Perturbation Confusion and Referential Transparency

Correct Functional Implementation of Forward-Mode AD

Jeffrey Mark Siskind
School of ECE
Purdue University

Barak A. Pearlmutter
The Hamilton Institute
NUI Maynooth

17th International Workshop on Implementation and Application of Functional Languages

Church Knew *Calculus*

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

Alonzo Church
(1941, *The Calculi of Lambda Conversion*, pg. 1¶4)

Examples of Nesting

$$\min_x \max_y g(x,y)$$

$$\min_x f(x, \max_y g(x,y))$$

$$\mathcal{F}\left\{ \frac{\mathrm{d}}{\mathrm{d}x} f(x,y) \right\}$$

Nested Differentiation

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}x+y\Big|_{y=1}\right)\right)\Big|_{x=1} = \quad ?$$

Nested Differentiation

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}x+y\Big|_{y=1}\right)\right)\Big|_{x=1} = 1$$

Nested Differentiation

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}x+y\Big|_{y=1}\right)\right)\Big|_{x=1} \stackrel{?}{=} 2$$

The Derivative Operator

A First-Class Higher-Order Function

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}x+y\Big|_{y=1}\right)\right)\Big|_{x=1}$$

$$\mathcal{D}\,f\;c=\frac{\mathrm{d}}{\mathrm{d}x}f(x)\Big|_{x=c}$$

The Derivative Operator

A First-Class Higher-Order Function

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(x\left(\frac{\mathrm{d}}{\mathrm{d}y}x+y\Big|_{y=1}\right)\right)\Big|_{x=1}$$

$$\mathcal{D}\,f\;c=\frac{\mathrm{d}}{\mathrm{d}x}f(x)\Big|_{x=c}$$

$$\mathcal{D}\;(\lambda x\;.\;x\times(\mathcal{D}\;(\lambda y\;.\;x+y)\;1))\;1$$

Dual Numbers

$$a + b\mathrm{i} \qquad (\mathrm{i}^2 = -1)$$

$$x + x'\varepsilon \qquad (\varepsilon^2 = 0)$$

(W. K. Clifford, Preliminary sketch of bi-quaternions, *Proc. London Math. Soc.* **4**:381–95, 1873)

Forward-Mode AD

$$\mathcal{E}\ (x + x'\varepsilon) \stackrel{\triangle}{=} x'$$

$$\mathcal{D}\, f\ c \stackrel{\triangle}{=} \mathcal{E}\ (f\ (c + \varepsilon))$$

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

Example of Forward-Mode AD

$$\frac{\mathrm{d}}{\mathrm{d}x} x^2 + x + 1 \Big|_{x=3}$$

Example of Forward-Mode AD

$$\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} = \mathcal{D}\,(\lambda x\,.\, x \times x + x + 1)\; 3$$

Example of Forward-Mode AD

$$\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} = \mathcal{D}\,(\lambda x\,.\, x \times x + x + 1)\; 3$$

Example of Forward-Mode AD

$$\begin{aligned}\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\ (\lambda x\ .\ x \times x + x + 1)\ 3 \\ &= \mathcal{E}\ ((\lambda x\ .\ x \times x + x + 1)\ (3 + \varepsilon))\end{aligned}$$

Example of Forward-Mode AD

$$\begin{aligned}\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\ &= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon))\end{aligned}$$

Example of Forward-Mode AD

$$
\begin{aligned}
\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3{+}\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,((9+6\varepsilon) + (3+\varepsilon) + 1)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,((9+6\varepsilon) + (3+\varepsilon) + 1)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,((9+6\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,(13+7\varepsilon)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1 \Big|_{x=3} &= \mathcal{D}\ (\lambda x\ .\ x \times x + x + 1)\ 3 \\
&= \mathcal{E}\ ((\lambda x\ .\ x \times x + x + 1)\ (3 + \varepsilon)) \\
&= \mathcal{E}\ ((3 + \varepsilon) \times (3 + \varepsilon) + (3 + \varepsilon) + 1) \\
&= \mathcal{E}\ ((9 + 6\varepsilon) + (3 + \varepsilon) + 1) \\
&= \mathcal{E}\ (13 + 7\varepsilon)
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3{+}\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,((9+6\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,(13 + 7\varepsilon) \\
&= 7
\end{aligned}
$$

