Denotational Semantics for Differentiable Programming with Manifolds

Student: Jesse Sigal
Supervisors: Luke Ong & Ohad Kammar
Category: Undergraduate

Department of Computer Science
University of Oxford

1 Problem and Motivation

Differentiating a user-defined function is required in various numerical programming situations. Examples include probabilistic programming, optimisation, and gradient-based machine learning. Obtaining the derivative without any further user effort and with complexity similar to the original function is desirable. Automatic differentiation (AD) attempts to achieve these goals. AD can calculate the derivative and value of a function simultaneously with only a ×3-4 slowdown [Griewank and Walther]. The reverse mode method of AD calculates all first-order partial derivatives of a function linearly w.r.t. the output dimension [ibid.]. Thus, reverse mode is ideal for high-dimension input, low-dimension output functions (such as neural networks composed with a cost function).

Smooth manifolds allow the extension of differential calculus beyond standard Euclidean spaces ($\mathbb{R}^n$). Optimisation tasks can take place on manifolds, such as a circle. One such example is fitting a periodic function to sampled data. A function $f\colon \mathbb{R} \to \mathbb{R}$ is periodic with period $p \in \mathbb{R}^{>0}$ if for all $x \in \mathbb{R}$, $f(x+p) = f(x)$. We can associate to f a function $\overline{f}\colon S^1 \to \mathbb{R}$ on the circle S^1 (where each point is given by its angle from the x-axis) by $\overline{f}(\theta) = f(\frac{p}{2\pi}\theta)$. The type $S^1 \to \mathbb{R}$ is a characterisation of periodicity, and thus it is more natural to fit a function of this type to periodic data than one of $\mathbb{R} \to \mathbb{R}$. Therefore, it is highly desirable for manifolds to be available as a data type.

AD is by no means solved. Manzyuk et al. discovered crippling bugs in a mature and performant system which stemmed from the interaction of higher-order functions and AD. Their solution also lacked sufficient theory to prove correctness against [Manzyuk et al.]. I believe my ongoing work on denotational semantics for differentiable programming is a useful tool for proving correctness of AD.

2 Background and Related Work

Smooth manifolds are one generalisation of Euclidean spaces ($\mathbb{R}^n$). Essentially, a smooth manifold is a space which 'locally' looks like $\mathbb{R}^n$ for some n and has enough nice properties to allow smooth functions to be defined on them. A smooth function is a function which can be differentiated an unbounded number of times at any point in its domain. For example, spheres and toruses are smooth manifolds. Around any point on a sphere or torus, if we 'zoom in', it is as if we were in the plane $\mathbb{R}^2$.

An important object for calculus on manifolds is the Jacobian. Consider a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$. We can decompose f into f_i's defined as $f_i := \pi_i \circ f$, the i^{th} coordinate of f. Then the Jacobian of f at $x \in \mathbb{R}^n$ is

$$\mathcal{J}_x f := \begin{bmatrix} \left.\frac{\partial f_1}{\partial x_1}\right|_x & \cdots & \left.\frac{\partial f_1}{\partial x_n}\right|_x \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial f_m}{\partial x_1}\right|_x & \cdots & \left.\frac{\partial f_m}{\partial x_n}\right|_x \end{bmatrix}$$

where $\frac{\partial f_i}{\partial x_j}|_x$ is the partial derivative of f_i with respect to x_j at x, assuming all such partial derivatives are defined. If each component function f_i is smooth (i.e., $\frac{\partial f_i}{\partial x_{j_1} \ldots x_{j_k}}$ exists for all sequences $j_1, \ldots j_k$), $\mathcal{J}_x f$ is the proper analogue to the single variable derivative. For example, the the chain rule for multivariable functions is $\mathcal{J}_x(g \circ f) = \mathcal{J}_{f(x)} g \times \mathcal{J}_x f$. Furthermore, the best local linear approximation of a smooth function f at a point $a \in \mathbb{R}^n$ is $f(x) \approx f(a) + (\mathcal{J}_a f) \times (x - a)$, where $(\mathcal{J}_a f) \times (x-a)$ is a matrix-vector product.

