Lazy Multivariate Higher-Order Forward-Mode AD

Barak A. Pearlmutter[1] Jeffrey Mark Siskind[2]

[1]Hamilton Institute, National University of Ireland Maynooth; barak@cs.nuim.ie
[2]School of Electrical and Computer Engineering, Purdue University; qobi@purdue.edu

Symposium on Principles of Programming Languages
18 January 2007

$$\mathcal{D}\, f\; c \;\triangleq\; \left.\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right|_{x=c}$$

Wengert (1964)

Higher-Order Forward-Mode AD

$$\mathcal{D}\ f\ c \quad \triangleq \quad \left.\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right|_{x=c}$$

$$\mathcal{D}^i\ f\ c \quad \triangleq \quad \left.\frac{\mathrm{d}^i f(x)}{\mathrm{d}x^i}\right|_{x=c}$$

Wengert (1964)

Higher-Order Forward-Mode AD

$$\mathcal{D}\, f\, c \;\triangleq\; \left.\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right|_{x=c}$$

$$\mathcal{D}^i\, f\, c \;\triangleq\; \left.\frac{\mathrm{d}^i f(x)}{\mathrm{d}x^i}\right|_{x=c}$$

$$[f(c), f'(c), f''(c), \ldots, f^{(i)}(c), \ldots]$$

Wengert (1964), Karczmarczuk (2001)

Lazy Multivariate Higher-Order Forward-Mode AD

$$\mathcal{D}\, f\, c \quad \triangleq \quad \left.\frac{\mathrm{d}f(x)}{\mathrm{d}x}\right|_{x=c}$$

$$\mathcal{D}^{i}\, f\, c \quad \triangleq \quad \left.\frac{\mathrm{d}^{i}f(x)}{\mathrm{d}x^{i}}\right|_{x=c}$$

$$[f(c), f'(c), f''(c), \ldots, f^{(i)}(c), \ldots]$$

$$\mathcal{D}^{[i_1,\ldots,i_n]}\, f\, [c_1, \ldots, c_n] \quad \triangleq \quad \left.\frac{\partial^{i_1+\cdots+i_n} f(x_1, \ldots, x_n)}{\partial {x_1}^{i_1} \cdots \partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}$$

Wengert (1964), Karczmarczuk (2001)

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε,

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε,

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1!

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c+\varepsilon$ is also a truncated power series.

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c+\varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by 1! (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

$(\mathcal{D}\, f)$ is $\mathcal{O}(1)$ relative to f (both space and time).

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\,f\;c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by $1!$ (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c+\varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

$(\mathcal{D}\,f)$ is $\mathcal{O}(1)$ relative to f (both space and time).

These $a+b\varepsilon$ are called *dual numbers* and can be represented as $\langle a, b\rangle$.

The Essence of Forward-Mode AD

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}\, f\ c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε, and
- multiply by $1!$ (noop).

Key idea: Only need output to be a **finite** truncated power series $a + b\varepsilon$.

The input $c + \varepsilon$ is also a truncated power series.

Can do a *nonstandard interpretation* of f over truncated power series.

Preserves control flow: Augments original values with derivatives.

$(\mathcal{D}\, f)$ is $\mathcal{O}(1)$ relative to f (both space and time).

These $a + b\varepsilon$ are called *dual numbers* and can be represented as $\langle a, b\rangle$.

(Analogous to complex numbers $a + bi$ represented as $\langle a, b\rangle$.)

Arithmetic on Truncated Power Series (i.e. Dual Numbers)

$$(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) + (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + \mathcal{O}(\varepsilon^2)$$

Arithmetic on Truncated Power Series (i.e. Dual Numbers)

$$(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) + (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) \times (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2))\\ = (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + \mathcal{O}(\varepsilon^2)\end{aligned}$$

Arithmetic on Truncated Power Series (i.e. Dual Numbers)

$$(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) + (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}&(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) \times (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2))\\ &\qquad = (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + \mathcal{O}(\varepsilon^2)\end{aligned}$$

$$u\ (x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (u\ x_0) + (x_1 \times (u'\ x_0))\varepsilon + \mathcal{O}(\varepsilon^2)$$

Arithmetic on Truncated Power Series (i.e. Dual Numbers)

$$(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) + (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) &\times (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) \\ &= (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + \mathcal{O}(\varepsilon^2)\end{aligned}$$

$$u\ (x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (u\ x_0) + (x_1 \times (u'\ x_0))\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}b\ ((x_0 + x_1\varepsilon &+ \mathcal{O}(\varepsilon^2)), (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2))) \\ &= (b\ (x_0, y_0)) + (x_1 \times (b^{(1,0)}\ (x_0, y_0)) + y_1 \times (b^{(0,1)}\ (x_0, y_0)))\varepsilon + \mathcal{O}(\varepsilon^2)\end{aligned}$$