Example of Forward-Mode AD

$$
\begin{aligned}
\left.\frac{\mathrm{d}}{\mathrm{d}x}\, x^2 + x + 1\right|_{x=3} &= \mathcal{D}\,(\lambda x\,.\,x \times x + x + 1)\;3 \\
&= \mathcal{E}\,((\lambda x\,.\,x \times x + x + 1)\;(3+\varepsilon)) \\
&= \mathcal{E}\,((3+\varepsilon) \times (3+\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,((9+6\varepsilon) + (3+\varepsilon) + 1) \\
&= \mathcal{E}\,(13+7\varepsilon) \\
&= 7
\end{aligned}
$$

Perturbation Confusion

$$\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1$$

Perturbation Confusion

$$\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1$$

Perturbation Confusion

$$\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1$$
$$= \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon))$$

Perturbation Confusion

$$\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1$$
$$= \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon))$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon)) \\
&\qquad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon) + y)\ 1))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon)) \\
&\qquad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1 + \varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1 + \varepsilon) + y)\ (1 + \varepsilon))))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon))))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon))))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon))))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2+2\varepsilon)))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon)))
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2)
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2)
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\qquad = \mathcal{E}\ (2 + 2\varepsilon)
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\qquad = \mathcal{E}\ (2 + 2\varepsilon)
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\quad = \mathcal{E}\ (2 + 2\varepsilon) \\
&\quad = 2
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\qquad = \mathcal{E}\ (2 + 2\varepsilon) \\
&\qquad = 2 \\
&\qquad \neq 1
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\qquad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2+2\varepsilon))) \\
&\qquad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\qquad = \mathcal{E}\ (2+2\varepsilon) \\
&\qquad = 2 \\
&\qquad \neq 1
\end{aligned}
$$

Perturbation Confusion

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1+\varepsilon)) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{D}\ (\lambda y\ .\ (1+\varepsilon) + y)\ 1)) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((\lambda y\ .\ (1+\varepsilon) + y)\ (1+\varepsilon)))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ ((1+\varepsilon) + (1+\varepsilon)))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times (\mathcal{E}\ (2 + 2\varepsilon))) \\
&\quad = \mathcal{E}\ ((1+\varepsilon) \times 2) \\
&\quad = \mathcal{E}\ (2 + 2\varepsilon) \\
&\quad = 2 \\
&\quad \neq 1
\end{aligned}
$$

Referential (non)Transparency

$$c\ x \stackrel{\triangle}{=} \mathcal{D}\ (\lambda y\ .\ x + y)\ 1$$

$$c\ x = 1 \qquad \forall x$$

$$(c\ x)$$

$$1$$

Referential (non)Transparency

$$c\ x \stackrel{\triangle}{=} \mathcal{D}\ (\lambda y\ .\ x + y)\ 1$$

$$c\ x = 1 \qquad \forall x$$

$$\lambda x\ .\ x \times (c\ x)$$

$$\lambda x\ .\ x \times 1$$

Referential (non)Transparency

$$c\ x \stackrel{\triangle}{=} \mathcal{D}\ (\lambda y\ .\ x + y)\ 1$$

$$c\ x = 1 \qquad \forall x$$

$$\mathcal{D}\ (\lambda x\ .\ x \times (c\ x))\ 1$$

$$\mathcal{D}\ (\lambda x\ .\ x \times 1)\ 1$$

Referential (non)Transparency

$$c\ x \stackrel{\triangle}{=} \mathcal{D}\ (\lambda y\ .\ x + y)\ 1$$

$$c\ x = 1 \qquad \forall x$$

$$\mathcal{D}\ (\lambda x\ .\ x \times (c\ x))\ 1 \stackrel{?}{=} 2$$

$$\mathcal{D}\ (\lambda x\ .\ x \times 1)\ 1 = 1$$

Avoiding Perturbation Confusion

$$\mathcal{E}_t \left(x + x'\varepsilon_t\right) \stackrel{\triangle}{=} x'$$

$$\mathcal{D}\, f\, c \stackrel{\triangle}{=} \mathcal{E}_t \left(f \left(c + \varepsilon_t\right)\right)$$