The definition of a derivative relies on the local behaviour of a function. Because a smooth manifold M is locally Euclidean, we can define an analogous concept. However, the characterisation of a Jacobian via best linear approximation of a smooth function uses the fact that $\mathbb{R}^n$ is a vector space as matrix-vector product is used. The solution involves a new smooth manifold $\mathcal{T}M$ called the *tangent bundle* (full explanation is out of scope). The tangent bundle construction is a functor. Thus, a smooth function $f\colon M \to N$ between smooth manifolds induces a smooth function $\mathcal{T}f\colon \mathcal{T}M \to \mathcal{T}N$ which is the generalised Jacobian of f. The functorality of $\mathcal{T}$ ensures $\mathcal{T}(g \circ f) = \mathcal{T}g \circ \mathcal{T}f$ (compare to the Jacobian chain rule) and is key to my approach.

Some AD implementations can be explained by the compositionality of the Jacobian. The account by Pearlmutter and Siskind uses compositionality to derive the multiple versions of AD, including reverse mode. Functional programming languages such as F# and Haskell currently have useful libraries for reverse mode AD, `DiffSharp` [Baydin et al.] and `ad` respectively.

The denotational semantics of differentiable programming is an active area of research. For example, the differential λ-calculus [Ehrhard and Regnier], originally related to linear substitution, has models in which derivative operation corresponds to a generalised version of differentiation [Blute et al.]. Plotkin's work on first-order differentiable programming[1,2] and Kammar, Staton, and Vákár's work on diffeo-

[1] Gordon Plotkin. *First-order differential programming.* Domains13 talk. 2018

[2] Gordon Plotkin. *Some Principles of Differential Programming Lan-*

logical spaces[3] incorporate rich data types and higher-order types. Elliott's work translates a Haskell program into a Cartesian category with inbuilt AD.

My work builds on the above research in a number of important ways:

- I focus on a simple and tractable first-order language, allowing intuitive understanding by being based off of standard calculus.
- I integrate smooth manifolds at a base level along with a syntax for defining and differentiating functions on manifolds.

3 Results and Contributions

The main preliminary results of my ongoing work are a simple first-order programming language which allows differentiation on manifolds and corresponding denotational semantics. The language includes conditionals, iteration, recursion, simple data types and manifolds as ground types, as well as facilities for defining functions on manifolds. Thus my work provides a guideline on how to implement a differentiable programming language and a framework in which to prove various properties, such as correctness of operator overloading or source-to-source transformation techniques.

In detail, the following types can be constructed

$$\gamma \quad := \quad \mathsf{unit} \mid G \mid \gamma_1 \times \cdots \times \gamma_n \mid \gamma_1 + \cdots + \gamma_n \mid \mathcal{T}\gamma$$

where G varies over smooth manifold ground types such as $\mathbb{R}$ and S^1. Note that the type constructor $\mathcal{T}$ corresponds to exactly to the tangent bundle mentioned previously. The syntax and typing judgements of the language form a standard functional language with the important addition of

$$\frac{\Delta \mid \Gamma, [x_i : \gamma_i] \vdash t : \gamma \qquad \Delta \mid \Gamma \vdash s_i : \mathcal{T}\gamma_i}{\Delta \mid \Gamma \vdash \dfrac{\partial t}{\partial (x_1, \ldots, x_n)} (s_1, \ldots, s_n) : \mathcal{T}\gamma}$$

to allow differentiation (Δ is a function context and Γ is a variable context). The above term differentiates t as a function of $(x_1, \ldots, x_n)$ at $(s_1, \ldots, s_n)$. Additionally, there is an important function $p_\gamma : \mathcal{T}\gamma \to \gamma$ at every type which satisfies

$$\left[\!\!\left[\Delta \mid \Gamma \vdash p\left(\frac{\partial t}{\partial x}(s) \right) : \gamma \right]\!\!\right] = \left[\!\!\left[\Delta \mid \Gamma \vdash \mathsf{let}\ x = p(s)\ \mathsf{in}\ t : \gamma \right]\!\!\right]$$

where $[\![e]\!]$ is the denotation of an expression e.

