Higher-Order, Higher-Order Automatic Differentiation

(early, incomplete draft—comments invited[*]*)*

Conal Elliott
conal@conal.net

Draft of January 23, 2020

1 Introduction

This note picks up where Elliott (2018) left off, and I assume the reader to be familiar with that paper and have it close at hand. I am circulating this follow-on in fairly rough form for early feedback, to then evolve in to a full research paper. The main new contributions are two senses of "*higher-order* automatic differentiation":

- derivatives of higher-order functions, and
- higher-order derivatives of functions, and

The former has been addressed in a recent paper (Vytiniotis et al., 2019), but in a way I find dissatisfying for a variety of reasons described in Section 9 and discussed at length with the authors.

Begin with the category of computably differentiable functions from Elliott (2018, Section 4.1):

$$\textbf{newtype}\ D\ a\ b = D\ (a \to b \times (a \multimap b))$$

where $a \multimap b$ is the type of linear maps from a to b. The function around which the automatic differentiation (AD) algorithm is organized simply "zips" together a function $f : a \to b$ and its derivative $\mathcal{D}\,f : a \to a \multimap b$:[1,2,3]

$$\begin{aligned}
&\hat{\mathcal{D}} : (a \to b) \to D\ a\ b \\
&\hat{\mathcal{D}}\,f = D\ (\lambda\, a \to (f\ a, \mathcal{D}\,f\ a)) \\
&\phantom{\hat{\mathcal{D}}\,f} = D\ (f \vartriangle \mathcal{D}\,f)
\end{aligned}$$

[*]For the latest version of this document and the repository location for questions, suggestions, bugs, etc, see `http://conal.net/papers/higher-order-ad`.

[1][Red bracketed remarks are notes to myself.]

[2]This paper generally uses Haskell notation but deviates slightly by using a single colon rather than double colon for type signatures. [Experimental.]

[3]The infix operators for function types ("$\to$") and linear maps ("$\multimap$") both associate to the right and have equal, very low precedence. For instance, "$a \to a \multimap b$" means $a \to (a \multimap b)$.

Note that this definition is not computable, since $\mathcal{D}$ is not (Pour-El and Richards, 1978, 1983). The whole specification of AD is then simply that $\hat{\mathcal{D}}$ is a homomorphism with respect to a standard compositional vocabulary of functions, namely that of cartesian categories, plus a collection of numeric primitives like (uncurried) addition and multiplication, *sin* and *cos*, etc. An example of such a homomorphism equation is $\hat{\mathcal{D}}\ g \circ \hat{\mathcal{D}}\ f = \hat{\mathcal{D}}\ (g \circ f)$, in which the only unknown is the meaning of the LHS ($\circ$), i.e., sequential composition in the category D. Solving the collection of such homomorphism equations yields correct-by-construction AD.

The function $\hat{\mathcal{D}}$ is invertible, i.e., $\hat{\mathcal{D}}^{-1} \circ \hat{\mathcal{D}} = id$, where $\hat{\mathcal{D}}^{-1}$ simply drops the derivative:[4]

$$\begin{aligned}
&\hat{\mathcal{D}}^{-1} : D\ a\ b \to (a \to b)\\
&\hat{\mathcal{D}}^{-1}\ (D\ h) = exl \circ h
\end{aligned}$$

Indeed, $\hat{\mathcal{D}}^{-1}$ is a left inverse of $\hat{\mathcal{D}}$:

$$\begin{aligned}
&\ \hat{\mathcal{D}}^{-1}\ (\hat{\mathcal{D}}\ f)\\
&= \hat{\mathcal{D}}^{-1}\ (D\ (f \vartriangle \mathcal{D}\ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition}\\
&= exl \circ (f \vartriangle \mathcal{D}\ f) && \text{-- } \hat{\mathcal{D}}^{-1} \text{ definition}\\
&= f && \text{-- cartesian law: } exl \circ (g \vartriangle h) = g
\end{aligned}$$

As defined so far, $\hat{\mathcal{D}}^{-1}$ is *not* a right inverse to $\hat{\mathcal{D}}$, since the linear map portion might not be the true derivative. We will thus *restrict* the category D to be the image of $\hat{\mathcal{D}}$, which is to say that $\hat{\mathcal{D}}$ is surjective, i.e., the derivative is correct.[5] This restriction guarantees that $\hat{\mathcal{D}}^{-1}$ is indeed a right inverse of $\hat{\mathcal{D}}$. Given $\hat{f} : D\ a\ b$ (with the mentioned restriction), there is an $f : a \to b$ such that $\hat{f} = \hat{\mathcal{D}}\ f$, so[6]

$$\begin{aligned}
&\ \hat{\mathcal{D}}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})\\
&= \hat{\mathcal{D}}\ (\hat{\mathcal{D}}^{-1}\ (\hat{\mathcal{D}}\ f)) && \text{-- } \hat{f} = \hat{\mathcal{D}}\ f\\
&= \hat{\mathcal{D}}\ f && \text{-- } \hat{\mathcal{D}}^{-1} \circ \hat{\mathcal{D}} = id\\
&= \hat{f} && \text{-- } \hat{\mathcal{D}}\ f = \hat{f}
\end{aligned}$$

Thus, $\hat{\mathcal{D}} \circ \hat{\mathcal{D}}^{-1} = id$ as well.

AD is often described as coming in forward and reverse "modes". For many practical applications (including deep learning and other high-dimensional optimization problems), reverse mode is much more efficient than forward mode. As typically presented, reverse mode is also much more complicated, but this difference appears to be due only to unfortunate choices in how to understand and implement AD. Instead, a single, simple algorithm works for forward, reverse, and other modes. Reverse mode is distinguished only by using a different linear map representation resulting from a simple classic trick (Elliott, 2018).

This general AD algorithm is justified by three main theorems about differentiation:

Theorem 1 (compose/"chain" rule)**.**

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

[4] This paper uses "*exl*" and "*exr*" to name left and right product projections (defined on cartesian categories), rather than Haskell's (function-only) "*fst*" and "*snd*".

[5] Haskell's type system is not expressive enough to capture this restriction by itself, so the restriction will be only implied in this draft. For more rigor, one could use a language (extension) with refinement types such as Liquid Haskell [ref] or a dependently-typed language such as Agda [ref] or Idris [ref].

[6] This reasoning hold for *any* surjective function with a left inverse.

Theorem 2 (cross rule)**.**

$$\mathcal{D}\ (f \times g)\ (a, b) = \mathcal{D}\ f\ a \times \mathcal{D}\ g\ b$$

Theorem 3 (linear rule)**.** *For all linear functions f,*

$$\mathcal{D}\ f\ a = f$$

In addition to these three theorems, we need a collection of facts about the derivatives of various mathematical operations, e.g., $\hat{\mathcal{D}}\ \mathit{sin}\ x = \mathit{scale}\ (\mathit{cos}\ x)$, where $\mathit{scale} : a \to a \multimap a$ is uncurried scalar multiplication (so $\mathit{scale}\ s$ is linear for all s).

2 Some Additional Properties of Differentiation

A few additional properties of differentiation will prove useful in extending Elliott (2018) to higher-order functions and higher-order derivatives.

2.1 Pair-Valued Domains

One half of the *curry*/*uncurry* isomorphism involves functions of pair-valued domains. The notion of partial derivatives is helpful for differentiating such functions.[7]

Lemma 4 (proved in Section A.1)**.** *Given a function $f : a \times b \to c$,*

$$\mathcal{D}\ f\ (a, b) = \mathcal{D}_l\ f\ (a, b) \triangledown \mathcal{D}_r\ f\ (a, b)$$

where $\mathcal{D}_l$ and $\mathcal{D}_r$ construct the ("first" and "second", or "left" and "right") "partial derivatives" of f at (a, b), defined as follows:

$$\begin{array}{l}\mathcal{D}_l : (a \times b \to c) \to a \times b \to (a \multimap c)\\ \mathcal{D}_l\ f\ (a, b) = \mathcal{D}\ (f \circ (, b))\ a\end{array}$$

$$\begin{array}{l}\mathcal{D}_r : (a \times b \to c) \to a \times b \to (b \multimap c)\\ \mathcal{D}_r\ f\ (a, b) = \mathcal{D}\ (f \circ (a,))\ b\end{array}$$

The notation "$(a,)$" and "$(b,)$" refers to right and left "sections" of pairing: $(, b)\ a = (a,)\ b = (a, b)$. Equivalently,

$$\begin{array}{l}\mathcal{D}_l\ f\ (a, b) = \mathcal{D}\ f\ (a, b) \circ \mathit{inl}\\ \mathcal{D}_r\ f\ (a, b) = \mathcal{D}\ f\ (a, b) \circ \mathit{inr}\end{array}$$

Note also that $f \circ (a,) = \mathit{curry}\ f\ a$ and $f \circ (, b) = \mathit{curry}'\ f\ b$, where

$$\begin{array}{l}\mathit{curry}\ \ f\ a\ b = f\ (a, b)\\ \mathit{curry}'\ f\ b\ a = f\ (a, b)\end{array}$$

As an example of how this decomposition of $\mathcal{D}\ f$ helps construct derivatives, suppose that f is *bilinear*, which is to say that f is linear in each argument, while holding the other constant. More formally *bilinearity* of f means that $f \circ (a,)$ and $f \circ (b,)$ (equivalently, $\mathit{curry}\ f\ a$ and $\mathit{curry}'\ f\ b$) are both linear for all a and b.

[7]Recall that, on linear maps, $(f \triangledown g)\ (a, b) = f\ a + g\ b$, $\mathit{inl}\ a = (a, 0)$, and $\mathit{inr}\ b = (0, b)$

Corollary 4.1. *If $f : a \times b \to c$ is bilinear then*

$$\mathcal{D}\ f\ (a, b) = f \circ (, b) \triangledown f \circ (a,)$$

Proof.

$$\begin{aligned}
&\mathcal{D}\ f\ (a, b) \\
&= \mathcal{D}_l\ f\ (a, b) \triangledown \mathcal{D}_r\ f\ (a, b) && \text{-- Lemma 4} \\
&= \mathcal{D}\ (f \circ (, b))\ a \triangledown \mathcal{D}\ (f \circ (a,))\ b && \text{-- } \mathcal{D}_l \text{ and } \mathcal{D}_r \text{ definitions} \\
&= f \circ (, b) \triangledown f \circ (a,) && \text{-- linearity}
\end{aligned}$$

□

For instance, the derivative of uncurried multiplication is given by the Leibniz product rule:

$$\begin{aligned}
&\mathcal{D}\ (uncurry\ (*))\ (a, b) \\
&= uncurry\ (*) \circ (, b) \triangledown uncurry\ (*) \circ (a,) \\
&= (*\ b) \triangledown (a\ *) \\
&= \lambda\, (da, db) \to da * b + a * db
\end{aligned}$$

More generally, consider differentiating interacts with uncurrying:

Corollary 4.2 (proved in Section A.2)**.**

$$\mathcal{D}\ (uncurry\ g)\ (a, b) = at\ b \circ \mathcal{D}\ g\ a \triangledown \mathcal{D}\ (g\ a)\ b$$

As a special case, let g be curried multiplication:

$$\begin{aligned}
&\mathcal{D}\ (uncurry\ (*)) \\
&= at\ b \circ \mathcal{D}\ (*)\ a \triangledown \mathcal{D}\ (a*)\ b \\
&= at\ b \circ (*) \triangledown (a*) \\
&= (*\ b) \triangledown (a\ *)
\end{aligned}$$

which agrees with the calculation above.

For cartesian closure, we'll need the derivative of another function with a pair-valued domain:

$$\begin{aligned}
&eval : (a \to b) \times a \to b \\
&eval\ (f, a) = f\ a \quad \text{-- on functions}
\end{aligned}$$

(Since *eval* is neither linear nor bilinear, Theorem 3 and Corollary 4.1 are inapplicable.) We'll need one more linear map operation, which is curried, reverse function application:[8]

$$\begin{aligned}
&at : a \to (a \to b) \multimap b \\
&at\ a\ df = df\ a
\end{aligned}$$

Corollary 4.3 (proved in Section A.3)**.**

$$\mathcal{D}\ eval\ (f, a) = at\ a \triangledown \mathcal{D}\ f\ a$$

[8] Linearity of *at a* follows from the usual definition of addition and scaling on functions.

2.2 Function-Valued Codomains

It will also be useful to calculate derivatives of functions with higher-order codomains.[9] We'll need another linear map operation, which is the indexed variant of $(\triangle)$ (and a specialization of Haskell's *flip* function):

$$\begin{aligned}&\mathit{fork}_F : (b \to a \multimap c) \to (a \multimap b \to c)\\&\mathit{fork}_F\ h = \lambda\, da\ b \to h\ b\ da\end{aligned}$$

Lemma 5 (proved in Section A.4)**.** *Given a function $g : a \to b \to c$,*

$$\mathcal{D}\ g\ a = \mathit{fork}_F\ (\lambda\, b \to \mathcal{D}\ (\mathit{at}\ b \circ g)\ a).$$

Corollary 5.1 (proved in Section A.5)**.**

$$\mathcal{D}\ (\mathit{curry}\ f)\ a = \mathit{fork}_F\ (\mathcal{D}_l\ f \circ (a,))$$

3 Cartesian Closure, first attempt

While D is a category and a *cartesian* category at that, as specified by $\hat{\mathcal{D}}$ being a cartesian functor, another question naturally arises. Can $\hat{\mathcal{D}}$ also be a cartesian *closed* functor? In other words, are there definitions of *eval*, *curry*, and *uncurry* on D such that

$$\begin{aligned}&\mathit{curry}\ (\hat{\mathcal{D}}\ f) = \hat{\mathcal{D}}\ (\mathit{curry}\ f)\\&\mathit{uncurry}\ (\hat{\mathcal{D}}\ g) = \hat{\mathcal{D}}\ (\mathit{uncurry}\ g)\\&\mathit{eval} = \hat{\mathcal{D}}\ \mathit{eval}\end{aligned}$$

As usual, we'll want to solve each homomorphism equation for its single unknown, which is a categorical operation on D (on the LHS).