Arithmetic on Truncated Power Series (i.e. Dual Numbers)

$$(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) + (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}
&(x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) \times (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2)) \\
&\quad = (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + \mathcal{O}(\varepsilon^2)
\end{aligned}$$

$$u\ (x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)) = (u\ x_0) + (x_1 \times (u'\ x_0))\varepsilon + \mathcal{O}(\varepsilon^2)$$

$$\begin{aligned}
&b\ ((x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)), (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2))) \\
&\quad = (b\ (x_0, y_0)) + (x_1 \times (b^{(1,0)}\ (x_0, y_0)) + y_1 \times (b^{(0,1)}\ (x_0, y_0)))\varepsilon + \mathcal{O}(\varepsilon^2)
\end{aligned}$$

Non-truncated is harder: Cannot ignore $\mathcal{O}(\varepsilon^2)$s.

Higher-Order Derivatives via Iteration

$$\mathcal{D}^i\, f\; c = \underbrace{(\mathcal{D}\;\cdots\;(\mathcal{D}}_{i}\; f)\cdots)\; c$$

Higher-Order Derivatives via Iteration

$$\mathcal{D}^i\, f\; c = \underbrace{(\mathcal{D}\;\cdots\;(\mathcal{D}}_{i}\; f)\cdots)\; c$$

$f : \mathbb{R} \to \mathbb{R}$.

Higher-Order Derivatives via Iteration

$$\mathcal{D}^i\, f\; c = \underbrace{(\mathcal{D}\; \cdots\; (\mathcal{D}}_{i}\; f) \cdots)\; c$$

$f : \mathbb{R} \rightarrow \mathbb{R}$.

$(\mathcal{D}\, f)$ lifts f to $\mathbb{D}(\mathbb{R}) \rightarrow \mathbb{D}(\mathbb{R})$.

Higher-Order Derivatives via Iteration

$$\mathcal{D}^i\, f\; c = \underbrace{(\mathcal{D}\ \cdots\ (\mathcal{D}}_{i}\ f)\cdots)\ c$$

$f : \mathbb{R} \to \mathbb{R}$.

$(\mathcal{D}\, f)$ lifts f to $\mathbb{D}(\mathbb{R}) \to \mathbb{D}(\mathbb{R})$.

$(\mathcal{D}\ (\mathcal{D}\, f))$ lifts f to $\mathbb{D}(\mathbb{D}(\mathbb{R})) \to \mathbb{D}(\mathbb{D}(\mathbb{R}))$.

Higher-Order Derivatives via Iteration

$$\mathcal{D}^i\, f\; c = \underbrace{(\mathcal{D}\ \cdots\ (\mathcal{D}}_{i}\ f)\cdots)\ c$$

$f : \mathbb{R} \to \mathbb{R}$.

$(\mathcal{D}\, f)$ lifts f to $\mathbb{D}(\mathbb{R}) \to \mathbb{D}(\mathbb{R})$.

$(\mathcal{D}\ (\mathcal{D}\, f))$ lifts f to $\mathbb{D}(\mathbb{D}(\mathbb{R})) \to \mathbb{D}(\mathbb{D}(\mathbb{R}))$.

Need mechanism to support arbitrary nesting of power series.

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i,

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i,

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i, and
- multiply by $i!$

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\; c$:

- evaluate f at the **term** $c + \varepsilon$ to get a **power series**,
- extract the coefficient of ε^i, and
- multiply by $i!$ (not a noop).

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\; c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i, and
- multiply by $i!$ (not a noop).

Good news: The input $c+\varepsilon$ is (a special case of) a power series.

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\ c$:

- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i, and
- multiply by $i!$ (not a noop).

Good news: The input $c+\varepsilon$ is (a special case of) a power series.

Can do a *nonstandard interpretation* of f over power series.

Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

$$f(c+\varepsilon) = \frac{f(c)}{0!} + \frac{f'(c)}{1!}\varepsilon + \frac{f''(c)}{2!}\varepsilon^2 + \cdots + \frac{f^{(i)}(c)}{i!}\varepsilon^i + \cdots$$

To compute $\mathcal{D}^i\, f\, c$:

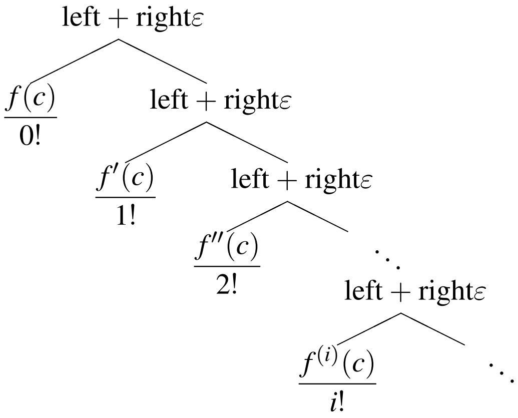
- evaluate f at the **term** $c+\varepsilon$ to get a **power series**,
- extract the coefficient of ε^i, and
- multiply by $i!$ (not a noop).

