Taking Derivatives of Functional Programs

AD in a Functional Framework

Jeffrey Mark Siskind
qobi@purdue.edu

School of Electrical and Computer Engineering
Purdue University

University of Hertfordshire
22 November 2005

Joint work with Barak A. Pearlmutter.

Outline

Outline

Higher-Order Functions

$$
\begin{aligned}
\textsc{Fold}\ (m, n, u, b, i) \triangleq\ & \textbf{if}\ \ m > n \\
& \textbf{then}\ \ i \\
& \textbf{else}\ \ b\ ((u\ m), (\textsc{Fold}\ (m+1, n, u, b, i)))
\end{aligned}
$$

Higher-Order Functions

$$\textsc{Fold}\ (m, n, u, b, i) \triangleq \begin{array}{l} \textbf{if}\ \ m > n \\ \textbf{then}\ \ i \\ \textbf{else}\ \ b\ ((u\ m), (\textsc{Fold}\ (m+1, n, u, b, i))) \end{array}$$

$$\sum_{i=m}^{n} \sin\ i$$

Higher-Order Functions

$$\textsc{Fold}\ (m, n, u, b, i) \triangleq \begin{array}{l} \textbf{if}\ \ m > n \\ \textbf{then}\ \ i \\ \textbf{else}\ \ b\ ((u\ m), (\textsc{Fold}\ (m+1, n, u, b, i))) \end{array}$$

$$\sum_{i=m}^{n} \sin\ i : \textsc{Fold}\ (m, n, \sin, +, 0)$$

Higher-Order Functions

$$\textsc{Fold}\ (m, n, u, b, i) \triangleq \begin{array}{l}\textbf{if}\ \ m > n\\ \textbf{then}\ \ i\\ \textbf{else}\ \ b\ ((u\ m), (\textsc{Fold}\ (m+1, n, u, b, i)))\end{array}$$

$$\sum_{i=m}^{n} \cos\ i : \textsc{Fold}\ (m, n, \cos, +, 0)$$

Higher-Order Functions

$$\textsc{Fold}\ (m, n, u, b, i) \triangleq \begin{array}{l}\textbf{if}\ \ m > n\\ \textbf{then}\ \ i\\ \textbf{else}\ \ b\ ((u\ m), (\textsc{Fold}\ (m+1, n, u, b, i)))\end{array}$$

$$\prod_{i=m}^{n} \sin\ i : \textsc{Fold}\ (m, n, \sin, \times, 1)$$

Lambda Expressions

Anonymous Functions

$$\sum_{i=m}^{n} i^2$$

Lambda Expressions

Anonymous Functions

$$\sum_{i=m}^{n} i^2$$

$$\mathrm{SQR}\ i \quad \triangleq \quad i \times i$$

Lambda Expressions

Anonymous Functions

$$\sum_{i=m}^{n} i^2 \quad = \quad \textsc{Fold}\,(m, n, \textsc{sqr}, +, 0)$$

$$\textsc{sqr}\ i \quad \triangleq \quad i \times i$$

Lambda Expressions

Anonymous Functions

$$\sum_{i=m}^{n} i^2 \quad = \quad \textsc{Fold}\,(m, n, (\lambda i\; i \times i), +, 0)$$

Nesting, Free Variables, and Closures

$$(\lambda x\, 2 \times x)\; 3 \quad = \quad 6$$

Nesting, Free Variables, and Closures

$$(\lambda x\; 2 \times x)\; 3 \quad = \quad 6$$

$$((\lambda x\; \lambda y\; x + y)\; 3)\; 4 \quad = \quad 7$$

Nesting, Free Variables, and Closures

$$
\begin{aligned}
(\lambda x\, 2 \times x)\; 3 &= 6 \\
(\lambda x\; \lambda y\; x + y)\; 3 &= ?
\end{aligned}
$$

Nesting, Free Variables, and Closures

$$(\lambda x\ 2 \times x)\ 3 = 6$$

$$(\lambda x\ \lambda y\ x + y)\ 3 = \langle \{x \mapsto 3\}, \lambda y\ x + y \rangle$$

It is, of course, not excluded that the range of arguments or range of values of a function should consist wholly or partly of functions. The derivative, as this notion appears in the elementary differential calculus, is a familiar mathematical example of a function for which both ranges consist of functions.

(p. 1 ¶4)

Church, A. (1941). *The Calculi of Lambda Conversion*. Princeton University Press, Princeton, NJ.

Gottfried Leibniz
|
Jacob Bernoulli
|
Johann Bernoulli
|
Leonhard Euler
|
Joseph Louis Lagrange
|
Simeon Poisson
|
Michel Chasles
|
Hubert Anson Newton
|
Eliakim Hastings Moore
|
Oswald Veblen
|
Alonzo Church

Outline

Derivatives

$$\frac{\mathrm{d}ax^2}{\mathrm{d}x} \rightsquigarrow 2ax$$

Derivatives

$$\frac{\mathrm{d}ax^2}{\mathrm{d}x} \leadsto 2ax$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : \underbrace{f}_{\mathbb{R}\rightarrow\mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}\rightarrow\mathbb{R}}$$

Derivatives

$$\frac{\mathrm{d}ax^2}{\mathrm{d}x} \rightsquigarrow 2ax$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : \underbrace{f}_{\mathbb{R}\to\mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}\to\mathbb{R}}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

Derivatives

$$\frac{\mathrm{d}ax^2}{\mathrm{d}x} \rightsquigarrow 2ax$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : \underbrace{f}_{\mathbb{R}\to\mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}\to\mathbb{R}}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

$$\mathcal{D} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

Derivatives

$$\frac{\mathrm{d}ax^2}{\mathrm{d}x} \leadsto 2ax$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : \underbrace{f}_{\mathbb{R}\to\mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}\to\mathbb{R}}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

$$\mathcal{D} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$$

$$\mathcal{D}\ \lambda x\ ax^2$$

Partial Derivatives

$$\frac{\partial a x^2 y^3}{\partial x} \qquad \frac{\partial a x^2 y^3}{\partial y}$$

Partial Derivatives

$$\frac{\partial a x^2 y^3}{\partial x} \qquad \frac{\partial a x^2 y^3}{\partial y}$$

$$\mathcal{D}\ \lambda x\ a x^2 y^3 \qquad \mathcal{D}\ \lambda y\ a x^2 y^3$$

Partial Derivatives

$$\frac{\partial ax^2y^3}{\partial x} \qquad \frac{\partial ax^2y^3}{\partial y}$$

$$\mathcal{D}\ \lambda x\ ax^2y^3 \qquad \mathcal{D}\ \lambda y\ ax^2y^3$$

$$\mathcal{D}_1\ \lambda(x,y)\ ax^2y^3 \qquad \mathcal{D}_2\ \lambda(x,y)\ ax^2y^3$$

Partial Derivatives

$$\frac{\partial a x^2 y^3}{\partial x} \qquad \frac{\partial a x^2 y^3}{\partial y}$$

$$\mathcal{D}\ \lambda x\ a x^2 y^3 \qquad \mathcal{D}\ \lambda y\ a x^2 y^3$$

$$\mathcal{D}_1\ \lambda(x,y)\ a x^2 y^3 \qquad \mathcal{D}_2\ \lambda(x,y)\ a x^2 y^3$$

$$\frac{\partial}{\partial x} : \underbrace{f}_{\mathbb{R}^n \to \mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}^n \to \mathbb{R}}$$

Partial Derivatives

$$\frac{\partial a x^2 y^3}{\partial x} \qquad \frac{\partial a x^2 y^3}{\partial y}$$

$$\mathcal{D}\ \lambda x\ a x^2 y^3 \qquad \mathcal{D}\ \lambda y\ a x^2 y^3$$

$$\mathcal{D}_1\ \lambda(x,y)\ a x^2 y^3 \qquad \mathcal{D}_2\ \lambda(x,y)\ a x^2 y^3$$

$$\frac{\partial}{\partial x} : \underbrace{f}_{\mathbb{R}^n \to \mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}^n \to \mathbb{R}}$$

$$\frac{\partial}{\partial x} : (\mathbb{R}^n \to \mathbb{R}) \to (\mathbb{R}^n \to \mathbb{R})$$

Partial Derivatives

$$\frac{\partial ax^2y^3}{\partial x} \qquad \frac{\partial ax^2y^3}{\partial y}$$

$$\mathcal{D}\ \lambda x\ ax^2y^3 \qquad \mathcal{D}\ \lambda y\ ax^2y^3$$

$$\mathcal{D}_1\ \lambda(x,y)\ ax^2y^3 \qquad \mathcal{D}_2\ \lambda(x,y)\ ax^2y^3$$

$$\frac{\partial}{\partial x} : \underbrace{f}_{\mathbb{R}^n \rightarrow \mathbb{R}} \mapsto \underbrace{f'}_{\mathbb{R}^n \rightarrow \mathbb{R}}$$

$$\frac{\partial}{\partial x} : (\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R})$$

$$\mathcal{D}_i : (\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R})$$

Gradients

$$\nabla\; f\; \mathbf{x} \quad = \quad (\mathcal{D}_1\; f\; \mathbf{x}), \ldots, (\mathcal{D}_n\; f\; \mathbf{x})$$

$$\nabla \quad : \quad (\mathbb{R}^n \rightarrow \mathbb{R}) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^n)$$

