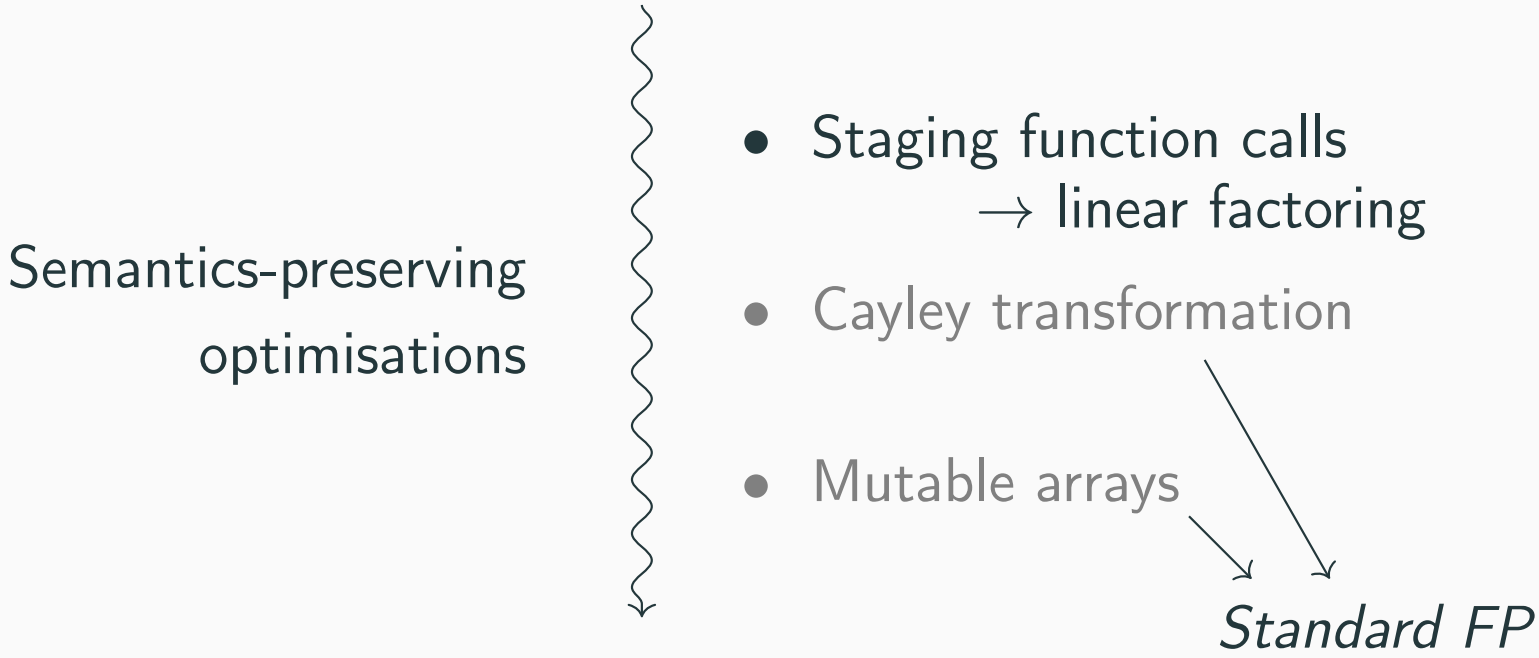
Semantics-preserving optimisations

- Staging function calls
  $\rightarrow$ linear factoring
- Cayley transformation
- Mutable arrays

Efficient reverse AD on Haskell98 *(mostly)*

Overview

Naive beautiful code transformation



Efficient reverse AD on Haskell98 *(mostly)*

Overview

Naive beautiful code transformation 1

Semantics-preserving optimisations

- Staging function calls 2
  $\rightarrow$ linear factoring
- Cayley transformation 3
- Mutable arrays 4

Standard FP

Efficient reverse AD on Haskell98 *(mostly)*

- Comparison with CHAD 5

Dual-numbers AD

Dual-numbers AD

Original program

$$\begin{aligned}&\lambda(x : \mathbb{R}, y : \mathbb{R}).\\ &\quad \textbf{let } z = x + y\\ &\quad \textbf{in } x \cdot z\end{aligned}$$

Forward AD

$$\begin{aligned}&\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}})\\ &\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}})).\\ &\qquad \textbf{let } (z, dz) = (x + y, dx + dy)\\ &\qquad \textbf{in } (x \cdot z, dz \cdot x + dx \cdot z)\end{aligned}$$

Reverse AD

$$\begin{aligned}&\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))\\ &\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))).\\ &\qquad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dx\ d + dy\ d)\\ &\qquad \textbf{in } (x \cdot z, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dz\ (x \cdot d) + dx\ (z \cdot d))\end{aligned}$$

Dual-numbers AD

Original program

$$\lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Forward AD

tangent

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}})$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}})).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, dx + dy)$$
$$\quad\quad \textbf{in } (x \cdot z, dz \cdot x + dx \cdot z)$$

Reverse AD

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

Dual-numbers AD

Original program

$$\lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Forward AD

tangent

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}})$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}})).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, dx + dy)$$
$$\quad\quad \textbf{in } (x \cdot z, dz \cdot x + dx \cdot z)$$

Reverse AD

cotangent

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

Dual-numbers AD

Original program

$$\lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Forward AD

tangent

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}})$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}})).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, dx + dy)$$
$$\quad\quad \textbf{in } (x \cdot z, dz \cdot x + dx \cdot z)$$

Reverse AD

cotangent

backpropagator

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

Dual-numbers AD

Original program

$$\lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Forward AD

tangent

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}})$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}})).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, dx + dy)$$
$$\quad\quad \textbf{in } (x \cdot z, dz \cdot x + dx \cdot z)$$

Reverse AD

cotangent

backpropagator

$$\lambda((x : \mathbb{R}, dx : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))$$
$$\quad ,(y : \mathbb{R}, dy : \underline{\mathbb{R}} \multimap (\underline{\mathbb{R}}, \underline{\mathbb{R}}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \underline{\mathbb{R}}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

- Inversion of control flow

Dual-numbers *reverse* AD

Dual-numbers AD: code transformation

$$\sigma, \tau ::= \mathbb{R} \mid () \mid (\sigma, \tau) \mid \sigma \to \tau \mid \mathsf{Int}$$

$$s, t ::= (x : \tau) \mid () \mid (s, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid s\ t$$
$$\mid \quad \lambda(x : \tau).\ t \mid \textbf{let}\ x : \tau = s\ \textbf{in}\ t \mid r \mid op(t_1, \ldots, t_n)$$

Dual-numbers AD: code transformation

$$\sigma, \tau ::= \mathbb{R} \mid () \mid (\sigma, \tau) \mid \sigma \to \tau \mid \mathsf{Int}$$

$$s, t ::= (x : \tau) \mid () \mid (s, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid s\ t$$
$$\mid \quad \lambda(x : \tau).\ t \mid \textbf{let}\ x : \tau = s\ \textbf{in}\ t \mid r \mid op(t_1, \ldots, t_n)$$

$$\begin{aligned}
\mathbf{D}_c[\mathbb{R}] &= \boxed{\begin{array}{c}\textbf{Forward AD:}\\ (\mathbb{R}, \underline{\mathbb{R}})\end{array}} \boxed{\begin{array}{c}\textbf{Reverse AD:}\\ (\mathbb{R}, \underline{\mathbb{R}} \multimap c)\end{array}} \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(\sigma, \tau)] &= (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \\
\mathbf{D}_c[\sigma \to \tau] &= \mathbf{D}_c[\sigma] \to \mathbf{D}_c[\tau] \\
\mathbf{D}_c[\mathsf{Int}] &= \mathsf{Int}
\end{aligned}$$

On environments: $\mathbf{D}_c[\varepsilon] = \varepsilon \qquad \mathbf{D}_c[\Gamma, x : \tau] = \mathbf{D}_c[\Gamma], x : \mathbf{D}_c[\tau]$

Dual-numbers AD: code transformation

$$\sigma, \tau ::= \mathbb{R} \mid () \mid (\sigma, \tau) \mid \sigma \to \tau \mid \mathsf{Int}$$

$$s, t ::= (x : \tau) \mid () \mid (s, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid s\ t$$
$$\mid \ \lambda(x : \tau).\ t \mid \textbf{let}\ x : \tau = s\ \textbf{in}\ t \mid r \mid op(t_1, \ldots, t_n)$$

$$\begin{aligned}
\mathbf{D}_c[\mathbb{R}] &= \boxed{\begin{array}{c}\textbf{Forward AD:}\\ (\mathbb{R}, \underline{\mathbb{R}})\end{array}} \quad \boxed{\begin{array}{c}\textbf{Reverse AD:}\\ (\mathbb{R}, \underline{\mathbb{R}} \multimap c)\end{array}} \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(\sigma, \tau)] &= (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \\
\mathbf{D}_c[\sigma \to \tau] &= \mathbf{D}_c[\sigma] \to \mathbf{D}_c[\tau] \\
\mathbf{D}_c[\mathsf{Int}] &= \mathsf{Int}
\end{aligned}$$

Ex.

$((\mathbb{R}, \mathsf{Int} \to \mathbb{R}), (\mathbb{R} \to \mathbb{R})) \quad \rightsquigarrow$
$((\mathbf{D}_c[\mathbb{R}], \mathsf{Int} \to \mathbf{D}_c[\mathbb{R}]), (\mathbf{D}_c[\mathbb{R}] \to \mathbf{D}_c[\mathbb{R}]))$

On environments: $\mathbf{D}_c[\varepsilon] = \varepsilon \qquad \mathbf{D}_c[\Gamma, x : \tau] = \mathbf{D}_c[\Gamma], x : \mathbf{D}_c[\tau]$

Dual-numbers AD: code transformation

$$\sigma, \tau ::= \mathbb{R} \mid () \mid (\sigma, \tau) \mid \sigma \to \tau \mid \mathsf{Int}$$

$$s, t ::= (x : \tau) \mid () \mid (s, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid s\ t$$
$$\mid \ \lambda(x : \tau).\ t \mid \mathbf{let}\ x : \tau = s\ \mathbf{in}\ t \mid r \mid op(t_1, \ldots, t_n)$$

$$\begin{aligned}
\mathbf{D}_c[\mathbb{R}] &= \boxed{\begin{array}{c}\textbf{Forward AD:}\\ (\mathbb{R}, \mathbb{R})\end{array}} \boxed{\begin{array}{c}\textbf{Reverse AD:}\\ (\mathbb{R}, \mathbb{R} \multimap c)\end{array}} \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(\sigma, \tau)] &= (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \\
\mathbf{D}_c[\sigma \to \tau] &= \mathbf{D}_c[\sigma] \to \mathbf{D}_c[\tau] \\
\mathbf{D}_c[\mathsf{Int}] &= \mathsf{Int}
\end{aligned}$$

Ex.

$((\mathbb{R}, \mathsf{Int} \to \mathbb{R}), (\mathbb{R} \to \mathbb{R})) \quad \rightsquigarrow$
$((\mathbf{D}_c[\mathbb{R}], \mathsf{Int} \to \mathbf{D}_c[\mathbb{R}]), (\mathbf{D}_c[\mathbb{R}] \to \mathbf{D}_c[\mathbb{R}]))$

On environments: $\mathbf{D}_c[\varepsilon] = \varepsilon \qquad \mathbf{D}_c[\Gamma, x : \tau] = \mathbf{D}_c[\Gamma], x : \mathbf{D}_c[\tau]$

Dual-numbers AD: code transformation

$$\sigma, \tau ::= \mathbb{R} \mid () \mid (\sigma, \tau) \mid \sigma \to \tau \mid \mathsf{Int}$$

$$s, t ::= (x : \tau) \mid () \mid (s, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid s\ t$$
$$\mid \ \lambda(x : \tau).\ t \mid \mathbf{let}\ x : \tau = s\ \mathbf{in}\ t \mid r \mid op(t_1, \ldots, t_n)$$

$$\begin{aligned}
\mathbf{D}_c[\mathbb{R}] &= \boxed{\text{Forward AD: } (\mathbb{R}, \mathbb{R})} \quad \boxed{\text{Reverse AD: } (\mathbb{R}, \mathbb{R} \multimap c)} \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(\sigma, \tau)] &= (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \\
\mathbf{D}_c[\sigma \to \tau] &= \mathbf{D}_c[\sigma] \to \mathbf{D}_c[\tau] \\
\mathbf{D}_c[\mathsf{Int}] &= \mathsf{Int}
\end{aligned}$$

Ex.