Good news: The input $c+\varepsilon$ is (a special case of) a power series.

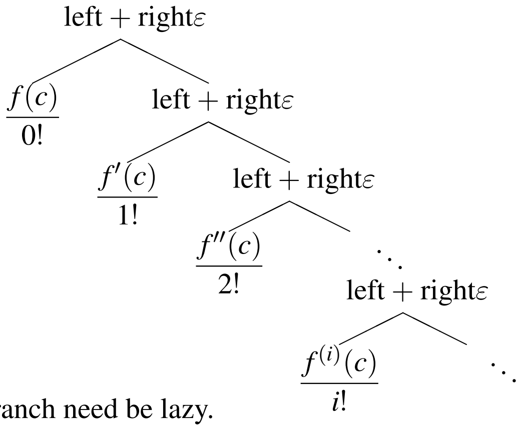
Can do a *nonstandard interpretation* of f over power series.

Bad news: The power series may be **infinite**.

Solution: Represent Power Series as Lazy Streams

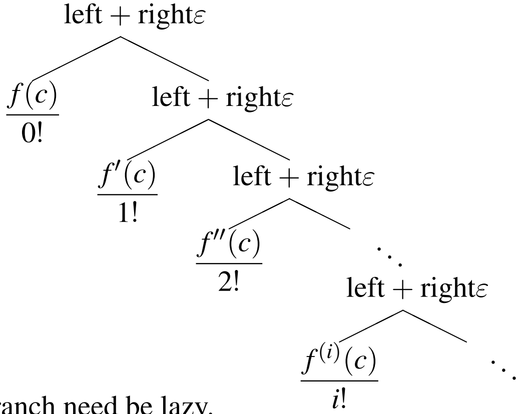


Solution: Represent Power Series as Lazy Streams



Only the right branch need be lazy.

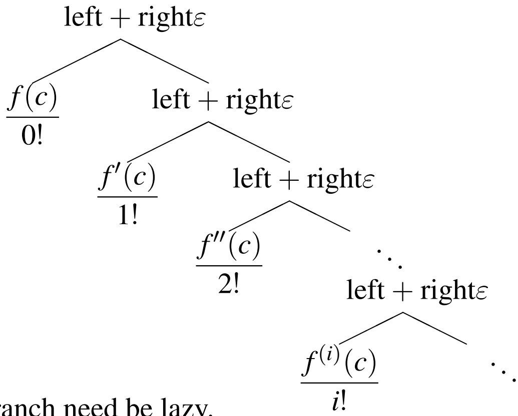
Solution: Represent Power Series as Lazy Streams



Only the right branch need be lazy.

API: $(\mathcal{Q}\ \varepsilon\ p)$ computes *quotient* of $\frac{p}{\varepsilon}$, analogous to forcing `cdr`.

Solution: Represent Power Series as Lazy Streams



Only the right branch need be lazy.

API: $(\mathcal{Q}\ \varepsilon\ p)$ computes *quotient* of $\frac{p}{\varepsilon}$, analogous to forcing `cdr`.

$(\mathcal{R}\ \varepsilon\ p)$ computes *remainder* of $\frac{p}{\varepsilon}$, analogous to `car`.

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty} \cdots \sum_{i_n=0}^{\infty} \frac{1}{i_1! \cdots i_n!} \left. \frac{\partial^{i_1+\cdots+i_n} f(x_1, \ldots, x_n)}{\partial {x_1}^{i_1} \cdots \partial {x_n}^{i_n}} \right|_{x_1=c_1,\ldots,x_n=c_n} {\varepsilon_1}^{i_1} \cdots {\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\ [c_1, \ldots, c_n]$:

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\, [c_1,\ldots,c_n]$:

- evaluate f

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\,f\,[c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\ [c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$ to get a **multivariate** power series,

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\,f\;[c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$,

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\, [c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$,

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\,f\,[c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$, and
- multiply by $i_1!\cdots i_n!$.

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty}\cdots\sum_{i_n=0}^{\infty}\frac{1}{i_1!\cdots i_n!}\left.\frac{\partial^{i_1+\cdots+i_n}f(x_1,\ldots,x_n)}{\partial {x_1}^{i_1}\cdots\partial {x_n}^{i_n}}\right|_{x_1=c_1,\ldots,x_n=c_n}{\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\,f\,[c_1,\ldots,c_n]$:

- evaluate f at $(c_1+\varepsilon_1),\ldots,(c_n+\varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1}\cdots{\varepsilon_n}^{i_n}$, and
- multiply by $i_1!\cdots i_n!$.

