Lazy Multivariate Higher-Order Forward-Mode AD

Barak A. Pearlmutter\textsuperscript{1}  Jeffrey Mark Siskind\textsuperscript{2}

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

Symposium on Principles of Programming Languages
18 January 2007
\[ D f \ c \triangleq \left. \frac{df(x)}{dx} \right|_{x=c} \]

Wengert (1964)
\[ \mathcal{D} f \Delta c \triangleq \left. \frac{df(x)}{dx} \right|_{x=c} \]

\[ \mathcal{D}^i f \Delta c \triangleq \left. \frac{d^i f(x)}{dx^i} \right|_{x=c} \]

Wengert (1964)
\[ \mathcal{D} f \ c \triangleq \left. \frac{df(x)}{dx} \right|_{x=c} \]

\[ \mathcal{D}^i f \ c \triangleq \left. \frac{d^i f(x)}{dx^i} \right|_{x=c} \]

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

Wengert (1964), Karczmarczuk (2001)
\[ \mathcal{D} f \ c \triangleq \frac{df(x)}{dx}\bigg|_{x=c} \]

\[ \mathcal{D}^i f \ c \triangleq \frac{d^i f(x)}{dx^i}\bigg|_{x=c} \]

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

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

Wengert (1964), Karczmarczuk (2001)
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 \( \mathcal{D} f \mid c \):

- evaluate \( f \)
The Essence of Forward-Mode AD

Taylor expansion:

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

To compute \( Df(c) \):

- 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 \( D f c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
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 \( D f c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \),
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 \(D_f c\):

- evaluate \(f\) at the term \(c + \varepsilon\) to get a power series,
- extract the coefficient of \(\varepsilon\),
The Essence of Forward-Mode AD

Taylor expansion:

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

To compute \( D f c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \), 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 \( \varepsilon \), 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 \(Df\): c:

- evaluate \(f\) at the term \(c + \varepsilon\) to get a power series,
- extract the coefficient of \(\varepsilon\), 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 \( Df \, c \):
- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \), 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 \( Df(c) \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \), 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 \( Df c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \), 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 \(D f c\):

- evaluate \(f\) at the term \(c + \varepsilon\) to get a power series,
- extract the coefficient of \(\varepsilon\), 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.

\((D f)\) is \(O(1)\) 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 $\varepsilon$, 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 \( Df \mid c \):
- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon \), 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. 

\( (Df) \) is \( O(1) \) 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)

\[(x_0 + x_1 \varepsilon + O(\varepsilon^2)) + (y_0 + y_1 \varepsilon + O(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + O(\varepsilon^2)\]

\[(x_0 + x_1 \varepsilon + O(\varepsilon^2)) \times (y_0 + y_1 \varepsilon + O(\varepsilon^2)) = (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + O(\varepsilon^2)\]
Arithmetic on Truncated Power Series (i.e. Dual Numbers)

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

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

\[u (x_0 + x_1 \varepsilon + \mathcal{O}(\varepsilon^2)) = (u x_0) + (x_1 \times (u' x_0)) \varepsilon + \mathcal{O}(\varepsilon^2)\]
Arithmetic on Truncated Power Series (i.e. Dual Numbers)

\[(x_0 + x_1 \varepsilon + O(\varepsilon^2)) + (y_0 + y_1 \varepsilon + O(\varepsilon^2)) = (x_0 + y_0) + (x_1 + y_1)\varepsilon + O(\varepsilon^2)\]

\[(x_0 + x_1 \varepsilon + O(\varepsilon^2)) \times (y_0 + y_1 \varepsilon + O(\varepsilon^2)) = (x_0 \times y_0) + (x_0 \times y_1 + x_1 \times y_0)\varepsilon + O(\varepsilon^2)\]

\[u (x_0 + x_1 \varepsilon + O(\varepsilon^2)) = (u x_0) + (x_1 \times (u' x_0))\varepsilon + O(\varepsilon^2)\]

\[b ((x_0 + x_1 \varepsilon + O(\varepsilon^2)), (y_0 + y_1 \varepsilon + O(\varepsilon^2))) = (b (x_0, y_0)) + (x_1 \times (b^{(1,0)} (x_0, y_0)) + y_1 \times (b^{(0,1)} (x_0, y_0)))\varepsilon + O(\varepsilon^2)\]
Arithmetic on Truncated Power Series (i.e. Dual Numbers)