3.1 Curry

Start with *curry*, simplifying the LHS:

$$\begin{aligned}&\ \ \mathit{curry}\ (\hat{\mathcal{D}}\ f)\\&= \mathit{curry}\ (D\ (f \triangle \mathcal{D}\ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition}\end{aligned}$$

Then the RHS:

$$\begin{aligned}&\ \ \hat{\mathcal{D}}\ (\mathit{curry}\ f)\\&= D\ (\mathit{curry}\ f \triangle \mathcal{D}\ (\mathit{curry}\ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition}\\&= D\ (\lambda\, a \to (\mathit{curry}\ f\ a, \mathcal{D}\ (\mathit{curry}\ f)\ a)) && \text{-- } (\triangle) \text{ on functions}\\&= D\ (\lambda\, a \to ((\lambda\, b \to f\ (a,b)), \mathit{fork}_F\ (\mathcal{D}_l\ f \circ (a,)))) && \text{-- Corollary 5.1}\\&= D\ (\lambda\, a \to ((\lambda\, b \to f\ (a,b)), \mathit{fork}_F\ (\lambda\, b \to \mathcal{D}_l\ f\ (a,b)))) && \text{-- } (\circ) \text{ on functions}\\&= D\ (\lambda\, a \to ((\lambda\, b \to f\ (a,b)), \mathit{fork}_F\ (\lambda\, b \to \mathcal{D}\ f\ (a,b) \circ \mathit{inl}))) && \text{-- Section A.1}\end{aligned}$$

[9] [The previous section and this one provide "adjoint" techniques in a sense that currying is an adjunction from functions from products to functions to functions. Is there something else interesting to say here?]

The last form uses f and $\mathcal{D}\ f$, which can be extracted from $\hat{\mathcal{D}}\ f = D\ (f \vartriangle \mathcal{D}\ f)$: Thus a sufficient condition for our homomorphic specification ($\mathit{curry}\ (\hat{\mathcal{D}}\ f) = \hat{\mathcal{D}}\ (\mathit{curry}\ f)$) is

$$\begin{array}{l} \mathit{curry}\ (D\ \mathit{ff}') = D\ (\lambda\ a \to ((\lambda\ b \to f\ (a,b)), \mathit{fork}_F\ (\lambda\ b \to f'\ (a,b) \circ \mathit{inl}))) \\ \quad \mathbf{where}\ (f, f') = \mathit{fork}^{-1}\ \mathit{ff}' \end{array}$$

The fork^{-1} function is half of an isomorphism that holds for all cartesian categories:

$$\begin{array}{l} \mathit{fork} : \mathit{Cartesian}\ k \Rightarrow (a\ \text{`}k\text{`}\ c) \times (a\ \text{`}k\text{`}\ d) \to (a\ \text{`}k\text{`}\ (c \times d)) \\ \mathit{fork} = \mathit{uncurry}\ (\vartriangle) \end{array}$$

$$\begin{array}{l} \mathit{fork}^{-1} : \mathit{Cartesian}\ k \Rightarrow (a\ \text{`}k\text{`}\ (c \times d)) \to (a\ \text{`}k\text{`}\ c) \times (a\ \text{`}k\text{`}\ d) \\ \mathit{fork}^{-1}\ h = (\mathit{exl} \circ h, \mathit{exr} \circ h) \end{array}$$

Lemma 6. *The pair of functions fork and fork^{-1} form an isomorphism in all cartesian categories and a linear isomorphism in the category of vector spaces and linear maps.*[10] (Proof: Exercise.)

Another such linear isomorphism can be found in cocartesian categories. The following types are specialized to biproduct categories (such as linear maps):

$$\begin{array}{l} \mathit{join} : \mathit{Cocartesian}\ k \Rightarrow (a\ \text{`}k\text{`}\ c) \times (a\ \text{`}k\text{`}\ d) \to (a\ \text{`}k\text{`}\ (c \times d)) \\ \mathit{join} = \mathit{uncurry}\ (\triangledown) \end{array}$$

$$\begin{array}{l} \mathit{join}^{-1} : \mathit{Cocartesian}\ k \Rightarrow (a\ \text{`}k\text{`}\ (c \times d)) \to (a\ \text{`}k\text{`}\ c) \times (a\ \text{`}k\text{`}\ d) \\ \mathit{join}^{-1}\ h = (h \circ \mathit{inl}, h \circ \mathit{inr}) \end{array}$$

Lemma 7. *The pair of functions join and join^{-1} form an isomorphism in all cocartesian categories and a linear isomorphism in the category of vector spaces and linear maps.* (Proof: Exercise.)

These two isomorphism pairs were used by Elliott (2018) to construct a correct-by-construction implementation of reverse-mode AD, by merely altering the representation of linear maps used in the simple, general AD algorithm.

Another useful operation is the *uncurried* version of the monoidal $(\times)$:

$$\begin{array}{l} \mathit{cross} : \mathit{Monoidal}\ k \Rightarrow (a\ \text{`}k\text{`}\ c) \times (b\ \text{`}k\text{`}\ d) \to ((a \times b)\ \text{`}k\text{`}\ (c \times d)) \\ \mathit{cross} = \mathit{uncurry}\ (\times) \end{array}$$

Lemma 8. *In the category of vector spaces and linear maps, the cross function is linear.* (Proof: Exercise.)

Although *fork* and fork^{-1} form an isomorphism and hence preserve information, fork^{-1} can result in a loss of efficiency, due to computation that can be (and often is) in common to a function f and its derivative $\mathcal{D}\ f$. Indeed, the definition of $\mathit{fork}^{-1}\ h$ above shows that h gets replicated. It's unclear how to avoid this redundancy problem in practice with currying when D is used to represent computably differentiable functions. Personal experience with compiling to

[10] [To do: name this category early (say "$\mathbf{Vec}_s$" for a semiring s) and refer to by name where needed.]

categories Elliott (2017) suggests that most uses of *curry* generated during translation from the λ calculus (e.g., Haskell) are in fact transformed away at compile time using various equational CCC laws. Still, it does seem an important question to explore.

Intriguingly, curried functions can also help eliminate redundant computation suggested by uncurried counterparts functions. Given a function $g : a \to b \to c$, it is sometimes convenient to "partially apply" g to an argument u and then apply the resulting $g\ u : b \to c$ to many different $v : b$. In some cases, a considerable amount of work can be done based solely on u, saving residual work to be done for different b values. In such situations, *uncurry* g loses this performance advantage.

3.2 Uncurry

Next, let's tackle *uncurry*, whose defining homomorphism is

$$\mathit{uncurry}\ (\hat{\mathcal{D}}\ g) = \hat{\mathcal{D}}\ (\mathit{uncurry}\ g)$$

Simplify the LHS:

$$\begin{aligned} &\mathit{uncurry}\ (\hat{\mathcal{D}}\ g) \\ =\ &\mathit{uncurry}\ (D\ (g \vartriangle \mathcal{D}\ g)) && \text{-- } \hat{\mathcal{D}} \text{ definition} \end{aligned}$$

Then the RHS:

$$\begin{aligned} &\hat{\mathcal{D}}\ (\mathit{uncurry}\ g) \\ =\ &D\ (\mathit{uncurry}\ g \vartriangle \mathcal{D}\ (\mathit{uncurry}\ g)) && \text{-- } \hat{\mathcal{D}} \text{ definition} \\ =\ &D\ (\lambda\,(a,b) \to (\mathit{uncurry}\ g\ (a,b), \mathcal{D}\ (\mathit{uncurry}\ g)\ (a,b))) && \text{-- } (\vartriangle) \text{ definition} \\ =\ &D\ (\lambda\,(a,b) \to (g\ a\ b, \mathcal{D}\ (\mathit{uncurry}\ g)\ (a,b))) && \text{-- } \mathit{uncurry} \text{ on functions} \\ =\ &D\ (\lambda\,(a,b) \to (g\ a\ b, \mathit{at}\ b \circ \mathcal{D}\ g\ a \triangledown \mathcal{D}\ (g\ a)\ b)) && \text{-- Corollary 4.2} \end{aligned}$$

Now we have a problem with solving the defining homomorphism above. Although we can extract g and $\mathcal{D}\ g$ from $\hat{\mathcal{D}}\ g$, we cannot extract $\mathcal{D}\ (g\ a)$. Or rather we can, but not computably.

3.3 Eval

We don't need to work out both *uncurry* and *eval*, since each can be defined in terms of the other:

$$\begin{aligned} \mathit{eval} &= \mathit{uncurry}\ \mathit{id} \\ \mathit{uncurry}\ g &= \mathit{eval} \circ (g \times \mathit{id}) \\ &= \mathit{eval} \circ \mathit{first}\ g \end{aligned}$$

Since we got stuck on *uncurry*, however, let's try *eval* as well to see if we learn anything new.

The corresponding homomorphism equation has a particularly simple form:

$$\mathit{eval} = \hat{\mathcal{D}}\ \mathit{eval}$$

It might appear that we have succeeded at the start, taking the equation to be a definition for *eval*. Recall, however, that $\hat{\mathcal{D}}$ is noncomputable, being defined via $\mathcal{D}$ (differentiation itself). Simplifying the RHS,

$$\begin{aligned}
&\hat{\mathcal{D}}\ \mathit{eval} \\
&= D\ (\mathit{eval} \vartriangle \mathcal{D}\ \mathit{eval}) && \text{-- } \hat{\mathcal{D}} \text{ definition} \\
&= D\ (\lambda\,(f,a) \to (\mathit{eval}\ (f,a), \mathcal{D}\ \mathit{eval}\ (f,a))) && \text{-- } (\vartriangle) \text{ on functions} \\
&= D\ (\lambda\,(f,a) \to (f\ a, \mathcal{D}\ \mathit{eval}\ (f,a))) && \text{-- } \mathit{eval} \text{ on functions} \\
&= D\ (\lambda\,(f,a) \to (f\ a, \mathit{at}\ a \triangledown \mathcal{D}\ f\ a)) && \text{-- Corollary 4.3}
\end{aligned}$$

As with uncurrying (Section 3.2), the final form is well-defined but is not a computable recipe, leaving us in a pickle. Next, let's look for some wiggle room.

4 Object Mapping

The choice of category-associated products and exponentials is a degree of freedom not exercised in the development of AD in Elliott (2018) (or above) and is tied closely to another such choice available in the general notion of *cartesian closed functor* in category theory. In general, a functor has two aspects:

- a mapping from arrows to arrows, and
- a mapping from objects to objects.

The functor $\hat{\mathcal{D}}$ defined (noncomputably) above implicitly chooses an *identity object mapping*, as evident in its type signature $\hat{\mathcal{D}} : (a \to b) \to D\ a\ b$. The type of $\hat{\mathcal{D}}$ plus the requirement that it be a cartesian *closed* functor implies that the object mapping aspect of $\hat{\mathcal{D}}$ is the identity. More generally, however, we can define an object mapping $O : \mathit{Type} \to \mathit{Type}$ for a new functor $\mathcal{D}_o$:[11]

$$\mathcal{D}_o : (a \to b) \to D\ (O\ a)\ (O\ b)$$

Each cartesian category k has its own notion of categorical product $a \times_k b$ (satisfying a universality property), and similarly for cocartesian categories (with categorical products and coproducts coinciding for biproduct categories). Likewise, each cartesian *closed* category k has its own notion of *exponential* objects $a \Rightarrow_k b$.

The generalized interface for cartesian closed categories with per-category exponentials is as follows:[12]

$$\begin{aligned}
&\textbf{class}\ \mathit{Cartesian}\ k \Rightarrow \mathit{CartesianClosed}\ k\ \textbf{where} \\
&\quad \textbf{type}\ (\Rightarrow_k) : \mathit{Type} \to \mathit{Type} \to \mathit{Type} \\
&\quad \mathit{curry} \quad\ \ : ((a \times_k b)\ \text{`}k\text{`}\ c) \to (a\ \text{`}k\text{`}\ (b \Rightarrow_k c)) \\
&\quad \mathit{uncurry} : (a\ \text{`}k\text{`}\ (b \Rightarrow_k c)) \to ((a \times_k b)\ \text{`}k\text{`}\ c) \\
&\quad \mathit{eval} \quad\ \ \ : ((a \Rightarrow_k b) \times_k a)\ \text{`}k\text{`}\ b
\end{aligned}$$

where $a \Rightarrow_k b$ is a type of "exponential objects" (first class functions/arrows) from a to b for the category k.

The property of being a closed cartesian functor requires O to preserve categorical products and exponentials, i.e.,

[11] [Experiment with different notation for $O\ a$, e.g., "$\bar{a}$".]

[12] These operations support higher-order programming and arise during translation from a typed lambda calculus (e.g., Haskell) to categorical vocabulary (Elliott, 2017).

$$\begin{aligned}
&O\ (a\ \times\ b) = O\ a \times_D\ O\ b\\
&O\ (a \to b) = O\ a \Rightarrow_D O\ b
\end{aligned}$$

The usual notion of cartesian products are working fine, so we'll continue to choose $a \times_D b = a \times b$. While $\hat{\mathcal{D}}$ being a closed cartesian functor (CCF) from $(\to)$ to D implies an noncomputable *eval* and *uncurry* (Section 3.2 and Section 3.3), our goal is to define $\Rightarrow_D$ and $\mathcal{D}_o$ such that $\mathcal{D}_o$ is a CCF with computable operations.

Consider again the homomorphic specification for *eval* (part of the CCF definition): $\mathit{eval} = \mathcal{D}_o\ \mathit{eval}$. The RHS *eval* (on functions) has type $(a \to b) \times a \to b$, while the LHS *eval* (on D) has type

$$\begin{aligned}
&\ \ D\ (O\ ((a \to b) \times a))\ (O\ b)\\
&= D\ (O\ (a \to b) \times O\ a)\ (O\ b)\\
&= D\ ((O\ a \Rightarrow_D O\ b) \times O\ a)\ (O\ b)
\end{aligned}$$

The difficulty with our attempt at *eval* in Section 3.3 was that we were given a (computable) function f, but we also needed its (noncomputable) derivative $\mathcal{D}\ f$. Similarly, with *uncurry* in Section 3.2, we were given $g : a \to b \to c$, and we needed not only $g\ a$ but also $\mathcal{D}\ (g\ a)$. In both cases the exponential object was a function, but we also needed its (computable) derivative.