(where t is a "tag" unique to each application of $\mathcal{D}$)

Avoiding Perturbation Confusion: an Example

$$\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1$$

Avoiding Perturbation Confusion: an Example

$$\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a))
\end{aligned}$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1))
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b))))
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b))))
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b)))
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1)
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1) \\
&\quad = \mathcal{E}_a\ (1 + \varepsilon_a)
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1) \\
&\quad = \mathcal{E}_a\ (1 + \varepsilon_a) \\
&\quad = 1
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1) \\
&\quad = \mathcal{E}_a\ (1 + \varepsilon_a) \\
&\quad = 1
\end{aligned}
$$

Avoiding Perturbation Confusion: an Example

$$
\begin{aligned}
&\mathcal{D}\ (\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ 1 \\
&\quad = \mathcal{E}_a\ ((\lambda x\ .\ x \times (\mathcal{D}\ (\lambda y\ .\ x + y)\ 1))\ (1 + \varepsilon_a)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{D}\ (\lambda y\ .\ (1 + \varepsilon_a) + y)\ 1)) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((\lambda y\ .\ (1 + \varepsilon_a) + y)\ (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ ((1 + \varepsilon_a) + (1 + \varepsilon_b)))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times (\mathcal{E}_b\ (2 + \varepsilon_a + \varepsilon_b))) \\
&\quad = \mathcal{E}_a\ ((1 + \varepsilon_a) \times 1) \\
&\quad = \mathcal{E}_a\ (1 + \varepsilon_a) \\
&\quad = 1
\end{aligned}
$$

Scheme Implementation (1/4)

```
1   (define primal cadr)

3   (define tangent caddr)

5   (define tag cadddr)

7   (define (bundle tag primal tangent) (list 'bundle primal tangent tag))

9   (define bundle?
10   (let ((pair? pair?)) (lambda (x) (and (pair? x) (eq? (car x) 'bundle)))))

12  (define pair?
13   (let ((pair? pair?)) (lambda (x) (and (pair? x) (not (bundle? x))))))

15  (define make-tag (let ((tag 0)) (lambda () (set! tag (+ tag 1)) tag)))

17  (define (e-t t x)
18   (if (bundle? x) (if (= (tag x) t) (tangent x) (e-t t (primal x))) 0))
```

Scheme Implementation (2/4)

```
20  (define (remove-tag t x)
21   (if (bundle? x)
22       (if (= (tag x) t)
23           (primal x)
24           (bundle (tag x) (remove-tag t (primal x)) (remove-tag t (tangent x))))
25       x))

27  (define (bring-tag-to-top t x) (bundle t (remove-tag t x) (e-t t x)))

29  (define (lift-real t x) (bundle t x 0))

31  (define (in? t x) (and (bundle? x) (or (= (tag x) t) (in? t (primal x)))))

33  (define (lift-real->real f df/dx)
34   (letrec ((self (lambda (x)
35                    (if (bundle? x)
36                        (bundle (tag x)
37                                (self (primal x))
38                                (* (df/dx (primal x)) (tangent x)))
39                        (f x)))))
40    self))

42  (define (lift-real*real->real f df/dx1 df/dx2)
43   (letrec ((self
44             (lambda (x1 x2)
45              (if (bundle? x1)
46                  (if (bundle? x2)
47                      (if (= (tag x1) (tag x2))
48                          (bundle
```