Jacobians

$$f \quad : \quad \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{f} \quad : \quad (\mathbb{R}^n \rightarrow \mathbb{R})^m$$

$$(\mathcal{J}\ f\ \mathbf{x})[i,j] \quad = \quad (\nabla\ (\mathbf{f}[i]))[j]$$

$$\mathcal{J} \quad : \quad (\mathbb{R}^n \rightarrow \mathbb{R}^m) \rightarrow (\mathbb{R}^n \rightarrow \mathbb{R}^{m\times n})$$

Operators

$\mathcal{D}$, ∇, and $\mathcal{J}$ are traditionally called *operators*.
A more modern term is *higher-order functions*.
Higher-order functions are common in mathematics, physics, and engineering:

> *summations, comprehensions, quantifications, optimizations, integrals, convolutions, filters, edge detectors, Fourier transforms, differential equations, Hamiltonians, ...*

The Chain Rule

$$(f \circ g)\ x = g\ (f\ x)$$

The Chain Rule

$$(f \circ g)\ x = g\ (f\ x)$$

$$\frac{\mathrm{d}g}{\mathrm{d}x} = \frac{\mathrm{d}g}{\mathrm{d}f}\frac{\mathrm{d}f}{\mathrm{d}x}$$

The Chain Rule

$$(f \circ g)\ x = g\ (f\ x)$$

$$\frac{\mathrm{d}g}{\mathrm{d}x} = \frac{\mathrm{d}g}{\mathrm{d}f}\frac{\mathrm{d}f}{\mathrm{d}x}$$

$$\mathcal{D}\ (f \circ g)\ x = (\mathcal{D}\ g\ (f\ x)) \times (\mathcal{D}\ f\ x)$$

The Chain Rule

$$(f \circ g)\ x = g\ (f\ x)$$

$$\frac{\mathrm{d}g}{\mathrm{d}x} = \frac{\mathrm{d}g}{\mathrm{d}f}\frac{\mathrm{d}f}{\mathrm{d}x}$$

$$\mathcal{D}\ (f \circ g)\ x = (\mathcal{D}\ g\ (f\ x)) \times (\mathcal{D}\ f\ x)$$

$$\mathcal{J}\ (f \circ g)\ \mathbf{x} = (\mathcal{J}\ g\ (f\ \mathbf{x})) \times (\mathcal{J}\ f\ \mathbf{x})$$

Outline

Outline

Straight-Line Code and Jacobians

$$\mathbf{x}_1 = f_1\ \mathbf{x}_0$$
$$\vdots$$
$$\mathbf{x}_n = f_n\ \mathbf{x}_{n-1}$$

$$f = f_1 \circ \cdots \circ f_n$$

$$\mathcal{J}\ f\ \mathbf{x}_0 = (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\ f_1\ \mathbf{x}_0)$$
$$(\mathcal{J}\ f\ \mathbf{x}_0)^\top = (\mathcal{J}\ f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^\top$$

One Way to Compute the Jacobian

$$\overline{\mathbf{X}'_1} = (\mathcal{J}\ f_1\ \mathbf{x}_0)$$
$$\overline{\mathbf{X}'_2} = (\mathcal{J}\ f_2\ \mathbf{x}_1) \times \overline{\mathbf{X}'_1}$$
$$\vdots$$
$$\overline{\mathbf{X}'_n} = (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \overline{\mathbf{X}'_{n-1}}$$

$$\overline{\mathbf{X}'_n} = \mathcal{J}\ f\ \mathbf{x}_0$$

Forward-Mode AD

$$\overrightarrow{\mathbf{x}_1} = (\mathcal{J}\ f_1\ \mathbf{x}_0) \times \overrightarrow{\mathbf{x}_0}$$
$$\vdots$$
$$\overrightarrow{\mathbf{x}_n} = (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \overrightarrow{\mathbf{x}_{n-1}}$$

$$\overrightarrow{\mathbf{x}_n} = (\mathcal{J}\ f\ \mathbf{x}_0) \times \overrightarrow{\mathbf{x}_0}$$

Wengert, R. E. (1964). A simple automatic derivative evaluation program. *Communications of the ACM*, **7**(8):463–4.

Interleaving Forward Mode

$$\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \end{aligned}$$

$$\begin{aligned} \overrightharpoon{\mathbf{x}_1} &= (\mathcal{J}\ f_1\ \mathbf{x}_0) \times \overrightharpoon{\mathbf{x}_0} \\ &\vdots \\ \overrightharpoon{\mathbf{x}_n} &= (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \overrightharpoon{\mathbf{x}_{n-1}} \end{aligned}$$

$$\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ \overrightharpoon{\mathbf{x}_1} &= (\mathcal{J}\ f_1\ \mathbf{x}_0) \times \overrightharpoon{\mathbf{x}_0} \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \\ \overrightharpoon{\mathbf{x}_n} &= (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \overrightharpoon{\mathbf{x}_{n-1}} \end{aligned}$$

Forward Mode as a Transformation

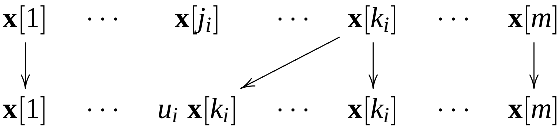
$$\left.\begin{array}{l} \mathbf{x}_1 = f_1\ \mathbf{x}_0 \\ \vdots \\ \mathbf{x}_n = f_n\ \mathbf{x}_{n-1} \end{array}\right\} \rightsquigarrow \left\{\begin{array}{l} \overrightharpoon{\mathbf{x}_1} = \overrightharpoon{f_1}\ \overrightharpoon{\mathbf{x}_0} \\ \vdots \\ \overrightharpoon{\mathbf{x}_n} = \overrightharpoon{f_n}\ \overrightharpoon{\mathbf{x}_{n-1}} \end{array}\right.$$

$$\begin{array}{rcl} \overrightharpoon{\mathbf{x}} & = & (\mathbf{x}, \overline{\mathbf{x}'}) \\ \overrightharpoon{f}\ (\mathbf{x}, \overline{\mathbf{x}'}) & = & ((f\ \mathbf{x}), ((\mathcal{J}\ f\ \mathbf{x}) \times \overline{\mathbf{x}'})) \end{array}$$

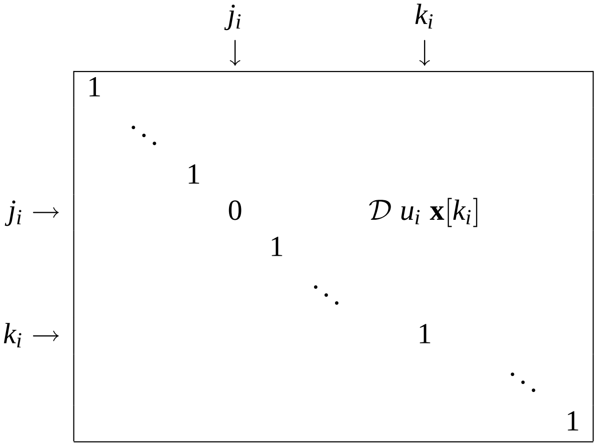
A Unary Sparse Function

$$(f_i\ \mathbf{x})[j_i] = u_i\ \mathbf{x}[k_i]$$

$$(f_i\ \mathbf{x})[j'] = \mathbf{x}[j'] \qquad j' \neq j_i$$



The Jacobian of a Unary Sparse Function

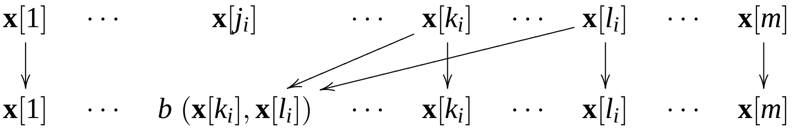


Computing $(\mathcal{J}\ f_i\ \mathbf{x}_{i-1}) \times \acute{\mathbf{x}}_{i-1}$ for a Unary Sparse Function

$$\begin{pmatrix} \acute{\mathbf{x}}[1] \\ \vdots \\ \acute{\mathbf{x}}[j_i-1] \\ (\mathcal{D}\ u_i\ \mathbf{x}[k_i]) \times \acute{\mathbf{x}}[k_i] \\ \acute{\mathbf{x}}[j_i+1] \\ \vdots \\ \acute{\mathbf{x}}[k_i] \\ \vdots \\ \acute{\mathbf{x}}[m] \end{pmatrix} = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & 0 & & & \mathcal{D}\ u_i\ \mathbf{x}[k_i] & & \\ & & & & 1 & & & & \\ & & & & & \ddots & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \begin{pmatrix} \acute{\mathbf{x}}[1] \\ \vdots \\ \acute{\mathbf{x}}[j_i-1] \\ \acute{\mathbf{x}}[j_i] \\ \acute{\mathbf{x}}[j_i+1] \\ \vdots \\ \acute{\mathbf{x}}[k_i] \\ \vdots \\ \acute{\mathbf{x}}[m] \end{pmatrix}$$

A Binary Sparse Function

$$
\begin{aligned}
(f_i\ \mathbf{x})[j_i] &= b_i\ (\mathbf{x}[k_i], \mathbf{x}[l_i]) & \\
(f_i\ \mathbf{x})[j'] &= \mathbf{x}[j'] & j' \neq j_i
\end{aligned}
$$