This analysis suggests that we include a derivative in the exponential object, simply by choosing $\Rightarrow_D$ to be D itself. Additionally, map scalars to themselves and cartesian products to cartesian products:

$$\begin{aligned}
&O\ \mathbb{R} = \mathbb{R}\\
&O\ (a\ \times\ b) = O\ a \times_D\ O\ b = O\ a \times O\ b\\
&O\ (a \to b) = O\ a \Rightarrow_D O\ b = D\ (O\ a)\ (O\ b)
\end{aligned}$$

We will need to convert between a and $O\ a$, which we can do with a family of *linear isomorphisms*[13] indexed by a:[14]

```
class HasO t where
  type O t
  o    : t → O t
  o⁻¹ : O t → t
```

For scalar types a and the unit type, $O\ a = a$, the isomorphism is trivial:

```
instance HasO ℝ where
  type O ℝ = ℝ
  o    = id
  o⁻¹ = id

instance HasO () where
  type O () = ()
```

[13] The implicit requirements for all *HasO* instances are thus that $o \circ o^{-1} = id$, $o^{-1} \circ o = id$, and *to* and o^{-1} are linear.

[14] [It may be more elegant to combine the functions o and o^{-1} into a single *isomorphism*.]

$$\begin{aligned} o &= id \\ o^{-1} &= id \end{aligned}$$

For products, convert components independently:[15]

$$\begin{array}{l} \textbf{instance}\ (HasO\ a, HasO\ b) \Rightarrow HasO\ (a \times b)\ \textbf{where} \\ \quad \textbf{type}\ O\ (a \times b) = O\ a \times O\ b \\ \quad o \quad\ = o \quad\ \times o \\ \quad o^{-1} = o^{-1} \times o^{-1} \end{array}$$

The new functor $\mathcal{D}_o$ converts its given $a \to b$ to $O\ a \to O\ b$ and then applies the $\hat{\mathcal{D}}$ functor:[16]

$$\begin{array}{l} (\Rightarrow) : (p' \to p) \to (q \to q') \to ((p \to q) \to (p' \to q')) \\ f \Rightarrow h = \lambda\, g \to h \circ g \circ f \end{array}$$

$$\begin{array}{l} wrap_o : (a \to b) \to (O\ a \to O\ b) \\ wrap_o = o^{-1} \Rightarrow o \end{array}$$

$$\begin{array}{l} wrap_o^{-1} : (O\ a \to O\ b) \to (a \to b) \\ wrap_o^{-1} = o \Rightarrow o^{-1} \end{array}$$

$$\begin{array}{l} \mathcal{D}_o : (a \to b) \to D\ (O\ a)\ (O\ b) \\ \mathcal{D}_o = \hat{\mathcal{D}} \circ wrap_o \end{array}$$

$$\begin{array}{l} \mathcal{D}_o^{-1} : D\ (O\ a)\ (O\ b) \to (a \to b) \\ \mathcal{D}_o^{-1} = wrap_o^{-1} \circ \hat{\mathcal{D}}^{-1} \end{array}$$

Lemma 9 (proved in Section A.6)**.** *$wrap_o$ and $wrap_o^{-1}$ form a linear isomorphism.*

Lemma 10 (proved in Section A.7)**.** *$\mathcal{D}_o$ and $\mathcal{D}_o^{-1}$ form a linear isomorphism.*

Lemma 11 (proved in Section A.8)**.** *$wrap_o$ is a cartesian functor.*

The cartesian category operations already defined on D (Elliott, 2018) are solutions to homomorphism equations saying that $\hat{\mathcal{D}}$ is a cartesian functor. Thanks to the simple, regular structure of o and o^{-1},

Theorem 12. *$\mathcal{D}_o$ is a cartesian functor.*

Proof: $\hat{\mathcal{D}}$ is a cartesian functor (Elliott, 2018), as is $wrap_o$ (Lemma 11), so $\mathcal{D}_o = \hat{\mathcal{D}} \circ wrap_o$ is also.

What about exponentials and cartesian *closure*? As mentioned above, $O\ (a \to b) = O\ a \Rightarrow_D O\ b = D\ (O\ a)\ (O\ b)$, which suggests using $\mathcal{D}_o$ and $\mathcal{D}_o^{-1}$ for o and o^{-1}:

$$\begin{array}{l} \textbf{instance}\ (HasO\ a, HasO\ b) \Rightarrow HasO\ (a \to b)\ \textbf{where} \\ \quad \textbf{type}\ O\ (a \to b) = D\ (O\ a)\ (O\ b) \\ \quad o \quad\ = \mathcal{D}_o \\ \quad o^{-1} = \mathcal{D}_o^{-1} \end{array}$$

A useful consequence:

[15] Recall that $(f \times g)\ (a, b) = (f\ a, g\ b)$, so $o\ (a, b) = (o \times o)\ (a, b) = (o\ a, o\ b)$, and similarly for o^{-1}.

[16] [Consider dropping the $(\Rightarrow)$ definition and uses here.]

Lemma 13 (proved in Section A.9)**.**

$$wrap_o\ (curry\ f) = \hat{\mathcal{D}} \circ curry\ (wrap_o\ f)$$

Corollary 13.1.

$$curry\ (wrap_o\ f) = \hat{\mathcal{D}}^{-1} \circ wrap_o\ (curry\ f)$$

Proof. Left-compose $\hat{\mathcal{D}}^{-1}$ with both sides of Lemma 13; then simplify and reverse the resulting equation. □

Let's now try to solve the CCF equations for $\mathcal{D}_o$. This time begin with *eval*:

Lemma 14 (proved in Section A.10)**.** *With the following (effective) definition of eval on D, $eval = \mathcal{D}_o\ eval$:*

$$eval = D\ (\lambda\,(D\ h, a) \rightarrow \mathbf{let}\ (b, f') = h\ a\ \mathbf{in}\ (b, at\ a \circ \hat{\mathcal{D}}^{-1} \triangledown f'))$$

For *uncurry*, use the standard definition $uncurry\ g = eval \circ first\ g$.

The definition of *curry* in Section 3.1 worked fine, but we'll need to check again, as we did with the cartesian category operations (Theorem 12). The homomorphism equation is $curry\ (\mathcal{D}_o\ f) = \mathcal{D}_o\ (curry\ f)$, to be solved for the unknown LHS *curry* (on D), with $f : a \times b \rightarrow c$. First let $f_o = wrap_o\ f$. Simplify the LHS:

$$\begin{aligned}
&\ curry\ (\mathcal{D}_o\ f) \\
=&\ curry\ (\hat{\mathcal{D}}\ (wrap_o\ f)) && \text{-- } \mathcal{D}_o \text{ definition} \\
=&\ curry\ (\hat{\mathcal{D}}\ f_o) && \text{-- } f_o \text{ definition} \\
=&\ curry\ (D\ (f_o \vartriangle \mathcal{D}\ f_o)) && \text{-- } \hat{\mathcal{D}} \text{ definition}
\end{aligned}$$

Then the RHS:[17]

Lemma 15 (proved in Section A.11)**.**

$$\begin{aligned}
&\mathcal{D}_o\ (curry\ f) = \\
&\quad D\ (\lambda\ a \rightarrow (D\ (\lambda\ b \rightarrow (f_o\ (a, b), \mathcal{D}_r\ f_o\ (a, b))) \\
&\qquad\qquad ,\lambda\ da \rightarrow D\ (\lambda\ b \rightarrow (\mathcal{D}_l\ f_o\ (a, b)\ da, at\ da \circ \mathcal{D}_r\ (\mathcal{D}_l\ f_o)\ (a, b)))))
\end{aligned}$$

where $f_o = wrap_o\, f$.

The RHS uses $f_o\ (a, b)$ and $\mathcal{D}\ f_o\ (a, b)$ (via its components $\mathcal{D}_l\ f_o\ (a, b)$ and $\mathcal{D}_r\ f_o\ (a, b)$), but it also uses a *second* partial derivative $\mathcal{D}_r\ (\mathcal{D}_l\ f_o)\ (a, b)$, which is not available from the *curry* argument $D\ (f_o \vartriangle \mathcal{D}\ f_o)$.

[17] [State, prove, and use a lemma about $\hat{\mathcal{D}}\ (g \circ f)\ a$ for linear g and another for linear f. Maybe also $\mathcal{D}_o\ (g \circ f)\ a$ for linear g or f.]

5 Where Are We?

Let's now reflect on what we've learned so far:

- The cartesian functor (CF) $\hat{\mathcal{D}} : (a \to b) \to D\ a\ b$ also forms a cartesian *closed* functor (CCF) with suitable definitions of *curry*, *uncurry*, and *eval*, but not computably (Section 3). More specifically, *curry* is computable, but *uncurry* and *eval* are not, since they need to synthesize derivatives of regular computable functions.

- General categorical functors can remap objects (here, types) as well as morphisms (here, functions). Exploiting this degree of freedom, define $\mathcal{D}_o : (a \to b) \to D\ (O\ a)\ (O\ b)$, where $O : \mathit{Type} \to \mathit{Type}$ replaces regular functions with computably differentiable functions, i.e., $O\ (u \to v) = D\ (O\ u)\ (O\ v)$. This new function is defined in terms of the old one, $\mathcal{D}_o = \hat{\mathcal{D}} \circ \mathit{wrap}_o$, and indeed $\mathcal{D}_o$ is a CF as well. In the absence of higher-order functions, O is the identity mapping, and $\mathcal{D}_o$ coincides with $\hat{\mathcal{D}}$.

- Computably satisfying the required homomorphism properties of $\mathcal{D}_o$ for *uncurry* and *eval* becomes easy, since the operations are *given* the required derivatives rather than having to synthesize them. Unfortunately, now *curry* becomes noncomputable because it has to synthesize partial *second* derivatives.

6 Higher-Order Derivatives

Where can we go from here? An obvious next step is to add second order derivatives to the representation of computably differentiable functions. It seem likely, however, that the CCF specification would reveal that *curry* needs at least third order derivatives, and so on. In other words, differentiation of higher-order functions requires all higher-order derivatives of functions.

In order to construct higher-order derivatives, it will help to examine the linearity properties of our familiar categorical vocabulary, which turns out to be mostly linear with just a bit of bilinearity. As noted in Elliott (2018), the categorical operation *id*; the cartesian operations *exl*, *exr*, *dup*; and the cocartesian operations *inl*, *inr*, and *jam* are all linear. Lemmas 6 and 7 have already noted that the functions *fork* and *join* (uncurried versions of $(\vartriangle)$ and $(\triangledown)$ defined in Section 3.1) are linear (as well as isomorphisms). Next, let *comp* be uncurried composition:[18]

$$\begin{aligned} &\mathit{comp} : \mathit{Category}\ k \Rightarrow (b\ `k`\ c) \times (a\ `k`\ b) \to (a\ `k`\ c) \\ &\mathit{comp} = \mathit{uncurry}\ (\circ) \end{aligned}$$

Lemma 16 (proved in Section A.12)**.** *On linear maps, comp is bilinear.*

Lemma 17 (proved in Section A.13)**.** *Given any bilinear function h:*

a. curry h a is linear for all a.

b. curry′ h b is linear for all b.

c. curry h and curry′ h are linear.

[18] [Maybe define *comp* only for linear maps.]

d. $\mathcal{D}\ h$ *is linear.*

Corollary 17.1. *On linear maps,*

a. $(g\ \circ)$ *is linear for all* g.

b. $(\circ\ f)$ *is linear for all* f.

c. $(\circ)$ *and* $\mathit{flip}\ (\circ)$ *are linear.*

d. $\mathcal{D}\ \mathit{comp}$ *is linear.*

These properties will help re-express Theorems 1 and 2 and related facts in a form more amenable to constructing higher derivatives:

Lemma 18 (proved in Section A.14)**.**

a. $\mathcal{D}\ (g \circ f) = \mathit{comp} \circ (\mathcal{D}\ g \circ f \vartriangle \mathcal{D}\ f)$.

b. $\mathcal{D}\ (f \times g) = \mathit{cross} \circ (\mathcal{D}\ f \times \mathcal{D}\ g)$.

c. $\mathcal{D}\ (f \vartriangle g) = \mathit{fork} \circ (\mathcal{D}\ f \vartriangle \mathcal{D}\ g)$.

d. For a linear *function* f, $\mathcal{D}\ f = \mathit{const}\ f$.

e. For any function $f : a \times b \to c$, $\mathcal{D}\ f = \mathit{join} \circ (\mathcal{D}_l\ f \times \mathcal{D}_r\ f)$.

f. For a bilinear *function* $f : a \times b \to c$, $\mathcal{D}\ f = \mathit{join} \circ (\mathit{curry}'\ f \times \mathit{curry}\ f) \circ \mathit{swap}$.

g. On linear maps, $\mathcal{D}\ \mathit{comp} = \mathit{join} \circ (\mathit{flip}\ (\circ) \times (\circ)) \circ \mathit{swap}$.