Scheme Implementation (3/4)

```
49                            (tag x1)
50                            (self (primal x1) (primal x2))
51                            (+ (* (df/dx1 (primal x1) (primal x2)) (tangent x1))
52                               (* (df/dx2 (primal x1) (primal x2)) (tangent x2))))
53                           (cond ((in? (tag x1) x2)
54                                  (self x1 (bring-tag-to-top (tag x1) x2)))
55                                 ((in? (tag x2) x1)
56                                  (self (bring-tag-to-top (tag x2) x1) x2))
57                                 (else (self x1 (lift-real (tag x1) x2)))))
58                       (self x1 (lift-real (tag x1) x2)))
59                   (if (bundle? x2)
60                       (self (lift-real (tag x2) x1) x2)
61                       (f x1 x2))))))
62     self))

64   (define (lift-real->boolean f)
65    (letrec ((self (lambda (x) (if (bundle? x) (self (primal x)) (f x)))))
66     self))

68   (define (lift-real*real->boolean f)
69    (letrec ((self (lambda (x1 x2)
70                     (if (bundle? x1)
71                         (if (bundle? x2)
72                             (self (primal x1) (primal x2))
73                             (self (primal x1) x2))
74                         (if (bundle? x2) (self x1 (primal x2)) (f x1 x2))))))
75     self))
```

Scheme Implementation (4/4)

```
77  (define + (lift-real*real->real + (lambda (x1 x2) 1) (lambda (x1 x2) 1)))

79  (define - (lift-real*real->real - (lambda (x1 x2) 1) (lambda (x1 x2) -1)))

81  (define * (lift-real*real->real * (lambda (x1 x2) x2) (lambda (x1 x2) x1)))

83  (define /
84   (lift-real*real->real
85    / (lambda (x1 x2) (/ 1 x2)) (lambda (x1 x2) (- 0 (/ x1 (* x2 x2))))))

87  (define sqrt (lift-real->real sqrt (lambda (x) (/ 1 (* 2 (sqrt x))))))

89  (define exp (lift-real->real exp (lambda (x) (exp x))))

91  (define log (lift-real->real log (lambda (x) (/ 1 x))))

93  (define sin (lift-real->real sin (lambda (x) (cos x))))

95  (define cos (lift-real->real cos (lambda (x) (- 0 (sin x)))))

97  (define atan (lift-real*real->real
98                atan
99                (lambda (x1 x2) (/ (- 0 x2) (+ (* x1 x1) (* x2 x2))))
100               (lambda (x1 x2) (/ x1 (+ (* x1 x1) (* x2 x2))))))

102 (define = (lift-real*real->boolean =))

104 (define < (lift-real*real->boolean <))

106 (define > (lift-real*real->boolean >))

108 (define <= (lift-real*real->boolean <=))
```

Example: Saddle Point $\min_x \max_y f(x,y)$

```
1  (define (abs x) (if (negative? x) (- 0 x) x))

3  (define (argmin f)
4   (let loop ((x 0))
5    (let ((x-prime (- x (/ ((derivative f) x)
6                           ((derivative (derivative f)) x)))))
7     (if (<= (abs (- x x-prime)) 1e-5) x (loop x-prime)))))

9  (define (argmax f) (argmin (lambda (x) (- 0 (f x)))))

11 (define (saddle-point f)
12  (let ((x* (argmin
13             (lambda (x) (f x (argmax (lambda (y) (f x y))))))))
14   (list x* (argmax (lambda (y) (f x* y))))))

16 (define (sqr x) (* x x))

18 (define (payoff x y) (- (sqr (- y 4)) (sqr (- x 3))))

20 (begin (newline) (write (saddle-point payoff)) (newline))
```

Functional Programming *vs* Referential Transparency

Problem!