The Jacobian of a Binary Sparse Function

$$
\begin{array}{c}
\begin{array}{ccc} j_i & k_i & l_i \\ \downarrow & \downarrow & \downarrow \end{array}\\
\begin{array}{l}
\\ \\ \\ j_i \rightarrow \\ \\ \\ k_i \rightarrow \\ \\ l_i \rightarrow \\ \\ \\
\end{array}
\left[\begin{array}{ccccccccccc}
1 & & & & & & & & & & \\
& \ddots & & & & & & & & & \\
& & 1 & & & & & & & & \\
& & & 0 & & & \mathcal{D}_1\; b_i\; (\mathbf{x}[k_i], \mathbf{x}[l_i]) & & \mathcal{D}_2\; b_i\; (\mathbf{x}[k_i], \mathbf{x}[l_i]) & & \\
& & & & 1 & & & & & & \\
& & & & & \ddots & & & & & \\
& & & & & & 1 & & & & \\
& & & & & & & \ddots & & & \\
& & & & & & & & 1 & & \\
& & & & & & & & & \ddots & \\
& & & & & & & & & & 1
\end{array}\right]
\end{array}
$$

Computing $(\mathcal{J}\; f_i\; \mathbf{x}_{i-1}) \times \overline{\mathbf{x}_{i-1}}$ for a Binary Sparse Function

$$
\begin{pmatrix}
\overline{\mathbf{x}}[1]\\
\vdots\\
\overline{\mathbf{x}}[j_i-1]\\
((\mathcal{D}_1\; b_i\; (\mathbf{x}[k_i],\mathbf{x}[l_i])) \times \overline{\mathbf{x}}[k_i]) + ((\mathcal{D}_2\; b_i\; (\mathbf{x}[k_i],\mathbf{x}[l_i])) \times \overline{\mathbf{x}}[l_i])\\
\overline{\mathbf{x}}[j_i+1]\\
\vdots\\
\overline{\mathbf{x}}[k_i]\\
\vdots\\
\overline{\mathbf{x}}[l_i]\\
\vdots\\
\overline{\mathbf{x}}[m]
\end{pmatrix}
=
\begin{pmatrix}
1 &&&&&&&&&&\\
& \ddots &&&&&&&&&\\
&& 1 &&&&&&&&\\
&&& 0 &&& \mathcal{D}_1\; b_i\; (\mathbf{x}[k_i],\mathbf{x}[l_i]) && \mathcal{D}_2\; b_i\; (\mathbf{x}[k_i],\mathbf{x}[l_i]) &&\\
&&&& 1 &&&&&&\\
&&&&& \ddots &&&&&\\
&&&&&& 1 &&&&\\
&&&&&&& \ddots &&&\\
&&&&&&&& 1 &&\\
&&&&&&&&& \ddots &\\
&&&&&&&&&& 1
\end{pmatrix}
\begin{pmatrix}
\overline{\mathbf{x}}[1]\\
\vdots\\
\overline{\mathbf{x}}[j_i-1]\\
\overline{\mathbf{x}}[j_i]\\
\overline{\mathbf{x}}[j_i+1]\\
\vdots\\
\overline{\mathbf{x}}[k_i]\\
\vdots\\
\overline{\mathbf{x}}[l_i]\\
\vdots\\
\overline{\mathbf{x}}[m]
\end{pmatrix}
$$

Forward Mode as a Sparse Transformation

$$
\begin{aligned}
x_{j_i} := u_i\ x_{k_i} \quad &\rightsquigarrow \quad \overrightarrow{x_{j_i}} := \overrightarrow{u_i}\ \overrightarrow{x_{k_i}} \\
x_{j_i} := b_i\ (x_{k_i}, x_{l_i}) \quad &\rightsquigarrow \quad \overrightarrow{x_{j_i}} := \overrightarrow{b_i}\ (\overrightarrow{x_{k_i}}, \overrightarrow{x_{l_i}})
\end{aligned}
$$

$$
\begin{aligned}
\overrightarrow{x} &= (x, \acute{x}) \\
\overrightarrow{u}\ (x, \acute{x}) &= ((u\ x), ((\mathcal{D}\ u\ x) \times \acute{x})) \\
\overrightarrow{b}\ ((x_1, \acute{x_1}), (x_2, \acute{x_2})) &= ((b\ (x_1, x_2)), \\
&\quad (((\mathcal{D}_1\ b\ (x_1, x_2)) \times \acute{x_1}) + ((\mathcal{D}_2\ b\ (x_1, x_2)) \times \acute{x_2})))
\end{aligned}
$$

Outline

Straight-Line Code and Jacobians

$$
\begin{aligned}
\mathbf{x}_1 &= f_1\ \mathbf{x}_0\\
&\vdots\\
\mathbf{x}_n &= f_n\ \mathbf{x}_{n-1}
\end{aligned}
$$

$$f = f_1 \circ \cdots \circ f_n$$

$$
\begin{aligned}
\mathcal{J}\ f\ \mathbf{x}_0 &= (\mathcal{J}\ f_n\ \mathbf{x}_{n-1}) \times \cdots \times (\mathcal{J}\ f_1\ \mathbf{x}_0)\\
(\mathcal{J}\ f\ \mathbf{x}_0)^\top &= (\mathcal{J}\ f_1\ \mathbf{x}_0)^\top \times \cdots \times (\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^\top
\end{aligned}
$$

Another Way to Compute the Jacobian

$$\begin{aligned}
\overleftarrow{\mathbf{x}_{n-1}} &= (\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^{\top} \\
\overleftarrow{\mathbf{x}_{n-2}} &= (\mathcal{J}\ f_{n-1}\ \mathbf{x}_{n-2})^{\top} \times \overleftarrow{\mathbf{x}_{n-1}} \\
&\vdots \\
\overleftarrow{\mathbf{x}_{0}} &= (\mathcal{J}\ f_1\ \mathbf{x}_{0})^{\top} \times \overleftarrow{\mathbf{x}_{1}}
\end{aligned}$$

$$\overleftarrow{\mathbf{x}_{0}} = (\mathcal{J}\ f\ \mathbf{x}_{0})^{\top}$$

Reverse-Mode AD

$$\grave{\overline{\mathbf{x}_{n-1}}} = (\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^\top \times \grave{\overline{\mathbf{x}_n}}$$

$$\vdots$$

$$\grave{\overline{\mathbf{x}_0}} = (\mathcal{J}\ f_1\ \mathbf{x}_0)^\top \times \grave{\overline{\mathbf{x}_1}}$$

$$\grave{\overline{\mathbf{x}_0}} = (\mathcal{J}\ f\ \mathbf{x}_0)^\top \times \grave{\overline{\mathbf{x}_n}}$$

Speelpenning, B. (1980). *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Reverse Mode Cannot be Interleaved

$$
\begin{aligned}
\mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\
&\vdots \\
\mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \\
\overleftarrow{\mathbf{x}_{n-1}} &= (\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^\top \times \overleftarrow{\mathbf{x}_n} \\
&\vdots \\
\overleftarrow{\mathbf{x}_0} &= (\mathcal{J}\ f_1\ \mathbf{x}_0)^\top \times \overleftarrow{\mathbf{x}_1}
\end{aligned}
$$

Reverse Mode via Backpropagators

$$
\begin{aligned}
\mathbf{x}_1 &= f_1\ \mathbf{x}_0\\
\overline{\mathbf{x}_1} &= \lambda \overleftharpoonup{\mathbf{x}}\ \overline{\mathbf{x}_0}\ ((\mathcal{J}\ f_1\ \mathbf{x}_0)^\top \times \overleftharpoonup{\mathbf{x}})\\
&\ \vdots\\
\mathbf{x}_n &= f_n\ \mathbf{x}_{n-1}\\
\overline{\mathbf{x}_n} &= \lambda \overleftharpoonup{\mathbf{x}}\ \overline{\mathbf{x}_{n-1}}\ ((\mathcal{J}\ f_n\ \mathbf{x}_{n-1})^\top \times \overleftharpoonup{\mathbf{x}})
\end{aligned}
$$

$$\overline{\mathbf{x}_n}\ \overleftharpoonup{\mathbf{x}_n}$$

Reverse Mode as a Transformation

$$\left.\begin{aligned}\mathbf{x}_1 &= f_1\ \mathbf{x}_0\\ &\vdots\\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1}\end{aligned}\right\} \rightsquigarrow \left\{\begin{aligned}\overleftarrow{\mathbf{x}_1} &= \overleftarrow{f_1}\ \overleftarrow{\mathbf{x}_0}\\ &\vdots\\ \overleftarrow{\mathbf{x}_n} &= \overleftarrow{f_n}\ \overleftarrow{\mathbf{x}_{n-1}}\end{aligned}\right.$$

$$\begin{aligned}\overleftarrow{\mathbf{x}} &= (\mathbf{x}, \bar{\mathbf{x}})\\ \overleftarrow{f}\ (\mathbf{x}, \bar{\mathbf{x}}) &= ((f\ \mathbf{x}), (\lambda\, \grave{\mathbf{x}}\ \bar{\mathbf{x}}\ ((\mathcal{J}\ f\ \mathbf{x})^\top \times \grave{\mathbf{x}})))\end{aligned}$$

Reverse Mode via a Tape

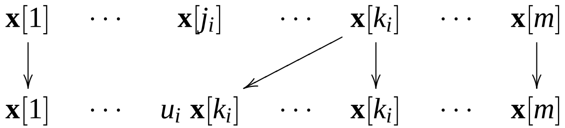
$$\left.\begin{aligned} \mathbf{x}_1 &= f_1\ \mathbf{x}_0 \\ &\vdots \\ \mathbf{x}_n &= f_n\ \mathbf{x}_{n-1} \end{aligned}\right\} \rightsquigarrow \left\{\begin{aligned} \overleftarrow{\mathbf{x}_1} &= \overleftarrow{f_1}\ \overleftarrow{\mathbf{x}_0} \\ &\vdots \\ \overleftarrow{\mathbf{x}_n} &= \overleftarrow{f_n}\ \overleftarrow{\mathbf{x}_{n-1}} \end{aligned}\right.$$

$$\begin{aligned} \overleftarrow{\mathbf{x}} &= \mathbf{x} \\ \overleftarrow{f}\ \mathbf{x} &= \textbf{begin}\ \overline{\mathbf{x}} := \lambda\, \grave{\mathbf{x}}\ \overline{\mathbf{x}}\ ((\mathcal{J}\ f\ \mathbf{x})^\top \times \grave{\mathbf{x}}); \\ &\qquad (f\ \mathbf{x})\ \textbf{end} \end{aligned}$$

