Differential Lambda Calculus

Christopher Hittner and Thomas Pyle
We pledge our honor that we have abided by the Stevens Honor System.
November 26, 2018
In this presentation, we will discuss research into the differential calculus, as well as demonstrate an implementation of a language that supports these features. We will also discuss the type system for such a language.
Differential Lambda Calculus
Motivation

We seek to extend the Lambda Calculus to support the use of differentiation; given a function, take its derivative.

We also seek to design a type system for this language.

We will consider the Lambda-Mu ($\lambda\mu$) Calculus developed by Thomas Ehrhard and Laurent Regnier.
Terms

We extend the grammar with the following terms:

\[ M ::= \ldots | D_i M \cdot N | \frac{\partial}{\partial x} M \cdot N \]

These enable us to differentiate \( M \) with respect to its \( i \)th argument given \( N \), or by a given variable name.

- \( D_1 \lambda x.x \cdot 1 \) is a term similar to finding \( \frac{d}{dx} x \).
- \( D_1 \lambda x.x \) is not a term because it must be associated with an application.
\( \lambda \mu \) requires several units of operational semantics:

\[
D_1 \lambda x. M \cdot N \rightarrow \lambda x. \frac{\partial M}{\partial x} \cdot N
\]

\[
D_{i+1} \lambda x. M \cdot N \rightarrow \lambda x. (D_i M \cdot N)
\]

\[
\delta_{x,y} = 1 \text{ if } x = y \text{ else } 0
\]

\[
\frac{\partial y}{\partial x} \cdot P \rightarrow \delta_{x,y} P
\]

\[
n \in \mathbb{R}
\]

\[
\frac{\partial}{\partial x} n \cdot P \rightarrow 0
\]
For example, consider \( f(x) = x^2 + 2 \). Clearly, \( \frac{df}{dx} = 2x \).

Equivalently, consider \( f = \lambda x. x^2 + 2 \). We can differentiate it like so:

\[
\begin{align*}
D_1 \lambda x. x^2 + 2 & \cdot 1 \\
\rightarrow \lambda x. \frac{d}{dx} (x^2 + 2) & \cdot 1 \\
\rightarrow \lambda x. 2x & \cdot 1 \\
\rightarrow \lambda x. 2x
\end{align*}
\]
On top of the default typed lambda calculus rules, we have little to add to the type system:

\[
\Gamma \vdash M : \sigma_1, \sigma_2, \ldots, \sigma_i \rightarrow \tau \quad \Gamma \vdash N : \sigma_i \\
\Gamma \vdash D_i M \cdot N : \sigma_1, \sigma_2, \ldots, \sigma_i \rightarrow \tau \\
\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \sigma \quad a, b \in \mathbb{R} \\
\Gamma \vdash aM + bN : \sigma
\]
Consider the previous example. The typing tree is roughly like so:

\[
\emptyset \vdash \lambda x. x^2 + 2 : \mathbb{R} \rightarrow \mathbb{R} \quad \emptyset \vdash 1 : \mathbb{R}
\]

\[
\emptyset \vdash D_1 \lambda x. x^2 + 2 \cdot 1 : \mathbb{R} \rightarrow \mathbb{R}
\]
The type system we’ve seen here could be thought of as obvious; the derivative of a $\mathbb{R} \rightarrow \mathbb{R}$ is clearly the same type.

Hence, we wish to explore typing for more complex means of differentiation.
Lomda
Motivation

The original goal of the project was to create a language that allowed for differentiation of expressions.

Here, we present Lomda (Λ) as an interpreted programming language that supports differentiation across multiple types, from ints and reals to tuples and lists (presented as vectors and matrices).
Lomda has a sizable range of features, including maps and folds. Their operational semantics are shown below:

map F over $[M_1, M_2, \ldots, M_n] \rightarrow [F(M_1), F(M_2), \ldots, F(M_n)]$

fold $[M_1, \ldots]$ into $F$ from $B \rightarrow$ fold $[M_2, \ldots]$ into $F$ from $F(B, M_1)$

fold $[]$ into $F$ from $B \rightarrow B$
To define our type system, we will first define our types as follows:

\[
\sigma ::= \mathbb{Z} \mid \mathbb{R} \mid \mathbb{B} \mid \Sigma^* \mid [\sigma] \mid \sigma \rightarrow \sigma \mid \sigma \ast \sigma \mid \frac{d\sigma}{d\sigma}
\]

Where \(\mathbb{Z} <: \mathbb{R}\)

We can see that most of these types are reasonable to have in the language (\(\mathbb{R}\) is reals, \(\mathbb{B}\) booleans, etc.).