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

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

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

\[b \ ((x_0 + x_1\varepsilon + \mathcal{O}(\varepsilon^2)), (y_0 + y_1\varepsilon + \mathcal{O}(\varepsilon^2))) = (b \ (x_0, y_0)) + (x_1 \times (b^{(1,0)} \ (x_0, y_0)) + y_1 \times (b^{(0,1)} \ (x_0, y_0)))\varepsilon + \mathcal{O}(\varepsilon^2)\]

Non-truncated is harder: Cannot ignore \(\mathcal{O}(\varepsilon^2)\)s.
\[ D^i f c = (D \cdots (D f) \cdots ) c \]
Higher-Order Derivatives via Iteration

\[ \mathcal{D}^i f \ c = (\mathcal{D} \cdots (\mathcal{D} \ f) \cdots)_i \ c \]

\( f : \mathbb{R} \rightarrow \mathbb{R} \).
Higher-Order Derivatives via Iteration

\[ \mathcal{D}^i f \ p = (\mathcal{D} \cdots (\mathcal{D} \ (f) \cdots) \ p \right) \]

\( f : \mathbb{R} \to \mathbb{R} \).

\((\mathcal{D} f)\) lifts \( f \) to \( \mathbb{D}(\mathbb{R}) \to \mathbb{D}(\mathbb{R}) \).
\[ \mathcal{D}^i f \ c = (\mathcal{D} \cdots (\mathcal{D} \ f) \cdots)_i \ c \]

\[ f : \mathbb{R} \rightarrow \mathbb{R}. \]

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

\((\mathcal{D} (\mathcal{D} f))\) lifts \(f\) to \(\mathbb{D}(\mathbb{D}(\mathbb{R})) \rightarrow \mathbb{D}(\mathbb{D}(\mathbb{R}))\).
Higher-Order Derivatives via Iteration

\[ D^i f \ c = (D \cdots (D f) \cdots)_i c \]

\( f : \mathbb{R} \rightarrow \mathbb{R} \).

(\( Df \)) lifts \( f \) to \( \mathbb{D}(\mathbb{R}) \rightarrow \mathbb{D}(\mathbb{R}) \).

(\( D(Df) \)) lifts \( f \) to \( \mathbb{D} (\mathbb{D}(\mathbb{R})) \rightarrow \mathbb{D}(\mathbb{D}(\mathbb{R})) \).

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 \( \mathcal{D}^i f \mid_c \):
Higher-Order Derivatives via Taylor Expansion

Taylor expansion:

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

To compute \( D^i f \): 

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
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 \( D^i f c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon^i \),
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 \(D^i f c\):

- evaluate \(f\) at the term \(c + \varepsilon\) to get a power series,
- extract the coefficient of \(\varepsilon^i\),
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 \( D^i f c \):

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon^i \), 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 \( D^i f \ c \):

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

- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon^i \), 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:

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

To compute \( D^i f c \):
- evaluate \( f \) at the term \( c + \varepsilon \) to get a power series,
- extract the coefficient of \( \varepsilon^i \), 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:

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

To compute \( D^i f \mid c \):

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

\[
\begin{align*}
\left. \frac{f(c)}{0!} \right. & + \left. \frac{f'(c)}{1!} \right. + \left. \frac{f''(c)}{2!} \right. + \cdots + \left. \frac{f^{(i)}(c)}{i!} \right. + \cdots \\
\left. \varepsilon \right. & + \left. \varepsilon \right. \\
\left. \varepsilon \right. & + \left. \varepsilon \right. \\
\left. \varepsilon \right. & + \left. \varepsilon \right. \\
\end{align*}
\]
Solution: Represent Power Series as Lazy Streams

\[
\begin{align*}
&\text{left + right} & \varepsilon \\
&\frac{f(c)}{0!} & \text{left + right} & \varepsilon \\
&\frac{f'(c)}{1!} & \text{left + right} & \varepsilon \\
&\frac{f''(c)}{2!} & \text{left + right} & \varepsilon \\
&\frac{f^{(i)}(c)}{i!} & \text{left + right} & \varepsilon \\
\end{align*}
\]

Only the right branch need be lazy.
Solution: Represent Power Series as Lazy Streams

Only the right branch need be lazy.