Good news: Can do a *nonstandard interpretation* of f over **multivariate** power series.

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty} \cdots \sum_{i_n=0}^{\infty} \frac{1}{i_1! \cdots i_n!} \left. \frac{\partial^{i_1+\cdots+i_n} f(x_1, \ldots, x_n)}{\partial {x_1}^{i_1} \cdots \partial {x_n}^{i_n}} \right|_{x_1=c_1,\ldots,x_n=c_n} {\varepsilon_1}^{i_1} \cdots {\varepsilon_n}^{i_n}$$

To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\ [c_1, \ldots, c_n]$:

- evaluate f at $(c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1} \cdots {\varepsilon_n}^{i_n}$, and
- multiply by $i_1! \cdots i_n!$.

Good news: Can do a *nonstandard interpretation* of f over **multivariate** power series.

Bad news: Need a distinct ε_i for each argument of f

Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

$$f((c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)) =$$

$$\sum_{i_1=0}^{\infty} \cdots \sum_{i_n=0}^{\infty} \frac{1}{i_1! \cdots i_n!} \left. \frac{\partial^{i_1+\cdots+i_n} f(x_1, \ldots, x_n)}{\partial {x_1}^{i_1} \cdots \partial {x_n}^{i_n}} \right|_{x_1=c_1,\ldots,x_n=c_n} {\varepsilon_1}^{i_1} \cdots {\varepsilon_n}^{i_n}$$

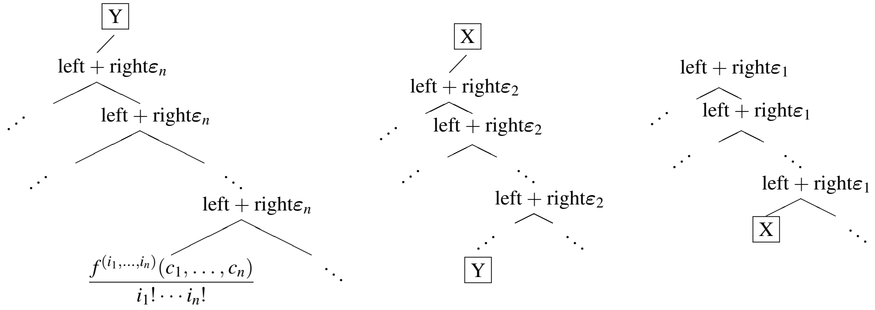
To compute $\mathcal{D}^{[i_1,\ldots,i_n]}\, f\; [c_1, \ldots, c_n]$:

- evaluate f at $(c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)$ to get a **multivariate** power series,
- extract the coefficient of ${\varepsilon_1}^{i_1} \cdots {\varepsilon_n}^{i_n}$, and
- multiply by $i_1! \cdots i_n!$.

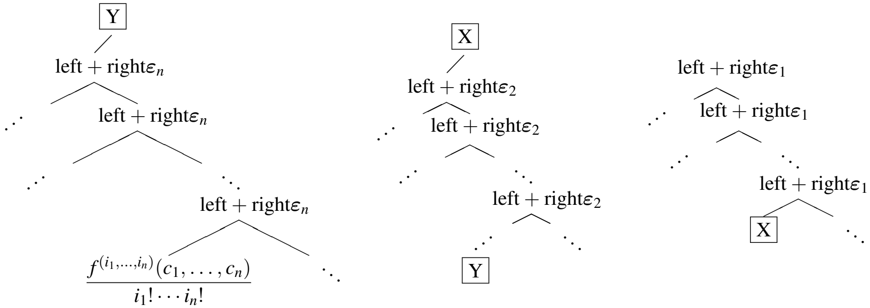
Good news: Can do a *nonstandard interpretation* of f over **multivariate** power series.

Bad news: Need a distinct ε_i for each argument of f
(and for each nested invocation of $\mathcal{D}$, **even in the univariate case**).