Cannot “gensym” tags in HASKELL

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> x + y) 1
15 should_be_one_a = d (\x -> x * (constant_one x)) 1
16 should_be_one_b = d (\x -> x * 1                ) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> x + y) 1
15 should_be_one_a = d (\x -> x * (constant_one x)) 1
16 should_be_one_b = d (\x -> x * 1                ) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> x + y) 1
15 should_be_one_a = d (\x -> x * (lift (constant_one x))) 1
16 should_be_one_b = d (\x -> x * (lift 1               )) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> (lift x) + y) 1
15 should_be_one_a = d (\x -> x * (constant_one x)) 1
16 should_be_one_b = d (\x -> x * 1                ) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> (lift x) + y) 1
15 should_be_one_a = d (\x -> x * (constant_one x)) 1
16 should_be_one_b = d (\x -> x * 1                ) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

Referential non-Transparency in HASKELL

```
1  data Num a => Bundle a = Bundle a a
2  instance Num a => Show (Bundle a) where
3     showsPrec p (Bundle x x') = showsPrec p [x,x']
4  instance Num a => Eq (Bundle a) where
5     (Bundle x x') == (Bundle y y') = (x == y)
6  lift z = Bundle z 0
7  instance Num a => Num (Bundle a) where
8     (Bundle x x')+(Bundle y y') = Bundle(x + y)(x' + y')
9     (Bundle x x')*(Bundle y y') = Bundle(x * y)(x * y' + x' * y)
10    fromInteger z = lift (fromInteger z)
11 instance Fractional a => Fractional (Bundle a) where
12    fromRational z = lift (fromRational z)
13 d f x = let (Bundle y y') = f (Bundle x 1) in y'
14 constant_one x = d (\y -> (lift x) + y) 1
15 should_be_one_a = d (\x -> x * (constant_one x)) 1
16 should_be_one_b = d (\x -> x * 1                ) 1
17 violation_of_referential_transparency =
18    should_be_one_a /= should_be_one_b
```

A Static Alternative

- Insert the right number of "`lift`"s around every use of a numeric variable.
- The right number is: the number of (dynamic) invocations of $\mathcal{D}$ that intervene between the binding of the variable and the use.
- If this number cannot be determined or is not unique, the code cannot be transformed.
- Aggregate data is also problematic.
- A suitably restrictive coding style can ensure that the "right number"s are apparent.
- Other static transformation methods are possible.

Context: Forward-Mode AD and Reverse-Mode AD

Transformation of $x \mapsto (f\ x)$ where $f : \mathbb{R}^n \to \mathbb{R}^m$

Forward-mode AD:

$$(\mathbb{R}^n \to \mathbb{R}^m) \to ((\mathbb{R}^n \times \mathbb{R}^n) \to (\mathbb{R}^m \times \mathbb{R}^m))$$

$$(x, \acute{x}) \mapsto ((f\ x), (\mathcal{J}\ f\ x) \times \acute{x})$$

Reverse-mode AD:

$$(\mathbb{R}^n \to \mathbb{R}^m) \to (\mathbb{R}^n \to (\mathbb{R}^m \times (\mathbb{R}^m \to \mathbb{R}^n)))$$

$$x \mapsto ((f\ x), (\lambda \grave{y}\ .\ (\mathcal{J}\ f\ x)^T \times \grave{y}))$$

Related Work

HASKELL	(Karczmarczuk, 1998, 1999, 2001; Nilsson, 2003)
SCHEME:	SCMUTILS (Sussman et al., 2001)
FORTRAN:	ADIFOR (Bischof et al., 1996)
C:	ADIC (Bischof et al., 1997)
MATLAB:	ADIMAT (Bischof et al., 2003)
MAPLE:	GRADIENT (Monagan & Neuenschwander, 1993)

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: Voice for FP + AD

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: high-performance SCHEME + AD

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: STALIN + AD

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: STALIN + gradients

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: STALIN $+\ \nabla$

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: STALIN∇

Future Work

Lambda: the ultimate calculus

Part of larger project to combine λ-calculus and AD.

- include both forward-mode and reverse-mode operators
- get numerically correct answers
- via correct transformation (time/space complexity)
- make $\overrightarrow{\mathcal{J}}$ and $\overleftarrow{\mathcal{J}}$ first class
- make them apply to all (differentiable) programs
- handle nested application (self-application of $\mathcal{J}$)
- good puns
  - mathematical: the $\lambda\nabla$-calculus
  - programming language: VLAD
  - implementation: STALIN∇
- manuscripts and code:
  http://www-bcl.cs.nuim.ie/~qobi/stalingrad/