Evolving the Incremental λ Calculus into a Model of Forward AD*

Robert Kelly† Barak A. Pearlmutter‡ Jeffrey Mark Siskind§

April 2016

Introduction

Formal transformations somehow resembling the usual derivative are surprisingly common in computer science, with two notable examples being derivatives of regular expressions [1] and derivatives of types [2, 3]. A newcomer to this list is the incremental λ-calculus, or ILC, a "theory of changes" that deploys a formal apparatus allowing the automatic generation of efficient update functions which perform incremental computation [4]. An example of this would be using the ILC derivative-like operator $\mathcal{D}$ to alter a function $f : B \to B$, which performs some major reorganization on a database (of type B), into the update function $\mathcal{D}\,f : B \to \Delta B \to \Delta B$. Here ΔB is the type of *changes* to B. So $\mathcal{D}\,f$, given an initial database, maps a *change* to that input database to a change to the output database. This in principle, and as shown in their work also in practice, allows enormous savings when the change to the input is small compared to the size of the input itself. Resemblance to the standard derivative can be exhibited by a simple example

$$\mathcal{D}\,(\lambda\,x\,.\,f\,(g\,x)) \rightsquigarrow (\lambda\,x\,x'\,.\,\mathcal{D}\,f\,(g\,x)\,(\mathcal{D}\,g\,x\,x')) \tag{1a}$$

or

$$\mathcal{D}\,(f \circ g)\,x \rightsquigarrow \mathcal{D}\,f\,(g\,x) \circ \mathcal{D}\,g\,x \tag{1b}$$

which seems suspiciously similar to the familiar Calculus 101 chain rule.

The ILC is not only defined, but given a formal machine-understandable definition—accompanied by mechanically verifiable proofs of various properties, including in particular correctness of various sorts. Here, we show how the ILC can be mutated into propagating tangents, thus serving as a model of Forward Accumulation Mode Automatic Differentiation.[1]

This mutation is done in several steps. These steps can also be applied to the proofs, resulting in machine-checked proofs of the correctness of this model of forward AD.

The Mutagenic Steps

There are two differences between the incremental λ calculus and forward AD. First, *changes* rather than *tangents* are propagated. These changes are elements of *change sets*, and constitute finite (i.e., not infinitesimal) modifications. (For example, a change to a list might consist of swapping the first two elements, and a change to a number might consist of increasing its value by 5.) In numerics, these would be "differences" rather than "differentials", and Δ rather than ∂. Second, the changes are passed as additional arguments instead of being bundled together with primal values. Passing changes in additional arguments makes great sense in the domain of incremental computation, where the whole point of the construction is to partially evaluate a function $\mathcal{D}\,f : \alpha \to \Delta\alpha \to \Delta\beta$ with respect to f's original input, yielding a mapping of changes to changes: $\Delta\alpha \to \Delta\beta$. But in the context of forward AD, we wish to propagate tangent values *in parallel* with primal values, which necessitates both bundling the "new" values with the original ones, and including the original output in the output of the transformed function.

We proceed to eliminate these two differences. This is done in two stages. First, considering only *power series* change sets to the base type $\mathbb{R}$. And second, *uncurrying* the outputs of the derivative operator and causing it to propagate change sets and primal values bundled together all the way through to its output. Truncating the power

*Extended abstract presented at the AD 2016 Conference, Sep 2016, Oxford UK.

†Corresponding Author, Dept of Computer Science, National University of Ireland Maynooth, funded by the Irish Research Council, rob.kelly@cs.nuim.ie

‡Dept of Computer Science, National University of Ireland Maynooth, barak@pearlmutter.net

§School of Electrical and Computer Engineering, Purdue University, qobi@purdue.edu

[1] The approach detailed here stands in contrast to the Simply Typed λ-Calculus of Forward Automatic Differentiation [5]. Aside from some issues with confluence, that work folded together levels of the hierarchy by not distinguishing numeric basis functions which operate on $\mathbb{R}$ from those which are lifted to operate on Dual numbers, while here these are distinguished. Moreover, here we have a framework for machine-readable machine-verified proofs of various correctness and efficiency properties. This approach differs from the Differential Lambda-Calculus [6] in analogous ways: complexity, machine-checked proofs, and explicit segregation of levels of differentiation.