What about \(\frac{d\sigma}{d\tau}\)?

We will see this in a bit
On top of the basic rules, we define some interesting cases:

\[
\frac{\Gamma \vdash F : \sigma \rightarrow \tau \quad \Gamma \vdash L : [\sigma]}{\Gamma \vdash \text{map } F \text{ over } L : [\tau]} \quad \text{T-MAP}
\]

\[
\frac{\Gamma \vdash L : [\tau] \quad \Gamma \vdash F : \sigma \rightarrow \tau \rightarrow \sigma \quad \Gamma \vdash B : \sigma}{\Gamma \vdash \text{fold } L \text{ into } F \text{ from } B : \sigma} \quad \text{T-FOLD}
\]
We need to define typing operations for differentiation. We define the following typing judgment:

\[
\frac{\Gamma \vdash x : \tau \land \Gamma \vdash M : \sigma}{\Gamma \vdash \frac{d}{dx} M : \frac{d\sigma}{d\tau}} \quad \text{T-DERIV}
\]

Again, what is \( \frac{d\sigma}{d\tau} \)?
We propose the following reductions of $\frac{d\sigma}{d\tau}$:

1. $\frac{d\mathbb{Z}}{d\mathbb{Z}} = \frac{d\mathbb{Z}}{d\mathbb{R}} = \mathbb{Z}$
2. $\frac{d\mathbb{R}}{d\mathbb{Z}} = \frac{d\mathbb{R}}{d\mathbb{R}} = \mathbb{R}$
3. $\frac{d}{d\tau}(\sigma_1 \ast \sigma_2) = (\frac{d}{d\tau} \sigma_1 \ast \frac{d}{d\tau} \sigma_2)$
4. $\frac{d}{d\tau} [\sigma] = [\frac{d\sigma}{d\tau}]$
5. Let $\rho \in \{\mathbb{Z}, \mathbb{R}\}$. Then,
   - $\frac{d\rho}{d[\sigma]} = [\frac{d\rho}{d\sigma}]$
   - $\frac{d\rho}{d(\sigma \ast \tau)} = (\frac{d\rho}{d\sigma} \ast \frac{d\rho}{d\tau})$
6. Let $\rho \in \{\mathbb{B}, \Sigma^*\}$. Then, $\frac{d\rho}{d\sigma}$ and $\frac{d\sigma}{d\rho}$ are undefined.
Examples

Lomda 0.1.0
Compiled May 4 2018 @ 20:17:51
The following configuration is in use:
optimize: 0 (default: 0)
use_types: 2 (default: 0)
verbosity: 0 (default: 0)
werror: 1 (default: 0)
Enter a program and press <enter> to execute, or one of the following:
'exit' - exit the interpreter
'q/quit' - exit the interpreter
>d/dx (x) -> x+1
a/a d/dx (lambda (x) x + 1) : (a -> (da/da + dZ/da))
x.dx/dx + d/dx (1) |
> let A = [[1, 0], [0, 1]], x = [1, 1]; d/dx A*x
  let A = [[1, 0], [0, 1]], x = [1, 1]; d/dx (A * x) : [[Z]]
  [1, 0, [0, 1]]
> let x = [1,2,3]; d/dx map (x) -> x*x over x
Z/b, Z/a  let x = [1, 2, 3]; d/dx (map lambda (x) x * x over x) : [[Z]]
  [2, 0, 0, [0, 4, 0], [0, 0, 6]]
> q
Under the Lomda system, we can handle more interesting types and their derivatives.

Clearly, a derivative’s type is not necessarily the same as the type of the source.
References

Thomas Ehrhard and Laurent Regnier.  
**The differential lambda-calculus.**  

Christopher Hittner.  
**Lomda repository.**  
https://github.com/themaddoctor1/Lomda/.

Oleksandr Manzyuk.  
**Tangent bundles in differential lambda-categories.**  
*MFPS*, 2013.

Lionel Vaux.  
**The differential lambda-mu-calculus.**  