Multivariate Power Series as Nested Univariate Power Series

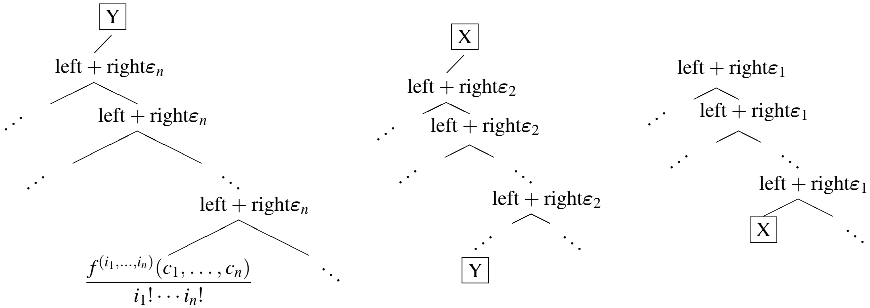


Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

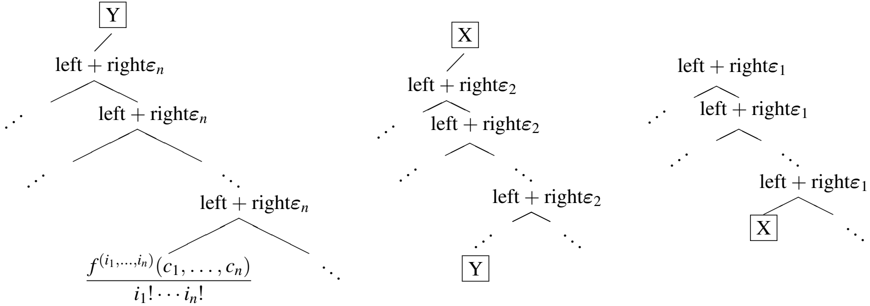
Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

Only the right branch need be lazy.

Multivariate Power Series as Nested Univariate Power Series

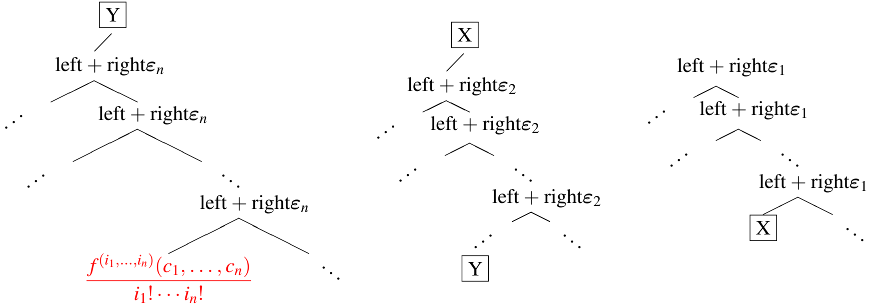


The left-branching depth is limited by the number of distinct ε_i.

Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Multivariate Power Series as Nested Univariate Power Series



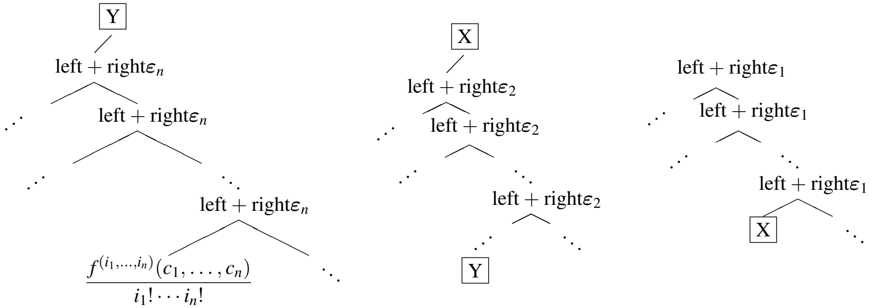
The left-branching depth is limited by the number of distinct ε_i.

Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

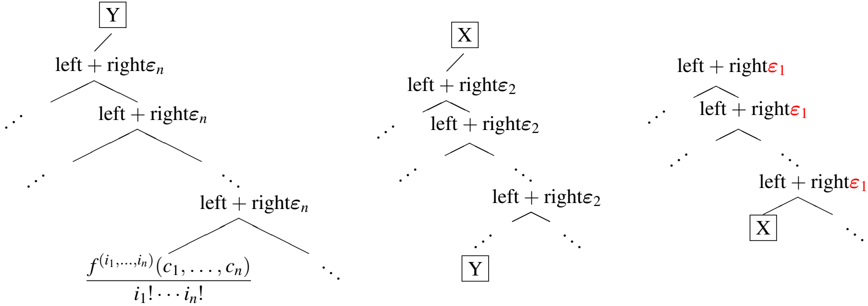
Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to **generate** a distinct type or tag for each ε_i.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

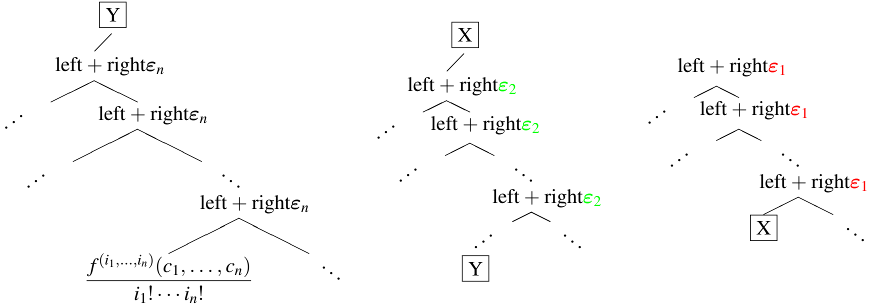
Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to **generate** a distinct type or tag for each ε_i.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