Let us now consider the task of constructing *all* orders of derivatives. The D category encapsulates a function f and its first derivative, i.e., the zeroth and first derivatives of f, which we might write as "$\hat{\mathcal{D}}\ f = \mathcal{D}^0\ f \vartriangle \mathcal{D}^1\ f$". Our new category will encapsulate *all* derivatives of f, i.e.,[19]

$$\mathcal{D}^*\ f = \mathcal{D}^0\ f \vartriangle \mathcal{D}^1\ f \vartriangle \mathcal{D}^2\ f \vartriangle \cdots$$

where

$$\begin{aligned}\mathcal{D}^0\ \ \ \ f &= f \\ \mathcal{D}^{n+1}\ f &= \mathcal{D}^n\ (\mathcal{D}\ f)\end{aligned}$$

Then

$$\begin{aligned}&\mathcal{D}^*\ f \\ =\ &\mathcal{D}^0\ f \vartriangle \mathcal{D}^1\ f \vartriangle \mathcal{D}^2\ f \vartriangle \mathcal{D}^3\ f \vartriangle \cdots \\ =\ &f \vartriangle \mathcal{D}^1\ f \vartriangle \mathcal{D}^2\ f \vartriangle \mathcal{D}^3\ f \vartriangle \cdots \\ =\ &f \vartriangle \mathcal{D}^0\ (\mathcal{D}\ f) \vartriangle \mathcal{D}^1\ (\mathcal{D}\ f) \vartriangle \mathcal{D}^2\ (\mathcal{D}\ f) \vartriangle \cdots \\ =\ &f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ f)\end{aligned}$$

[19]Take $\vartriangle$ to be *right*-associative.

which we can take as a recursive definition of $\mathcal{D}^*$. Define a corresponding type of infinitely differentiable functions:[20]

$$\textbf{type } D^*\ a\ b = a \to T\ a\ b$$

$$\textbf{type } T\ a\ b = b \times T\ a\ (a \multimap b)$$

$$\begin{array}{l}\mathcal{D}^* : (a \to b) \to D^*\ a\ b \\ \mathcal{D}^*\ f = f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ f)\end{array}$$

We will want to find cartesian category operations for D^* such that $\mathcal{D}^*$ is a cartesian functor (CF), which will be coinductively assumed at several points below.

Start with the constant-zero function[21]: $\mathit{zero} : a \to b$:

$$\begin{array}{ll} \quad \mathcal{D}^*\ \mathit{zero} & \\ = \mathit{zero} \vartriangle \mathcal{D}^*\ (\mathcal{D}\ \mathit{zero}) & \text{-- } \mathcal{D}^* \text{ definition} \\ = \mathit{zero} \vartriangle \mathcal{D}^*\ \mathit{zero} & \text{-- } \mathcal{D}\ \mathit{zero} = \mathit{const}\ \mathit{zero} = \mathit{zero} \\ = \mathit{zero} \vartriangle \mathit{zero} & \text{-- coinduction} \\ = \mathit{zero} & \text{-- Zero on pairs} \end{array}$$

Then constant functions more generally:

$$\begin{array}{ll} \quad \mathcal{D}^*\ (\mathit{const}\ b) & \\ = \mathit{const}\ b \vartriangle \mathcal{D}^*\ (\mathcal{D}\ (\mathit{const}\ b)) & \text{-- } \mathcal{D}^* \text{ definition} \\ = \mathit{const}\ b \vartriangle \mathcal{D}^*\ \mathit{zero} & \text{-- } \mathcal{D}\ (\mathit{const}\ b) = \mathit{zero} \\ = \mathit{const}\ b \vartriangle \mathit{zero} & \text{-- above} \end{array}$$

Next, linear functions f:

$$\begin{array}{ll} \quad \mathcal{D}^*\ f & \\ = f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ f) & \text{-- } \mathcal{D}^* \text{ definition} \\ = f \vartriangle \mathcal{D}^*\ (\mathit{const}\ f) & \text{-- } f \text{ linearity} \\ = f \vartriangle \mathit{const}\ f \vartriangle \mathit{zero} & \text{-- above} \end{array}$$

We will have several uses of this formula, so name it:

$$\begin{array}{l}\mathit{linear} : (a \multimap b) \to D^*\ a\ b \\ \mathit{linear}\ f = f \vartriangle \mathit{const}\ f \vartriangle \mathit{zero}\end{array}$$

For instance, the following definitions of *id*, *exl* and *exr* satisfy the associated homomorphism (cartesian functor) properties:

$$\begin{array}{ll} \mathit{id} & = \mathit{linear}\ \mathit{id} \\ \mathit{exl} & = \mathit{linear}\ \mathit{exl} \\ \mathit{exr} & = \mathit{linear}\ \mathit{exr} \end{array}$$

Next, *bilinear* functions g:

[20] For notational simplicity, we'll drop the **newtype** isomorphisms.

[21] As usual, types are restricted to vector spaces over a common field, which we can take to be $\mathbb{R}$

$$\begin{aligned}
&\mathcal{D}^*\ g \\
&= g \vartriangle \mathcal{D}^*\ (\mathcal{D}\ g) && \text{-- } \mathcal{D}^* \text{ definition} \\
&= g \vartriangle \mathit{linear}\ (\mathcal{D}\ g) && \text{-- derivative of bilinear is linear} \\
&= g \vartriangle \mathit{linear}\ (\mathit{join} \circ (\mathit{curry}'\ g \times \mathit{curry}\ g) \circ \mathit{swap}) && \text{-- Lemma 18f}
\end{aligned}$$

Specialize to uncurried linear map composition:

$$\begin{aligned}
&\mathcal{D}^*\ \mathit{comp} \\
&= \mathit{comp} \vartriangle \mathit{linear}\ (\mathit{join} \circ (\mathit{curry}'\ \mathit{comp} \times \mathit{curry}\ \mathit{comp}) \circ \mathit{swap}) && \text{-- above} \\
&= \mathit{comp} \vartriangle \mathit{linear}\ (\mathit{join} \circ (\mathit{flip}\ (\circ) \times (\circ)) \circ \mathit{swap}) && \text{-- } \mathit{comp} \text{ definition}
\end{aligned}$$

Name $\mathcal{D}^*\ \mathit{comp}$ for future use:

$$\begin{aligned}
&\mathit{comp}' : D^*\ ((b \multimap c) \times (a \multimap b))\ (a \multimap c) \\
&\mathit{comp}' = \mathit{comp} \vartriangle \mathit{linear}\ (\mathit{join} \circ (\mathit{flip}\ (\circ) \times (\circ)) \circ \mathit{swap})
\end{aligned}$$

Then sequential compositions:

$$\begin{aligned}
&\mathcal{D}^*\ (g \circ f) \\
&= g \circ f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ (g \circ f)) && \text{-- } \mathcal{D}^* \text{ definition} \\
&= g \circ f \vartriangle \mathcal{D}^*\ (\mathit{comp} \circ (\mathcal{D}\ g \circ f \vartriangle \mathcal{D}\ f)) && \text{-- Lemma 18a} \\
&= g \circ f \vartriangle \mathcal{D}^*\ \mathit{comp} \circ (\mathcal{D}^*\ (\mathcal{D}\ g) \circ \mathcal{D}^*\ f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ f)) && \text{-- coinduction} \\
&= g \circ f \vartriangle \mathit{comp}' \circ (\mathcal{D}^*\ (\mathcal{D}\ g) \circ \mathcal{D}^*\ f \vartriangle \mathcal{D}^*\ (\mathcal{D}\ f)) && \text{-- above}
\end{aligned}$$

Note that all of the components here (g, f, $\mathcal{D}^*\ (\mathcal{D}\ g)$, $\mathcal{D}^*\ f$, and $\mathcal{D}^*\ (\mathcal{D}\ f)$) are available in $\mathcal{D}^*\ g$ and $\mathcal{D}^*\ f$, so we have a computable recipe for $(\circ)$ on D^*. [To do: fill in the details.]

Finally, $f \vartriangle g$:

$$\begin{aligned}
&\mathcal{D}^*\ (f \vartriangle g) \\
&= (f \vartriangle g) \vartriangle \mathcal{D}^*\ (\mathcal{D}\ (f \vartriangle g)) && \text{-- } \mathcal{D}^* \text{ definition} \\
&= (f \vartriangle g) \vartriangle \mathcal{D}^*\ (\mathit{fork} \circ (\mathcal{D}\ f \vartriangle \mathcal{D}\ g)) && \text{-- Lemma 18c} \\
&= (f \vartriangle g) \vartriangle \mathcal{D}^*\ \mathit{fork} \circ (\mathcal{D}^*\ (\mathcal{D}\ f) \vartriangle \mathcal{D}^*\ (\mathcal{D}\ g)) && \text{-- coinduction} \\
&= (f \vartriangle g) \vartriangle \mathit{linear}\ \mathit{fork} \circ (\mathcal{D}^*\ (\mathcal{D}\ f) \vartriangle \mathcal{D}^*\ (\mathcal{D}\ g)) && \text{-- } \mathit{fork} \text{ linearity (Lemma 6)}
\end{aligned}$$

Again, the components here (f, g, $\mathcal{D}^*\ (\mathcal{D}\ f)$, and $\mathcal{D}^*\ (\mathcal{D}\ g)$) are all available from $\mathcal{D}^*\ f$ and $\mathcal{D}^*\ g$, so we have a computable recipe for $(\vartriangle)$ on D^*. [To do: fill in the details.]

Working here

7 Avoiding redundant computation

The $\hat{\mathcal{D}}$ functor was carefully chosen to enable elimination of redundant computation between a function and its derivative. The potential for redundancy is apparent in the chain rule (Theorem 1):

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

This theorem reveals that computation of $(g \circ f)\ a$ and $\mathcal{D}\ (g \circ f)\ a$ at both involve computing $f\ a$. Since sequential composition is a very commonly used building block of computations, it is thus typical for functions and their derivatives to involve common work. This fact motivates the choice $\hat{\mathcal{D}}\ f = f \vartriangle \mathcal{D}\ f$ over $\mathcal{D}_0^{+}\ f = (f, \mathcal{D}\ f)$ (Elliott, 2018, Section 3.1). While both options can give rise to compositional (functorial) AD, $\mathcal{D}_0^{+}$ precludes sharing of work, while $\hat{\mathcal{D}}$ enables such sharing, with just a bit of care:

$$D\ \hat{g} \circ D\ \hat{f} = D\ (\lambda\, a \to \mathbf{let}\ \{(b, f') = \hat{f}\ a; (c, g') = \hat{g}\ b\}\ \mathbf{in}\ (c, g' \circ f'))$$

We can calculate this definition in a categorical/pointfree form using Lemma 18a:[22,23]

$$\begin{aligned}
&\hat{\mathcal{D}}\ (g \circ f) \\
&= D\ (g \circ f \vartriangle \mathcal{D}\ (g \circ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition} \\
&= D\ (g \circ f \vartriangle \mathit{comp} \circ (\mathcal{D}\ g \circ f \vartriangle \mathcal{D}\ f)) && \text{-- Lemma 18a} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ (g \circ f \vartriangle (\mathcal{D}\ g \circ f \vartriangle \mathcal{D}\ f))) && \text{-- Lemma 19c below} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ ((g \circ f \vartriangle \mathcal{D}\ g \circ f) \vartriangle \mathcal{D}\ f)) && \text{-- [justify this step]} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ ((g \vartriangle \mathcal{D}\ g) \circ f \vartriangle \mathcal{D}\ f)) && \text{-- Gibbons (2002, Section 1.5.1).} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ (\mathit{unD}\ (\hat{\mathcal{D}}\ g) \circ f \vartriangle \mathcal{D}\ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ \mathit{first}\ (\mathit{unD}\ (\hat{\mathcal{D}}\ g)) \circ (f \vartriangle \mathcal{D}\ f)) && \text{-- Lemma 19b below} \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ \mathit{first}\ (\mathit{unD}\ (\hat{\mathcal{D}}\ g)) \circ \mathit{unD}\ (\hat{\mathcal{D}}\ f)) && \text{-- } \hat{\mathcal{D}} \text{ definition}
\end{aligned}$$

We can thus define

$$\begin{aligned}
D\ \hat{g} \circ D\ \hat{f} &= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ \mathit{first}\ (\mathit{unD}\ (D\ \hat{g})) \circ \mathit{unD}\ (D\ \hat{f})) \\
&= D\ (\mathit{second}\ \mathit{comp} \circ \mathit{assoc}_R \circ \mathit{first}\ \hat{g} \circ \hat{f})
\end{aligned}$$

with the consequence that $\hat{\mathcal{D}}\ g \circ \hat{\mathcal{D}}\ f = \hat{\mathcal{D}}\ (g \circ f)$. In this form, $\hat{f}$ and $\hat{g}$ each appear once, so as long as $D\ \hat{f}$ and $D\ \hat{g}$ are nonredundant, $D\ \hat{g} \circ D\ \hat{f}$ will be nonredundant as well. Inlining the definitions of *comp* and of *second*, assoc_R, and *first* for functions and then simplifying yields the pointful definition above.

Lemma 19. *The following properties hold for* $(\vartriangle)$*:*

a. $(h \times k) \circ (f \vartriangle g) = h \circ f \vartriangle k \circ g$

b. $\mathit{first}\ h \circ (f \vartriangle g) = h \circ f \vartriangle g$

c. $\mathit{second}\ k \circ (f \vartriangle g) = f \vartriangle k \circ g$

Proof: For a, see Gibbons (2002, Section 1.5.1). Then b and c follow as corollaries from the definitions $\mathit{first}\ h = h \times \mathit{id}$ and $\mathit{second}\ k = \mathit{id} \times k$.

Working here

[22] [Define and use a variant of $\hat{\mathcal{D}}$ that omits D. Then introduce D in "We can thus define ...".]

[23] The assoc_R operation in monoidal categories is defined for functions as $\mathit{assoc}_R\ ((a, b), c) = (a, (b, c))$.