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

Diagram:

```
left + right $\varepsilon$

\[ f(c) \]
\[ \frac{f'(c)}{1!} \]
\[ \frac{f''(c)}{2!} \]
\[ \frac{f(i)(c)}{i!} \]

\dots

left + right $\varepsilon$

left + right $\varepsilon$

left + right $\varepsilon$

left + right $\varepsilon$
```
Solution: Represent Power Series as Lazy Streams

\[
\begin{align*}
\left. f(c) \right|_{0!} & \left. + \right. \left. \varepsilon \right. \\
\left. f'(c) \right|_{1!} & \left. + \right. \left. \varepsilon \right. \\
\left. f''(c) \right|_{2!} & \left. + \right. \left. \varepsilon \right. \\
\left. \cdots \right. & \left. \cdots \right. \\
\left. f^{(i)}(c) \right|_{i!} & \left. + \right. \left. \varepsilon \right. \\
\end{align*}
\]

Only the right branch need be lazy.

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

\((R \varepsilon p)\) computes *remainder* of \(\frac{p}{\varepsilon}\), analogous to \(\text{car}\).
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 \( D^{[i_1, \ldots, i_n]} f [c_1, \ldots, c_n] \):
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 \( D^{[i_1, \ldots, i_n]} f \ [c_1, \ldots, c_n] \):

- evaluate \( f \)
Higher-Order Multivariate Derivatives

Multivariate Taylor expansion:

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

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

- evaluate \( f \) at \((c_1 + \varepsilon_1), \ldots, (c_n + \varepsilon_n)\)
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 \( D^{[i_1, \ldots, i_n]} f [c_1, \ldots, c_n] \):

- 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:

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

\[ \sum_{i_1=0}^{\infty} \cdots \sum_{i_n=0}^{\infty} \frac{1}{i_1! \cdots i_n!} \frac{\partial^{i_1+\cdots+i_n} f(x_1, \ldots, x_n)}{\partial x_1^{i_1} \cdots \partial x_n^{i_n}} \bigg|_{x_1=c_1, \ldots, x_n=c_n} \]

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

- 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:

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

To compute \( D^{[i_1, \ldots, i_n]} f [c_1, \ldots, c_n] \):
- 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:

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

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

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

- 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:

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

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

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

- 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:

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

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

- 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 \( \varepsilon_i \) for each argument of \( f \)
Multivariate Taylor expansion:

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

To compute \(D[i_1, \ldots, i_n] f [c_1, \ldots, c_n]\):

- 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 \(\varepsilon_i\) for each argument of \(f\) (and for each nested invocation of \(D\), even in the univariate case).
Multivariate Power Series as Nested Univariate Power Series

\[ f(i_1, \ldots, i_n)(c_1, \ldots, c_n) \]

\[ i_1! \cdots i_n! \]
The left-branching depth is limited by the number of distinct $\varepsilon_i$. 

The expression for $f(i_1,\ldots,i_n)(c_1,\ldots,c_n)$ is:

$$f(i_1,\ldots,i_n)(c_1,\ldots,c_n) = \frac{i_1! \cdots i_n!}{i_1! \cdots i_n!}$$
Multivariate Power Series as Nested Univariate Power Series

The left-branching depth is limited by the number of distinct $\varepsilon_i$. 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 $\varepsilon_i$.
Only the right branch need be lazy.
The same $(Q \varepsilon p)$ and $(R \varepsilon p)$ API is sufficient.
The left-branching depth is limited by the number of distinct $\varepsilon_i$.