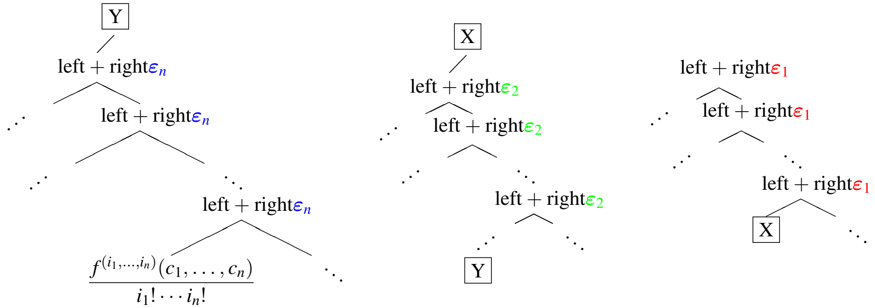
Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to generate a distinct type or tag for each ε_i.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

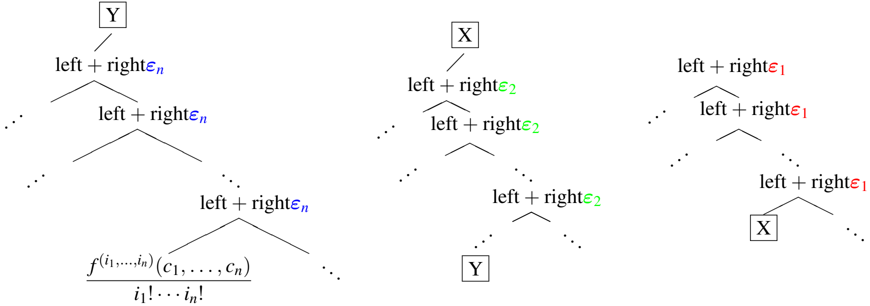
Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to **generate** a distinct type or tag for each ε_i.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

Only the right branch need be lazy.

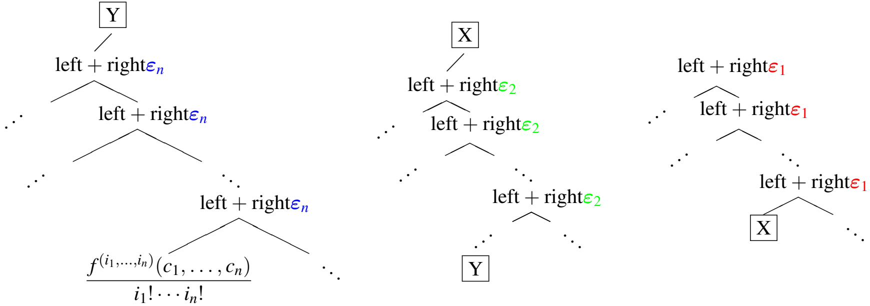
The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to **generate** a distinct type or tag for each ε_i.

Cannot do this in a referentially transparent language.

Multivariate Power Series as Nested Univariate Power Series



The left-branching depth is limited by the number of distinct ε_i.

Only the right branch need be lazy.

The same $(\mathcal{Q}\ \varepsilon\ p)$ and $(\mathcal{R}\ \varepsilon\ p)$ API is sufficient.

Good news: Each higher-order partial derivative appears **exactly once**.

Bad news: Need to **generate** a distinct type or tag for each ε_i.

Cannot do this in a referentially transparent language.

Painfully ironic: Cannot *implement* $\mathcal{D}$ in a referentially transparent language even though $\mathcal{D}$ itself *is* referentially transparent!

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon^0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon 0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon 0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon 0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon 0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$
- Requires u', the derivative of u.

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon^0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon^0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$
- Requires u', the derivative of u.
- Requires that u' be (recursively) evaluated under the nonstandard interpretation to compute $u'\ (x + x'\varepsilon)$.

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon 0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon 0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$
- Requires u', the derivative of u.
- Requires that u' be (recursively) evaluated under the nonstandard interpretation to compute $u'\ (x + x'\varepsilon)$.
- Binary primitives can be curried when arguments are power series over different εs.