A Unary Sparse Function

$$(f_i\ \mathbf{x})[j_i] = u_i\ \mathbf{x}[k_i]$$

$$(f_i\ \mathbf{x})[j'] = \mathbf{x}[j'] \qquad j' \neq j_i$$



The Jacobian of a Unary Sparse Function

$$
\begin{array}{r}
\\ \\ \\ \\ j_i \longrightarrow \\ \\ \\ k_i \longrightarrow \\ \\ \\
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
 & & & j_i & & & k_i & & \\
 & & & \downarrow & & & \downarrow & &
\end{array}\\
\left[\begin{array}{ccccccccc}
1 & & & & & & & & \\
 & \ddots & & & & & & & \\
 & & 1 & & & & & & \\
 & & & 0 & & & \mathcal{D}\; u_i\; \mathbf{x}[k_i] & & \\
 & & & & 1 & & & & \\
 & & & & & \ddots & & & \\
 & & & & & & 1 & & \\
 & & & & & & & \ddots & \\
 & & & & & & & & 1
\end{array}\right]
\end{array}
$$

The Transpose of the Jacobian of a Unary Sparse Function

$$\begin{pmatrix}
1 &&&&&&&& \\
& \ddots &&&&&&& \\
&& 1 &&&&&& \\
&&& 0 &&& \mathcal{D}\; u_i\; \mathbf{x}[k_i] && \\
&&&& 1 &&&& \\
&&&&& \ddots &&& \\
&&&&&& 1 && \\
&&&&&&& \ddots & \\
&&&&&&&& 1
\end{pmatrix}^{\top}
=
\begin{pmatrix}
1 &&&&&&&& \\
& \ddots &&&&&&& \\
&& 1 &&&&&& \\
&&& 0 &&&&& \\
&&&& 1 &&&& \\
&&&&& \ddots &&& \\
&&& \mathcal{D}\; u_i\; \mathbf{x}[k_i] &&& 1 && \\
&&&&&&& \ddots & \\
&&&&&&&& 1
\end{pmatrix}$$

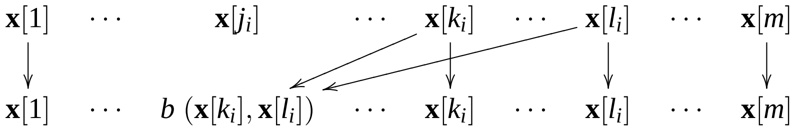
Computing $(\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \grave{\mathbf{x}}_i$ for a Unary Sparse Function

$$
\begin{pmatrix}
\grave{\mathbf{x}}[1] \\
\vdots \\
\grave{\mathbf{x}}[j_i-1] \\
0 \\
\grave{\mathbf{x}}[j_i+1] \\
\vdots \\
((\mathcal{D}\ u_i\ \mathbf{x}[k_i]) \times \grave{\mathbf{x}}[j_i]) + \grave{\mathbf{x}}[k_i] \\
\vdots \\
\grave{\mathbf{x}}[m]
\end{pmatrix}
=
\begin{pmatrix}
1 & & & & & & & & \\
& \ddots & & & & & & & \\
& & 1 & & & & & & \\
& & & 0 & & & & & \\
& & & & 1 & & & & \\
& & & & & \ddots & & & \\
& & & \mathcal{D}\ u_i\ \mathbf{x}[k_i] & & & 1 & & \\
& & & & & & & \ddots & \\
& & & & & & & & 1
\end{pmatrix}
\begin{pmatrix}
\grave{\mathbf{x}}[1] \\
\vdots \\
\grave{\mathbf{x}}[j_i-1] \\
\grave{\mathbf{x}}[j_i] \\
\grave{\mathbf{x}}[j_i+1] \\
\vdots \\
\grave{\mathbf{x}}[k_i] \\
\vdots \\
\grave{\mathbf{x}}[m]
\end{pmatrix}
$$

A Binary Sparse Function

$$(f_i\ \mathbf{x})[j_i] = b_i\ (\mathbf{x}[k_i], \mathbf{x}[l_i])$$

$$(f_i\ \mathbf{x})[j'] = \mathbf{x}[j'] \qquad j' \neq j_i$$



The Jacobian of a Binary Sparse Function

$$
\begin{array}{r|ccccccccccc|}
 & & & & j_i & & & k_i & & l_i & & \\
 & & & & \downarrow & & & \downarrow & & \downarrow & & \\
\hline
 & 1 & & & & & & & & & & \\
 & & \ddots & & & & & & & & & \\
 & & & 1 & & & & & & & & \\
j_i \rightarrow & & & & 0 & & & \mathcal{D}_1\ b_i\ (\mathbf{x}[k_i], \mathbf{x}[l_i]) & & \mathcal{D}_2\ b_i\ (\mathbf{x}[k_i], \mathbf{x}[l_i]) & & \\
 & & & & & 1 & & & & & & \\
 & & & & & & \ddots & & & & & \\
k_i \rightarrow & & & & & & & 1 & & & & \\
 & & & & & & & & \ddots & & & \\
l_i \rightarrow & & & & & & & & & 1 & & \\
 & & & & & & & & & & \ddots & \\
 & & & & & & & & & & & 1 \\
\hline
\end{array}
$$

The Transpose of the Jacobian of a Binary Sparse Function

$$
\begin{pmatrix}
1 &&&&&&&&&& \\
& \ddots &&&&&&&&& \\
&& 1 &&&&&&&& \\
&&& 0 &&& \mathcal{D}_1\, b_i\, (\mathbf{x}[k_i], \mathbf{x}[l_i]) && \mathcal{D}_2\, b_i\, (\mathbf{x}[k_i], \mathbf{x}[l_i]) && \\
&&&& 1 &&&&&& \\
&&&&& \ddots &&&&& \\
&&&&&& 1 &&&& \\
&&&&&&& \ddots &&& \\
&&&&&&&& 1 && \\
&&&&&&&&& \ddots & \\
&&&&&&&&&& 1
\end{pmatrix}^{\top}
=
\begin{pmatrix}
1 &&&&&&&&&& \\
& \ddots &&&&&&&&& \\
&& 1 &&&&&&&& \\
&&& 0 &&&&&&& \\
&&&& 1 &&&&&& \\
&&&&& \ddots &&&&& \\
&&& \mathcal{D}_1\, b_i\, (\mathbf{x}[k_i], \mathbf{x}[l_i]) &&& 1 &&&& \\
&&&&&&& \ddots &&& \\
&&& \mathcal{D}_2\, b_i\, (\mathbf{x}[k_i], \mathbf{x}[l_i]) &&&&& 1 && \\
&&&&&&&&& \ddots & \\
&&&&&&&&&& 1
\end{pmatrix}
$$

Computing $(\mathcal{J}\ f_i\ \mathbf{x}_{i-1})^\top \times \grave{\mathbf{x}}_i$ for a Binary Sparse Function

$$\begin{pmatrix}
\grave{\mathbf{x}}[1]\\
\vdots\\
\grave{\mathbf{x}}[j_i-1]\\
0\\
\grave{\mathbf{x}}[j_i+1]\\
\vdots\\
((\mathcal{D}_1\ b_i\ (\mathbf{x}[k_i],\mathbf{x}[l_i]))\times\grave{\mathbf{x}}[j_i])+\grave{\mathbf{x}}[k_i]\\
\vdots\\
((\mathcal{D}_2\ b_i\ (\mathbf{x}[k_i],\mathbf{x}[l_i]))\times\grave{\mathbf{x}}[j_i])+\grave{\mathbf{x}}[l_i]\\
\vdots\\
\grave{\mathbf{x}}[m]
\end{pmatrix}
=
\begin{pmatrix}
1 &&&&&&&&&&\\
&\ddots&&&&&&&&&\\
&&1&&&&&&&&\\
&&&0&&&&&&&\\
&&&&1&&&&&&\\
&&&&&\ddots&&&&&\\
&&&\mathcal{D}_1\ b_i\ (\mathbf{x}[k_i],\mathbf{x}[l_i])&&&1&&&&\\
&&&&&&&\ddots&&&\\
&&&\mathcal{D}_2\ b_i\ (\mathbf{x}[k_i],\mathbf{x}[l_i])&&&&&1&&\\
&&&&&&&&&\ddots&\\
&&&&&&&&&&1
\end{pmatrix}
\begin{pmatrix}
\grave{\mathbf{x}}[1]\\
\vdots\\
\grave{\mathbf{x}}[j_i-1]\\
\grave{\mathbf{x}}[j_i]\\
\grave{\mathbf{x}}[j_i+1]\\
\vdots\\
\grave{\mathbf{x}}[k_i]\\
\vdots\\
\grave{\mathbf{x}}[l_i]\\
\vdots\\
\grave{\mathbf{x}}[m]
\end{pmatrix}$$