$((\mathbb{R}, \mathsf{Int} \to \mathbb{R}), (\mathbb{R} \to \mathbb{R})) \quad \rightsquigarrow$
$((\mathbf{D}_c[\mathbb{R}], \mathsf{Int} \to \mathbf{D}_c[\mathbb{R}]), (\mathbf{D}_c[\mathbb{R}] \to \mathbf{D}_c[\mathbb{R}]))$

On environments: $\mathbf{D}_c[\varepsilon] = \varepsilon \qquad \mathbf{D}_c[\Gamma, x : \tau] = \mathbf{D}_c[\Gamma], x : \mathbf{D}_c[\tau]$

Dual-numbers reverse AD

$$\mathbf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathbf{D}_c[(\sigma, \tau)] = (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \qquad \ldots$$

$$\begin{aligned}
\mathbf{D}_c[x : \tau] &= x : \mathbf{D}_c[\tau] \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(s, t)] &= (\mathbf{D}_c[s], \mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[s\ t] &= \mathbf{D}_c[s]\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathbf{D}_c[\tau]).\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\textbf{let } x : \tau = s \textbf{ in } t] &= \textbf{let } x : \mathbf{D}_c[\tau] = \mathbf{D}_c[s] \textbf{ in } \mathbf{D}_c[t] \\
\mathbf{D}_c[r] &= \ldots \\
\mathbf{D}_c[op(t_1, \ldots, t_n)] &= \ldots
\end{aligned}$$

$$\frac{\Gamma \vdash t : \tau}{\mathbf{D}_c[\Gamma] \vdash \mathbf{D}_c[t] : \mathbf{D}_c[\tau]}$$

Dual-numbers reverse AD

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathsf{D}_c[(\sigma, \tau)] = (\mathsf{D}_c[\sigma], \mathsf{D}_c[\tau]) \qquad \ldots$$

$$\begin{aligned}
\mathsf{D}_c[x : \tau] &= x : \mathsf{D}_c[\tau] \\
\mathsf{D}_c[()] &= () \\
\mathsf{D}_c[(s, t)] &= (\mathsf{D}_c[s], \mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[s\ t] &= \mathsf{D}_c[s]\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathsf{D}_c[\tau]).\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\mathbf{let}\ x : \tau = s\ \mathbf{in}\ t] &= \mathbf{let}\ x : \mathsf{D}_c[\tau] = \mathsf{D}_c[s]\ \mathbf{in}\ \mathsf{D}_c[t] \\
\mathsf{D}_c[r] &= \ldots \\
\mathsf{D}_c[op(t_1, \ldots, t_n)] &= \ldots
\end{aligned}$$

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathsf{D}_c[(\sigma, \tau)] = (\mathsf{D}_c[\sigma], \mathsf{D}_c[\tau]) \qquad \ldots$$

$$\begin{aligned}
\mathsf{D}_c[x : \tau] &= x : \mathsf{D}_c[\tau] \\
\mathsf{D}_c[()] &= () \\
\mathsf{D}_c[(s, t)] &= (\mathsf{D}_c[s], \mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[s\ t] &= \mathsf{D}_c[s]\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathsf{D}_c[\tau]).\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\mathbf{let}\ x : \tau = s\ \mathbf{in}\ t] &= \mathbf{let}\ x : \mathsf{D}_c[\tau] = \mathsf{D}_c[s]\ \mathbf{in}\ \mathsf{D}_c[t] \\
\mathsf{D}_c[r] &= (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0}) \\
\mathsf{D}_c[op(t_1, \ldots, t_n)] &= \ldots
\end{aligned}$$

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathsf{D}_c[(\sigma, \tau)] = (\mathsf{D}_c[\sigma], \mathsf{D}_c[\tau]) \qquad \ldots$$

$$\begin{aligned}
\mathsf{D}_c[x : \tau] &= x : \mathsf{D}_c[\tau] \\
\mathsf{D}_c[()] &= () \\
\mathsf{D}_c[(s, t)] &= (\mathsf{D}_c[s], \mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathsf{D}_c[t]) \\
\mathsf{D}_c[s\ t] &= \mathsf{D}_c[s]\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathsf{D}_c[\tau]).\ \mathsf{D}_c[t] \\
\mathsf{D}_c[\mathbf{let}\ x : \tau = s\ \mathbf{in}\ t] &= \mathbf{let}\ x : \mathsf{D}_c[\tau] = \mathsf{D}_c[s]\ \mathbf{in}\ \mathsf{D}_c[t] \\
\mathsf{D}_c[r] &= (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0}) \\
\mathsf{D}_c[op(t_1, \ldots, t_n)] &= \mathbf{let}\ (x_1, d_1) = \mathsf{D}_c[t_1]\ \ldots\ (x_n, d_n) = \mathsf{D}_c[t_n] \\
&\quad\ \mathbf{in}\ (op(x_1, \ldots, x_n) \\
&\qquad ,\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad\qquad d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))
\end{aligned}$$

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD

$$\mathbf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathbf{D}_c[(\sigma, \tau)] = (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \qquad \ldots$$

$$\begin{aligned}
\mathbf{D}_c[x : \tau] &= x : \mathbf{D}_c[\tau] \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(s, t)] &= (\mathbf{D}_c[s], \mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[s\ t] &= \mathbf{D}_c[s]\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathbf{D}_c[\tau]).\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\mathbf{let}\ x : \tau = s\ \mathbf{in}\ t] &= \mathbf{let}\ x : \mathbf{D}_c[\tau] = \mathbf{D}_c[s]\ \mathbf{in}\ \mathbf{D}_c[t] \\
\mathbf{D}_c[r] &= (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0}) \\
\mathbf{D}_c[op(t_1, \ldots, t_n)] &= \mathbf{let}\ (x_1, d_1) = \mathbf{D}_c[t_1]\ \ldots\ (x_n, d_n) = \mathbf{D}_c[t_n] \\
&\quad\ \mathbf{in}\ (op(x_1, \ldots, x_n) \\
&\qquad ,\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad\qquad d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))
\end{aligned}$$

$$\frac{\Gamma \vdash t : \tau}{\mathbf{D}_c[\Gamma] \vdash \mathbf{D}_c[t] : \mathbf{D}_c[\tau]}$$

$(op : \mathbb{R}^n \to \mathbb{R})$

Dual-numbers reverse AD

$$\mathbf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \qquad \mathbf{D}_c[(\sigma, \tau)] = (\mathbf{D}_c[\sigma], \mathbf{D}_c[\tau]) \qquad \ldots$$

$$\frac{\Gamma \vdash t : \tau}{\mathbf{D}_c[\Gamma] \vdash \mathbf{D}_c[t] : \mathbf{D}_c[\tau]}$$

$$\begin{aligned}
\mathbf{D}_c[x : \tau] &= x : \mathbf{D}_c[\tau] \\
\mathbf{D}_c[()] &= () \\
\mathbf{D}_c[(s, t)] &= (\mathbf{D}_c[s], \mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{fst}(t)] &= \mathsf{fst}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[\mathsf{snd}(t)] &= \mathsf{snd}(\mathbf{D}_c[t]) \\
\mathbf{D}_c[s\ t] &= \mathbf{D}_c[s]\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\lambda(x : \tau).\ t] &= \lambda(x : \mathbf{D}_c[\tau]).\ \mathbf{D}_c[t] \\
\mathbf{D}_c[\mathbf{let}\ x : \tau = s\ \mathbf{in}\ t] &= \mathbf{let}\ x : \mathbf{D}_c[\tau] = \mathbf{D}_c[s]\ \mathbf{in}\ \mathbf{D}_c[t] \\
\mathbf{D}_c[r] &= (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0}) \\
\mathbf{D}_c[op(t_1, \ldots, t_n)] &= \mathbf{let}\ (x_1, d_1) = \mathbf{D}_c[t_1]\ \ldots\ (x_n, d_n) = \mathbf{D}_c[t_n] \\
&\quad\ \mathbf{in}\ (op(x_1, \ldots, x_n) \\
&\qquad ,\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad\qquad d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))
\end{aligned}$$

$(op : \mathbb{R}^n \to \mathbb{R})$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma \to \tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma \to \tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to \mathsf{D}_c[\tau]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to \mathsf{D}_c[\mathbb{R}]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to \mathsf{D}_c[\mathbb{R}]}$$

Dual-numbers reverse AD: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap c)}$$

Dual-numbers reverse AD: top-level function

$c = \sigma$

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{D}_c[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap c)}$$

Dual-numbers reverse AD: top-level function

$c = \sigma$

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathbf{D}_\sigma[\Gamma] \vdash \mathbf{D}_\sigma[t] : \mathbf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$

Dual-numbers reverse AD: top-level function

$c = \sigma$

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathbf{D}_\sigma[\Gamma] \vdash \mathbf{D}_\sigma[t] : \mathbf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$

Dual-numbers reverse AD: top-level function

$c = \sigma$

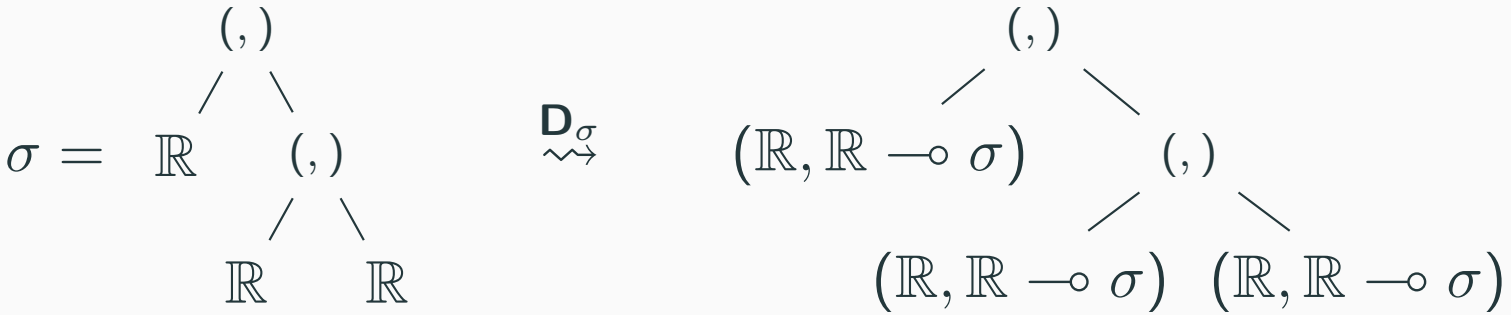
$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathbf{D}_\sigma[\Gamma] \vdash \mathbf{D}_\sigma[t] : \mathbf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$

$$\sigma = \begin{array}{ccccc} & & (,) & & \\ & \swarrow & & \searrow & \\ \mathbb{R} & & & & (,) \\ & & & \swarrow & & \searrow \\ & & \mathbb{R} & & & & \mathbb{R} \end{array}$$

Dual-numbers reverse AD: top-level function

$c = \sigma$

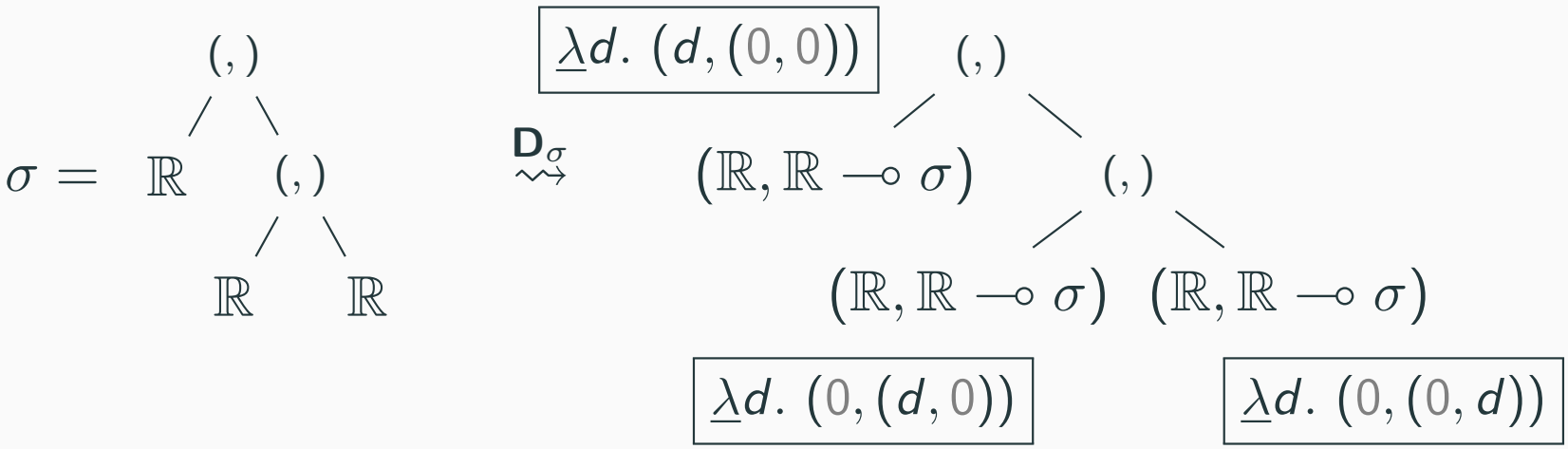
$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathbf{D}_\sigma[\Gamma] \vdash \mathbf{D}_\sigma[t] : \mathbf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$



Dual-numbers reverse AD: top-level function

$c = \sigma$

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathbf{D}_\sigma[\Gamma] \vdash \mathbf{D}_\sigma[t] : \mathbf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$



But is it fast?