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon^0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon^0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$
- Requires u', the derivative of u.
- Requires that u' be (recursively) evaluated under the nonstandard interpretation to compute $u'\ (x + x'\varepsilon)$.
- Binary primitives can be curried when arguments are power series over different εs.
- Can rename when arguments are power series over the same ε:

$$b\ ((x + x'\varepsilon), (y + y'\varepsilon)) = (b\ ((x + x'\varepsilon), (y + y'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon]$$

Arithmetic on Non-Truncated Power Series

- Unary primitives:

$$u\ (x + x'\varepsilon) = (u\ x) + ((\mathcal{C}_{\varepsilon^0}\ (u'\ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon$$

- Bookkeeping: $\mathcal{C}_{\varepsilon^0}$, $[\varepsilon \mapsto \xi]$, $[\xi \mapsto \varepsilon]$
- Requires u', the derivative of u.
- Requires that u' be (recursively) evaluated under the nonstandard interpretation to compute $u'\ (x + x'\varepsilon)$.
- Binary primitives can be curried when arguments are power series over different εs.
- Can rename when arguments are power series over the same ε:

$$b\ ((x + x'\varepsilon), (y + y'\varepsilon)) = (b\ ((x + x'\varepsilon), (y + y'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon]$$

- Read the paper for the details.

Wrap Up

- Functional programming has had little impact on numerical computing.

Wrap Up

- Functional programming has had little impact on numerical computing.
- Many important numeric concepts are higher-order functions.

Wrap Up

- Functional programming has had little impact on numerical computing.
- Many important numeric concepts are higher-order functions.
- For functional programming to interest numerical computing, it should provide useful numeric constructs.

Wrap Up

- Functional programming has had little impact on numerical computing.
- Many important numeric concepts are higher-order functions.
- For functional programming to interest numerical computing, it should provide useful numeric constructs.
- For instance: *exact efficient derivatives!*

Wrap Up

- Functional programming has had little impact on numerical computing.
- Many important numeric concepts are higher-order functions.
- For functional programming to interest numerical computing, it should provide useful numeric constructs.
- For instance: *exact efficient derivatives!*
- We have shown how to implement an unrestricted multivariate higher-order derivative operator using forward-mode AD.

Contingency Slides

Forward AD of Non-Scalar Functions

Discussed scalar functions for expository simplicity

- Can generalize higher-order scalar derivative

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

  to higher-order vector directional derivative

$$\mathcal{J} : \mathbb{N} \times (\mathbb{R}^n \to \mathbb{R}^m) \to (\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m)$$

- using same mechanisms: find directional i-th derivative $\mathcal{J}\ i\ f\ \mathbf{c}\ \mathbf{c}'$ of $f : \mathbb{R}^n \to \mathbb{R}^m$ at $\mathbf{c} : \mathbb{R}^n$ in direction $\mathbf{c}' : \mathbb{R}^n$ by calculating

$$\mathbf{y} = f\ [c_1 + c_1'\varepsilon, \ldots, c_n + c_n'\varepsilon]$$

  and extracting

$$[y_1', \ldots, y_m'] = [\mathcal{C}_{\varepsilon^i}\ y_1, \ldots, \mathcal{C}_{\varepsilon^i}\ y_m]$$

Representation and Factorials: A Technicality

Two alternatives for representing

$$x(\varepsilon) = x_0 + x_1\varepsilon + x_2\varepsilon^2 + x_3\varepsilon^3 + \cdots$$

Representation and Factorials: A Technicality

Two alternatives for representing

$$x(\varepsilon) = x_0 + x_1\varepsilon + x_2\varepsilon^2 + x_3\varepsilon^3 + \cdots$$

- Tower of Coefficients (without factorials)

$$\langle x_0, \langle x_1, \langle x_2, \langle x_3, \ldots\rangle\rangle\rangle\rangle$$

- Tower of Derivatives (with factorials)

$$\begin{aligned}
&\langle x(0), \langle x'(0), \langle x''(0), \langle x'''(0), \ldots\rangle\rangle\rangle\rangle \\
&\quad = \langle 0! \times x_0, \langle 1! \times x_1, \langle 2! \times x_2, \langle 3! \times x_3, \ldots\rangle\rangle\rangle\rangle \\
&\quad\quad = \langle x_0, \langle x_1, \langle 2 \times x_2, \langle 6 \times x_3, \ldots\rangle\rangle\rangle\rangle
\end{aligned}$$

Representation and Factorials: A Technicality

Two alternatives for representing

$$x(\varepsilon) = x_0 + x_1\varepsilon + x_2\varepsilon^2 + x_3\varepsilon^3 + \cdots$$

- Tower of Coefficients (without factorials)

$$\langle x_0, \langle x_1, \langle x_2, \langle x_3, \ldots\rangle\rangle\rangle\rangle$$