Sparse Reverse Mode via a Tape

$$
\begin{array}{lcl}
x_{j_i} := u_i\ x_{k_i} & \rightsquigarrow & \bar{x} := \lambda[\,]\ \textbf{begin}\ \overleftharpoon{x_{k_i}} \mathrel{+}:= (\mathcal{D}\ u_i\ x_{k_i}) \times \overleftharpoon{x_{j_i}};\\
& & \qquad\qquad\qquad\ \ \overleftharpoon{x_{j_i}} := 0;\\
& & \qquad\qquad\qquad\ \ \bar{x}\ [\,]\ \textbf{end};\\
& & x_{j_i} := u_i\ x_{k_i}
\end{array}
$$

$$
\begin{array}{lcl}
x_{j_i} := b_i\ (x_{k_i}, x_{l_i}) & \rightsquigarrow & \bar{x} := \lambda[\,]\ \textbf{begin}\ \overleftharpoon{x_{k_i}} \mathrel{+}:= (\mathcal{D}_1\ b_i\ (x_{k_i}, x_{l_i})) \times \overleftharpoon{x_{j_i}};\\
& & \qquad\qquad\qquad\ \ \overleftharpoon{x_{l_i}} \mathrel{+}:= (\mathcal{D}_2\ b_i\ (x_{k_i}, x_{l_i})) \times \overleftharpoon{x_{j_i}};\\
& & \qquad\qquad\qquad\ \ \overleftharpoon{x_{j_i}} := 0;\\
& & \qquad\qquad\qquad\ \ \bar{x}\ [\,]\ \textbf{end};\\
& & x_{j_i} := b_i\ (x_{k_i}, x_{l_i})
\end{array}
$$

Outline

Reverse Mode on Imperative Programs

$$\begin{array}{rcl} & \vdots & \\ x_2 & := & u\ x_1 \\ & & \\ x_4 & := & b\ (x_2, x_3) \\ & \vdots & \end{array}$$

Reverse Mode on Imperative Programs

$$\left.\begin{array}{rcl} & \vdots & \\ x_2 & := & u\ x_1 \\ & & \\ x_4 & := & b\ (x_2, x_3) \\ & \vdots & \end{array}\right\}\ \textit{forward phase}$$

$$\left.\begin{array}{rcl} & \vdots & \\ \overleftarrow{x_2} & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \overleftarrow{x_4} \\ \overleftarrow{x_3} & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \overleftarrow{x_4} \\ & & \\ \overleftarrow{x_1} & +:= & (\mathcal{D}\ u\ x_1) \times \overleftarrow{x_2} \\ & \vdots & \end{array}\right\}\ \textit{reverse phase}$$

Reverse Mode on Imperative Programs

$$\left.\begin{array}{lcl} & \vdots & \\ x_2 & := & u\ x_1 \\ & & \\ x_4 & := & b\ (x_2, x_3) \\ & \vdots & \end{array}\right\}\ \textit{forward phase}$$

$$\left.\begin{array}{lcl} & \vdots & \\ \grave{x}_2 & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \grave{x}_4 \\ \grave{x}_3 & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \grave{x}_4 \\ & & \\ \grave{x}_1 & +:= & (\mathcal{D}\ u\ x_1) \times \grave{x}_2 \\ & \vdots & \end{array}\right\}\ \textit{reverse phase}$$

Reverse Mode on Imperative Programs

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
x_2 & := & u\ x_1 \\
x_4 & := & b\ (x_2, x_3) \\
 & \vdots &
\end{array}
\right\}\ \textit{forward phase}
$$

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
\overleftarrow{x_2} & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \overleftarrow{x_4} \\
\overleftarrow{x_3} & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \overleftarrow{x_4} \\
\overleftarrow{x_1} & +:= & (\mathcal{D}\ u\ x_1) \times \overleftarrow{x_2} \\
 & \vdots &
\end{array}
\right\}\ \textit{reverse phase}
$$

Reverse Mode on Imperative Programs

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
x_2 & := & u\ x_1 \\
 & & \\
x_4 & := & b\ (x_2, x_3) \\
 & \vdots &
\end{array}
\right\} \textit{forward phase}
$$

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
\grave{\overline{x}}_2 & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \grave{\overline{x}}_4 \\
\grave{\overline{x}}_3 & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \grave{\overline{x}}_4 \\
 & & \\
\grave{\overline{x}}_1 & +:= & (\mathcal{D}\ u\ x_1) \times \grave{\overline{x}}_2 \\
 & \vdots &
\end{array}
\right\} \textit{reverse phase}
$$

Reverse Mode on Imperative Programs

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
x_2 & := & u\ x_1 \\
x_2 & := & b\ (x_2, x_3) \\
 & \vdots &
\end{array}
\right\}\ \textit{forward phase}
$$

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
\overleftarrow{x_2} & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \overleftarrow{x_2} \\
\overleftarrow{x_3} & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \overleftarrow{x_2} \\
\overleftarrow{x_1} & +:= & (\mathcal{D}\ u\ x_1) \times \overleftarrow{x_2} \\
 & \vdots &
\end{array}
\right\}\ \textit{reverse phase}
$$

Reverse Mode on Imperative Programs

$$\begin{array}{rcl} & \vdots & \\ x_2 & := & u\ x_1 \\ x_2 & := & b\ (x_2, x_3) \\ & \vdots & \end{array} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} \textit{forward phase}$$

$$\begin{array}{rcl} & \vdots & \\ \overline{x}_2 & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \overline{x}_2 \\ \overline{x}_3 & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \overline{x}_2 \\ \overline{x}_1 & +:= & (\mathcal{D}\ u\ x_1) \times \overline{x}_2 \\ & \vdots & \end{array} \left.\vphantom{\begin{array}{c}a\\a\\a\\a\\a\end{array}}\right\} \textit{reverse phase}$$

Reverse Mode on Imperative Programs

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
x_2 & := & u\ x_1 \\
\text{PUSH}\ x_2 & & \\
 & & \\
x_2 & := & b\ (x_2, x_3) \\
 & \vdots &
\end{array}
\right\} \textit{forward phase}
$$

$$
\left.
\begin{array}{lcl}
 & \vdots & \\
\text{POP}\ x_2 & & \\
\overleftarrow{x_2} & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \overleftarrow{x_2} \\
\overleftarrow{x_3} & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \overleftarrow{x_2} \\
 & & \\
\overleftarrow{x_1} & +:= & (\mathcal{D}\ u\ x_1) \times \overleftarrow{x_2} \\
 & \vdots &
\end{array}
\right\} \textit{reverse phase}
$$

Reverse Mode on Imperative Programs

$$
\left.\begin{array}{lll}
 & \vdots & \\
\text{PUSH } x_1 & & \\
x_2 & := & u\ x_1 \\
\text{PUSH } x_2 & & \\
\text{PUSH } x_3 & & \\
x_2 & := & b\ (x_2, x_3) \\
 & \vdots &
\end{array}\right\} \textit{forward phase}
$$

$$
\left.\begin{array}{lll}
 & \vdots & \\
\text{POP } x_3 & & \\
\text{POP } x_2 & & \\
\grave{x}_2 & +:= & (\mathcal{D}_1\ b\ (x_2, x_3)) \times \grave{x}_2 \\
\grave{x}_3 & +:= & (\mathcal{D}_2\ b\ (x_2, x_3)) \times \grave{x}_2 \\
\text{POP } x_1 & & \\
\grave{x}_1 & +:= & (\mathcal{D}\ u\ x_1) \times \grave{x}_2 \\
 & \vdots &
\end{array}\right\} \textit{reverse phase}
$$

Notation

In the following slides, I use $\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}_i$, and $\mathbf{y}_i$ to denote tuples of scalar variables, i.e. (x_{47}, x_{19}, x_{33}).

Notation

In the following slides, I use $\mathbf{x}$, $\mathbf{y}$, $\mathbf{x}_i$, and $\mathbf{y}_i$ to denote tuples of scalar variables, i.e. (x_{47}, x_{19}, x_{33}).

I use $\grave{\overline{\mathbf{x}}}$, $\grave{\overline{\mathbf{y}}}$, $\grave{\overline{\mathbf{x}}}_i$, and $\grave{\overline{\mathbf{y}}}_i$ to denote tuples of corresponding sensitivities of scalar variables, i.e. $(\grave{\overline{x}}_{47}, \grave{\overline{x}}_{19}, \grave{\overline{x}}_{33})$.