The calculation of $\hat{\mathcal{D}}\ (g \circ f)$ above is somewhat tedious, and it's unclear how to extend it to higher derivatives. Some of the complexity comes from routing $\mathcal{D}\ f$ (i.e., $\mathit{exr} \circ \hat{\mathcal{D}}\ f$) around $\hat{\mathcal{D}}\ g$ to compose derivatives $(\mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a)$. The motivation for this routing arises from an asymmetry in D, namely that it maps just a primal value to a primal *and* derivative:

$$\begin{aligned}&\hat{\mathcal{D}}' : (a \to b) \to (a \to b \times (a \multimap b))\\&\hat{\mathcal{D}}'\ f = f \vartriangle \mathcal{D}\ f\end{aligned}$$

Suppose instead that we thread the primal/derivative pairs *in* as well as out.[24,25]

$$\begin{aligned}&\tilde{\mathcal{D}}' : (a \to b) \to \forall z.\, a \times (z \multimap a) \to b \times (z \multimap b)\\&\forall f\ q. \tilde{\mathcal{D}}'\ f \circ \hat{\mathcal{D}}'\ q = \hat{\mathcal{D}}'\ (f \circ q) \quad \text{-- specification}\end{aligned}$$

Moreover, $\tilde{\mathcal{D}}'$ suffices to compute $\hat{\mathcal{D}}'$:

$$\begin{aligned}&\ \ \hat{\mathcal{D}}'\ f\\&= \hat{\mathcal{D}}'\ (f \circ \mathit{id})\\&= \tilde{\mathcal{D}}'\ f \circ \hat{\mathcal{D}}'\ \mathit{id}\\&= \tilde{\mathcal{D}}'\ f \circ (\mathit{id} \vartriangle \mathcal{D}\ \mathit{id})\\&= \tilde{\mathcal{D}}'\ f \circ (\lambda\, z \to (z, \mathit{id}))\end{aligned}$$

Now consider sequential composition:

$$\begin{aligned}&\ \ \tilde{\mathcal{D}}'\ (g \circ f) \circ \hat{\mathcal{D}}'\ q\\&= \hat{\mathcal{D}}'\ ((g \circ f) \circ q) && \text{-- specification of } \tilde{\mathcal{D}}'\\&= \hat{\mathcal{D}}'\ (g \circ (f \circ q)) && \text{-- associativity of } (\circ)\\&= \tilde{\mathcal{D}}'\ g \circ \hat{\mathcal{D}}'\ (f \circ q) && \text{-- specification of } \tilde{\mathcal{D}}'\\&= \tilde{\mathcal{D}}'\ g \circ (\tilde{\mathcal{D}}'\ f \circ \hat{\mathcal{D}}'\ q) && \text{-- specification of } \tilde{\mathcal{D}}'\\&= (\tilde{\mathcal{D}}'\ g \circ \tilde{\mathcal{D}}'\ f) \circ \hat{\mathcal{D}}'\ q && \text{-- associativity of } (\circ)\end{aligned}$$

Hence $\tilde{\mathcal{D}}'\ (g \circ f) = \tilde{\mathcal{D}}'\ g \circ \tilde{\mathcal{D}}'\ f$ by the following lemma.[26]

Lemma 20. *For any $f : a \to b \to c$, if $\mathit{uncurry}\ f$ is surjective and $\forall x : a.\ g \circ f\ x = g' \circ f\ x$, then $g = g'$.*

Proof.

$$\begin{aligned}&\ \ g \circ \mathit{uncurry}\, f\\&= \lambda\,(x, y) \to g\ (\mathit{uncurry}\ f\ (x, y)) && \text{-- } \eta \text{ conversion}\\&= \lambda\,(x, y) \to g\ (f\ x\ y) && \text{-- } \mathit{uncurry} \text{ on functions}\end{aligned}$$

[24][To do: find a (non-effective) *definition* for $\tilde{\mathcal{D}}'$. Well, I know how to define $\tilde{\mathcal{D}}'$ in this case:

$$\tilde{\mathcal{D}}'\ f\ (a, q') = (b, f' \circ q')\ \textbf{where}\ (b, f') = \hat{\mathcal{D}}'\ f\ a$$

How to extend to higher derivatives?]

[25]This formulation is similar to the use of dual numbers in forward-mode AD [ref].

[26][Move lemma and proof to the appendix.]

$$\begin{array}{ll}
= \mathit{uncurry}\ (\lambda\, x\ y \rightarrow g\ (f\ \ x\ \ y)) & \text{-- } \mathit{uncurry} \text{ on functions} \\
= \mathit{uncurry}\ (\lambda\, x\ y \rightarrow (g \circ f\ \ x)\ \ y) & \text{-- } (\circ) \text{ on functions} \\
= \mathit{uncurry}\ (\lambda\, x \rightarrow g \circ f\ \ x) & \text{-- } \eta \text{ conversion} \\
= \mathit{uncurry}\ (\lambda\, x \rightarrow g' \circ f\ \ x) & \text{-- assumption} \\
= \mathit{uncurry}\ (\lambda\, x\ y \rightarrow (g' \circ f\ \ x)\ \ y) & \text{-- } \eta \text{ conversion} \\
= \mathit{uncurry}\ (\lambda\, x\ y \rightarrow g'\ (f\ \ x\ \ y)) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\,(x, y) \rightarrow g'\ (f\ \ x\ \ y) & \text{-- } \mathit{uncurry} \text{ on functions} \\
= \lambda\,(x, y) \rightarrow g'\ (\mathit{uncurry}\ f\ \ (x, y)) & \text{-- } \mathit{uncurry} \text{ on functions} \\
= g' \circ \mathit{uncurry}\ f & \text{-- } \eta \text{ conversion}
\end{array}$$

Since *uncurry* f is surjective, $g = g'$. □

Likewise, consider *id*:

$$\begin{array}{l}
\quad \tilde{\mathcal{D}}'\ \mathit{id} \circ \hat{\mathcal{D}}'\ q \\
= \hat{\mathcal{D}}'\ (\mathit{id} \circ q) \\
= \hat{\mathcal{D}}'\ q \\
= \mathit{id} \circ \hat{\mathcal{D}}'\ q
\end{array}$$

By Lemma 20, $\tilde{\mathcal{D}}'\ \mathit{id} = \mathit{id}$.

$$\boxed{\text{Working here}}$$

Parallel composition:

$$\begin{array}{ll}
\quad \tilde{\mathcal{D}}'\ (f \vartriangle g) \circ \hat{\mathcal{D}}'\ q & \\
= \hat{\mathcal{D}}'\ ((f \vartriangle g) \circ q) & \text{-- } \tilde{\mathcal{D}}' \text{ specification} \\
= \hat{\mathcal{D}}'\ (f \circ q \vartriangle g \circ q) & \text{-- Cartesian law} \\
= \hat{\mathcal{D}}'\ (f \circ q) \blacktriangle \hat{\mathcal{D}}'\ (g \circ q) & \text{-- [for suitable } (\blacktriangle)\text{, probably as with } (\vartriangle) \text{ for } D] \\
= \tilde{\mathcal{D}}'\ f \circ \hat{\mathcal{D}}'\ q \blacktriangle \tilde{\mathcal{D}}'\ g \circ \hat{\mathcal{D}}'\ q & \text{-- } \tilde{\mathcal{D}}' \text{ specification} \\
= (\tilde{\mathcal{D}}'\ f \blacktriangle \tilde{\mathcal{D}}'\ g) \circ \hat{\mathcal{D}}'\ q & \text{-- [To prove about } (\blacktriangle)]
\end{array}$$

8 What's Next?

[Yet to come:

- Avoid redundant computation in D^*. Doing so is fairly easy in D (zeroth and first derivatives), but I don't yet see how in D^* (all derivatives).
- Spell out the *Category* and *Cartesian* instances that result from solving the cartesian functor equations as in Section 6.
- Cartesian *closure* (*curry* and *eval*/*uncurry*) for D^*, exploiting higher-order derivatives.
- Variation of $\mathcal{D}^*$: $\mathcal{D}_*\ f = f \vartriangle \mathcal{D}\ (\mathcal{D}_*\ f)$.

]

9 Related Work

The most closely related work I'm aware of is by Vytiniotis et al. (2019)[27], who also define an algorithm around the language of cartesian closed categories. There appear to be some significant shortcomings, however, at least when considered as an extension to Elliott (2018):

- Although the work is referred to as "differentiable programming", it appears to lack a specification and proof that match this claim, i.e., one defined by the mathematical operation of differentiation. As such, it's unclear to me whether the algorithm is about differentiation or something else. In contrast, the specification at the center of Elliott (2018) (and the extensions described above) is just (Fréchet) differentiation itself, combined with the original function as needed by the chain rule, or rather the requirement that the function-with-derivative satisfies a standard collection of homomorphism properties. Correctness of the algorithm was defined as faithfulness to this simple specification, and the algorithm is systematically derived from this specification and hence is correct by construction.

- Functions are already well-defined as a vector space, and thus linear maps (including derivatives) are as well, but the authors chose a different notion. They write

  > [...] what should be the tangent space of a function type? Perhaps surprisingly, a function type itself is not the right answer. We provide two possible implementations for function tangents and differentiable currying, and explain the tradeoffs.

  There is no explanation, however, of what makes their answers "right" and the unsurprising answer wrong. It is unclear what it could possibly mean for their answer to be right, since the usual notion of derivative of a function $f : a \to b$ between vector spaces has type $a \to a \multimap b$ for all vector spaces a and b, *including function types.* This observation seems to contradict the claim that the tangent space for a function types is not a function type.

- The algorithm presented is limited to reverse mode rather than a general AD algorithm as in Elliott (2018) and the work described above.

Another related paper is Brunel et al. (2019). The authors write (in Section 1)

> However, Elliot's approach is still restricted to first-order programs (i.e., computational graphs): as far as we understand, the functor D is cartesian but not cartesian closed, so the higher-order primitives (λ-abstraction and application) lack a satisfactory treatment. This is implicit in Sect. 4.4 of Elliott (2018), where the author states that he only uses biproduct categories: it is well-known that non-trivial cartesian closed biproduct categories do not exist.

The confusion here—which was mistakenly encouraged by Elliott (2018)—is the idea that the category of differentiable functions itself is (or need be) a biproduct category. Rather, all that was needed is that the various representations of *linear maps* (derivatives) are biproduct categories. This requirement is easily satisfied by construction, since these representations are all calculated from their denotation (linear functions, itself a biproduct category) via simple cocartesian functors.

[27] I am in the middle of an in-depth conversation with authors.

References

Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. *CoRR*, abs/1909.13768, 2019.

Conal Elliott. Compiling to categories. In *Proceedings of the ACM on Programming Languages (ICFP)*, 2017.

Conal Elliott. The essence of automatic differentiation. In *Proceedings of the ACM on Programming Languages (ICFP)*, 2018.

Jeremy Gibbons. Calculating functional programs. In *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, volume 2297 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

Marian Boykan Pour-El and Ian Richards. Differentiability properties of computable functions—A summary. *Acta Cybernetica*, 4(1):123–125, 1978.

Marian Boykan Pour-El and Ian Richards. Computability and noncomputability in classical analysis. *Transactions of the American Mathematical Society*, 275(2):539–560, 1983.

Dimitrios Vytiniotis, Dan Belov, Richard Wei, Gordon Plotkin, and Martin Abadi. The differentiable curry. October 2019. To appear in the Program Transformations workshop at NeurIPS 2019.

A Proofs

A.1 Lemma 4

Suppose we have a function $f : a \times b \to c$, and we want to compute its derivative at a point in its (pair-valued) domain. Because linear maps (derivatives) form a cocartesian category,[28]

$$\mathcal{D}\ f\ (a,b) = \mathcal{D}\ f\ (a,b) \circ \mathit{inl} \triangledown \mathcal{D}\ f\ (a,b) \circ \mathit{inr}$$

Noting that (for linear maps) $\mathit{inl}\ da = (da, 0)$ and $\mathit{inr}\ db = (0, db)$, we can see that the "partial derivatives" ($\mathcal{D}\ f\ (a,b) \circ \mathit{inl}$ and $\mathcal{D}\ f\ (a,b) \circ \mathit{inr}$) allow only one half of a pair to change.

Next, note that $\mathcal{D}\ f\ (a,b) \circ \mathit{inl} = \mathcal{D}\ (f \circ (,b))\ a$, by the following equational reasoning:

$$\begin{array}{ll}
\ \ \mathcal{D}\ (f \circ (,b))\ a & \\
= \mathcal{D}\ f\ ((,b)\ a) \circ \mathcal{D}\ (,b)\ a & \text{-- chain rule (Theorem 1)} \\
= \mathcal{D}\ f\ (a,b) \circ \mathcal{D}\ (,b)\ a & \text{-- } (,b) \text{ definition} \\
= \mathcal{D}\ f\ (a,b) \circ \mathcal{D}\ (\mathit{inl} + \mathit{const}\ (0,b))\ a & \text{-- } \mathit{inl} \text{ on functions, and meaning of } (,b) \\
= \mathcal{D}\ f\ (a,b) \circ (\mathcal{D}\ \mathit{inl}\ a + \mathcal{D}\ (\mathit{const}\ (0,b))\ a) & \text{-- linearity of } (+) \\
= \mathcal{D}\ f\ (a,b) \circ \mathcal{D}\ \mathit{inl}\ a & \text{-- } \mathcal{D}\ (\mathit{const}\ z)\ a = 0 \\
= \mathcal{D}\ f\ (a,b) \circ \mathit{inl} & \text{-- linearity of } \mathit{inl}\text{; Theorem 3}
\end{array}$$

Likewise, $\mathcal{D}\ f\ (a,b) \circ \mathit{inr} = \mathcal{D}\ (f \circ (a,))\ b$.

[28]The cocartesian law $h = h \circ \mathit{inl} \triangledown h \circ \mathit{inr}$ is dual to the cartesian law $h = \mathit{exl} \circ h \vartriangle \mathit{exr} \circ h$ (Gibbons, 2002).