Only the right branch need be lazy.

The same $(Q \varepsilon p)$ and $(R \varepsilon p)$ 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 $\varepsilon_i$.
Only the right branch need be lazy.
The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

---

Pearlmutter & Siskind (Hamilton & Purdue) | Lazy Multivariate Higher-Order Forward AD | POPL/January 2007 | 9/15
The left-branching depth is limited by the number of distinct $\varepsilon_i$.

Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 
Multivariate Power Series as Nested Univariate Power Series

The left-branching depth is limited by the number of distinct $\varepsilon_i$.

Only the right branch need be lazy.

The same $(Q \varepsilon p)$ and $(R \varepsilon p)$ 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 $\varepsilon_i$. 
The left-branching depth is limited by the number of distinct $\varepsilon_i$.

Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) 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 $\varepsilon_i$. 

The left-branching depth is limited by the number of distinct $\varepsilon_i$. Only the right branch need be lazy.

The same ($Q \varepsilon p$) and ($R \varepsilon p$) API is sufficient.
Multivariate Power Series as Nested Univariate Power Series

\[ f(i_1, \ldots, i_n)(c_1, \ldots, c_n) = \frac{i_1! \cdots i_n!}{i_1! \cdots i_n!} \]

The left-branching depth is limited by the number of distinct \( \varepsilon_i \).
Only the right branch need be lazy.
The same \((Q \varepsilon p)\) and \((R \varepsilon p)\) 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 \( \varepsilon_i \).
Cannot do this in a referentially transparent language.
Multivariate Power Series as Nested Univariate Power Series

\[ f(i_1, \ldots, i_n)(c_1, \ldots, c_n) \]

\[ i_1! \cdots i_n! \]

The left-branching depth is limited by the number of distinct \( \varepsilon_i \).

Only the right branch need be lazy.

The same \( (Q \varepsilon p) \) and \( (R \varepsilon p) \) 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 \( \varepsilon_i \).

Cannot do this in a referentially transparent language.

**Painfully ironic:** Cannot *implement* \( D \) in a referentially transparent language even though \( D \) itself *is* referentially transparent!
Unary primitives:

\[ u \ (x + x' \varepsilon) = (u \ x) + \ ((C_{\varepsilon^0} \ (u' \ (x + x' \varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x') \varepsilon \]
Arithmetic on Non-Truncated Power Series

- **Unary primitives:**
  
  \[ u (x + x' \varepsilon) = (u x) + ((C_{\varepsilon^0} (u' (x + x' \varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x') \varepsilon \]

- **Bookkeeping:** \( C_{\varepsilon^0}, [\varepsilon \mapsto \xi], [\xi \mapsto \varepsilon] \)
Arithmetic on Non-Truncated Power Series

- Unary primitives:

  \[ u (x + x' \varepsilon) = (u \, x) + ((C_{\varepsilon^0} \, (u' \, (x + x' \varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x')\varepsilon \]

- Bookkeeping: \( C_{\varepsilon^0}, [\varepsilon \mapsto \xi], [\xi \mapsto \varepsilon] \)

- Requires \( u' \), the derivative of \( u \).
Arithmetic on Non-Truncated Power Series

- **Unary primitives:**

  \[ u \left( x + x' \varepsilon \right) = (u \cdot x) + \left( (C_{\varepsilon}^{0} \left( u' \left( x + x' \varepsilon \right) [\varepsilon \mapsto \xi] \right)) [\xi \mapsto \varepsilon] \times x' \right) \varepsilon \]

- **Bookkeeping:** \( C_{\varepsilon}^{0}, [\varepsilon \mapsto \xi], [\xi \mapsto \varepsilon] \)

- Requires \( u' \), the derivative of \( u \).