An important equivalence witnessed by the constructed semantics is what we call the if-rule,

$$\left[\!\!\left[\Delta \mid \Gamma \vdash \frac{\partial(\mathsf{if}\ b\ \mathsf{then}\ v\ \mathsf{else}\ u)}{\partial z}(s) : \mathcal{T}\gamma \right]\!\!\right]$$

$$=$$

$$\left[\!\!\left[\Delta \mid \Gamma \vdash \mathsf{if}\ (\mathsf{let}\ z = p(s)\ \mathsf{in}\ b)\ \mathsf{then}\ \frac{\partial v}{\partial z}(s)\ \ \mathsf{else}\ \frac{\partial u}{\partial z}(s) : \mathcal{T}\gamma \right]\!\!\right]$$

which shows the derivative of the conditional is can be ignored. The if-rule is often broken by implementations of AD, for example the Haskell `ad`. It is a reasonable choice, however, as my semantics force non-standard definitions of functions such as $<: \mathbb{R} \times \mathbb{R} \to \mathsf{Bool} = \mathsf{unit} + \mathsf{unit}$, which is undefined on $\{(x, x) : x \in \mathbb{R}\}$.

An example of an easily expressible program is function fitting of periodic data with known period. Given three periods worth of data, we wish to fit a sine wave or triangle wave to the data. Figure 1 shows a possible instance of the problem with candidate models.

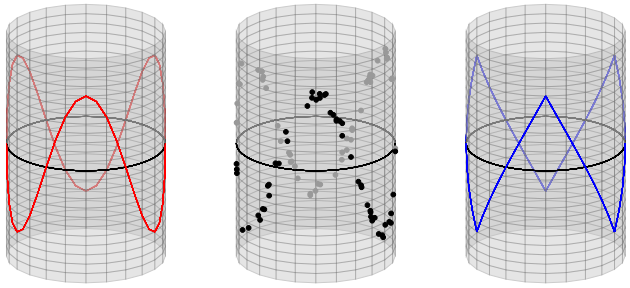

Figure 1: Graphs of $sin(3\theta)$ (left), three periods of random data (center), and $T(3\theta)$ (right) on $\theta \in S^1$

Figure 2 is an implementation of gradient based model fitting. The model is either a sine or triangle wave with parameters changing amplitude a and phase p. The error of a model is the sum of the vertical distance squared between the model and the data. The function $fitter$ calculates the derivative of the error with respect to the model error. The function dM uses syntax and concepts beyond the scope of this abstract, specifically $\lceil\cdot\rceil : (\mathcal{T}S^1 \times \mathcal{T}\mathbb{R} + \mathcal{T}S^1 \times \mathcal{T}\mathbb{R}) \to \mathcal{T}\mathsf{Model}$ and $\odot : S^1 \times \mathbb{R} \to \mathcal{T}S^1, \mathbb{R} \times \mathbb{R} \to \mathcal{T}\mathbb{R}$. Intuitively, $dM(m, 1, 0)$ allows calculation of a derivative w.r.t the a parameter of the models, and $dM(m, 0, 1)$ w.r.t p. The functions $step_{a,p}$ are also not defined, but essentially return a perturbation of their arguments in the direction given by $de_{a,p} : \mathcal{T}\mathbb{R}$.

My work, although first-order, can be extended in a limited way to a higher-order order environment. My language can be embedded in another functional language by way of an EDSL. The host language would then interpret the EDSL via my semantics where equations such as the if-rule could provide re-write rules. However, much is gained. Often, the advantage of higher-order functions is their ability to work on and organise structures. Thus higher-order functions could be used to aid in the construction of EDSL terms, allowing the normal niceties while avoiding the hairy issue of differentiating higher-order functions. For example, the Haskell library `ad` can be viewed as an EDSL, so my work could provide formalism to check against to avoid the type of error found by Manzyuk et al.

4 Approach and Uniqueness

The following is a short technical description of my approach. Currently, I have constructed the denotational semantics using an extension of the category of smooth manifolds with partial (smooth) maps as morphisms, hereafter called $\mathcal{C}$. I then take the countable coproduct completion of C, and by using the open subset inclusion as a system of monics, I apply the partial map construction of Fiore to create a category $p\mathcal{C}$. The category $p\mathcal{C}$ contains countable coproducts and finite partial products, and together they exhibit a partial distributive structure. Following Plotkin, the partiality equips each $p\mathcal{C}$-homset an ω-cpo structure, allowing us to interpret iteration and recursion in $p\mathcal{C}$. I also show that the differential structure

guages. POPL keynote. 2018. URL: `https://popl18.sigplan.org/event/popl-2018-papers-keynote-some-principles-of-differential-programming-languages`