Subroutines and Tapes

Unary Primitives

$$u : x \mapsto y$$

Subroutines and Tapes

Unary Primitives

$$u : x \mapsto y$$

$$\overleftarrow{u} : x \mapsto y \quad \triangleq \quad \left\{ \begin{array}{l} \text{PUSH } x \\ y \;:=\; u\,x \end{array} \right.$$

$$\bar{u} : \overleftharpoon{y} \mapsto \overleftharpoon{x} \quad \triangleq \quad \left\{ \begin{array}{l} \text{POP } x \\ \overleftharpoon{x} \;+\!:=\; (\mathcal{D}\,u\,x) \times \overleftharpoon{y} \end{array} \right.$$

Subroutines and Tapes

Binary Primitives

$$b : (x, y) \mapsto z$$

Subroutines and Tapes

Binary Primitives

$$b : (x, y) \mapsto z$$

$$\overset{\leftharpoonup}{b} : (x, y) \mapsto z \quad \triangleq \quad \left\{ \begin{array}{lll} \text{PUSH } x & & \\ \text{PUSH } y & & \\ z & := & b\ (x, y) \end{array} \right.$$

$$\overline{b} : \overset{\leftharpoondown}{z} \mapsto (\overset{\leftharpoondown}{x}, \overset{\leftharpoondown}{y}) \quad \triangleq \quad \left\{ \begin{array}{lll} \text{POP } x & & \\ \text{POP } y & & \\ \overset{\leftharpoondown}{x} & +:= & (\mathcal{D}_1\ b\ (x, y)) \times \overset{\leftharpoondown}{z} \\ \overset{\leftharpoondown}{y} & +:= & (\mathcal{D}_2\ b\ (x, y)) \times \overset{\leftharpoondown}{z} \end{array} \right.$$

Subroutines and Tapes

User-Defined Functions

$$f : \mathbf{x} \mapsto \mathbf{y} \quad \stackrel{\triangle}{=} \quad \left\{ \begin{array}{lcl} \mathbf{y}_1 & := & f_1\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n & := & f_n\ \mathbf{x}_n \end{array} \right.$$

Subroutines and Tapes

User-Defined Functions

$$f : \mathbf{x} \mapsto \mathbf{y} \quad \triangleq \quad \left\{ \begin{array}{lcl} \mathbf{y}_1 & := & f_1\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n & := & f_n\ \mathbf{x}_n \end{array} \right.$$

$$\overleftarrow{f} : \mathbf{x} \mapsto \mathbf{y} \quad \triangleq \quad \left\{ \begin{array}{lcl} \mathbf{y}_1 & := & \overleftarrow{f_1}\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n & := & \overleftarrow{f_n}\ \mathbf{x}_n \end{array} \right.$$

$$\bar{f} : \grave{\mathbf{y}} \mapsto \grave{\mathbf{x}} \quad \triangleq \quad \left\{ \begin{array}{lcl} \grave{\mathbf{x}}_n & +:= & \bar{f}_n\ \grave{\mathbf{y}}_n \\ & \vdots & \\ \grave{\mathbf{x}}_1 & +:= & \bar{f}_1\ \grave{\mathbf{y}}_1 \end{array} \right.$$

Representing the Tape as Function Arguments and Results

Unary Primitives

$$u : x \mapsto y$$

$$\overleftarrow{u} : x \mapsto (y, x) \quad \triangleq \quad \{\ y \ := \ u\, x$$

$$\overline{u} : (x, \grave{\overline{y}}) \mapsto \grave{\overline{x}} \quad \triangleq \quad \{\ \grave{\overline{x}} \ +:= \ (\mathcal{D}\, u\, x) \times \grave{\overline{y}}$$

Representing the Tape as Function Arguments and Results

Binary Primitives

$$b : (x, y) \mapsto z$$

$$\overleftarrow{b} : (x, y) \mapsto (z, (x, y)) \quad \triangleq \quad \left\{\; z \;:=\; b\;(x, y) \right.$$

$$\overline{b} : ((x, y), \grave{\overline{z}}) \mapsto (\grave{\overline{x}}, \grave{\overline{y}}) \quad \triangleq \quad \left\{ \begin{array}{lll} \grave{\overline{x}} & +:= & (\mathcal{D}_1\; b\; (x, y)) \times \grave{\overline{z}} \\ \grave{\overline{y}} & +:= & (\mathcal{D}_2\; b\; (x, y)) \times \grave{\overline{z}} \end{array} \right.$$

Representing the Tape as Function Arguments and Results

User-Defined Functions

$$f : \mathbf{x} \mapsto \mathbf{y} \quad \triangleq \quad \left\{ \begin{array}{lcl} \mathbf{y}_1 & := & f_1\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n & := & f_n\ \mathbf{x}_n \end{array} \right.$$

$$\overleftarrow{f} : \mathbf{x} \mapsto (\mathbf{y}, (\mathbf{t}_1, \ldots, \mathbf{t}_n)) \quad \triangleq \quad \left\{ \begin{array}{lcl} \mathbf{y}_1, \mathbf{t}_1 & := & \overleftarrow{f_1}\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n, \mathbf{t}_n & := & \overleftarrow{f_n}\ \mathbf{x}_n \end{array} \right.$$

$$\overline{f} : ((\mathbf{t}_1, \ldots, \mathbf{t}_n), \grave{\mathbf{y}}) \mapsto \grave{\mathbf{x}} \quad \triangleq \quad \left\{ \begin{array}{lcl} \grave{\mathbf{x}}_n & +:= & \overline{f_n}\ (\mathbf{t}_n, \grave{\mathbf{y}}_n) \\ & \vdots & \\ \grave{\mathbf{x}}_1 & +:= & \overline{f_1}\ (\mathbf{t}_1, \grave{\mathbf{y}}_1) \end{array} \right.$$

Representing the Tape as Closures

Unary Primitives

$$u : x \mapsto y$$

$$\overleftarrow{u} : x \mapsto (y, \bar{u}) \triangleq \begin{cases} y & := \quad u\, x \\ \bar{u} : \grave{y} \mapsto \grave{x} & \triangleq \quad \{\ \grave{x} \quad +:= \quad (\mathcal{D}\, u\, x) \times \grave{y} \end{cases}$$

Representing the Tape as Closures

Binary Primitives

$$b : (x, y) \mapsto z$$

$$\overleftarrow{b} : (x, y) \mapsto (z, \bar{b}) \triangleq \begin{cases} z & := \quad b\,(x, y) \\ \bar{b} : \grave{z} \mapsto (\grave{x}, \grave{y}) & \triangleq \quad \begin{cases} \grave{x} & +:= \quad (\mathcal{D}_1\, b\,(x, y)) \times \grave{z} \\ \grave{y} & +:= \quad (\mathcal{D}_2\, b\,(x, y)) \times \grave{z} \end{cases} \end{cases}$$

Representing the Tape as Closures

User-Defined Functions

$$f : \mathbf{x} \mapsto \mathbf{y} \quad \triangleq \quad \left\{ \begin{array}{lll} \mathbf{y}_1 & := & f_1\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n & := & f_n\ \mathbf{x}_n \end{array} \right.$$

$$\overleftarrow{f} : \mathbf{x} \mapsto (\mathbf{y}, \overline{f}) \quad \triangleq \quad \left\{ \begin{array}{lll} \mathbf{y}_1, \overline{f_1} & := & \overleftarrow{f_1}\ \mathbf{x}_1 \\ & \vdots & \\ \mathbf{y}_n, \overline{f_n} & := & \overleftarrow{f_n}\ \mathbf{x}_n \\ \overline{f} : \grave{\mathbf{y}} \mapsto \grave{\mathbf{x}} & \triangleq & \left\{ \begin{array}{lll} \grave{\mathbf{x}}_n & +:= & \overline{f_n}\ \grave{\mathbf{y}}_n \\ & \vdots & \\ \grave{\mathbf{x}}_1 & +:= & \overline{f_1}\ \grave{\mathbf{y}}_1 \end{array} \right. \end{array} \right.$$

Details for Handling Closures Omitted

Outline

Traditional Formulation of AD as Transformations

Forward Mode: $\mathbb{R}^n \rightarrow \mathbb{R}^m \rightsquigarrow (\mathbb{R}^n \times \mathbb{R}^n) \rightarrow (\mathbb{R}^m \times \mathbb{R}^m)$

Reverse Mode: $\mathbb{R}^n \rightarrow \mathbb{R}^m \rightsquigarrow (\mathbb{R}^n \rightarrow (\mathbb{R}^m \times \mathbb{R}^l)) \times ((\mathbb{R}^m \times \mathbb{R}^l) \rightarrow \mathbb{R}^n)$

New Formulation of AD as Higher-Order Functions

Perturbation Types

$$\overrightharpoon{\textbf{null}} = \textbf{null}$$

$$\overrightharpoon{\mathbb{R}} = \mathbb{R}$$

$$\overrightharpoon{\tau_1 \times \tau_2} = \overrightharpoon{\tau_1} \times \overrightharpoon{\tau_2}$$

$$\overrightharpoon{\tau_1 \xrightarrow{\tau_1', \ldots, \tau_n'} \tau_2} = \overrightharpoon{\tau_1'} \times \cdots \times \overrightharpoon{\tau_n'}$$

New Formulation of AD as Higher-Order Functions

Forward Types

$$\overrightharpoon{\mathbf{null}} = \mathbf{null} \times \acute{\mathbf{null}}$$

$$\overrightharpoon{\mathbb{R}} = \mathbb{R} \times \acute{\mathbb{R}}$$

$$\overrightharpoon{\tau_1 \times \tau_2} = \overrightharpoon{\tau_1} \times \overrightharpoon{\tau_2}$$

$$\overrightharpoon{\tau_1 \xrightarrow{\tau'_1,\ldots,\tau'_n} \tau_2} = \overrightharpoon{\tau_1} \xrightarrow{\overrightharpoon{\tau'_1},\ldots,\overrightharpoon{\tau'_n}} \overrightharpoon{\tau_2}$$

New Formulation of AD as Higher-Order Functions

Sensitivity Types

$$\overleftarrow{\textbf{null}} = \textbf{null}$$

$$\overleftarrow{\mathbb{R}} = \mathbb{R}$$

$$\overleftarrow{\tau_1 \times \tau_2} = \overleftarrow{\tau_1} \times \overleftarrow{\tau_2}$$

$$\overleftarrow{\tau_1 \xrightarrow{\tau'_1,\ldots,\tau'_n} \tau_2} = \overleftarrow{\tau'_1} \times \cdots \times \overleftarrow{\tau'_n}$$