- Tower of Derivatives (with factorials)

$$\begin{aligned}\langle x(0), \langle x'(0), \langle x''(0), \langle x'''(0), \ldots\rangle\rangle\rangle\rangle \\ = \langle 0! \times x_0, \langle 1! \times x_1, \langle 2! \times x_2, \langle 3! \times x_3, \ldots\rangle\rangle\rangle\rangle \\ = \langle x_0, \langle x_1, \langle 2 \times x_2, \langle 6 \times x_3, \ldots\rangle\rangle\rangle\rangle\end{aligned}$$

- Identical in truncated case
- Fungible: trade off which "left shift" is fast,

$$\mathcal{Q}\ \varepsilon\ x(\varepsilon) = \frac{1}{\varepsilon}(x(\varepsilon) - x(0)) \qquad \text{or} \qquad \frac{\mathrm{d}}{\mathrm{d}\varepsilon}x(\varepsilon)$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

$$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{x} \ldots \boxed{y} \ldots)\ c) \ldots)\ c$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

-

$$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \color{red}{\mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c}$$

-

$$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots} \boxed{x} \color{red}{\ldots} \boxed{\color{red}{y}} \color{red}{\ldots)\ c)} \ldots)\ c$$

-

$$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots} \boxed{(x+x)} \color{red}{\ldots} \boxed{(x+\color{red}{y})} \color{red}{\ldots)\ c)} \ldots)\ c$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \color{red}{\mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c}$$
- $$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots} \boxed{x} \color{red}{\ldots} \boxed{\color{red}{y}} \color{red}{\ldots)\ c)} \ldots)\ c$$
- $$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots} \boxed{(x+x)} \color{red}{\ldots} \boxed{(x+\color{red}{y})} \color{red}{\ldots)\ c)} \ldots)\ c$$

$$\text{referential transparency} \implies x = \color{red}{y} \qquad (\forall \text{ cases})$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{x} \ldots \boxed{y} \ldots)\ c) \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{(x+x)} \ldots \boxed{(x+y)} \ldots)\ c) \ldots)\ c$$

$$\text{referential transparency} \implies x = y \qquad (\forall\ \text{cases})$$

$$x = y \implies x + x = x + y$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{x} \ldots \boxed{y} \ldots)\ c) \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{(x+x)} \ldots \boxed{(x+y)} \ldots)\ c) \ldots)\ c$$

$$\begin{aligned} \text{referential transparency} &\implies x = y \qquad (\forall \text{ cases}) \\ x = y &\implies x + x = x + y \\ x + x = x + y &\implies \text{getting the wrong answer} \end{aligned}$$

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{x} \ldots \boxed{y} \ldots)\ c) \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{(x+x)} \ldots \boxed{(x+y)} \ldots)\ c) \ldots)\ c$$

$$\begin{aligned}
\text{referential transparency} &\implies x = y \qquad (\forall\ \text{cases})\\
x = y &\implies x + x = x + y\\
x + x = x + y &\implies \text{getting the wrong answer}
\end{aligned}$$

therefore

Implementation of $\mathcal{D}$ *vs.* Referential Transparency

$$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \color{red}{\mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c}$$

$$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots}\boxed{x}\color{red}{\ldots}\boxed{\color{red}{y}}\color{red}{\ldots)\ c)}\ldots)\ c$$

$$\mathcal{D}\ (\lambda x\ \ldots \color{red}{(\mathcal{D}\ (\lambda y\ \ldots}\boxed{(x+x)}\color{red}{\ldots}\boxed{(x+\color{red}{y})}\color{red}{\ldots)\ c)}\ldots)\ c$$

$$\begin{aligned}
\text{referential transparency} &\implies x = \color{red}{y} \qquad (\forall\ \text{cases})\\
x = \color{red}{y} &\implies x + x = x + \color{red}{y}\\
x + x = x + \color{red}{y} &\implies \text{getting the wrong answer}
\end{aligned}$$

therefore

$$\text{getting the right answer} \implies \neg\text{referential transparency}$$

- $$\mathcal{D}\ (\lambda x\ \ldots \boxed{x} \ldots)\ c \qquad\qquad \mathcal{D}\ (\lambda y\ \ldots \boxed{y} \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{x} \ldots \boxed{y} \ldots)\ c) \ldots)\ c$$

- $$\mathcal{D}\ (\lambda x\ \ldots (\mathcal{D}\ (\lambda y\ \ldots \boxed{(x+x)} \ldots \boxed{(x+y)} \ldots)\ c) \ldots)\ c$$

$$\begin{aligned} \text{referential transparency} &\implies x = y \qquad (\forall\ \text{cases}) \\ x = y &\implies x + x = x + y \\ x + x = x + y &\implies \text{getting the wrong answer} \end{aligned}$$

therefore

$$\text{getting the right answer} \implies \neg\text{referential transparency}$$

(Oops)