The example again

Original program

$$t = \lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Reverse AD

$$\mathbf{D}_c[t] = \lambda((x : \mathbb{R}, dx : \mathbb{R} \multimap (\mathbb{R}, \mathbb{R}))$$
$$\quad ,(y : \mathbb{R}, dy : \mathbb{R} \multimap (\mathbb{R}, \mathbb{R}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

The example again

Original program

$$t = \lambda(x : \mathbb{R}, y : \mathbb{R}).$$
$$\quad \textbf{let } z = x + y$$
$$\quad \textbf{in } x \cdot z$$

Reverse AD

$$\mathbf{D}_c[t] = \lambda((x : \mathbb{R}, dx : \mathbb{R} \multimap (\mathbb{R}, \mathbb{R}))$$
$$\quad ,(y : \mathbb{R}, dy : \mathbb{R} \multimap (\mathbb{R}, \mathbb{R}))).$$
$$\quad\quad \textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\quad\quad \textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

The example again

Original program

$$t = \lambda(x : \mathbb{R}, y : \mathbb{R}).$$

$$+ y$$

Reverse AD

$$\mathbf{D}_c[t] = \lambda((x : \mathbb{R}, c$$

$$, (y : \mathbb{R}, c$$

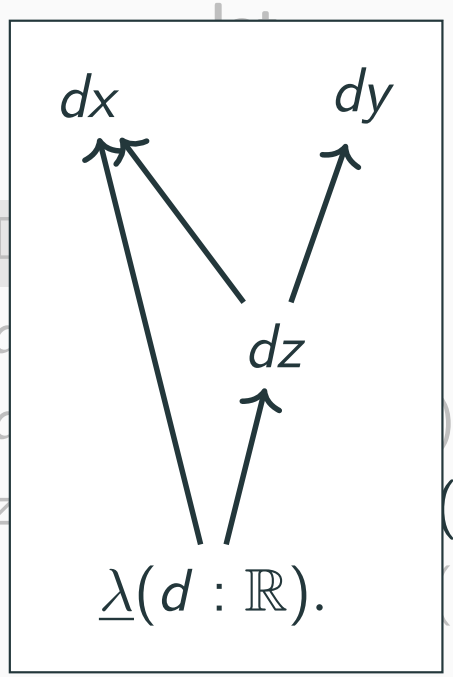
$$).$$

$$\textbf{let } (z$$

$$(d : \mathbb{R}).\ dx\ d + dy\ d)$$

$$\textbf{in } (x$$

$$(x \cdot d) + dx\ (z \cdot d))$$



The complexity issue

$$
\begin{aligned}
t_{\mathsf{bad}} = \lambda(x_0 : \mathbb{R}).\ & \textbf{let } x_1 = op(x_0, x_0) \qquad (\text{for any } op) \\
& \textbf{in let } x_2 = op(x_1, x_1) \\
& \quad \vdots \\
& \textbf{in let } x_n = op(x_{n-1}, x_{n-1}) \\
& \textbf{in } x_n
\end{aligned}
$$

The complexity issue

$$
\begin{aligned}
t_{\mathsf{bad}} = \lambda(x_0 : \mathbb{R}).\ & \textbf{let } x_1 = op(x_0, x_0) \qquad \text{(for any } op\text{)} \\
& \textbf{in let } x_2 = op(x_1, x_1) \\
& \quad \vdots \\
& \textbf{in let } x_n = op(x_{n-1}, x_{n-1}) \\
& \textbf{in } x_n
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{D}_c[t_{\mathsf{bad}}] = \lambda(x_0 : \mathbb{R}, \bar{x}_0 : \mathbb{R} \multimap c). \\
&\quad \textbf{let } (x_1, \bar{x}_1) = (op(x_0, x_0), \underline{\lambda}(d : \mathbb{R}).\ \bar{x}_0\ (\partial_1 op(x_0, x_0) \cdot d) + \\
&\qquad\qquad\qquad\qquad\qquad\qquad \bar{x}_0\ (\partial_2 op(x_0, x_0) \cdot d)) \\
&\quad \vdots \\
&\quad \textbf{in let } (x_n, \bar{x}_n) = (op(x_{n-1}, x_{n-1}), \underline{\lambda}(d : \mathbb{R}).\ \bar{x}_{n-1}\ (\partial_1 op(x_{n-1}, \ldots \\
&\quad \textbf{in } (x_n, \bar{x}_n)
\end{aligned}
$$

The complexity issue

$t_{\mathsf{bad}} = \lambda(x_0 : \mathbb{R}).\ \mathbf{let}\ x$ $x_0, x_0)$ (for any op)

$\mathbf{in}\ \mathbf{le}$ $op(x_1, x_1)$

$\vdots$

$\mathbf{in}\ \mathbf{le}$ $op(x_{n-1}, x_{n-1})$

$\mathbf{in}\ x$

$\mathsf{D}_c[t_{\mathsf{bad}}] = \lambda(x_0 : \mathbb{R}, \bar{x}_0 : \mathbb{R} -$

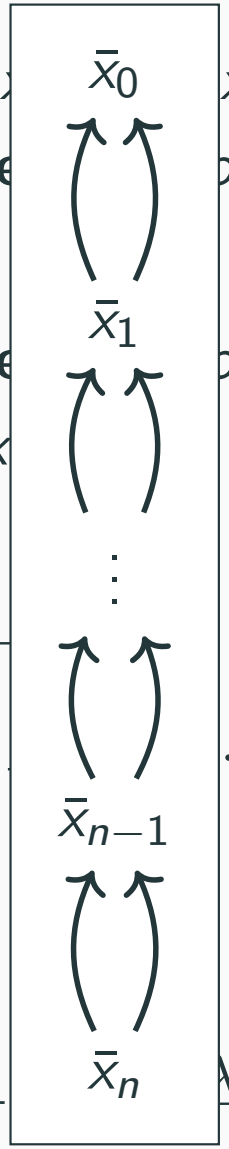
$\mathbf{let}\ (x_1, \bar{x}_1) = (op(x_0, x_0),$ $.\ \bar{x}_0\ (\partial_1 op(x_0, x_0) \cdot d) +$

$\bar{x}_0\ (\partial_2 op(x_0, x_0) \cdot d))$

$\vdots$

$\mathbf{in}\ \mathbf{let}\ (x_n, \bar{x}_n) = (op(x_{n-1}$ $\lambda(d : \mathbb{R}).\ \bar{x}_{n-1}\ (\partial_1 op(x_{n-1}, \ldots$

$\mathbf{in}\ (x_n, \bar{x}_n)$



Naive dual-numbers reverse AD

- Elegant
- Compositional & extensible
    - All the action happens at the scalars

- Easy to prove correct
- Strong parallel with dual-numbers forward AD
    - A type class abstracts over both

- Exponential time complexity...

Staging function calls: linear factoring

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

$$\{dz \mapsto 2 \cdot 1\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

$$\{dz \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

$$\{dz \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

$\{dz \mapsto 2\}$

$\{dx \mapsto 5 \cdot 1\}$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$

$y = 3$

$d = 1$

$z = 5$

$\{dz \mapsto 2\}$

$\{dx \mapsto 5\}$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$\{dz \mapsto 2\}$ $\{dx \mapsto 5\}$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$
(call dz at 2)
$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$
$$\{dx \mapsto 5 + 2, dy \mapsto 2\}$$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$
$$\{dx \mapsto 5 + 2, dy \mapsto 2\}$$
$$\{dx \mapsto 7, dy \mapsto 2\}$$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$
$$\{dx \mapsto 5 + 2, dy \mapsto 2\}$$
$$\{dx \mapsto 7, dy \mapsto 2\}$$

$$dx = \underline{\lambda}(d : \mathbb{R}).\ (d, \underline{0})$$
$$dy = \underline{\lambda}(d : \mathbb{R}).\ (\underline{0}, d)$$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$
$$\{dx \mapsto 5 + 2, dy \mapsto 2\}$$
$$\{dx \mapsto 7, dy \mapsto 2\}$$
$$(7, 0) + (0, 2) = (7, 2)$$

$$dx = \underline{\lambda}(d : \mathbb{R}).\ (d, \underline{0})$$
$$dy = \underline{\lambda}(d : \mathbb{R}).\ (\underline{0}, d)$$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls

$$\textbf{let } (z, dz) = (x + y, \underline{\lambda}(d : \mathbb{R}).\ dx\ d + dy\ d)$$
$$\textbf{in } (x \cdot z, \underline{\lambda}(d : \mathbb{R}).\ dz\ (x \cdot d) + dx\ (z \cdot d))$$

$x = 2$
$y = 3$
$d = 1$
$z = 5$

$$\{dz \mapsto 2\} \quad \cup \quad \{dx \mapsto 5\}$$
$$\{dx \mapsto 5, dz \mapsto 2\}$$

(call dz at 2)

$$\{dx \mapsto 5\} \quad \cup \quad (\{dx \mapsto 2\} \quad \cup \quad \{dy \mapsto 2\})$$
$$\{dx \mapsto 5\} \quad \cup \quad \{dx \mapsto 2, dy \mapsto 2\}$$
$$\{dx \mapsto 5 + 2, dy \mapsto 2\}$$
$$\{dx \mapsto 7, dy \mapsto 2\}$$
$$(7, 0) + (0, 2) = (7, 2)$$

$dx = \underline{\lambda}(d : \mathbb{R}).\ (d, \underline{0})$
$dy = \underline{\lambda}(d : \mathbb{R}).\ (\underline{0}, d)$

Linear factoring: $f\ x + f\ y = f\ (x + y)$

Staging linear function calls: implementation

- "$\{dx \mapsto 5, dz \mapsto 2\}$"
  - Um, equality on functions?

Staging linear function calls: implementation

- "$\{dx \mapsto 5, dz \mapsto 2\}$"
    - Um, equality on functions?
- "(call dz at 2)"
    - How do we choose?

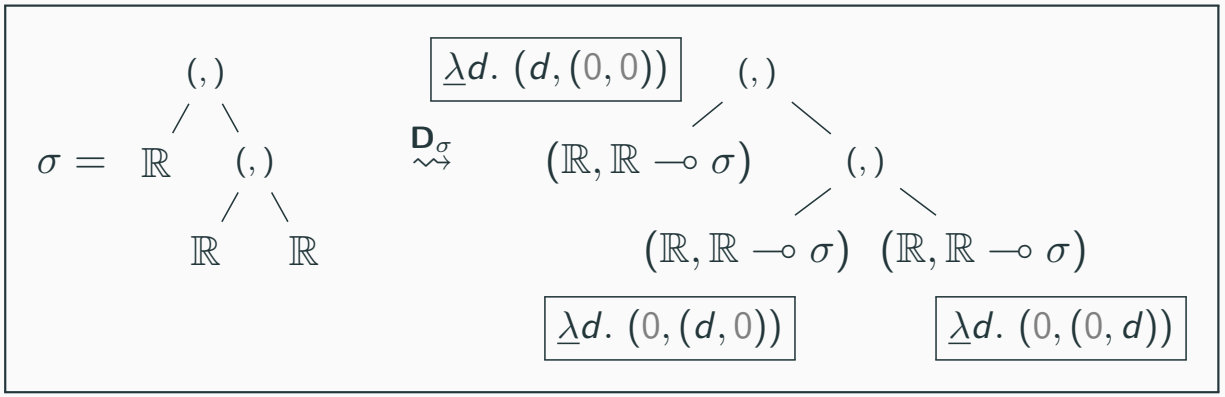
Staging linear function calls: implementation

- "$\{dx \mapsto 5, dz \mapsto 2\}$"
  - Um, equality on functions?
- "(call dz at 2)"
  - How do we choose?

Staging linear function calls: resolve order

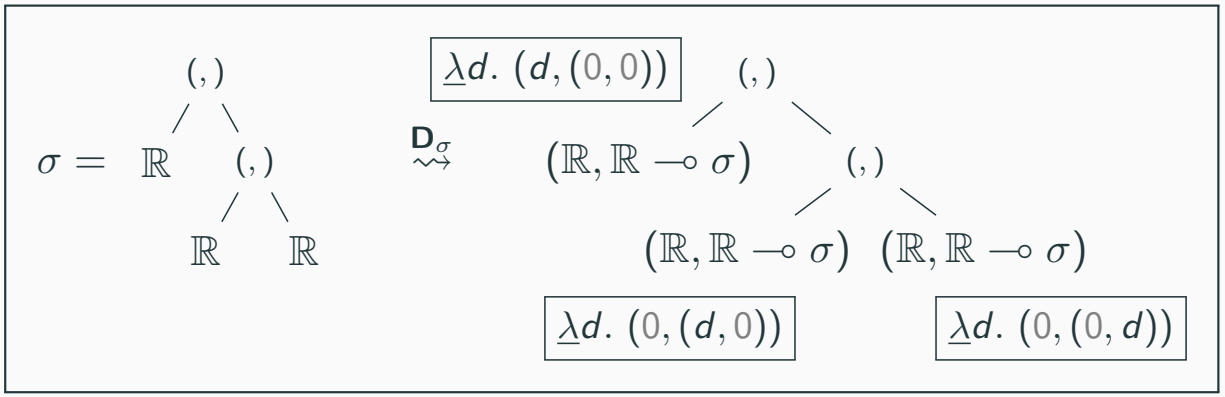
Three shapes of backpropagators:

Staging linear function calls: resolve order



Three shapes of backpropagators:

Staging linear function calls: resolve order



Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$

Staging linear function calls: resolve order

$$\boxed{\mathbf{D}_c[r] = (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0})}$$

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$

Staging linear function calls: resolve order

$$\boxed{\mathbf{D}_c[r] = (r, \underline{\lambda}(d : \mathbb{R}).\ \underline{0})}$$