New Formulation of AD as Higher-Order Functions

Reverse Types

$$\overleftarrow{\mathbf{null}} = \mathbf{null}$$

$$\overleftarrow{\mathbb{R}} = \mathbb{R}$$

$$\overleftarrow{\tau_1 \times \tau_2} = \overleftarrow{\tau_1} \times \overleftarrow{\tau_2}$$

$$\overleftarrow{\tau_1 \xrightarrow{\tau'_1,\ldots,\tau'_n} \tau_2} = \overleftarrow{\tau_1} \xrightarrow{\overleftarrow{\tau'_1},\ldots,\overleftarrow{\tau'_n}} (\overleftarrow{\tau_2} \times (\grave{\overline{\tau_2}} \to (\grave{\overline{\tau'_1}} \times \cdots \times \grave{\overline{\tau'_n}}) \times \grave{\overline{\tau_1}}))$$

New Formulation of AD as Higher-Order Functions

Forward Mode: $\overrightarrow{\mathcal{J}} : \tau \rightarrow \overrightarrow{\tau}$

Reverse Mode: $\overleftarrow{\mathcal{J}} : \tau \rightarrow \overleftarrow{\tau}$

Outline

Derivatives

$$\mathcal{D}\, f\, x \triangleq \textsc{Tangent}\ ((\overrightarrow{\mathcal{J}}\ f)\ (x \blacktriangleright 1))$$

$$\mathcal{D}\, f\, x \triangleq \textsc{Cdr}\ ((\textsc{Cdr}\ ((\overleftarrow{\mathcal{J}}\ f)\ (\overleftarrow{\mathcal{J}}\ x)))\ 1)$$

Roots using Newton-Raphson

$$\begin{aligned}\textsc{Root}\ (f, x_0, \epsilon) \triangleq\ & \textbf{let}\ x' \triangleq x_0 - \frac{f\ x_0}{\mathcal{D}\ f\ x_0} \\ & \textbf{in if}\ |x_0 - x'| \leq \epsilon\ \textbf{then}\ x_0\ \textbf{else}\ \textsc{Root}\ (f, x', \epsilon)\end{aligned}$$

Univariate Minimizer

Line Search

$$\textsc{LineSearch}\ (f, x_0, \epsilon) \triangleq \textsc{Root}\ ((\mathcal{D}\ f), x_0, \epsilon)$$

Gradients

$$\nabla\; f\; x \;\triangleq\; \textbf{let}\; n \triangleq \textsc{Length}\; x \;\textbf{in}\; \textsc{Map}\; ((\lambda i\; \textsc{Tangent}\; ((\overrightarrow{\mathcal{J}}\; f)\; (x \blacktriangleright e_{i,n}))), (\iota\; n))$$

$$\nabla\; f\; x \;\triangleq\; \textsc{Cdr}\; ((\textsc{Cdr}\; ((\overleftarrow{\mathcal{J}}\; f)\; (\overleftarrow{\mathcal{J}}\; x)))\; 1)$$

Multivariate Minimizer

Gradient Descent

$$
\begin{array}{l}
\textsc{GradientDescent}\ (f, x_0, \epsilon) \triangleq \\
\quad \textbf{let}\ g \triangleq \nabla\ f\ x_0 \\
\quad \textbf{in if}\ \|g\| \leq \epsilon \\
\qquad \textbf{then}\ x_0 \\
\qquad \textbf{else}\ \textsc{GradientDescent} \\
\qquad\qquad (f, (x_0 + ((\textsc{LineSearch}\ ((\lambda k\ f\ (x_0 + (k \times g))), \epsilon)) \times g)), \epsilon)
\end{array}
$$

Saddle Points

Continuous Two-Person Zero Sum Games

$\mathbf{x} : \mathbb{R}^m$

$\mathbf{y} : \mathbb{R}^n$

$\textsc{Payoff} : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$

$$\min_{\mathbf{x}} \max_{\mathbf{y}} \textsc{Payoff}\,(\mathbf{x}, \mathbf{y})$$

Saddle Points

Continuous Two-Person Zero Sum Games

$\mathbf{x} : \mathbb{R}^m$
$\mathbf{y} : \mathbb{R}^n$
$\textsc{Payoff} : \mathbb{R}^m \times \mathbb{R}^n \longrightarrow \mathbb{R}$
$\min_{\mathbf{x}} \max_{\mathbf{y}} \textsc{Payoff}\ (\mathbf{x}, \mathbf{y})$

$$\begin{aligned}(\mathbf{x}^*, \mathbf{y}^*) = \mathbf{let}\ \mathbf{x}^* &\triangleq \textsc{Argmin}\ ((\lambda\mathbf{x}\ \textsc{Max}\ ((\lambda\mathbf{y}\ \textsc{Payoff}\ (\mathbf{x}, \mathbf{y})), \mathbf{y}_0, \epsilon)), \mathbf{x}_0, \epsilon) \\ \mathbf{in}\ &(\mathbf{x}^*, (\textsc{Argmax}\ ((\lambda\mathbf{y}\ \textsc{Payoff}\ (\mathbf{x}^*, \mathbf{y})), \mathbf{y}_0, \epsilon)))\end{aligned}$$

Saddle Points

Continuous Two-Person Zero Sum Games

$\mathbf{x} : \mathbb{R}^m$
$\mathbf{y} : \mathbb{R}^n$
$\textsc{Payoff} : \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R}$
$\min_{\mathbf{x}} \max_{\mathbf{y}} \textsc{Payoff}\ (\mathbf{x}, \mathbf{y})$

$$\begin{aligned}(\mathbf{x}^*, \mathbf{y}^*) = \mathbf{let}\ \mathbf{x}^* &\triangleq \textsc{Argmin}\ ((\lambda\mathbf{x}\ \textsc{Max}\ ((\lambda\mathbf{y}\ \textsc{Payoff}\ (\mathbf{x}, \mathbf{y})), \mathbf{y}_0, \epsilon)), \mathbf{x}_0, \epsilon)\\ \mathbf{in}\ &(\mathbf{x}^*, (\textsc{Argmax}\ ((\lambda\mathbf{y}\ \textsc{Payoff}\ (\mathbf{x}^*, \mathbf{y})), \mathbf{y}_0, \epsilon)))\end{aligned}$$

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.

Carl Gauss
|
Christoph Gudermann
|
Karl Weierstrass
|
Hermann Schwarz
|
Leopold Fejér
|
John von Neumann

Function Inversion

$$f^{-1}\, y \triangleq \textsc{Root}\ ((\lambda x\ |(f\ x) - y|), x_0, \epsilon)$$

Neural Nets

$$\textsc{Neuron}\ (\mathbf{w}, \mathbf{x}) \triangleq \textsc{Sigmoid}\ (\mathbf{w} \cdot \mathbf{x})$$

$$\textsc{NeuralNet}\ ([\mathbf{w}''; \mathbf{w}'_1; \ldots; \mathbf{w}'_m], \mathbf{x}) \triangleq$$
$$\quad \textsc{Neuron}\ (\mathbf{w}'', [\textsc{Neuron}\ (\mathbf{w}'_1, \mathbf{x}); \ldots; \textsc{Neuron}\ (\mathbf{w}'_m, \mathbf{x})])$$

$$\textsc{Error}\ \mathbf{w} \triangleq$$
$$\quad \|[y_1; \ldots; y_n] - [\textsc{NeuralNet}\ (\mathbf{w}, \mathbf{x}_1); \ldots; \textsc{NeuralNet}\ (\mathbf{w}, \mathbf{x}_n)]\|$$

$$\textsc{GradientDescent}\ (\textsc{Error}, \mathbf{w}_0, \epsilon)$$

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**:533–6.

Supervised Machine Learning

Function Approximation

$$\text{ERROR}\ \theta \triangleq \|[y_1; \ldots; y_n] - [f\ (\theta, \mathbf{x}_1); \ldots; f\ (\theta, \mathbf{x}_n)]\|$$

$$\text{GRADIENTDESCENT}\ (\text{ERROR}, \theta_0, \epsilon)$$

Maximum Likelihood Estimation

$$\textsc{GradientDescent}\left(\left(\lambda\theta\left(-\prod_{\mathbf{x}\in\mathcal{X}} P(\mathbf{x}|\theta)\right)\right), \theta_0, \epsilon\right)$$

Fisher, R. A. (1922). On the mathematical foundations of theoretical statistics. *Philos. Trans. Roy. Soc. London Ser. A*, **222**:309–68.

Engineering Design

$$
\begin{aligned}
&\textsc{PerformanceOf SplineControlPoints} \triangleq \\
&\quad \textbf{let } \textsc{wing} \triangleq \textsc{SplineToSurface SplineControlPoints};\\
&\qquad \textsc{airflow} \triangleq \textsc{PDEsolver}\ (\textsc{wing}, \textsc{NavierStokes});\\
&\qquad \textsc{lift}, \textsc{drag} \triangleq \textsc{SurfaceIntegral}\ (\textsc{wing}, \textsc{airflow}, \textsc{force});\\
&\qquad \textsc{performance} \triangleq \textsc{DesignMetric}\ (\textsc{lift}, \textsc{drag}, (\textsc{weight wing}))\\
&\quad \textbf{in } \textsc{performance}
\end{aligned}
$$

$$\textsc{GradientDescent}\ (\textsc{PerformanceOf}, \textsc{SplineControlPoints}_0, \epsilon)$$

Outline

Our Approach is Efficient

Our Approach is Efficient

- source-to-source transformation

Our Approach is Efficient

- source-to-source transformation
- no overloading
- no interpretation of tape

Our Approach is Efficient

- source-to-source transformation
- no overloading
- no interpretation of tape
- transformation conceptually done reflectively at run-time

Our Approach is Efficient

- source-to-source transformation
- no overloading
- no interpretation of tape
- transformation conceptually done reflectively at run-time
- sophisticated compilation techniques can move transformation to compile-time

Our Approach Exhibits Closure Properties

Our Approach Exhibits Closure Properties

- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to *any* function

Our Approach Exhibits Closure Properties

- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to *any* function including $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ themselves.