A.2 Corollary 4.2

$$\begin{aligned}
&\mathcal{D}\ (\mathit{uncurry}\ g)\ (a,b) && \\
={}&\mathcal{D}_l\ (\mathit{uncurry}\ g)\ (a,b) \triangledown \mathcal{D}_r\ (\mathit{uncurry}\ g)\ (a,b) && \text{-- Lemma 4} \\
={}&\mathcal{D}\ (\mathit{uncurry}\ g \circ (,b))\ a \triangledown \mathcal{D}\ (\mathit{uncurry}\ g \circ (a,))\ b && \text{-- } \mathcal{D}_l \text{ and } \mathcal{D}_r \text{ definitions} \\
={}&\mathcal{D}\ (\lambda\, a' \rightarrow \mathit{uncurry}\ g\ (a',b))\ a \triangledown && \text{-- } \eta \text{ expansion and simplification} \\
&\mathcal{D}\ (\lambda\, b' \rightarrow \mathit{uncurry}\ g\ (a,b'))\ b && \\
={}&\mathcal{D}\ (\lambda\, a' \rightarrow g\ a'\ b)\ a \triangledown \mathcal{D}\ (\lambda\, b' \rightarrow g\ a\ b')\ b && \text{-- } \mathit{uncurry} \text{ on functions} \\
={}&\mathcal{D}\ (\mathit{at}\ b \circ g)\ a \triangledown \mathcal{D}\ (g\ a)\ b && \text{-- } \mathit{at} \text{ definition and } \eta \text{ reduction} \\
={}&\mathcal{D}\ (\mathit{at}\ b)\ (g\ a) \circ \mathcal{D}\ g\ a \triangledown \mathcal{D}\ (g\ a)\ b && \text{-- chain rule (Theorem 1)} \\
={}&\mathit{at}\ b \circ \mathcal{D}\ g\ a \triangledown \mathcal{D}\ (g\ a)\ b && \text{-- linearity of } \mathit{at}
\end{aligned}$$

A.3 Corollary 4.3

$$\begin{aligned}
&\mathcal{D}\ \mathit{eval}\ (f,a) && \\
={}&\mathcal{D}_l\ \mathit{eval}\ (f,a) \triangledown \mathcal{D}_r\ \mathit{eval}\ (f,a) && \text{-- Lemma 4} \\
={}&\mathcal{D}\ (\mathit{eval} \circ (,a))\ f \triangledown \mathcal{D}\ (\mathit{eval} \circ (f,))\ a && \text{-- } \mathcal{D}_l \text{ and } \mathcal{D}_r \text{ alternative definitions} \\
={}&\mathcal{D}\ (\mathit{at}\ a)\ f \triangledown \mathcal{D}\ f\ a && \text{-- } \mathit{eval} \text{ on functions; } \mathit{at} \text{ definition} \\
={}&\mathit{at}\ a \triangledown \mathcal{D}\ f\ a && \text{-- linearity of } \mathit{at}\ a \\
={}&\lambda\,(\mathit{df},\mathit{dx}) \rightarrow \mathit{df}\ a + \mathcal{D}\ f\ a\ \mathit{dx} && \text{-- } (\triangledown)\ \textit{on linear maps};\ \mathit{at} \text{ definition}
\end{aligned}$$

Alternatively, calculate $\mathcal{D}\ \mathit{eval}$ via *uncurry*:

$$\begin{aligned}
&\mathcal{D}\ \mathit{eval}\ (f,a) && \\
={}&\mathcal{D}\ (\mathit{uncurry}\ \mathit{id})\ (f,a) && \text{-- } \mathit{eval} = \mathit{uncurry}\ \mathit{id} \\
={}&\mathit{at}\ a \circ \mathcal{D}\ \mathit{id}\ a \triangledown \mathcal{D}\ (\mathit{id}\ f)\ a && \text{-- Corollary 4.2} \\
={}&\mathit{at}\ a \circ \mathit{id} \triangledown \mathcal{D}\ f\ a && \text{-- } \mathit{id} \text{ linearity} \\
={}&\mathit{at}\ a \triangledown \mathcal{D}\ f\ a && \text{-- } \mathit{id} \text{ as identity}
\end{aligned}$$

A.4 Lemma 5

$$\begin{aligned}
&\mathit{fork}_F\ (\lambda\, b \rightarrow \mathcal{D}\ (\mathit{at}\ b \circ g)\ a) && \\
={}&\lambda\, \mathit{da}\ b \rightarrow \mathcal{D}\ (\mathit{at}\ b \circ g)\ a\ \mathit{da} && \text{-- } \mathit{fork}_F \text{ definition} \\
={}&\lambda\, \mathit{da}\ b \rightarrow (\mathcal{D}\ (\mathit{at}\ b)\ (g\ a) \circ \mathcal{D}\ g\ a)\ \mathit{da} && \text{-- chain rule (Theorem 1)} \\
={}&\lambda\, \mathit{da}\ b \rightarrow (\mathit{at}\ b \circ \mathcal{D}\ g\ a)\ \mathit{da} && \text{-- } \mathit{at}\ b \text{ linearity} \\
={}&\lambda\, \mathit{da}\ b \rightarrow \mathit{at}\ b\ (\mathcal{D}\ g\ a\ \mathit{da}) && \text{-- } (\circ) \text{ on functions} \\
={}&\lambda\, \mathit{da}\ b \rightarrow \mathcal{D}\ g\ a\ \mathit{da}\ b && \text{-- } \mathit{at} \text{ definition} \\
={}&\mathcal{D}\ g\ a && \text{-- } \eta \text{ reduction (twice)}
\end{aligned}$$

A.5 Corollary 5.1

$$\begin{array}{ll}
\quad \mathcal{D}\;(\mathit{curry}\;f)\;a & \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}\;(\mathit{at}\;b \circ \mathit{curry}\;f))\;a & \text{-- Lemma 5} \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}\;(\lambda\,a \to \mathit{at}\;b\;(\mathit{curry}\;f\;a)))\;a & \text{-- } (\circ) \text{ on functions} \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}\;(\lambda\,a \to \mathit{curry}\;f\;a\;b))\;a & \text{-- } \mathit{at} \text{ definition} \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}\;(\lambda\,a \to f\;(a,b)))\;a & \text{-- } \mathit{curry} \text{ on functions} \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}\;(f \circ (,b)))\;a & \text{-- } (,b) \text{ definition} \\
= \mathit{fork}_F\;(\lambda\,b \to \mathcal{D}_l\;f\;(a,b)) & \text{-- } \mathcal{D}_l \text{ definition} \\
= \mathit{fork}_F\;(\mathcal{D}_l\;f \circ (a,)) & \text{-- } (a,) \text{ definition}
\end{array}$$

A.6 Lemma 9

The functions wrap_o and wrap_o^{-1} form an isomorphism:

$$\begin{array}{ll}
\quad \mathit{wrap}_o^{-1}\;(\mathit{wrap}_o\;f) & \\
= \mathit{wrap}_o^{-1}\;(o \circ f \circ o^{-1}) & \text{-- } \mathit{wrap}_o \text{ definition} \\
= o^{-1} \circ (o \circ f \circ o^{-1}) \circ o & \text{-- } \mathit{wrap}_o^{-1} \text{ definition} \\
= (o^{-1} \circ o) \circ f \circ (o^{-1} \circ o) & \text{-- } (\circ) \text{ associativity} \\
= \mathit{id} \circ f \circ \mathit{id} & \text{-- } o^{-1} \circ o = \mathit{id} \\
= f & \text{-- } \mathit{id} \text{ is identity for } (\circ)
\end{array}$$

$$\begin{array}{ll}
\quad \mathit{wrap}_o\;(\mathit{wrap}_o^{-1}\;h) & \\
= \mathit{wrap}_o\;(o^{-1} \circ h \circ o) & \text{-- } \mathit{wrap}_o^{-1} \text{ definition} \\
= o \circ (o^{-1} \circ h \circ o) \circ o^{-1} & \text{-- } \mathit{wrap}_o \text{ definition} \\
= (o \circ o^{-1}) \circ h \circ (o \circ o^{-1}) & \text{-- } (\circ) \text{ associativity} \\
= \mathit{id} \circ h \circ \mathit{id} & \text{-- } o \circ o^{-1} = \mathit{id} \\
= h & \text{-- } \mathit{id} \text{ is identity for } (\circ)
\end{array}$$

Linearity of wrap_o and wrap_o^{-1} follows from two facts:

- $(\circ\, f)$ is linear for all f.
- $(g\, \circ)$ is linear for all *linear* g.

Proof: exercise.

A.7 Lemma 10

The functions $\mathcal{D}_o$ and $\mathcal{D}_o^{-1}$ form an isomorphism:

$$\begin{array}{ll}
\quad \mathcal{D}_o^{-1} \circ \mathcal{D}_o & \\
= \mathit{wrap}_o^{-1} \circ \hat{\mathcal{D}}^{-1} \circ \hat{\mathcal{D}} \circ \mathit{wrap}_o & \text{-- } \mathcal{D}_o^{-1} \text{ and } \mathcal{D}_o \text{ definitions} \\
= \mathit{wrap}_o^{-1} \circ \mathit{wrap}_o & \text{-- } \hat{\mathcal{D}}^{-1} \circ \hat{\mathcal{D}} = \mathit{id} \\
= \mathit{id} & \text{-- } \mathit{wrap}_o^{-1} \circ \mathit{wrap}_o = \mathit{id}
\end{array}$$

$$\quad \mathcal{D}_o \circ \mathcal{D}_o^{-1}$$

$= \hat{\mathcal{D}} \circ \mathit{wrap}_o \circ \mathit{wrap}_o^{-1} \circ \mathcal{D}_o^{-1}$ -- $\mathcal{D}_o$ and $\mathcal{D}_o^{-1}$ definitions
$= \hat{\mathcal{D}} \circ \mathcal{D}_o^{-1}$ -- $\mathit{wrap}_o^{-1} \circ \mathit{wrap}_o = \mathit{id}$
$= \mathit{id}$ -- $\hat{\mathcal{D}}^{-1} \circ \hat{\mathcal{D}} = \mathit{id}$

Linearity of $\mathcal{D}_o$ and $\mathcal{D}_o^{-1}$ follows from linearity of $\hat{\mathcal{D}}$ and $\hat{\mathcal{D}}^{-1}$ and Lemma 9.

A.8 Lemma 11

The proof that wrap_o is a cartesian functor mainly exploit the regular structure of o and o^{-1}:

$\mathit{wrap}_o\ \mathit{id}$
$= o \circ \mathit{id} \circ o^{-1}$ -- wrap_o definition
$= o \circ o^{-1}$ -- id is identity for $(\circ)$
$= \mathit{id}$ -- $o \circ o^{-1} = \mathit{id}$

$\mathit{wrap}_o\ (g \circ f)$
$= o \circ g \circ f \circ o^{-1}$ -- wrap_o definition
$= o \circ g \circ o^{-1} \circ o \circ f \circ o^{-1}$ -- $o^{-1} \circ o = \mathit{id}$
$= (o \circ g \circ o^{-1}) \circ (o \circ f \circ o^{-1})$ -- $\hat{\mathcal{D}}$ is a functor
$= \mathit{wrap}_o\ g \circ \mathit{wrap}_o\ f$ -- wrap_o definition

$\mathit{wrap}_o\ (f \times g)$ -- $\mathcal{D}_o$ definition
$= o \circ (f \times g) \circ o^{-1}$ -- wrap_o definition
$= (o \times o) \circ (f \times g) \circ (o^{-1} \times o^{-1})$ -- o on products
$= o \circ f \circ o^{-1} \times o \circ g \circ o^{-1}$ -- monoidal category law
$= o \circ f \circ o^{-1} \times o \circ g \circ o^{-1}$ -- $\hat{\mathcal{D}}$ is a monoidal functor
$= \mathit{wrap}_o\ f \times \mathit{wrap}_o\ g$ -- wrap_o definition

$\mathit{wrap}_o\ \mathit{exl}$ -- $\mathcal{D}_o$ definition
$= o \circ \mathit{exl} \circ o^{-1}$ -- wrap_o definition
$= o \circ \mathit{exl} \circ (o^{-1} \times o^{-1})$ -- o^{-1} on products
$= o \circ o^{-1} \circ \mathit{exl}$ -- $\mathit{exl} \circ (f \times g) = f \circ \mathit{exl}$ for cartesian categories
$= \mathit{exl}$ -- $o \circ o^{-1} = \mathit{id}$

$\mathit{wrap}_o\ \mathit{exr}$ -- $\mathcal{D}_o$ definition
$= o \circ \mathit{exr} \circ o^{-1}$ -- wrap_o definition
$= o \circ \mathit{exr} \circ (o^{-1} \times o^{-1})$ -- o^{-1} on products
$= o \circ o^{-1} \circ \mathit{exr}$ -- $\mathit{exr} \circ (f \times g) = g \circ \mathit{exr}$ for cartesian categories
$= \mathit{exr}$ -- $o \circ o^{-1} = \mathit{id}$

$\mathit{wrap}_o\ \mathit{dup}$ -- $\mathcal{D}_o$ definition
$= o \circ \mathit{dup} \circ o^{-1}$ -- wrap_o definition
$= o \circ (o^{-1} \times o^{-1}) \circ \mathit{dup}$ -- $\mathit{dup} \circ f = (f \times f) \circ \mathit{dup}$ for cartesian categories
$= o \circ o^{-1} \circ \mathit{dup}$ -- o^{-1} on products
$= \mathit{dup}$ -- $o \circ o^{-1} = \mathit{id}$

A.9 Lemma 13

$$
\begin{aligned}
&\mathit{wrap}_o\ (\mathit{curry}\ f) \\
&= o \circ \mathit{curry}\ f \circ o^{-1} && \text{-- } \mathit{wrap}_o \text{ definition} \\
&= \mathcal{D}_o \circ \mathit{curry}\ f \circ o^{-1} && \text{-- } o \text{ on functions} \\
&= \hat{\mathcal{D}} \circ \mathit{wrap}_o \circ \mathit{curry}\ f \circ o^{-1} && \text{-- } \mathcal{D}_o \text{ definition} \\
&= \hat{\mathcal{D}} \circ \mathit{curry}\ (\mathit{wrap}_o\ f) && \text{-- below}
\end{aligned}
$$

For this last step,

$$
\begin{aligned}
&\mathit{wrap}_o \circ \mathit{curry}\ f \circ o^{-1} \\
&= \lambda\ a \to \mathit{wrap}_o\ (\mathit{curry}\ f\ (o^{-1}\ a)) && \text{-- } \eta \text{ expansion} \\
&= \lambda\ a \to o \circ \mathit{curry}\ f\ (o^{-1}\ a) \circ o^{-1} && \text{-- } \mathit{wrap}_o \text{ definition} \\
&= \lambda\ a\ b \to o\ (\mathit{curry}\ f\ (o^{-1}\ a)\ (o^{-1}\ b)) && \text{-- } \eta \text{ expansion} \\
&= \lambda\ a\ b \to o\ (f\ (o^{-1}\ a, o^{-1}\ b)) && \text{-- } \mathit{curry} \text{ on functions} \\
&= \lambda\ a\ b \to o\ (f\ (o^{-1}\ (a, b))) && \text{-- } o^{-1} \text{ on pairs} \\
&= \lambda\ a\ b \to \mathit{wrap}_o\ f\ (a, b) && \text{-- } \mathit{wrap}_o \text{ definition} \\
&= \mathit{curry}\ (\mathit{wrap}_o\ f) && \text{-- } \mathit{curry} \text{ on functions}
\end{aligned}
$$