- Requires that \( u' \) be (recursively) evaluated under the nonstandard interpretation to compute \( u' \left( x + x' \varepsilon \right) \).
Arithmetic on Non-Truncated Power Series

- **Unary primitives:**
  \[
  u (x + x' \varepsilon) = (u \times x) + (((C_{\varepsilon \rightarrow 0} (u' (x + x' \varepsilon) \rightarrow [\varepsilon \rightarrow \xi]))) \times [\xi \rightarrow \varepsilon] \times x') \varepsilon
  \]

- **Bookkeeping:** \(C_{\varepsilon \rightarrow 0}, [\varepsilon \rightarrow \xi], [\xi \rightarrow \varepsilon]\)
- **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 \(\varepsilon\)s.
Arithmetic on Non-Truncated Power Series

- **Unary primitives:**

\[
u (x + x'\varepsilon) = (u \ x) + \left((C_{\varepsilon^0} \ (u' \ (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \times x'\varepsilon\right)\]

- **Bookkeeping:** \(C_{\varepsilon^0}, [\varepsilon \mapsto \xi], [\xi \mapsto \varepsilon]\)
- **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 \(\varepsilon\)s.
- **Can rename** when arguments are power series over the same \(\varepsilon\):

\[
b \ ((x + x'\varepsilon), (y + y'\varepsilon)) = (b \ ((x + x'\varepsilon), (y + y'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon]\]
Arithmetic on Non-Truncated Power Series

- **Unary primitives:**
  \[
  u (x + x'\varepsilon) = (u x) + \left( (C_{\varepsilon^0} (u' (x + x'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon] \right) \times x'
  \]

- **Bookkeeping:** \( C_{\varepsilon^0}, [\varepsilon \mapsto \xi], [\xi \mapsto \varepsilon] \)
- **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 \( \varepsilon \)s.
- **Can rename** when arguments are power series over the same \( \varepsilon \):
  \[
  b ((x + x'\varepsilon), (y + y'\varepsilon)) = (b ((x + x'\varepsilon), (y + y'\varepsilon)[\varepsilon \mapsto \xi]))[\xi \mapsto \varepsilon]
  \]
- **Read the paper** for the details.
Wrap Up

- Functional programming has had little impact on numerical computing.
Functional programming has had little impact on numerical computing.

Many important numeric concepts are higher-order functions.
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.
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!*
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.
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 \( \mathcal{J} i f \ c \ c' \) of \( f : \mathbb{R}^n \to \mathbb{R}^m \) at \( c : \mathbb{R}^n \) in direction \( c' : \mathbb{R}^n \) by calculating

\[ y = f \left[ c_1 + c'_1 \varepsilon, \ldots, c_n + c'_n \varepsilon \right] \]

and extracting

\[ [y'_1, \ldots, y'_m] = [C_{\varepsilon i} y_1, \ldots, C_{\varepsilon i} y_m] \]
Two alternatives for representing

\[ x(\varepsilon) = x_0 + x_1 \varepsilon + x_2 \varepsilon^2 + x_3 \varepsilon^3 + \cdots \]
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)**
  \[ \langle x(0), \langle x'(0), \langle x''(0), \langle x'''(0), \ldots \rangle \rangle \rangle \rangle = \langle 0! \times x_0, \langle 1! \times x_1, \langle 2! \times x_2, \langle 3! \times x_3, \ldots \rangle \rangle \rangle = \langle x_0, \langle x_1, \langle 2 \times x_2, \langle 6 \times x_3, \ldots \rangle \rangle \rangle \]
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)
  \[ \langle x(0), \langle x'(0), \langle x''(0), \langle x'''(0), \ldots \rangle \rangle \rangle \rangle \]
  \[ = \langle 0! \times x_0, \langle 1! \times x_1, \langle 2! \times x_2, \langle 3! \times x_3, \ldots \rangle \rangle \rangle \rangle \]
  \[ = \langle x_0, \langle x_1, \langle 2 \times x_2, \langle 6 \times x_3, \ldots \rangle \rangle \rangle \rangle \]