Our Approach Exhibits Closure Properties

- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to *any* function including $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ themselves.
- The output of $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ are functions.

Our Approach Exhibits Closure Properties

- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to *any* function including $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ themselves.
- The output of $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ are functions.
- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to the output of $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$.

Our Approach Exhibits Closure Properties

- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to *any* function including $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ themselves.
- The output of $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ are functions.
- Can apply $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ to the output of $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$.
- Can take derivatives of arbitrary order.

Our Approach Enhances Modularity

$$\textsc{Argmin}_1\ (f, f') \quad \triangleq \quad \ldots$$

Our Approach Enhances Modularity

$$\text{ARGMIN}_1\ (f, f') \quad \triangleq \quad \ldots$$

$$\ldots \text{ARGMIN}_1\ (f, f') \ldots$$

Our Approach Enhances Modularity

$$\begin{aligned}\textsc{Argmin}_1\ (f, f') &\triangleq \ldots\\ \textsc{Argmin}_2\ (f, f', f'') &\triangleq \ldots\end{aligned}$$

$$\ldots\textsc{Argmin}_1\ (f, f')\ldots$$

Our Approach Enhances Modularity

$$\begin{aligned}\textsc{Argmin}_1\ (f, f') &\triangleq \ldots\\ \textsc{Argmin}_2\ (f, f', f'') &\triangleq \ldots\end{aligned}$$

$$\ldots\textsc{Argmin}_2\ (f, f', f'')\ldots$$

Our Approach Enhances Modularity

$$\textsc{Argmin}_1\, f \;\stackrel{\triangle}{=}\; \ldots(\overleftrightarrow{\mathcal{J}}\; f)\ldots$$

Our Approach Enhances Modularity

$$\textsc{Argmin}_1\, f \;\triangleq\; \ldots(\overleftrightarrow{\mathcal{J}}\; f)\ldots$$

$$\ldots\textsc{Argmin}_1\, f\ldots$$

Our Approach Enhances Modularity

$$\textsc{Argmin}_1\, f \;\stackrel{\triangle}{=}\; \ldots(\overleftrightarrow{\mathcal{J}}\, f)\ldots$$

$$\textsc{Argmin}_2\, f \;\stackrel{\triangle}{=}\; \ldots(\overleftrightarrow{\mathcal{J}}\, f)\ldots(\overleftrightarrow{\mathcal{J}}\, (\overleftrightarrow{\mathcal{J}}\, f))\ldots$$

$$\ldots\textsc{Argmin}_1\, f\ldots$$

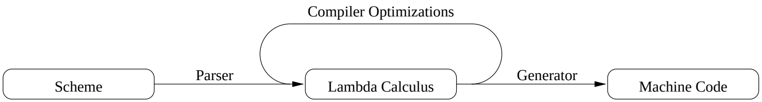
Our Approach Enhances Modularity

$$\textsc{Argmin}_1\, f \triangleq \ldots(\overleftrightarrow{\mathcal{J}}\, f)\ldots$$

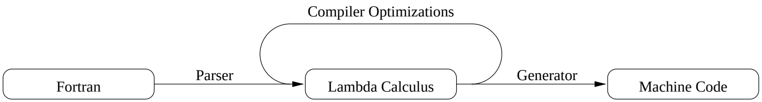
$$\textsc{Argmin}_2\, f \triangleq \ldots(\overleftrightarrow{\mathcal{J}}\, f)\ldots(\overleftrightarrow{\mathcal{J}}\,(\overleftrightarrow{\mathcal{J}}\, f))\ldots$$

$\ldots\textsc{Argmin}_2\, f\ldots$

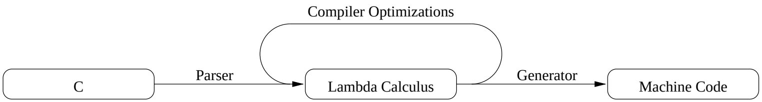
Lambda the Ultimate Intermediate Language



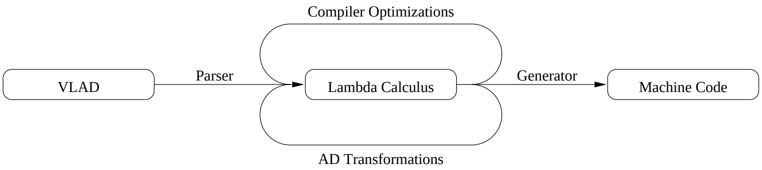
Lambda the Ultimate Intermediate Language



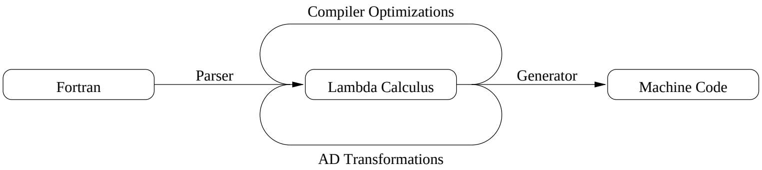
Lambda the Ultimate Intermediate Language



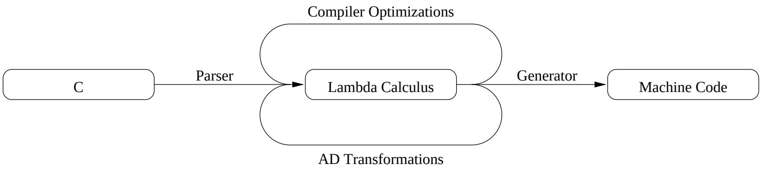
Lambda the Ultimate Intermediate Language *for AD*



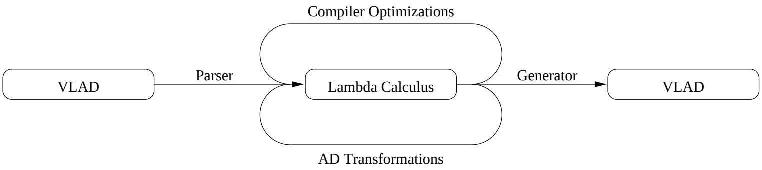
Lambda the Ultimate Intermediate Language *for AD*



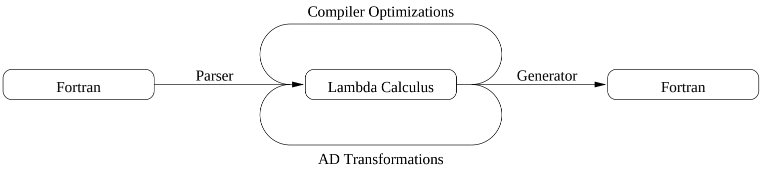
Lambda the Ultimate Intermediate Language *for AD*



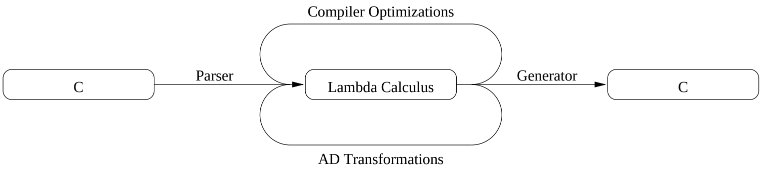
Lambda the Ultimate Intermediate Language *for AD*



Lambda the Ultimate Intermediate Language *for AD*



Lambda the Ultimate Intermediate Language *for AD*



Our Work

Our Work

- Prior Work

Our Work

- Prior Work

    STALIN compiler for SCHEME

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution
  20 × FORTRAN

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution
  20 $\times$ FORTRAN

- Current Work

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution
  $20 \times$ FORTRAN

- Current Work

  theory: $\lambda\nabla$-calculus

Our Work

- Prior Work

STALIN	compiler for SCHEME
	ruthless, brutal, good at execution
	$20 \times$ FORTRAN

- Current Work

theory:	$\lambda\nabla$-calculus
	λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution
  $20 \times$ FORTRAN

- Current Work

  theory: $\lambda\nabla$-calculus
  λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
  language: VLAD

Our Work

- Prior Work

STALIN	compiler for SCHEME
	ruthless, brutal, good at execution
	$20 \times$ FORTRAN

- Current Work

theory:	$\lambda\nabla$-calculus
	λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
language:	VLAD
	SCHEME $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$

Our Work

- Prior Work

STALIN	compiler for SCHEME
	ruthless, brutal, good at execution
	$20 \times$ FORTRAN

- Current Work

theory:	$\lambda\nabla$-calculus
	λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
language:	VLAD
	SCHEME $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
	<u>F</u>unctional <u>L</u>anguage for <u>AD</u>

Our Work

- Prior Work

STALIN	compiler for SCHEME
	ruthless, brutal, good at execution
	$20 \times$ FORTRAN

- Current Work

theory:	$\lambda\nabla$-calculus
	λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
language:	VLAD
	SCHEME $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
	Functional Language for AD
compiler:	STALIN∇

Our Work

- Prior Work

  STALIN compiler for SCHEME
  ruthless, brutal, good at execution
  $20 \times$ FORTRAN

- Current Work

  theory: $\lambda\nabla$-calculus
  λ-calculus $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
  language: VLAD
  SCHEME $+ \overrightarrow{\mathcal{J}} + \overleftarrow{\mathcal{J}}$
  Functional Language for AD
  compiler: STALIN∇

manuscripts and code:
`http://www-bcl.cs.nuim.ie/~qobi/stalingrad/`