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$

Staging linear function calls: resolve order

$$
\begin{array}{l}
\mathbf{D}_c[op(t_1, \ldots, t_n)] = \\
\quad \textbf{let } \ldots \\
\quad \textbf{in } (op(\ldots), \underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots))
\end{array}
$$

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$

Staging linear function calls: resolve order

$$
\begin{array}{l}
\mathbf{D}_c[op(t_1, \ldots, t_n)] = \\
\quad \textbf{let } \ldots \\
\quad \textbf{in } (op(\ldots), \underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots))
\end{array}
$$

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$
- $\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots)$

Staging linear function calls: resolve order

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$
- $\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots)$

Staging linear function calls: resolve order

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$
- $\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots)$ $\quad \leftarrow$ come from closure

Staging linear function calls: resolve order

Three shapes of backpropagators:

- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- $\underline{\lambda}(d : \mathbb{R}).\ \underline{0}$
- $\underline{\lambda}(d : \mathbb{R}).\ d_1\ (\ldots) + \cdots + d_n\ (\ldots)$ $\quad \leftarrow$ come from closure

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs
  $\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c)$

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c)$$

$$\rightsquigarrow \quad \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap c))$$

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c)$$
$$\rightsquigarrow \quad \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap c))$$
$$\rightsquigarrow \quad \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs

$$\begin{aligned} & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \\ \rightsquigarrow\ & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap c)) \\ \rightsquigarrow\ & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c)) \end{aligned}$$

$$\mathsf{Staged}\, c = (c, \mathsf{Map}\ \mathsf{Int}\ (\mathbb{R} \multimap \mathsf{Staged}\, c, \mathbb{R}))$$

$$(c, \{1 \mapsto (dx, 5), 3 \mapsto (dz, 2)\}) \ \equiv\ c + dx\ 5 + dz\ 2$$

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- Idea: sequentially number backpropagators with increasing IDs

$$\begin{aligned} & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, \mathbb{R} \multimap c) \\ \rightsquigarrow\ & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap c)) \\ \rightsquigarrow\ & \mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c)) \end{aligned}$$

$$\mathsf{Staged}\, c = (c, \mathsf{Map}\ \mathsf{Int}\ (\mathbb{R} \multimap \mathsf{Staged}\, c, \mathbb{R}))$$

$$(c, \{1 \mapsto (dx, 5), 3 \mapsto (dz, 2)\}) \ \equiv\ c + dx\ 5 + dz\ 2$$

- Consistent numbering

A backpropagator will only call other backpropagators that were created *earlier* at runtime.

Staging linear function calls: numbering

- $\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$

Staging linear function calls: numbering

- $\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$
- $$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})}$$
- $\mathsf{D}_c[\sigma \to \tau] = \mathsf{D}_c[\sigma] \to \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})$

Staging linear function calls: numbering

- $\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\ c))$
- $$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})}$$
- $\mathsf{D}_c[\sigma \to \tau] = \mathsf{D}_c[\sigma] \to \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})$
- $d_i\ (\partial_i op(x_1, \ldots, x_n) \cdot d) \quad \rightsquigarrow \quad \mathsf{SCall}\ d_i\ (\partial_i op(x_1, \ldots, x_n) \cdot d)$

Staging linear function calls: numbering

- $\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$
- $$\frac{\Gamma \vdash t : \tau}{\mathsf{D}_c[\Gamma] \vdash \mathsf{D}_c[t] : \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})}$$
- $\mathsf{D}_c[\sigma \to \tau] = \mathsf{D}_c[\sigma] \to \mathsf{Int} \to (\mathsf{D}_c[\tau], \mathsf{Int})$
- $d_i\ (\partial_i op(x_1, \ldots, x_n) \cdot d) \quad \rightsquigarrow \quad \mathsf{SCall}\ d_i\ (\partial_i op(x_1, \ldots, x_n) \cdot d)$
- $\mathsf{SCall} : (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c) \to \mathbb{R} \multimap \mathsf{Staged}\, c$
  $\mathsf{SCall}\ (i, f)\ a = (\underline{0}, \{i \mapsto (f, a)\})$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to (\mathbb{R}, \mathbb{R} \multimap \sigma)}$$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to \mathsf{Int} \to ((\mathbb{R}, \mathbb{R} \multimap \sigma), \mathsf{Int})}$$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to \mathsf{Int} \to ((\mathbb{R}, \mathbb{R} \multimap \sigma), \mathsf{Int})}$$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to \mathsf{Int} \to ((\mathbb{R}, \mathbb{R} \multimap \mathsf{Staged}\ \sigma), \mathsf{Int})}$$

Staging linear function calls: top-level function

$$\frac{\Gamma \vdash t : \sigma \to \mathbb{R}}{\mathsf{D}_\sigma[\Gamma] \vdash \mathsf{D}_\sigma[t] : \mathsf{D}_\sigma[\sigma] \to \mathsf{Int} \to ((\mathbb{R}, \mathbb{R} \multimap \mathsf{Staged}\ \sigma), \mathsf{Int})}$$

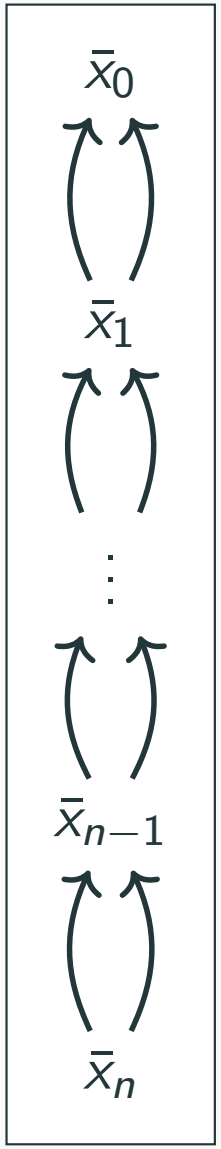
We still need to call those functions!

Staging linear function calls: resolving

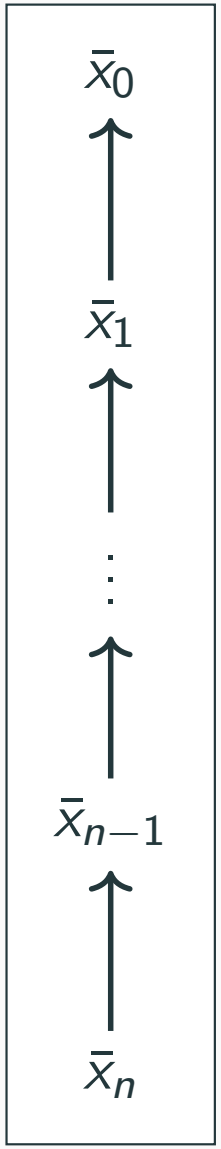
$$\text{Staged}\, c = (c, \text{Map Int}\ (\mathbb{R} \multimap \text{Staged}\, c, \mathbb{R}))$$

$$\begin{aligned}
&\text{ResolveStaged} : \text{Staged}\, c \multimap c \\
&\text{ResolveStaged}\ (c, m) = \\
&\quad \textbf{if}\ m \ \text{is empty} \\
&\qquad \textbf{then}\ c \\
&\qquad \textbf{else let}\ i = \text{highest key in}\ m \\
&\qquad\qquad \textbf{in let}\ (f, a) = \text{lookup}\ i\ \text{in}\ m \\
&\qquad\qquad \textbf{in let}\ m' = \text{delete}\ i\ \text{from}\ m \\
&\qquad\qquad \textbf{in}\ \text{ResolveStaged}\ (f\ a +_{\text{Staged}} (c, m'))
\end{aligned}$$

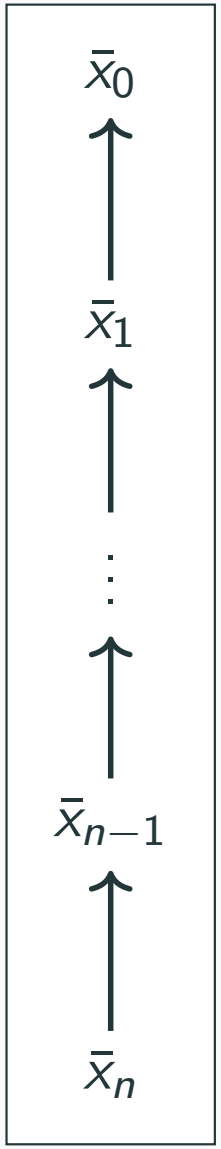
Linear factoring: results



Linear factoring: results



Linear factoring: results



Linear factoring: remaining complexity issues

Linear factoring: remaining complexity issues

$$\text{Staged}\, c = (c, \text{Map Int}\ (\mathbb{R} \multimap \text{Staged}\, c, \mathbb{R}))$$

$$\begin{aligned}
\mathsf{D}_c[r] &= \lambda i.\ \big((r, (i, \underline{\lambda}(d : \mathbb{R}).\ 0_{\text{Staged}})), i + 1\big) \\
\mathsf{D}_c[op(t_1, \ldots, t_n)] &= \\
&\ldots (op(\ldots), (i, \underline{\lambda}(d : \mathbb{R}).\ \text{SCall}\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad \text{SCall}\ d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))) \ldots
\end{aligned}$$

Linear factoring: remaining complexity issues

$$\text{Staged}\, c = (c, \text{Map Int}\ (\mathbb{R} \multimap \text{Staged}\, c, \mathbb{R}))$$

$$\begin{aligned}
\mathbf{D}_c[r] &= \lambda i.\ \big((r, (i, \underline{\lambda}(d : \mathbb{R}).\ 0_{\text{Staged}})), i + 1\big) \\
\mathbf{D}_c[op(t_1, \ldots, t_n)] &= \\
&\ldots (op(\ldots), (i, \underline{\lambda}(d : \mathbb{R}).\ \text{SCall}\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad \text{SCall}\ d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))) \ldots
\end{aligned}$$

- 0_{Staged}, $+_{\text{Staged}}$
- SCall

Linear factoring: remaining complexity issues

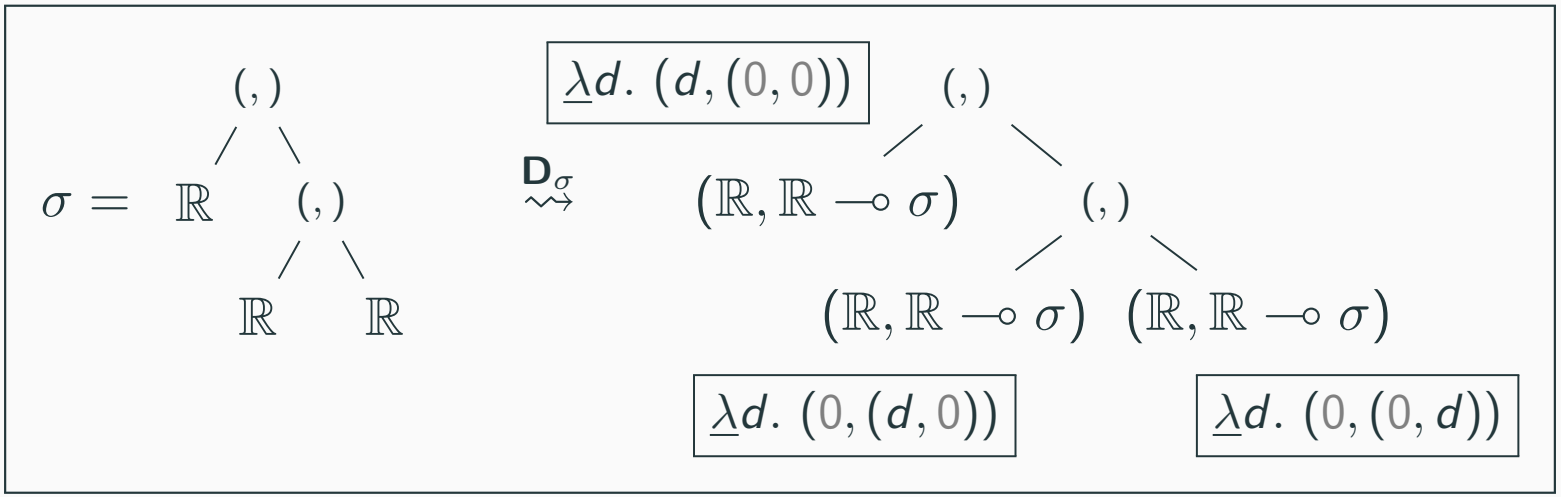
$$\text{Staged}\, c = (c, \text{Map Int}\ (\mathbb{R} \multimap \text{Staged}\, c, \mathbb{R}))$$

$$\begin{aligned}
\mathbf{D}_c[r] &= \lambda i.\ \big((r, (i, \underline{\lambda}(d : \mathbb{R}).\ 0_{\text{Staged}})), i + 1\big) \\
\mathbf{D}_c[op(t_1, \ldots, t_n)] &= \\
&\ldots (op(\ldots), (i, \underline{\lambda}(d : \mathbb{R}).\ \text{SCall}\ d_1\ (\partial_1 op(x_1, \ldots, x_n) \cdot d) + \cdots + \\
&\qquad \text{SCall}\ d_n\ (\partial_n op(x_1, \ldots, x_n) \cdot d))) \ldots
\end{aligned}$$