[3] Matthijs Vákár, Ohad Kammar, and Sam Staton. *Diffeological Spaces and Semantics for Differential Programming*. Domains13 talk. 2018

$$
\begin{aligned}
&\mathsf{type}\ \mathsf{Model} = \mathbb{R} \times S^1 + \mathbb{R} \times S^1 \ \text{// Sine + Triangle} \\
&\mathsf{type}\ \mathsf{Data} = (\mathbb{R} \times S^1, \ldots, \mathbb{R} \times S^1) \ \text{// } n\text{-tuple} \\
&\text{// Implementation of model} \\
&\quad impl : \mathsf{Model} \times S^1 \to \mathbb{R} \\
&\mathsf{def}\ impl\,(model, \theta) = \\
&\quad \mathsf{case}\ model\ \mathsf{of} \\
&\qquad \mathsf{inl}\,(a, p) \to a * sin(3 * \theta + p) \\
&\qquad \mathsf{inr}\,(a, p) \to a * T(3 * \theta + p) \\
&\text{// Error of model at data pair} \\
&\quad error : \mathsf{Model} \times (S^1 \times \mathbb{R}) \to \mathbb{R} \\
&\mathsf{def}\ error\,(model, pair) = \\
&\quad \mathsf{let}\ (y, \theta) = pair\ \mathsf{in}\ (y - impl(model, \theta))^2 \\
&\text{// Total error of model across data} \\
&\quad totalError : \mathsf{Model} \times \mathsf{Data} \to \mathbb{R} \\
&\mathsf{def}\ totalError\,(model, data) = \\
&\quad \mathsf{let}\ (p_1, \ldots, p_n) = data \\
&\quad \mathsf{in}\ error(model, p_1) + \cdots + error(model, p_n) \\
&\text{// Allows differentiation of a model} \\
&\quad dM : \mathsf{Model} \times \mathbb{R} \times \mathbb{R} \to \mathcal{T}\mathsf{Model} \\
&\mathsf{def}\ dM\,(model, da, dp) = \\
&\quad \mathsf{case}\ model\ \mathsf{of} \\
&\qquad \mathsf{inl}\,(a, p) \to \lceil \mathsf{inl}\,(a \odot da, p \odot dp) \rceil \\
&\qquad \mathsf{inr}\,(a, p) \to \lceil \mathsf{inr}\,(a \odot da, p \odot dp) \rceil \\
&\text{// Update model given derivatives} \\
&\quad update : \mathsf{Model} \times \mathcal{T}\mathbb{R} \times \mathcal{T}\mathbb{R} \to \mathsf{Model} \\
&\mathsf{def}\ update\,\big(model, de_a, de_p\big) = \\
&\quad \mathsf{case}\ model\ \mathsf{of} \\
&\qquad \mathsf{inl}\,(a, p) \to \mathsf{inl}\,\big(step_a(a, de_a), step_p(p, de_p)\big) \\
&\qquad \mathsf{inr}\,(a, p) \to \mathsf{inr}\,\big(step_a(a, de_a), step_p(p, de_p)\big) \\
&\text{// Makes the model more fit w.r.t. to data} \\
&\quad fitter : \mathsf{Model} \times \mathsf{Data} \to \mathsf{Model} \\
&\mathsf{def}\ fitter\,(model, data) = \\
&\quad \mathsf{let}\ de_a = \frac{\partial(totalError(m, data))}{\partial m}\,\big(dM(model, 1, 0)\big)\ \ \mathsf{in} \\
&\quad \mathsf{let}\ de_p = \frac{\partial(totalError(m, data))}{\partial m}\,\big(dM(model, 0, 1)\big)\ \ \mathsf{in} \\
&\quad update(model, de_a, de_p) \\
&\mathsf{let}\ data = \ll \text{fixed data} \gg\ \ \mathsf{in} \\
&\mathsf{let}\ model_0 = \ll \text{initial model} \gg\ \ \mathsf{in} \\
&\mathsf{let}\ model_1 = fitter(model_0, data)\ \mathsf{in} \\
&\mathsf{let}\ model_2 = fitter(model_1, data)\ \mathsf{in} \\
&fitter(model_2, data)
\end{aligned}
$$

Figure 2: Optimisation program for fitting periodic data. The method used is gradient descent, with three descent steps

of the manifolds, realised through the tangent bundle functor, is compatible with the ω-cpo structure, thus allowing the use of derivatives.