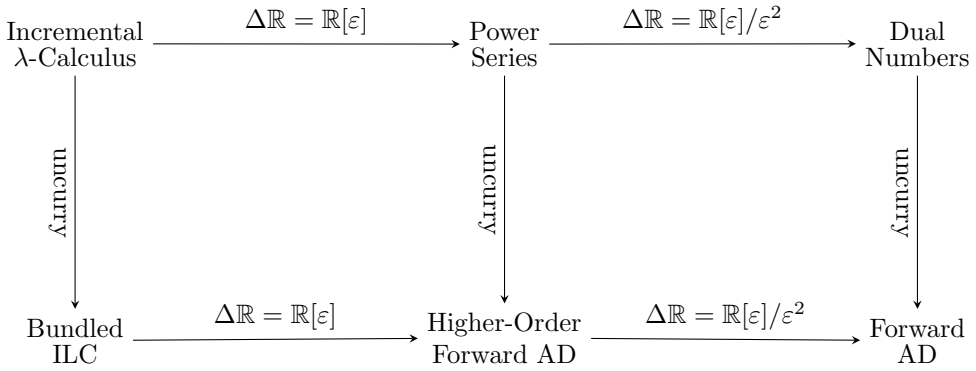

Figure 1: Mutating the Incremental λ-Calculus (ILC) into Forward-Mode Automatic Differentiation (Forward AD).

series changes them into Dual Numbers [7], yielding the familiar Forward AD. A commutative diagram of these steps is shown in Figure 1. The original ILC is in the top left, with relevant changes indicated with transitions to new states or nodes. Each of these edges leads to a different combination of forward AD in the ILC. The *power series* and *uncurry* steps can be taken in either order, so the diagram should commute.

Let us describe these two steps in a bit more detail.

Step One: Power Series

To see how power series change sets are introduced, we note that the ILC allows change sets to be defined for any base type τ. These change sets need only obey a particular set of axioms, which in our context amounts to associativity of addition of real numbers. We constrain ourselves to consider only change sets to reals: the base type $\mathbb{R}$. We then represent these change sets not as differences, but instead as power series (in some variable ε) with a zero constant term. This means that the change set of $x : \mathbb{R}$ is a term of the form $\langle \mathbf{zps}_\varepsilon \rangle$, where

$$\langle \mathbf{zps}_\varepsilon \rangle ::= 0 \mid \varepsilon * \langle \mathbf{ps}_\varepsilon \rangle \tag{2a}$$

$$\langle \mathbf{ps}_\varepsilon \rangle ::= \mathbb{R} \mid \mathbb{R} + \langle \mathbf{zps}_\varepsilon \rangle \tag{2b}$$

$$\Delta\mathbb{R} \equiv \langle \mathbf{zps}_\varepsilon \rangle \tag{2c}$$

For a specific value of ε (possibly subject to conditions of convergence) this would take on a particular numeric value. We further define an operator **coeff** which takes a nonnegative integer index and a power series in ε wrapped in a λ expression, i.e., $(\lambda\varepsilon \,.\, \langle \mathbf{ps}_\varepsilon \rangle)$, and yields the requested coefficient of the given power series.

$$\mathbf{coeff}\ 0\ (\lambda\varepsilon \,.\, r) \rightsquigarrow r \qquad (\text{where } \varepsilon \notin \mathrm{FV}(r)) \tag{3a}$$

$$\mathbf{coeff}\ 0\ (\lambda\varepsilon \,.\, r + \varepsilon * e) \rightsquigarrow r \qquad (\text{where } \varepsilon \notin \mathrm{FV}(r)) \tag{3b}$$

$$\mathbf{coeff}\ 0\ (\lambda\varepsilon \,.\, \varepsilon * e) \rightsquigarrow 0 \tag{3c}$$

$$\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, r + \varepsilon * e) \rightsquigarrow \mathbf{coeff}\ (i-1)\ (\lambda\varepsilon \,.\, e) \qquad (\text{where } i > 0 \text{ and } \varepsilon \notin \mathrm{FV}(r)) \tag{3d}$$

$$\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, \varepsilon * e) \rightsquigarrow \mathbf{coeff}\ (i-1)\ (\lambda\varepsilon \,.\, e) \qquad (\text{where } i > 0) \tag{3e}$$

For instance,

$$\mathbf{coeff}\ 2\ (\lambda\varepsilon \,.\, 0.1 + \varepsilon * (0.2 + \varepsilon * (0.3 + \varepsilon * (0.4 + \varepsilon * (0.5 + \cdots))))) \rightsquigarrow 0.3$$

Useful properties of such a change set are straightforward to establish: closure under the derivatives of the numeric basis functions, and dependence during such operators of coefficients only on coefficients of the same or lower order. The first property is necessary for consistency, while the second allows these power series to be truncated at ε^2, thus yielding the tangents of standard forward AD. With this machinery, we could define the familiar derivative $\mathbf{diff} : (\mathbb{R} \to \mathbb{R}) \to (\mathbb{R} \to \mathbb{R})$, for instance $\mathbf{diff}\ \sin = \cos$, as

$$\mathbf{diff}\ f\ x \equiv \mathbf{coeff}\ 1\ (\lambda\varepsilon \,.\, (\mathcal{D}\ f\ x\ (\varepsilon * 1))) \tag{4}$$