$$\text{SCall}\ (i, f)\ a = (\underline{0}, \{i \mapsto (f, a)\})$$

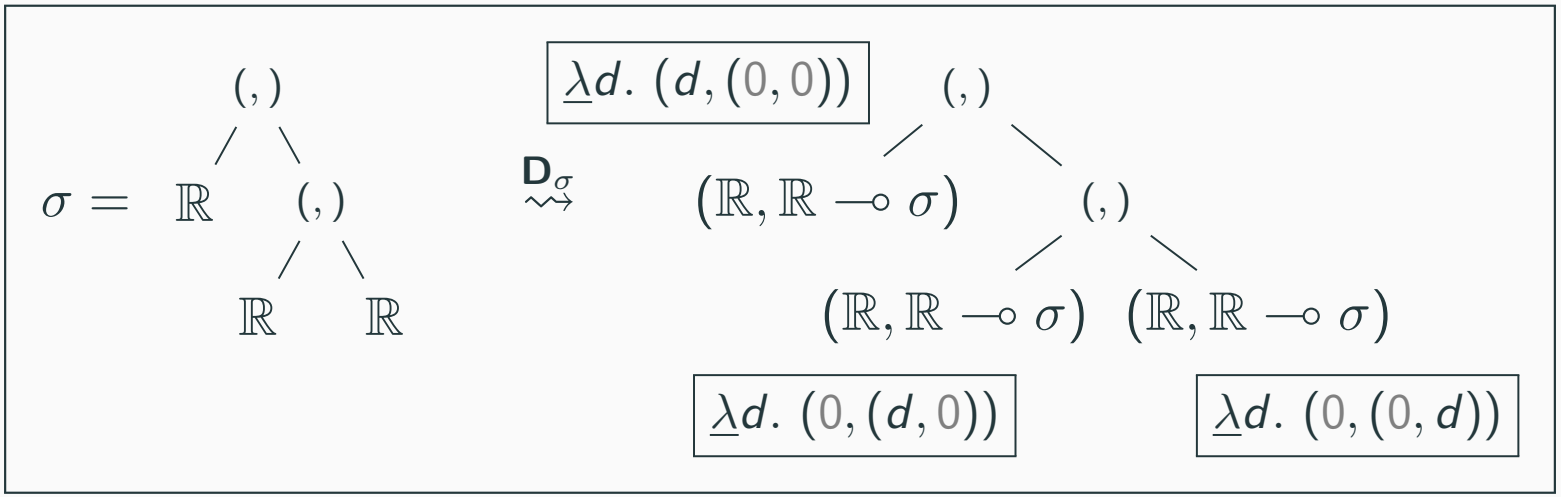
- 0_{Staged}, $+_{\text{Staged}}$
- SCall

Linear factoring: remaining complexity issues



- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall

Linear factoring: remaining complexity issues



- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$

Linear factoring: remaining complexity issues

$$
\begin{array}{l}
\mathsf{ResolveStaged} : \mathsf{Staged}\, c \multimap c \\
\mathsf{ResolveStaged}\ (c, m) = \\
\quad \textbf{if } m \text{ is empty} \\
\qquad \textbf{then } c \\
\qquad \textbf{else let } i = \text{highest key in } m \\
\qquad\qquad \textbf{in let } (f, a) = \text{lookup } i \text{ in } m \\
\qquad\qquad \textbf{in let } m' = \text{delete } i \text{ from } m \\
\qquad\qquad \textbf{in } \mathsf{ResolveStaged}\ (f\ a +_{\mathsf{Staged}} (c, m'))
\end{array}
$$

- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$

Linear factoring: remaining complexity issues

$$
\begin{array}{l}
\mathsf{ResolveStaged} : \mathsf{Staged}\, c \multimap c \\
\mathsf{ResolveStaged}\ (c, m) = \\
\quad \textbf{if } m \text{ is empty} \\
\qquad \textbf{then } c \\
\qquad \textbf{else let } i = \text{highest key in } m \\
\qquad\qquad \textbf{in let } (f, a) = \text{lookup } i \text{ in } m \\
\qquad\qquad \textbf{in let } m' = \text{delete } i \text{ from } m \\
\qquad\qquad \textbf{in } \mathsf{ResolveStaged}\ (f\ a +_{\mathsf{Staged}} (c, m'))
\end{array}
$$

- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

Linear factoring: remaining complexity issues

- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

Linear factoring: remaining complexity issues

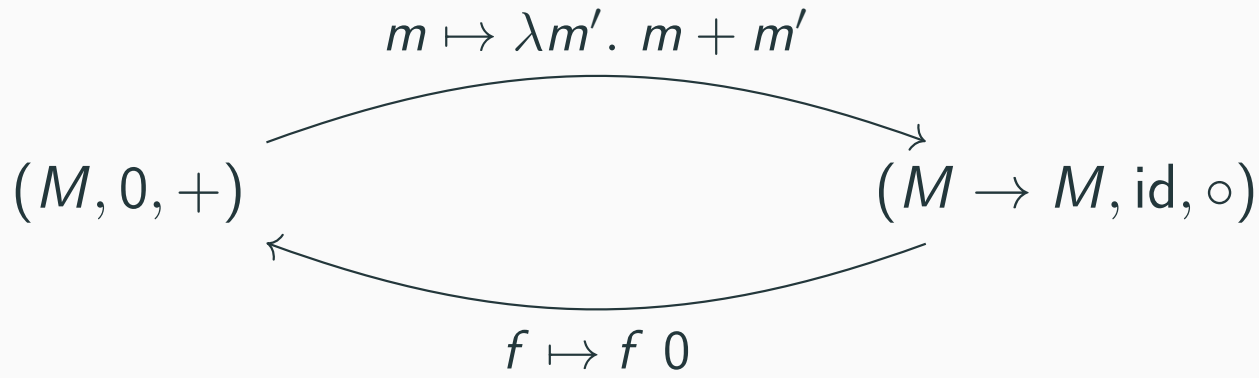
- 0_{Staged}, $+_{\text{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

Cayley

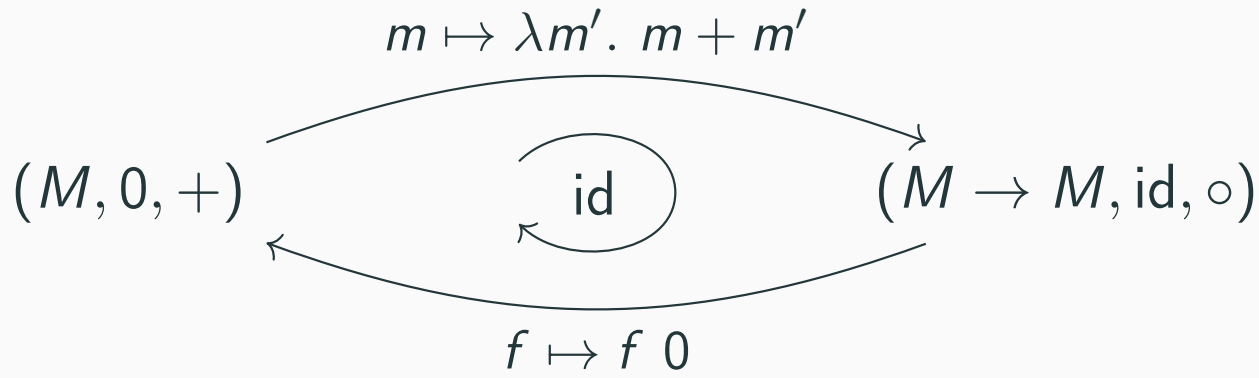
arrays

Cayley-transforming Staged c

Cayley for monoids



Cayley for monoids



Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c)))$$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\ c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\ c \to \mathsf{Staged}\ c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$
- $\mathsf{SCall} : \mathsf{Int} \to (\mathbb{R} \multimap \mathsf{Staged}\, c) \to (\mathbb{R} \multimap \mathsf{Staged}\, c)$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\ c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\ c \to \mathsf{Staged}\ c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$
- $\mathsf{SCall} : \mathsf{Int} \to (\mathbb{R} \multimap \mathsf{Staged}\ c) \to (\mathbb{R} \multimap \mathsf{Staged}\ c)$
$$\mathsf{Int} \to (\mathbb{R} \multimap (\mathsf{Staged}\ c \to \mathsf{Staged}\ c))$$
$$\to (\mathbb{R} \multimap (\mathsf{Staged}\ c \to \mathsf{Staged}\ c))$$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$
- $\mathsf{SCall} : \mathsf{Int} \to (\mathbb{R} \multimap \mathsf{Staged}\, c) \to (\mathbb{R} \multimap \mathsf{Staged}\, c)$
  $\mathsf{Int} \to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$
  $\to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$
- Input backpropagators:

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$
- $\mathsf{SCall} : \mathsf{Int} \to (\mathbb{R} \multimap \mathsf{Staged}\, c) \to (\mathbb{R} \multimap \mathsf{Staged}\, c)$
  $$\mathsf{Int} \to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$$
  $$\to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$$
- Input backpropagators:

$\underline{\lambda} d.\ (d, (0, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, yz).\ (x + d, yz)$
$\underline{\lambda} d.\ (0, (d, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y + d, z))$
$\underline{\lambda} d.\ (0, (0, d))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y, z + d))$

Cayley-transform

$$\mathsf{D}_c[\mathbb{R}] = (\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap \mathsf{Staged}\, c))$$
$$(\mathbb{R}, (\mathsf{Int}, \mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c)))$$

- $0_{\mathsf{Staged}}, +_{\mathsf{Staged}} \quad \rightsquigarrow \quad \mathsf{id}, \circ$
- $\mathsf{SCall} : \mathsf{Int} \to (\mathbb{R} \multimap \mathsf{Staged}\, c) \to (\mathbb{R} \multimap \mathsf{Staged}\, c)$
$$\mathsf{Int} \to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$$
$$\to (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c))$$

- Input backpropagators:

$\underline{\lambda} d.\ (d, (0, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, yz).\ (x + d, yz)$
$\underline{\lambda} d.\ (0, (d, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y + d, z))$
$\underline{\lambda} d.\ (0, (0, d))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y, z + d))$

Mutable arrays: removing log-factors

What log-factors?

- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

Cayley

arrays

What log-factors?

- 0_{Staged}, $+_{\mathsf{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

Cayley

arrays

Input backpropagators:

$\underline{\lambda} d.\ (d, (0, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, yz).\ (x + d, yz)$
$\underline{\lambda} d.\ (0, (d, 0))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y + d, z))$
$\underline{\lambda} d.\ (0, (0, d))$	$\rightsquigarrow$	$\underline{\lambda} d\ (x, (y, z)).\ (x, (y, z + d))$

What log-factors?

- 0_{Staged}, $+_{\text{Staged}}$
- SCall
- $\underline{\lambda}(d : \mathbb{R}).\ (0, \ldots, 0, d, 0, \ldots, 0)$
- Logarithmic Map operations in ResolveStaged

} Cayley

} arrays

Input backpropagators:

$$\boxed{\underline{\lambda} d.\ (d, (0, 0))} \quad \rightsquigarrow \quad \boxed{\underline{\lambda} d\ (x, yz).\ (x + d, yz)}$$

$$\boxed{\underline{\lambda} d.\ (0, (d, 0))} \quad \rightsquigarrow \quad \boxed{\underline{\lambda} d\ (x, (y, z)).\ (x, (y + d, z))}$$

$$\boxed{\underline{\lambda} d.\ (0, (0, d))} \quad \rightsquigarrow \quad \boxed{\underline{\lambda} d\ (x, (y, z)).\ (x, (y, z + d))}$$

$$\text{Staged}\, c = (c \qquad\qquad , \text{Map Int}\ (\mathbb{R} \multimap (\text{Staged}\, c \to \text{Staged}\, c), \mathbb{R}))$$

$$\downarrow$$

$$\text{Staged}\, c = (\text{Map Int}\ \mathbb{R}, \text{Map Int}\ (\mathbb{R} \multimap (\text{Staged}\, c \to \text{Staged}\, c), \mathbb{R}))$$

Mutable arrays

$$\mathsf{Staged}\, c = (\mathsf{Map}\ \mathsf{Int}\ \mathbb{R}, \mathsf{Map}\ \mathsf{Int}\ (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c), \mathbb{R}))$$

$$\downarrow$$

$$\mathsf{Staged}\, c = (\mathsf{MArray}\ \ \mathbb{R}, \mathsf{MArray}\ (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c), \mathbb{R}))$$