Equivalently, $\mathit{curry}\ (\mathit{wrap}_o\ f) = \hat{\mathcal{D}}^{-1} \circ \mathit{wrap}_o\ (\mathit{curry}\ f)$.[29]

A.10 $\mathcal{D}_o$ and *eval*

The homomorphism equation is $\mathit{eval} = \mathcal{D}_o\ \mathit{eval}$. Simplifying the RHS,

$$
\begin{aligned}
&\mathcal{D}_o\ \mathit{eval} \\
&= \hat{\mathcal{D}}\ (\mathit{wrap}_o\ \mathit{eval}) && \text{-- } \mathcal{D}_o \text{ definition} \\
&= \hat{\mathcal{D}}\ (o \circ \mathit{eval} \circ o^{-1}) && \text{-- } \mathit{wrap}_o \text{ definition} \\
&= \hat{\mathcal{D}}\ (o \circ \mathit{eval} \circ (\mathcal{D}_o^{-1} \times o^{-1})) && \text{-- } o^{-1} \text{ on } (a \to b) \times a \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to (o \circ \mathit{eval} \circ (\mathcal{D}_o^{-1} \times o^{-1}))\ (\hat{f}, a)) && \text{-- } \eta \text{ expansion} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to o\ (\mathit{eval}\ (\mathcal{D}_o^{-1}\ \hat{f}, o^{-1}\ a))) && \text{-- } (\circ) \text{ and } (\times) \text{ on functions} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to o\ (\mathcal{D}_o^{-1}\ \hat{f}\ (o^{-1}\ a))) && \text{-- } \mathit{eval} \text{ on functions} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to o\ (\mathit{wrap}_o^{-1}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})\ (o^{-1}\ a))) && \text{-- } \mathcal{D}_o^{-1} \text{ definition} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to o\ ((o^{-1} \circ \hat{\mathcal{D}}^{-1}\ \hat{f} \circ o)\ (o^{-1}\ a))) && \text{-- } \mathit{wrap}_o^{-1} \text{ definition} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to o\ (o^{-1}\ (\hat{\mathcal{D}}^{-1}\ \hat{f}\ (o\ (o^{-1}\ a))))) && \text{-- } (\circ) \text{ on functions} \\
&= \hat{\mathcal{D}}\ (\lambda\,(\hat{f}, a) \to \hat{\mathcal{D}}^{-1}\ \hat{f}\ a) && \text{-- } o \circ o^{-1} = \mathit{id} \\
&= \hat{\mathcal{D}}\ (\mathit{uncurry}\ \hat{\mathcal{D}}^{-1}) && \text{-- } \mathit{uncurry} \text{ on functions} \\
&= D\ (\lambda\,(\hat{f}, a) \to (\mathit{uncurry}\ \hat{\mathcal{D}}^{-1}\ (\hat{f}, a), \mathcal{D}\ (\mathit{uncurry}\ \hat{\mathcal{D}}^{-1})\ (\hat{f}, a))) && \text{-- } \hat{\mathcal{D}} \text{ definition} \\
&= D\ (\lambda\,(\hat{f}, a) \to (\hat{\mathcal{D}}^{-1}\ \hat{f}\ a, \mathcal{D}\ (\mathit{uncurry}\ \hat{\mathcal{D}}^{-1})\ (\hat{f}, a))) && \text{-- } \mathit{uncurry} \text{ on functions} \\
&= D\ (\lambda\,(\hat{f}, a) \to (\hat{\mathcal{D}}^{-1}\ \hat{f}\ a, \mathit{at}\ a \circ \mathcal{D}\ \hat{\mathcal{D}}^{-1}\ \hat{f} \triangledown \mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})\ a)) && \text{-- Section A.2} \\
&= D\ (\lambda\,(\hat{f}, a) \to (\hat{\mathcal{D}}^{-1}\ \hat{f}\ a, \mathit{at}\ a \circ \hat{\mathcal{D}}^{-1} \triangledown \mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})\ a)) && \text{-- } \hat{\mathcal{D}}^{-1} \text{ linearity}
\end{aligned}
$$

Now note that

[29] [Maybe this form will help simplify another proof.]

$$\begin{aligned}
&\hat{f} \\
&= \hat{\mathcal{D}}\ (\hat{\mathcal{D}}^{-1}\ \hat{f}) && \text{-- } \hat{\mathcal{D}} \circ \hat{\mathcal{D}}^{-1} = id \\
&= D\ (\hat{\mathcal{D}}^{-1}\ \hat{f} \mathbin{\triangle} \mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})) && \text{-- } \hat{\mathcal{D}} \text{ definition}
\end{aligned}$$

Letting $D\ h = \hat{f}$, we have

$$\begin{aligned}
h\ a &= (\hat{\mathcal{D}}^{-1}\ \hat{f} \mathbin{\triangle} \mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f}))\ a \\
&= (\hat{\mathcal{D}}^{-1}\ \hat{f}\ a, \mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f})\ a)
\end{aligned}$$

A bit of refactoring then replaces $\hat{\mathcal{D}}^{-1}\ \hat{f}\ a$ and (the noncomputable) $\mathcal{D}\ (\hat{\mathcal{D}}^{-1}\ \hat{f}\ a)$, yielding a *computable* form:

$$\begin{aligned}
&\mathcal{D}_o\ eval \\
&= \ ... \\
&= D\ (\lambda\,(D\ h, a) \rightarrow \mathbf{let}\ (b, f') = h\ a\ \mathbf{in}\ (b, at\ a \circ \hat{\mathcal{D}}^{-1} \mathbin{\triangledown} f'))
\end{aligned}$$

Since this calculation was fairly involved, let's get a sanity check on the types in the final form:

$$\begin{aligned}
(D\ h, a) &: O\ ((a \rightarrow b) \times a) \\
&: D\ (O\ a)\ (O\ b) \times O\ a \\
D\ h &: D\ (O\ a)\ (O\ b) \\
a &: O\ a \\
h &: O\ a \rightarrow O\ b \times (O\ a \multimap O\ b) \\
h\ a &: O\ b \times (O\ a \multimap O\ b)
\end{aligned}$$

$$\begin{aligned}
(b, f') &: O\ b \times (O\ a \multimap O\ b) \\
b &: O\ b \\
f' &: O\ a \multimap O\ b
\end{aligned}$$

$$\begin{aligned}
\hat{\mathcal{D}}^{-1} &: D\ (O\ a)\ (O\ b) \multimap (O\ a \rightarrow O\ b) \\
at\ a &: (O\ a \rightarrow O\ b) \multimap O\ b \\
at\ a \circ \hat{\mathcal{D}}^{-1} &: D\ (O\ a)\ (O\ b) \multimap O\ b \\
at\ a \circ \hat{\mathcal{D}}^{-1} \mathbin{\triangledown} f' &: D\ (O\ a)\ (O\ b) \times O\ a \multimap O\ b \\
(b, at\ a \circ \hat{\mathcal{D}}^{-1} \mathbin{\triangledown} f') &: O\ b \times (D\ (O\ a)\ (O\ b) \times O\ a \multimap O\ b) \\
\lambda\,(D\ h, a) \rightarrow ...\mathbf{in}\ (b, at\ a \circ \hat{\mathcal{D}}^{-1} \mathbin{\triangledown} f') &: O\ ((a \rightarrow b) \times a) \rightarrow O\ b \times (D\ (O\ a)\ (O\ b) \times O\ a \multimap O\ b) \\
D\ (\lambda\,(D\ h, a) \rightarrow ...\mathbf{in}\ (b, at\ a \circ \hat{\mathcal{D}}^{-1} \mathbin{\triangledown} f')) &: D\ (O\ ((a \rightarrow b) \times a))\ (O\ b)
\end{aligned}$$

$$\begin{aligned}
eval &: (a \rightarrow b) \times a \rightarrow b \\
\mathcal{D}_o\ \ eval &: D\ (O\ ((a \rightarrow b) \times a))\ (O\ b)
\end{aligned}$$

A.11 Lemma 15

Letting $f_o = wrap_o\, f$,

$$\begin{aligned}
&\mathcal{D}_o\ (curry\ f) \\
&= \hat{\mathcal{D}}\ (wrap_o\ (curry\ f)) && \text{-- } \mathcal{D}_o \text{ definition}
\end{aligned}$$

$$
\begin{array}{ll}
= \hat{\mathcal{D}}\ (\hat{\mathcal{D}} \circ \mathit{curry}\ (\mathit{wrap}_o\ f)) & \text{-- Lemma 13} \\
= \hat{\mathcal{D}}\ (\hat{\mathcal{D}} \circ \mathit{curry}\ f_o) & \text{-- } f_o \text{ definition} \\
= D\ ((\hat{\mathcal{D}} \circ \mathit{curry}\ f_o) \vartriangle \mathcal{D}\ (\hat{\mathcal{D}} \circ \mathit{curry}\ f_o)) & \text{-- } \hat{\mathcal{D}} \text{ definition} \\
= D\ (\lambda\, a \rightarrow \hat{\mathcal{D}}\ (\mathit{curry}\ f_o\ a), \mathcal{D}\ (\hat{\mathcal{D}} \circ \mathit{curry}\ f_o)\ a) & \text{-- } (\vartriangle) \text{ definition} \\
= D\ (\lambda\, a \rightarrow \hat{\mathcal{D}}\ (\mathit{curry}\ f_o\ a), \hat{\mathcal{D}} \circ \mathcal{D}\ (\mathit{curry}\ f_o)\ a) & \text{-- chain rule; linearity of } \hat{\mathcal{D}} \\
= D\ (\lambda\, a \rightarrow \hat{\mathcal{D}}\ (\mathit{curry}\ f_o\ a), \hat{\mathcal{D}} \circ \mathit{fork}_F\ (\mathcal{D}_l\ f_o \circ (a,))) & \text{-- Theorem 2}
\end{array}
$$

Now, separately simplify the two main parts of this last form.

$$
\begin{array}{ll}
\ \ \hat{\mathcal{D}}\ (\mathit{curry}\ f_o\ a) & \\
= D\ (\lambda\, b \rightarrow (f_o\ (a,b), \mathcal{D}_r\ f_o\ (a,b))) & \text{-- } \hat{\mathcal{D}} \text{ definition and Lemma 4}
\end{array}
$$

$$
\begin{array}{ll}
\ \ \hat{\mathcal{D}} \circ \mathit{fork}_F\ (\mathcal{D}_l\ f_o \circ (a,)) & \\
= \hat{\mathcal{D}} \circ (\lambda\, da\ b \rightarrow (\mathcal{D}_l\ f_o \circ (a,))\ b\ da) & \text{-- } \mathit{fork}_F \text{ definition} \\
= \hat{\mathcal{D}} \circ (\lambda\, da\ b \rightarrow \mathcal{D}_l\ f_o\ (a,b)\ da) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, da \rightarrow \hat{\mathcal{D}}\ (\lambda\, b \rightarrow \mathcal{D}_l\ f_o\ (a,b)\ da) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, da \rightarrow \hat{\mathcal{D}}\ (\lambda\, b \rightarrow \mathit{at}\ da\ (\mathcal{D}_l\ f_o\ (a,b))) & \text{-- } \mathit{at} \text{ definition} \\
= \lambda\, da \rightarrow \hat{\mathcal{D}}\ (\mathit{at}\ da \circ \mathcal{D}_l\ f_o \circ (a,)) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, da \rightarrow D\ (\lambda\, b \rightarrow ((\mathit{at}\ da \circ \mathcal{D}_l\ f_o \circ (a,))\ b, \mathcal{D}\ (\mathit{at}\ da \circ \mathcal{D}_l\ f_o \circ (a,))\ b)) & \text{-- } \hat{\mathcal{D}} \text{ definition} \\
= \lambda\, da \rightarrow D\ (\lambda\, b \rightarrow (\mathcal{D}_l\ f_o\ (a,b)\ da, \mathcal{D}\ (\mathit{at}\ da \circ \mathcal{D}_l\ f_o \circ (a,))\ b)) & \text{-- } (\circ) \text{ on functions}
\end{array}
$$

Now simplify the remaining differentiated composition:

$$
\begin{array}{ll}
\ \ \mathcal{D}\ (\mathit{at}\ da \circ \mathcal{D}_l\ f_o \circ (a,))\ b & \\
= \mathit{at}\ da \circ \mathcal{D}\ (\mathcal{D}_l\ f_o \circ (a,))\ b & \text{-- chain rule; linearity of } \mathit{at}\ da \\
= \mathit{at}\ da \circ \mathcal{D}_r\ (\mathcal{D}_l\ f_o)\ (a,b) & \text{-- Lemma 4}
\end{array}
$$

Putting the pieces back together,

$$
\begin{array}{l}
\mathcal{D}_o\ (\mathit{curry}\ f) = \\
\quad D\ (\lambda\, a \rightarrow (D\ (\lambda\, b \rightarrow (f_o\ (a,b), \mathcal{D}_r\ f_o\ (a,b))) \\
\qquad ,\lambda\, da \rightarrow D\ (\lambda\, b \rightarrow (\mathcal{D}_l\ f_o\ (a,b)\ da, \mathit{at}\ da \circ \mathcal{D}_r\ (\mathcal{D}_l\ f_o)\ (a,b)))))
\end{array}
$$

A.12 Lemma 16

To show that $\mathit{comp} = \mathit{uncurry}\ (\circ)$ is bilinear, we can show that it is linear in each argument, which is to say $\mathit{curry}\ \mathit{comp}\ g = (g\ \circ)$ and $\mathit{curry}'\ \mathit{comp}\ f = (\circ\ f)$ are linear for all g and f.