By defining **coeff** to distribute over algebraic datatypes

$$\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, \mathbf{Constructor}\ e_1\ \cdots\ e_n) \rightsquigarrow \mathbf{Constructor}\ (\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, e_1))\ \cdots\ (\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, e_n)) \tag{5a}$$

and post-compose over functions

$$\mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, (\lambda x \,.\, e)) \rightsquigarrow (\lambda x \,.\, \mathbf{coeff}\ i\ (\lambda\varepsilon \,.\, e)) \qquad (\text{where } x \neq \varepsilon) \tag{5b}$$

this machinery can find directional derivatives of functions with non-scalar output, including Church-encoded output.

In this formulation, the tagging necessary to distinguish distinct nested invocations of derivative-taking operators [8, 9] is handled by the standard λ-calculus mechanisms for avoiding variable capture during β-substitution, e.g., α-renaming.

Step Two: Uncurrying and Bundling

The second step is uncurrying arguments, and bundling the output. We need to change the type of the derivative operator from

$$\mathcal{D} : (t_1 \to t_2 \to \cdots \to t_n \to u) \to (t_1 \to \Delta t_1 \to t_2 \to \Delta t_2 \to \cdots \to t_n \to \Delta t_n \to \Delta u) \tag{6}$$

to

$$\hat{\mathcal{D}} : (t_1 \to t_2 \to \cdots \to t_n \to u) \to (Ft_1 \to Ft_2 \to \cdots \to Ft_n \to Fu) \tag{7}$$

where Ft is isomorphic to $t \times \Delta t$, a primal value bundled with its change set. If we define $F(t_1 \to t_2) = Ft_1 \to Ft_2$ then this yields a simpler type signature,

$$\hat{\mathcal{D}} : t \to Ft \tag{8}$$

The mechanics of this change are straightforward, requiring that the ILC reductions be modified to take the new shape. Note that, thus uncurried and carrying primal and change set values in tandem, the chain rules of Equation 1 are simplified: $\hat{\mathcal{D}}\ (f \circ g) \rightsquigarrow \hat{\mathcal{D}}\ f \circ \hat{\mathcal{D}}\ g$.

Acknowledgments

This work was supported, in part, by Science Foundation Ireland grant 09/IN.1/I2637 and by NSF grant 1522954-IIS. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] Janusz A. Brzozowski. Derivatives of regular expressions. *Journal of the ACM (JACM)*, 11(4):481, 1964.

[2] Conor McBride. The derivative of a regular type is its type of one-hole contexts, 2001. URL http://strictlypositive.org/diff.pdf. Available online.

[3] Michael Abbott, Neil Ghani, Thorsten Altenkirch, and Conor Mcbride. ∂ for data: Differentiating data structures. *Fundamenta Informaticae*, 65(1-2):1–28, August 2004. URL http://strictlypositive.org/dfordata.pdf.

[4] Yufei Cai, Paolo G. Giarrusso, Tillmann Rendel, and Klaus Ostermann. A theory of changes for higher-order languages: Incrementalizing λ-calculi by static differentiation. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 145–55, 2014. doi: 10.1145/2594291.2594304. URL https://inc-lc.github.io/resources/pldi14-ilc-author-final.pdf. See arXiv:1312.0658.

[5] Oleksandr Manzyuk. A simply typed λ-calculus of forward automatic differentiation. In *Mathematical Foundations of Programming Semantics Twenty-eighth Annual Conference*, pages 259–73, Bath, UK, June 6–9 2012. URL http://dauns.math.tulane.edu/~mfps/mfps28proc.pdf.

[6] Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3):1–41, December 2003.

[7] William Kingdon Clifford. Preliminary sketch of bi-quaternions. *Proceedings of the London Mathematical Society*, 4:381–95, 1873.

[8] Jeffrey Mark Siskind and Barak A. Pearlmutter. Nesting forward-mode AD in a functional framework. *Higher-Order and Symbolic Computation*, 21(4):361–76, 2008. doi: 10.1007/s10990-008-9037-1.

[9] Oleksandr Manzyuk, Barak A. Pearlmutter, Alexey Andreyevich Radul, David R. Rush, and Jeffrey Mark Siskind. Confusion of tagged perturbations in forward automatic differentiation of higher-order functions. *Higher-Order and Symbolic Computation*, 2015. To appear. See also arXiv:1211.4892.