Mutable arrays

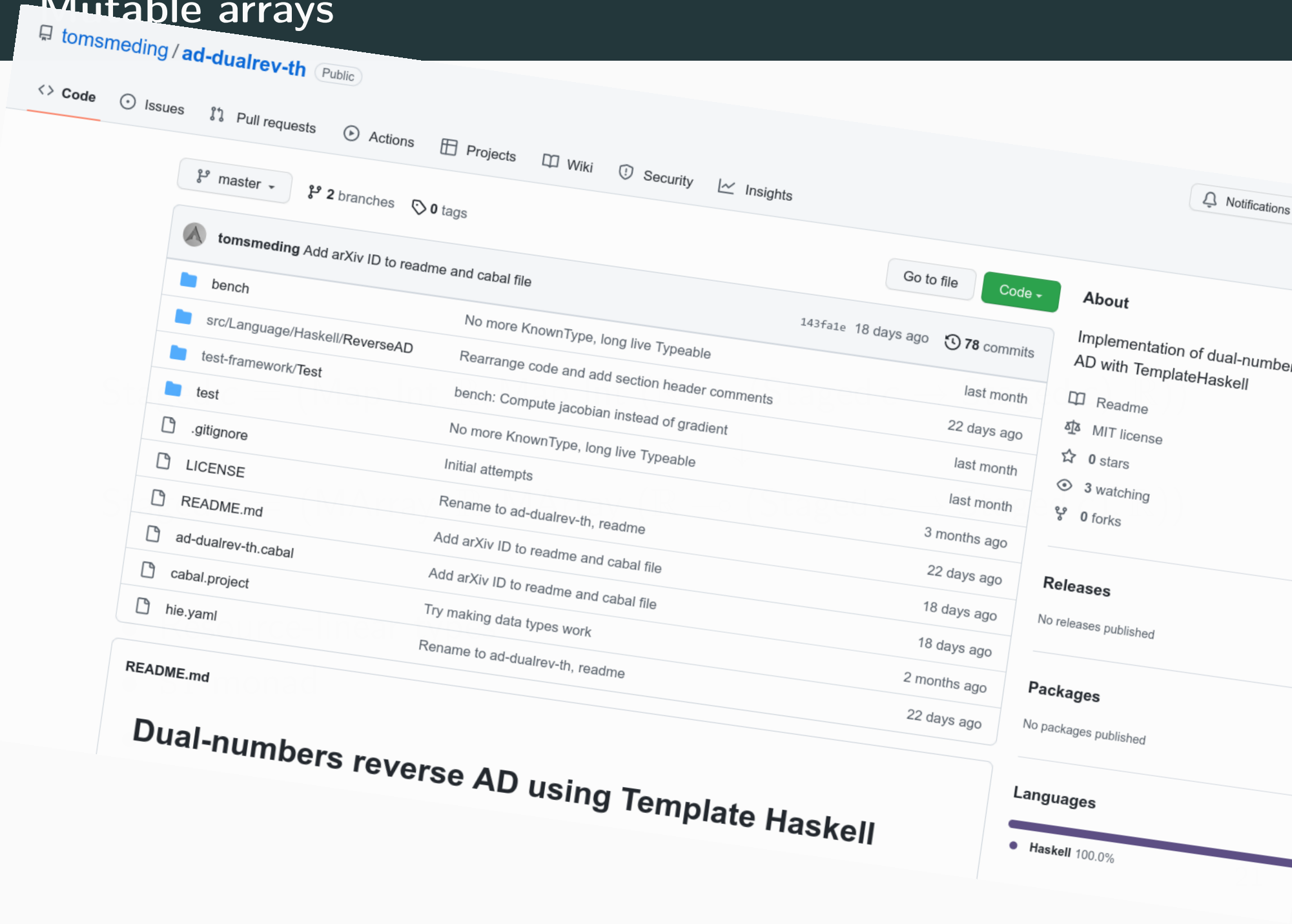
$$\mathsf{Staged}\, c = (\mathsf{Map}\ \mathsf{Int}\ \mathbb{R}, \mathsf{Map}\ \mathsf{Int}\ (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c), \mathbb{R}))$$

$$\downarrow$$

$$\mathsf{Staged}\, c = (\mathsf{MArray}\ \ \mathbb{R}, \mathsf{MArray}\ (\mathbb{R} \multimap (\mathsf{Staged}\, c \to \mathsf{Staged}\, c), \mathbb{R}))$$

- Resource-linear types
- `ST` monad
- ...

Mutable arrays



tomsmeding / ad-dualrev-th Public

Code Issues Pull requests Actions Projects Wiki Security Insights Notifications

master 2 branches 0 tags

tomsmeding Add arXiv ID to readme and cabal file

Go to file Code

bench	No more KnownType, long live Typeable	143fa1e 18 days ago 78 commits
src/Language/Haskell/ReverseAD	Rearrange code and add section header comments	last month
test-framework/Test	bench: Compute jacobian instead of gradient	22 days ago
test	No more KnownType, long live Typeable	last month
.gitignore	Initial attempts	last month
LICENSE	Rename to ad-dualrev-th, readme	3 months ago
README.md	Add arXiv ID to readme and cabal file	22 days ago
ad-dualrev-th.cabal	Add arXiv ID to readme and cabal file	18 days ago
cabal.project	Try making data types work	18 days ago
hie.yaml	Rename to ad-dualrev-th, readme	2 months ago

README.md

Dual-numbers reverse AD using Template Haskell

22 days ago

About

Implementation of dual-numbe AD with TemplateHaskell

Readme
MIT license
0 stars
3 watching
0 forks

Releases

No releases published

Packages

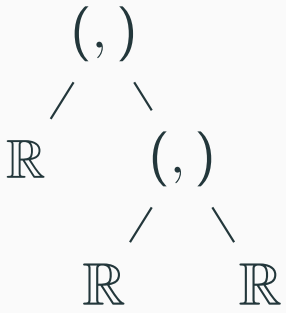
No packages published

Languages

Haskell 100.0%

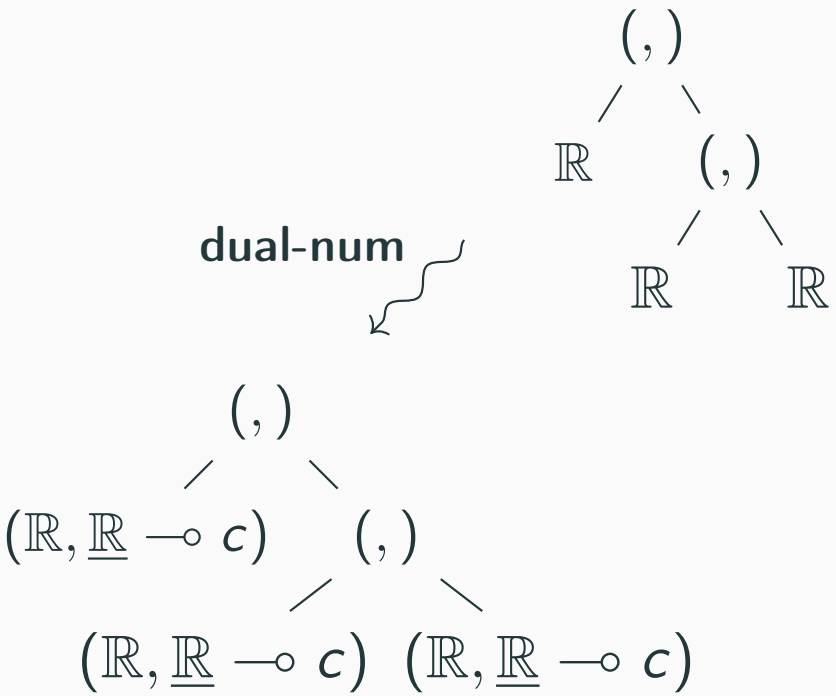
Links to other algorithms

Relation to CHAD

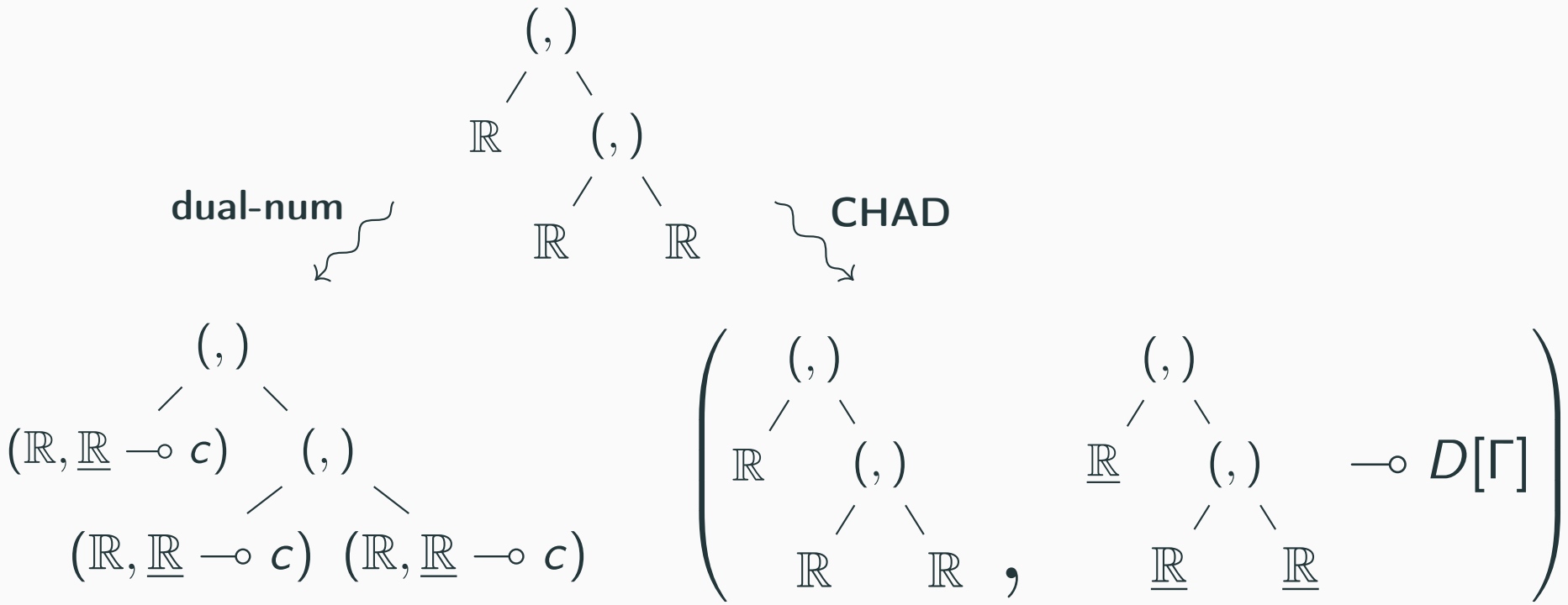


"CHAD: Combinatory Homomorphic Automatic Differentiation" (https://arxiv.org/abs/2103.15776, Matthijs Vákár & Tom Smeding)