- Identical in truncated case

- Fungible: trade off which “left shift” is fast,

\[ Q \varepsilon x(\varepsilon) = \frac{1}{\varepsilon} (x(\varepsilon) - x(0)) \]

or

\[ \frac{d}{d\varepsilon} x(\varepsilon) \]
Implementation of $\mathcal{D}$ vs. Referential Transparency
Implementation of $\mathcal{D}$ vs. Referential Transparency

\[
\mathcal{D} (\lambda x \ldots x \ldots) c \quad \mathcal{D} (\lambda y \ldots y \ldots) c
\]
Implementation of $\mathcal{D}$ vs. Referential Transparency

Observation I

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

Observation II

$\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots \boxed{x} \ldots \boxed{y} \ldots) \ c) \ldots) \ c$
Implementation of $\mathcal{D}$ vs. Referential Transparency

\[ \mathcal{D} (\lambda x \ldots \underline{x} \ldots) \ c \hspace{1cm} \mathcal{D} (\lambda y \ldots \underline{y} \ldots) \ c \]

\[ \mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots \underline{x} \ldots \underline{y} \ldots) \ c) \ldots) \ c \]

\[ \mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots (\underline{x} + \underline{x}) \ldots (\underline{x} + \underline{y}) \ldots) \ c) \ldots) \ c \]
Implementation of $\mathcal{D}$ vs. Referential Transparency

\[
\begin{align*}
\mathcal{D} (\lambda x \ldots [x] \ldots) c & \quad \mathcal{D} (\lambda y \ldots [y] \ldots) c \\
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots [x] \ldots [y] \ldots) c) \ldots) c & \\
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots ([x + x] \ldots ([x + y] \ldots) c) \ldots) c) \ldots) c
\end{align*}
\]

referential transparency $\implies x = y \quad \forall$ cases
Implementation of $\mathcal{D}$ vs. Referential Transparency

\[
\mathcal{D} (\lambda x \ldots [x] \ldots) \ c \quad \mathcal{D} (\lambda y \ldots [y] \ldots) \ c
\]

\[
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots [x] \ldots [y] \ldots) \ c) \ldots) \ c
\]

\[
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots ([x + x] \ldots [x + y] \ldots) \ c) \ldots) \ c)
\]

referential transparency $\implies x = y \quad (\forall \ cases)$

\[
x = y \implies x + x = x + y
\]
Implementation of \( D \) vs. Referential Transparency

Observation I

\[
D (\lambda x \ldots [x] \ldots) \ c \quad D (\lambda y \ldots [y] \ldots) \ c
\]

Observation II

\[
D (\lambda x \ldots (D (\lambda y \ldots [x] \ldots [y] \ldots) \ c) \ldots) \ c
\]

Observation III

\[
D (\lambda x \ldots (D (\lambda y \ldots ([x + x] \ldots [x + y]) \ldots) \ c) \ldots) \ c
\]

Referential transparency \( \implies \ x = y \quad (\forall \text{ cases}) \)

\[
x = y \ \implies \ x + x = x + y
\]

\[
x + x = x + y \ \implies \ \text{getting the wrong answer}
\]
Implementation of $\mathcal{D}$ vs. Referential Transparency

Observation I

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

Observation II

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

Observation III

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

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

referential transparency $\implies x = y$ (\forall cases)

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

$x + x = x + y \implies$ getting the wrong answer

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

\[
\mathcal{D} (\lambda x \ldots x \ldots) c \\
\mathcal{D} (\lambda y \ldots y \ldots) c
\]

\[
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots x \ldots y \ldots) c) \ldots) c
\]

\[
\mathcal{D} (\lambda x \ldots (\mathcal{D} (\lambda y \ldots (x + x) \ldots (x + y) \ldots) c) \ldots) c
\]

referential transparency $\implies x = y$ (\forall cases)

\[
x = y \implies x + x = x + y
\]

\[
x + x = x + y \implies \text{getting the wrong answer}
\]

therefore

\[
\text{getting the right answer} \implies \neg \text{referential transparency}
\]
Implementation of $\mathcal{D}$ vs. Referential Transparency

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

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

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

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

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

$x + x = x + y \implies \text{getting the wrong answer}$

therefore

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

(Oops)