During my ongoing summer research internship and fourth year project, I plan to analyse how the existing methods of implementing reverse mode AD can lead to a categorical version. For example, as the Haskell library `ad` uses stateful side-effects, I will try to add the state monad to my semantics to create a purely categorical account of reverse mode AD. I will also investigate to what extent inductive data types can be added.

The main novelty of my approach is that manifolds are included as ground types and the semantics are specified with respect to tangent bundles. Plotkin's work also includes operational aspects and symbolic differentiation. My work focuses on some of the categorical structure of his work. Elliott's work extracts a high-level structure from AD, implements it in Haskell with categorical ideas, relates it to existing work, and generalises. I will try to connect my approach to his by unifying his high-level structure with my semantics. My work also allows nesting of derivatives internal to the constructed category by virtue of smooth manifold structure of the tangent bundle.

5 Further Directions

I plan to investigate operation semantics for my language and prove it correct with respect to my denotational semantics. In a similar vein, I plan to investigate which methods of implementing AD can be justified with my semantics. For example, the if-rule shifts the use of the derivative construct 'inwards'. An operator overloading implementation could be proven correct given enough similar rules. Additionally, extending existing AD implementations to work with smooth manifolds appears feasible.

My work also generalises beyond automatic differentiation. My work can be seen as a fairly limited extension of work by Cockett and Cruttwell. Their work investigates the interaction between Cartesian tangent categories and Cartesian differential categories, and restriction structures on each (which allowing consideration of partial maps). The first structure is a generalisation of my approach (without coproducts) and the second of Elliott's. Thus I believe my language can be interpreted with little change in a much more general setting. I ultimately hope to create a generalised framework which encompasses many types of automatic differentiation.

References

Baydin, A. G. et al. 'Automatic differentiation in machine learning: a survey'. In: *arXiv preprint arXiv:1502.05767* (2015).

Blute, Richard, Thomas Ehrhard, and Christine Tasson. 'A convenient differential category'. In: *CoRR* abs/1006.3140 (2010). arXiv: 1006.3140. URL: http://arxiv.org/abs/1006.3140.

Cockett, J. R. B. and G. S. H. Cruttwell. 'Differential Structure, Tangent Structure, and SDG'. In: *Applied Categorical Structures* 22.2 (Apr. 2014), pp. 331–417. ISSN: 1572-9095. DOI: 10.1007/s10485-013-9312-0. URL: https://doi.org/10.1007/s10485-013-9312-0.

Ehrhard, Thomas and Laurent Regnier. 'The differential lambda-calculus'. In: *Theoretical Computer Science* 309.1 (2003), pp. 1–41. ISSN: 0304-3975. DOI: https://doi.org/10.1016/S0304-3975(03)00392-X. URL: http://www.sciencedirect.com/science/article/pii/S030439750300392X.

Elliott, Conal. 'The simple essence of automatic differentiation (Differentiable functional programming made easy)'. In: (Apr. 2018). ICFP, to appear.

Fiore, Marcelo P. *Axiomatic Domain Theory in Categories of Partial Maps*. New York, NY, USA: Cambridge University Press, 1996. ISBN: 0-521-57188-x.

Griewank, A. and A. Walther. *Evaluating Derivatives*. Second. Society for Industrial and Applied Mathematics, 2008. DOI: 10.1137/1.9780898717761. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898717761. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898717761.

Manzyuk, Oleksandr et al. *Confusion of Tagged Perturbations in Forward Automatic Differentiation of Higher-Order Functions*. Working Paper arXiv:1211.4892. 2012. URL: http://eprints.maynoothuniversity.ie/6552/.

Pearlmutter, Barak A. and Jeffrey Mark Siskind. 'Reverse-mode AD in a Functional Framework: Lambda the Ultimate Backpropagator'. In: *ACM Trans. Program. Lang. Syst.* 30.2 (Mar. 2008), 7:1–7:36. ISSN: 0164-0925. DOI: 10.1145/1330017.1330018. URL: http://doi.acm.org/10.1145/1330017.1330018.