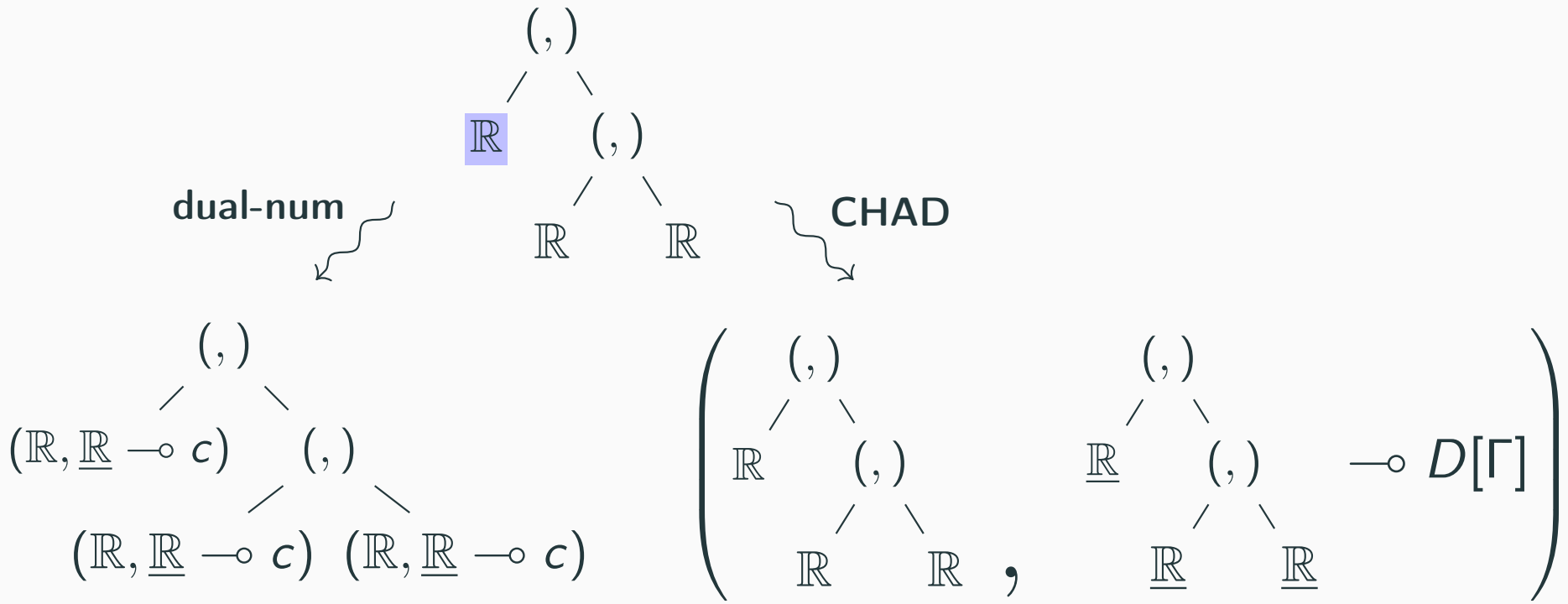
Relation to CHAD



Relation to CHAD

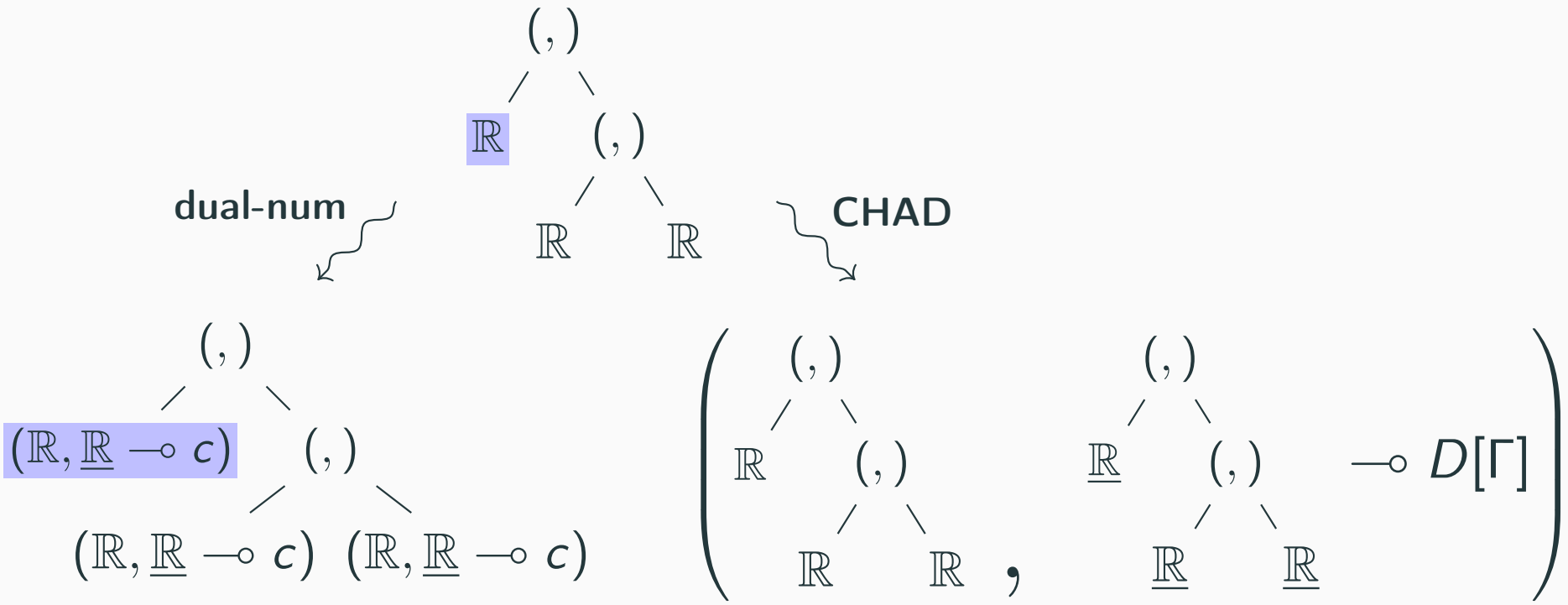


Relation to CHAD



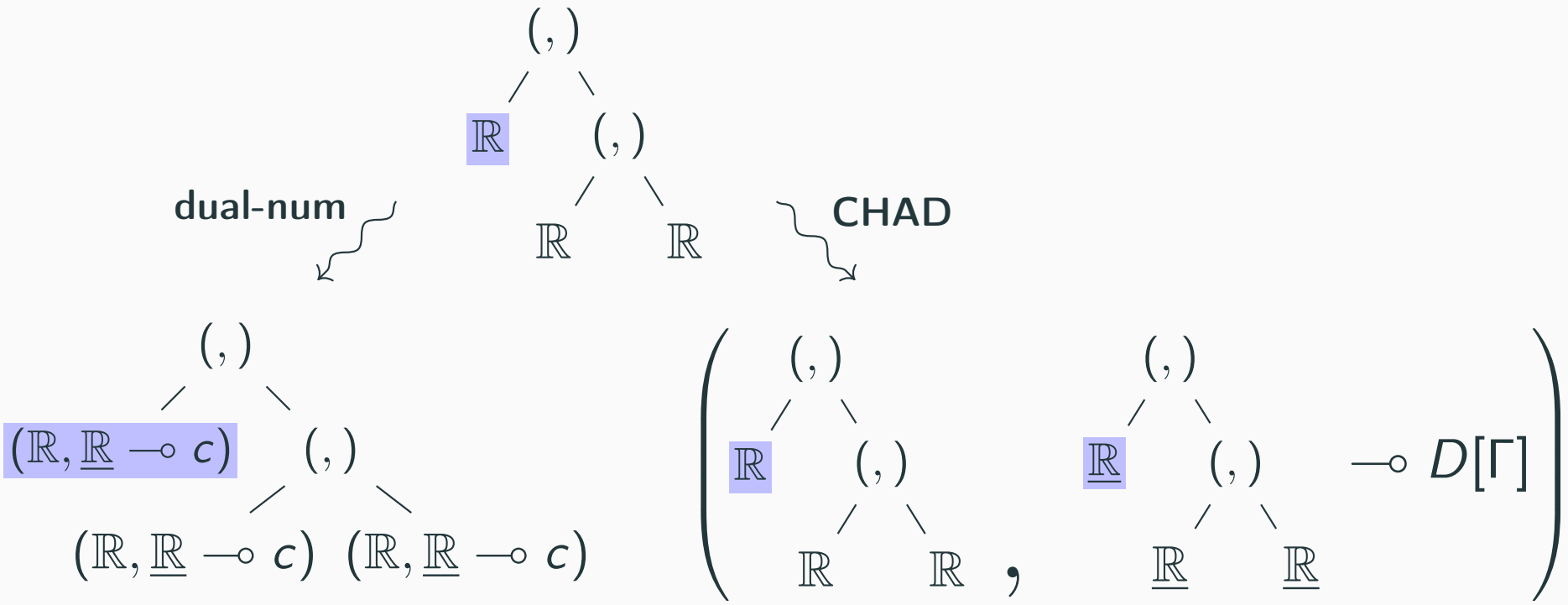
fst t

Relation to CHAD



$$\text{fst } t \quad \overset{D_c}{\rightsquigarrow} \quad \text{fst } (D_c[t])$$

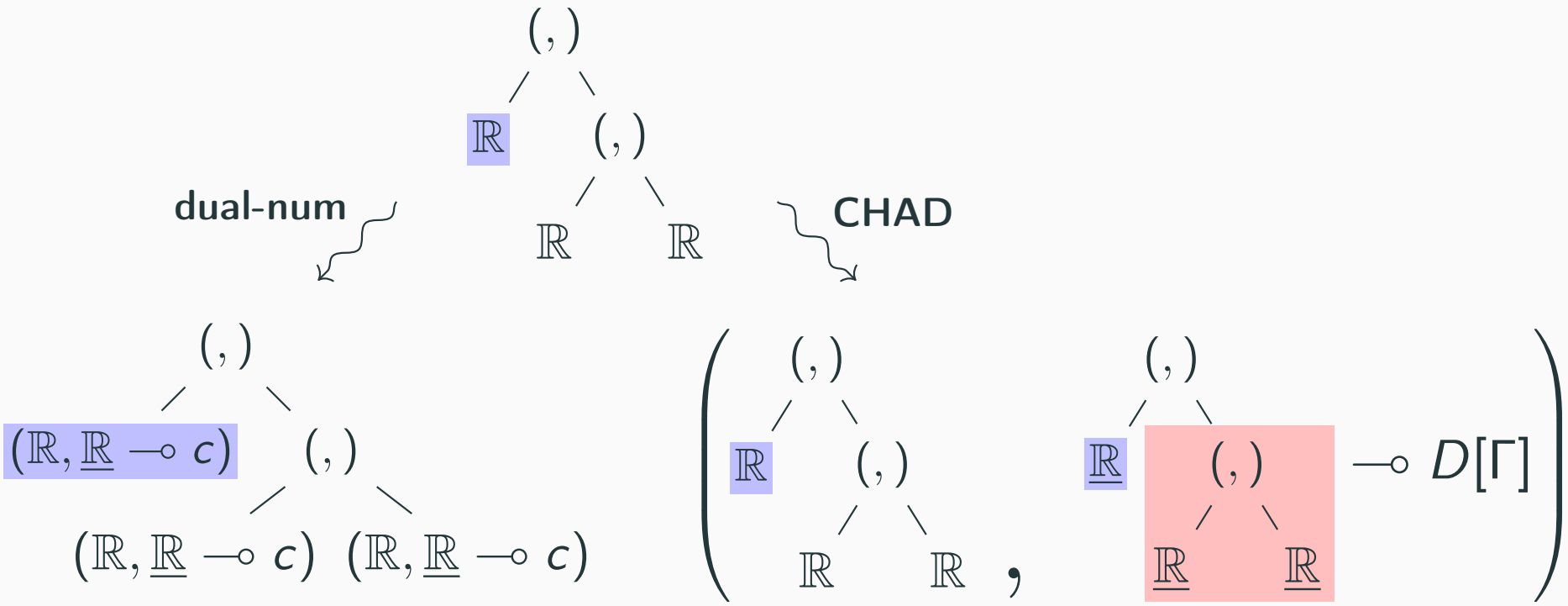
Relation to CHAD



$$\text{fst } t \quad \overset{D_c}{\rightsquigarrow} \quad \text{fst } (D_c[t])$$

$$\overset{CHAD}{\rightsquigarrow} \quad \textbf{let } (x, \bar{x}) = D[t] \textbf{ in } (\text{fst } x, \underline{\lambda} d.\ \bar{x}\ (d, \underline{0}))$$