First, $(\circ\ f)$ is linear for *any* function f (not just linear):

$$
\begin{array}{ll}
\ \ (\circ\ f)\ (g + g') & \\
= (g + g') \circ f & \text{-- left section definition} \\
= \lambda\, a \rightarrow (g + g')\ (f\ a) & \text{-- } \eta \text{ expansion} \\
= \lambda\, a \rightarrow g\ (f\ a) + g'\ (f\ a) & \text{-- addition on functions} \\
= (\lambda\, a \rightarrow g\ (f\ a)) + (\lambda\, a \rightarrow g'\ (f\ a)) & \text{-- addition on functions}
\end{array}
$$

$$
\begin{array}{ll}
= (g \circ f) + (g' \circ f) & \text{-- } (\circ) \text{ on functions} \\
= (\circ f)\ g + (\circ f)\ g' & \text{-- left section definition}
\end{array}
$$

$$
\begin{array}{ll}
\quad (\circ f)\ (s \cdot g) & \\
= (s \cdot g) \circ f & \text{-- left section definition} \\
= \lambda\, a \rightarrow (s \cdot g)\ (f\ \ a) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, a \rightarrow s \cdot g\ (f\ \ a) & \text{-- scaling on functions} \\
= s \cdot (\lambda\, a \rightarrow g\ (f\ \ a)) & \text{-- scaling on functions} \\
= s \cdot (g \circ f) & \text{-- } (\circ) \text{ on functions} \\
= s \cdot (\circ f)\ g & \text{-- left section definition}
\end{array}
$$

Second, $(g\ \circ)$ is linear for any *linear* functions g:

$$
\begin{array}{ll}
\quad (g\ \circ)\ (f + f') & \\
= g \circ (f + f') & \text{-- right section definition} \\
= \lambda\, a \rightarrow g\ ((f + f')\ a) & \text{-- } \eta \text{ expansion} \\
= \lambda\, a \rightarrow g\ (f\ \ a + f'\ \ a) & \text{-- addition on functions} \\
= \lambda\, a \rightarrow g\ (f\ \ a) + g\ (f'\ \ a) & \text{-- linearity of } g \\
= (\lambda\, a \rightarrow g\ (f\ \ a)) + (\lambda\, a \rightarrow g\ (f'\ \ a)) & \text{-- addition on functions} \\
= (g \circ f) + (g \circ f') & \text{-- } (\circ) \text{ on functions} \\
= (g\ \circ)\ f + (g\ \circ)\ f' & \text{-- right section definition}
\end{array}
$$

$$
\begin{array}{ll}
\quad (g\ \circ)\ (s \cdot f) & \\
= g \circ (s \cdot f) & \text{-- right section definition} \\
= \lambda\, a \rightarrow g\ ((s \cdot f)\ a) & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, a \rightarrow g\ (s \cdot f\ \ a) & \text{-- scaling on functions} \\
= \lambda\, a \rightarrow s \cdot g\ (f\ \ a) & \text{-- linearity of } g \\
= s \cdot (g \circ f) & \text{-- scaling on functions} \\
= s \cdot (g\ \circ)\ f & \text{-- right section definition}
\end{array}
$$

A.13 Lemma 17

Given any bilinear function h,

a. *curry* $h\ a$ is linear for all linear functions g:

$$
\begin{array}{ll}
\quad \mathit{curry}\ h\ a\ (b + b') & \\
= h\ (a, b + b') & \text{-- } \mathit{curry} \text{ on functions} \\
= h\ (a, b) + h\ (a, b') & \text{-- bilinearity of } h \\
= \mathit{curry}\ h\ a\ b + \mathit{curry}\ h\ a\ b' & \text{-- } \mathit{curry} \text{ on functions}
\end{array}
$$

$$
\begin{array}{ll}
\quad \mathit{curry}\ h\ a\ (s \cdot b) & \\
= h\ (a, s \cdot b) & \text{-- } \mathit{curry} \text{ on functions} \\
= s \cdot h\ (a, b) & \text{-- bilinearity of } h
\end{array}
$$

b. *curry'* $h\ b$ is linear for all functions b: Proof similar to *curry* $h\ a$.

c. *curry* h and *curry*$'$ h are linear:

$$\begin{array}{ll}
\quad curry\ h\ (a + a') & \\
= \lambda\, b \to curry\ h\ (a + a')\ b & \text{-- } \eta \text{ expansion} \\
= \lambda\, b \to h\ (a + a', b) & \text{-- } curry \text{ on functions} \\
= \lambda\, b \to h\ (a, b) + h\ (a', b) & \text{-- bilinearity of } h \\
= (\lambda\, b \to h\ (a, b)) + (\lambda\, b \to h\ (a', b)) & \text{-- addition on functions} \\
= curry\ h\ a + curry\ h\ a' & \text{-- } curry \text{ on functions}
\end{array}$$

$$\begin{array}{ll}
\quad curry\ h\ (s \cdot a) & \\
= \lambda\, b \to curry\ h\ (s \cdot a)\ b & \text{-- } \eta \text{ expansion} \\
= \lambda\, b \to h\ (s \cdot a', b) & \text{-- } curry \text{ on functions} \\
= \lambda\, b \to s \cdot h\ (a, b) & \text{-- bilinearity of } h \\
= s \cdot (\lambda\, b \to h\ (a, b)) & \text{-- scaling on functions} \\
= s \cdot curry\ h\ a & \text{-- } curry \text{ on functions}
\end{array}$$

Similarly for *curry*$'$ h.

d. $\mathcal{D}\ h$ is linear:

$$\begin{array}{ll}
\quad \mathcal{D}\ h\ ((a, b) + (a', b')) & \\
= \mathcal{D}\ h\ (a + a', b + b') & \text{-- } (+) \text{ on functions} \\
= h \circ (, b + b')\ \triangledown\ h \circ (a + a') & \text{-- Corollary 4.1} \\
= \lambda\, (da, db) \to h\ (da, b + b') + h\ (a + a', db) & \text{-- } (\triangledown) \text{ on functions} \\
= \lambda\, (da, db) \to h\ (da, b) + h\ (da, b') + h\ (a, db) + h\ (a', db) & \text{-- bilinearity of } h \\
= \lambda\, (da, db) \to h\ (da, b) + h\ (a, db) + h\ (da, b') + h\ (a', db) & \text{-- commutativity of } (+) \\
= (\lambda\, (da, db) \to h\ (da, b\ ) + h\ (a\ , db)) + & \\
\quad (\lambda\, (da, db) \to h\ (da, b') + h\ (a', db)) & \text{-- } (+) \text{ on functions} \\
= \mathcal{D}\ h\ (a, b) + \mathcal{D}\ h\ (a', b') & \text{-- Corollary 4.1}
\end{array}$$

Similarly for scaling.

A.14 Lemma 18

a. Sequential composition:

$$\begin{array}{ll}
\quad \mathcal{D}\ (g \circ f) & \\
= \lambda\, a \to \mathcal{D}\ (g \circ f)\ a & \text{-- } \eta \text{ expansion} \\
= \lambda\, a \to \mathcal{D}\ g\ (f\ a) \circ \mathcal{D}\ f\ a & \text{-- chain rule (Theorem 1)} \\
= \lambda\, a \to (\mathcal{D}\ g \circ f)\ a \circ \mathcal{D}\ f\ a & \text{-- } (\circ) \text{ on functions} \\
= \lambda\, a \to (\circ)\ ((\mathcal{D}\ g \circ f)\ a)\ (\mathcal{D}\ f\ a) & \text{-- alternative notation} \\
= \lambda\, a \to uncurry\ (\circ)\ ((\mathcal{D}\ g \circ f)\ a, \mathcal{D}\ f\ a) & \text{-- } uncurry \text{ on functions} \\
= \lambda\, a \to comp\ ((\mathcal{D}\ g \circ f)\ a, \mathcal{D}\ f\ a) & \text{-- } comp \text{ definition} \\
= comp \circ (\lambda\, a \to ((\mathcal{D}\ g \circ f)\ a, \mathcal{D}\ f\ a)) & \text{-- } (\circ) \text{ on functions} \\
= comp \circ (\mathcal{D}\ g \circ f \vartriangle \mathcal{D}\ f) & \text{-- } (\vartriangle) \text{ definition}
\end{array}$$

b. Cross:

$$\begin{array}{ll}
\quad \mathcal{D}\ (f \times g) & \\
= \lambda\,(a, b) \to \mathcal{D}\ (f \times g)\ (a, b) & \text{-- } \eta \text{ expansion} \\
= \lambda\,(a, b) \to \mathcal{D}\ f\ \ a \times \mathcal{D}\ g\ \ b & \text{-- cross rule (Theorem 2)} \\
= \lambda\,(a, b) \to \mathit{uncurry}\ (\times)\ (\mathcal{D}\ f\ \ a, \mathcal{D}\ g\ \ b) & \text{-- } \mathit{uncurry} \text{ on functions} \\
= \lambda\,(a, b) \to \mathit{cross}\ (\mathcal{D}\ f\ \ a, \mathcal{D}\ g\ \ b) & \text{-- } \mathit{cross} \text{ definition} \\
= \lambda\,(a, b) \to \mathit{cross}\ ((\mathcal{D}\ f \times \mathcal{D}\ g)\ (a, b)) & \text{-- } (\times) \text{ on functions} \\
= \mathit{cross} \circ (\mathcal{D}\ f \times \mathcal{D}\ g) & \text{-- } (\circ) \text{ on functions}
\end{array}$$

c. Fork:

$$\begin{array}{ll}
\quad \mathcal{D}\ (f \vartriangle g) & \\
= \mathcal{D}\ ((f \times g) \circ \mathit{dup}) & \text{-- cartesian law} \\
= \lambda\,a \to \mathcal{D}\ ((f \times g) \circ \mathit{dup})\ \ a & \text{-- } \eta \text{ expansion} \\
= \lambda\,a \to \mathcal{D}\ (f \times g)\ (\mathit{dup}\ \ a) \circ \mathcal{D}\ \mathit{dup}\ \ a & \text{-- chain rule (Theorem 1)} \\
= \lambda\,a \to \mathcal{D}\ (f \times g)\ (a, a) \circ \mathcal{D}\ \mathit{dup}\ \ a & \text{-- } \mathit{dup} \text{ for functions} \\
= \lambda\,a \to \mathcal{D}\ f\ \ a \times \mathcal{D}\ g\ \ a \circ \mathcal{D}\ \mathit{dup}\ \ a & \text{-- cross rule (Theorem 2)} \\
= \lambda\,a \to \mathcal{D}\ f\ \ a \times \mathcal{D}\ g\ \ a \circ \mathit{dup} & \text{-- } \mathit{dup} \text{ linearity} \\
= \lambda\,a \to \mathcal{D}\ f\ \ a \vartriangle \mathcal{D}\ g\ \ a & \text{-- cartesian law} \\
= \mathit{fork} \circ (\mathcal{D}\ f \vartriangle \mathcal{D}\ g) & \text{-- } \mathit{fork} \text{ definition}
\end{array}$$

d. A linear function f,

$$\begin{array}{ll}
\quad \mathcal{D}\ f & \\
= \lambda\,a \to \mathcal{D}\ f\ \ a & \text{-- } \eta \text{ expansion} \\
= \lambda\,a \to f & \text{-- Theorem 3} \\
= \mathit{const}\ f & \text{-- } \mathit{const} \text{ definition}
\end{array}$$

e. Any function $f : a \times b \to c$,

$$\begin{array}{ll}
\quad \mathcal{D}\ f & \\
= \lambda\,(a, b) \to \mathcal{D}\ f\ \ (a, b) & \text{-- } \eta \text{ expansion} \\
= \lambda\,(a, b) \to \mathcal{D}_l\ f\ \ (a, b) \triangledown \mathcal{D}_r\ f\ \ (a, b) & \text{-- Lemma 4} \\
= \mathit{join} \circ (\mathcal{D}_l\ f \vartriangle \mathcal{D}_r\ f) & \text{-- } \mathit{join} \text{ definition}
\end{array}$$

f. A *bilinear* function $f : a \times b \to c$,

$$\begin{array}{ll}
\quad \mathcal{D}\ f & \\
= \lambda\,(a, b) \to \mathcal{D}\ f\ \ (a, b) & \text{-- } \eta \text{ expansion} \\
= \lambda\,(a, b) \to f \circ (, b) \triangledown f \circ (a,) & \text{-- Corollary 4.1} \\
= \lambda\,(a, b) \to \mathit{curry}'\ f\ \ b \triangledown \mathit{curry}\ f\ \ a & \text{-- section definitions} \\
= \lambda\,(a, b) \to \mathit{join}\ (\mathit{curry}'\ f\ \ b, \mathit{curry}\ f\ \ a) & \text{-- } \mathit{join} = \mathit{uncurry}\ (\triangledown) \\
= \lambda\,(a, b) \to \mathit{join}\ ((\mathit{curry}'\ f \times \mathit{curry}\ f)\ (b, a)) & \text{-- } (\times) \text{ on functions} \\
= \lambda\,(a, b) \to \mathit{join}\ ((\mathit{curry}'\ f \times \mathit{curry}\ f)\ (\mathit{swap}\ (a, b))) & \text{-- } \mathit{swap} \text{ on functions} \\
= \mathit{join} \circ (\mathit{curry}'\ f \times \mathit{curry}\ f) \circ \mathit{swap} & \text{-- } (\circ) \text{ on functions}
\end{array}$$

g. Uncurried composition on linear maps,

$$
\begin{aligned}
&\mathcal{D}\ \mathit{comp} \\
&= \mathit{join} \circ (\mathit{curry}'\ \mathit{comp} \times \mathit{curry}\ \mathit{comp}) \circ \mathit{swap} && \text{-- previous } (\mathit{comp} \text{ is bilinear}) \\
&= \mathit{join} \circ (\mathit{flip}\ (\circ) \times (\circ)) \circ \mathit{swap} && \text{-- } \mathit{comp} \text{ definition}
\end{aligned}
$$