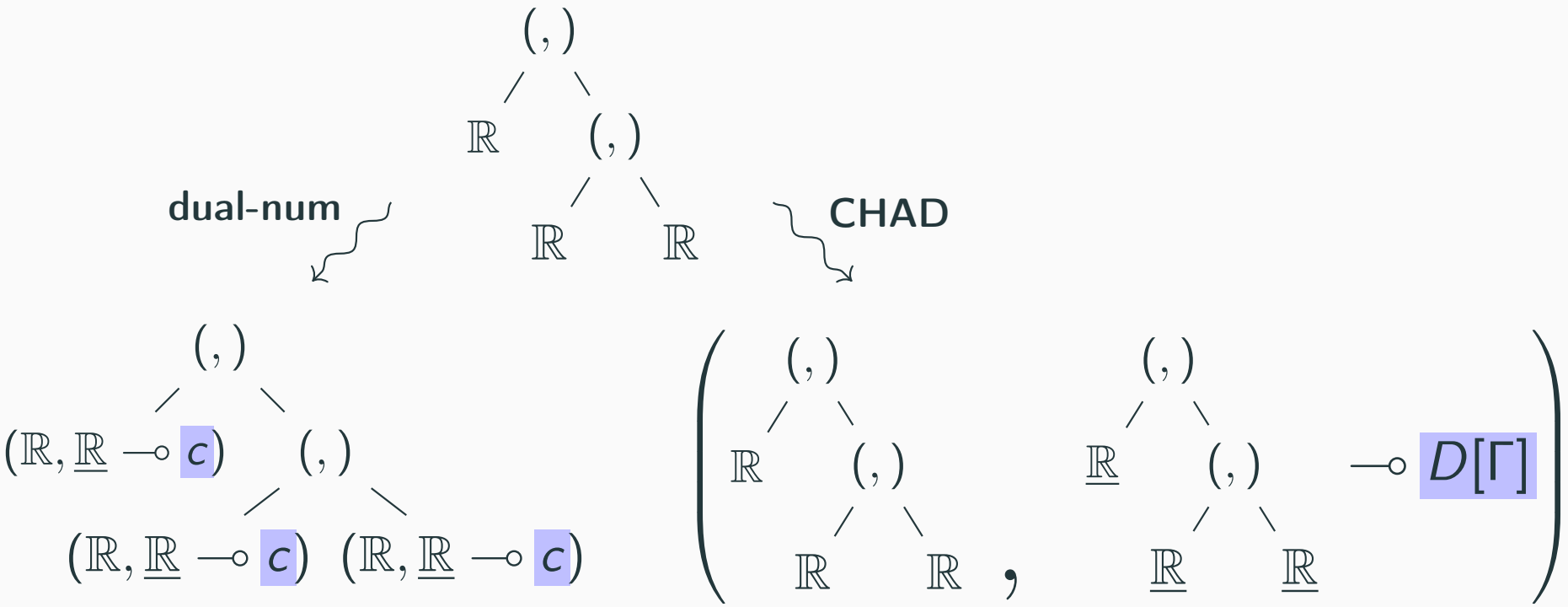
Relation to CHAD



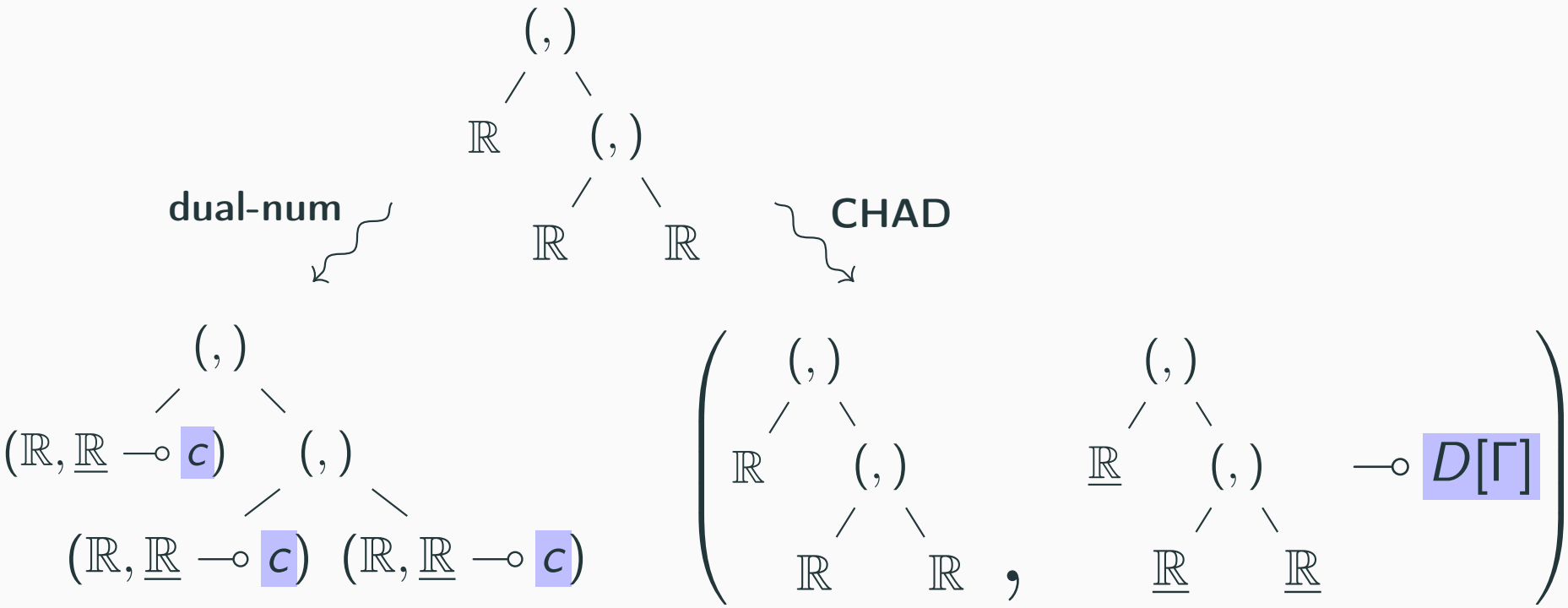
$$\text{fst } t \quad \overset{D_c}{\rightsquigarrow} \quad \text{fst } (D_c[t])$$

$$\overset{CHAD}{\rightsquigarrow} \quad \textbf{let } (x, \bar{x}) = D[t] \textbf{ in } (\text{fst } x, \underline{\lambda} d.\ \bar{x}\ (d, \underline{0}))$$

Relation to CHAD

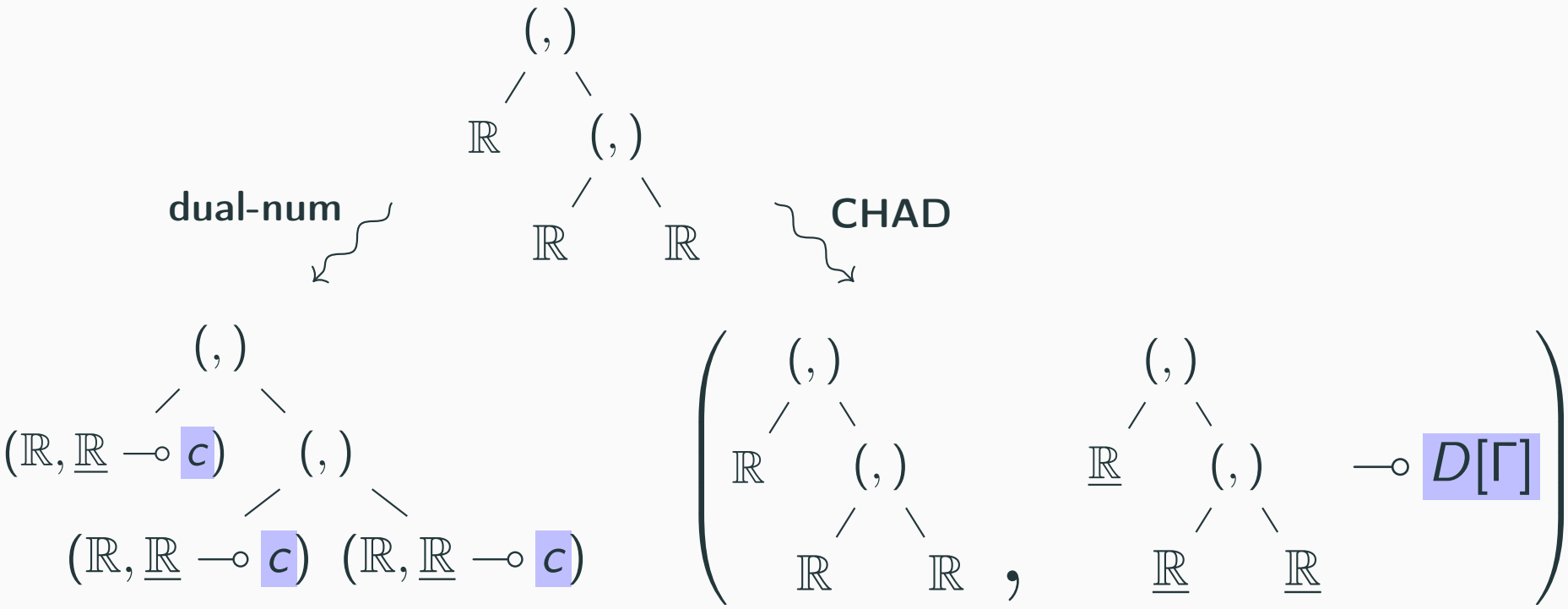


Relation to CHAD



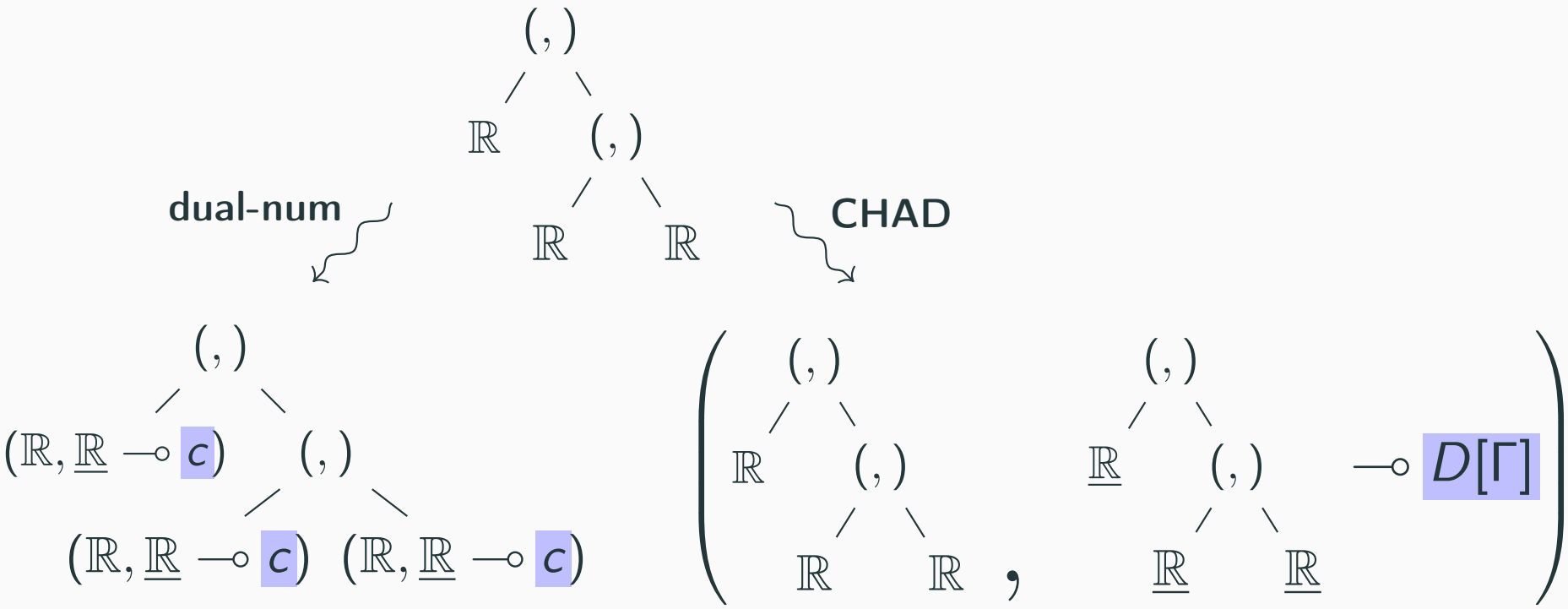
- Calls preceding backpropagators from closure
- Needs linear factoring

Relation to CHAD



- Calls preceding backpropagators from closure
- Needs linear factoring
- Differentiates only the current term
- Explosion from sharing prevented in **let**

Relation to CHAD

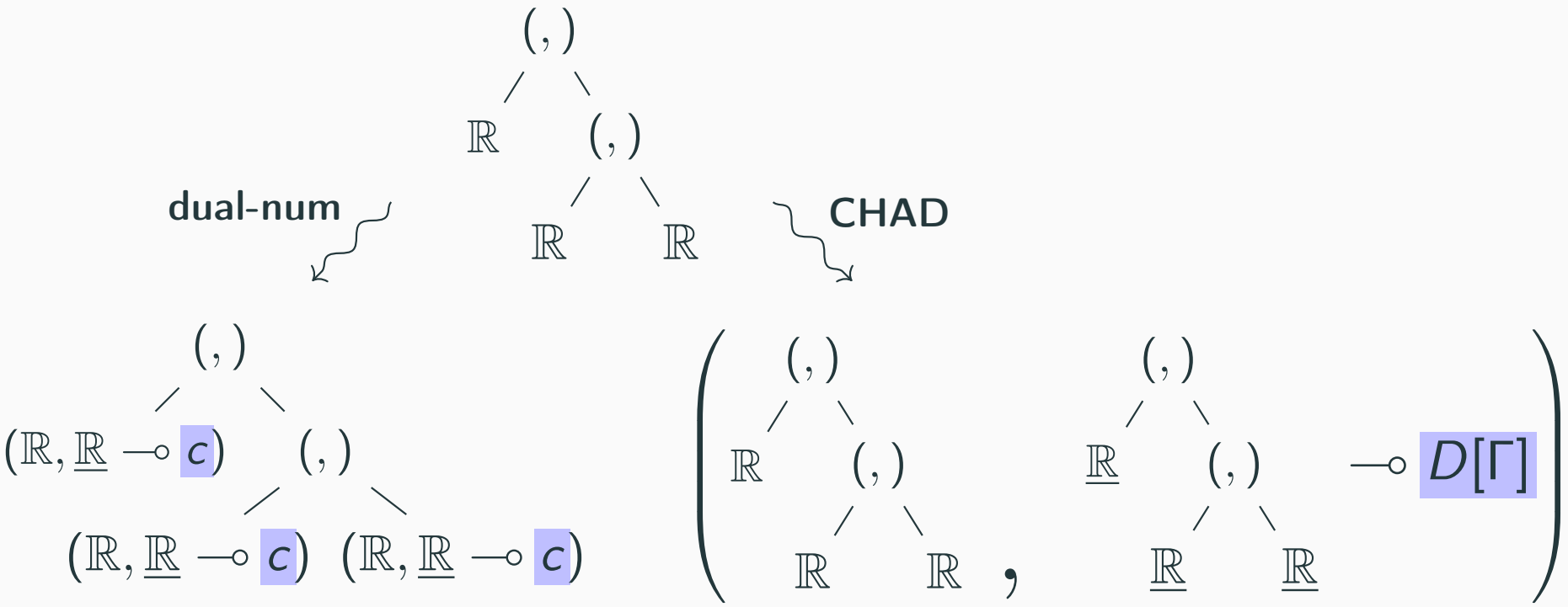


- Calls preceding backpropagators from closure
- Needs linear factoring
- Simple code transform

- Differentiates only the current term
- Explosion from sharing prevented in **let**

"CHAD: Combinatory Homomorphic Automatic Differentiation" (https://arxiv.org/abs/2103.15776, *Matthijs Vákár & Tom Smeding)*

Relation to CHAD



- Calls preceding backpropagators from closure
- Needs linear factoring
- Simple code transform

- Differentiates only the current term
- Explosion from sharing prevented in **let**
- Somewhat more complex code transform

"CHAD: Combinatory Homomorphic Automatic Differentiation" (https://arxiv.org/abs/2103.15776, Matthijs Vákár & Tom Smeding)

Summary

- Dual-numbers reverse AD becomes efficient with staging function calls (for linear factoring), Cayley-transformation and mutable arrays.

Summary

- Dual-numbers reverse AD becomes efficient with staging function calls (for linear factoring), Cayley-transformation and mutable arrays.

- Preprint:

  https://arxiv.org/abs/2205.11368

  "Dual-Numbers Reverse AD, Efficiently"

  (Tom Smeding & Matthijs Vákár)

- On Haskell using TemplateHaskell:

  https://github.com/tomsmeding/ad-dualrev-th