Implementing Tensor Calc functions in Diderot

Charisee Chiw

April 1, 2018

1 Overview

Shorthand for code .
Path = https://github.com/cchiw
Exs = *Path*/latte/
DATm = *Path*/DATm
Diderot_Dev = *Path*/Diderot-Dev
Vis15 = vis15

Shorthand for Text .
[Doc] = *Exs*/writeup/paper.pdf
[dissertation] = Chiw'17 dissertation
[AST paper] = Chiw'17 ICSE-AST paper

2 New and extended syntax

Functionality: Closed formed expressions: polynomials, ...
Syntax:

- Define an ofield with ofield#k(d)[β]
- Declare a polynomial with cfexp(exp,v_0,...) and evaluate that oField with "inst()"
  - "exp" is the core computation that includes operators on and between variables
  - "v_0" is the variable we differentiate in respect to. We accept 1-3 "v" terms
  - cfexp(): tensor[β] $\times$ tensor[i] ... $\rightarrow$ ofield#k(d)[β]
  - inst(): ofield#k(d)[α] $\times$ tensor[i] $\cdots \rightarrow$ tensor[α]

Branch: *Diderot_Dev*
Text: see [Doc]
Issues: :) Many :)

- Need to define/initiate all variables before cfexp() is called.
- OField type doesn't describe types for multiple inputs, need to change typechecker
- not user-proof
- plenty of examples but something this new needs more extensive testing

Examples:exs/dfn_cfe

- Base Case
  $[f_v = v]$: X1/v.diderot, $[f_v = v \bullet v]$: X1/vv.diderot, $[f_v = (v \bullet v) * v]$: X1/vvv.diderot, and $[f_s = s * s * s]$: X2/sss.diderot
- Multiple variables in core computation and differentiate in respect to one variable
  $[f_x = (1 - \frac{|x|}{y})^4]$ Sphere : X3/sphere.diderot, $[f_x = (x - cutPos) \bullet curNorm]$ clip : X3/clip.diderot
  $[f_x = (\frac{1}{\frac{|x|}{y}} * (1 - \frac{|x|}{y}))^3]$ Circle : X4/circle.diderot, and $[f_x = (1 - |x|/y)^4]$ Enr : X4/enr.diderot
- Multiple variables in core computation and differentiate in respect to multiple variables
  $[f_{sv} = s * v]$: X5/m1.diderot, $[f_{svx} = s * v + x]$: X5/m2.diderot, and $[f_{abc} = a^3bc^2]$: X5/m3.diderot

3 Defining fields with closed form expressions

3.1 Mini-Tutorial

It is natural to define a function with an expression.

$$F(x) = x^2$$

In the surface language we added function cfexp() where the first argument exp is an expression and the second x is an input variable.

```
tensor [] exp = x²;
ofield #2(2)[] F = cfexp(exp,x); //define F with variable x
tensor [2] v = [3,7];
tensor [] outF = inst(F,v); //evaluate F with argument v
```

We commonly refer to the right-hand-side to variable F as a cfexp (closed-form expression). The cfe is created with variable x, but is actually evaluated with v. The cfexp is a Diderot "ofield" type. The user can apply other operators on the cfexp including differentiation.

```
ofield #1(2)[2] GF = ∇F;
tensor [] outGF = inst(GF,v);
```

The differentiation of the cfexp is computed in respect to the variable v. We illustrate the expected structure below:

$$outF = F(v) = \; v_0^2 + v_1^2 \qquad outF = \nabla F(v) = \begin{bmatrix} 2 * v_0 \\ 2 * v_1 \end{bmatrix}$$

A function can be defined with multiple variables.

$$F(a, b) = a + b$$

and similarly a cfexp can be defined with multiple variables

```
real[] a = 1; real b = 7;
tensor [] exp = a+b;
ofield#k(d)[] G = cfexp(exp,a);
ofield#k(d)[] H = cfexp(exp,a,b);
ofield#k(d)[] I = cfexp(exp,b);
```

The distinction between G and H is that differentiation is applied in respect to either one or two variable, respectively.

$$\nabla G_a = \nabla a + b \qquad \text{and} \qquad \nabla H_{ab} = \nabla a + \nabla b \qquad \text{and} \qquad \nabla I_b = a + \nabla b$$

3.2 Implementation

Representation As an ongoing example, consider the function $F(x) = x^3$. We can define this function with the following syntax:

```
real p;
ofield #1(2)[] F = cfexp(p³,p);
tensor [] out = F(pos);
```

The language is translated inside the compiler as:

$$\underset{init}{\Longrightarrow} \begin{array}{l} \text{F} = \lambda()\big\langle PolyWrap_p(p^3)\big\rangle() \\ \text{out} = \lambda(F,x)\langle F(x)\rangle(\text{F,x}) \end{array} \qquad \underset{subst}{\Longrightarrow} \text{out} = \lambda(x)\big\langle PolyWrap_p(p^3)(x)\big\rangle(\text{x})$$

Replace polynomial variable .
The variable p represents a vector of length 2, where p= [X,Y].
The term P_0 represents the 0th component of the vector, or X. In the following, we will use the terms X and Y, in place of P_0 and P_1.

The polynomial variable is instantiated with the position.
$\Rightarrow$ out=$\lambda(p)\langle e\rangle$(x) where e=$p * p * p$.
The EIN term (p) is replaced with an EIN term that represents the vector components. In the 2-d case there are two terms

indexed with a constant index in $p = X\delta_{0i} + Y\delta_{1i}$

Occurrances for P are replaced inside the expression:
$\to (P_0\delta_{0i} + P_1\delta_{1i}) * (P_0\delta_{0i} + P_1\delta_{1i}) * (P_0\delta_{0i} + P_1\delta_{1i})$.
$= (X\delta_{0i} + Y\delta_{1i}) * (X\delta_{0i} + Y\delta_{1i}) * (X\delta_{0i} + Y\delta_{1i})$.

Normalization .
Similar terms are collected:

$$P_0 * P_0 \to {P_0}^2 \quad \text{or} \quad X * X \to X^2$$

The differentiation operator is distributed over the EIN term, as usual, and pushed to a polynomial term

$$\frac{\partial}{\partial x_i}({P_0}^2 + e) \to \frac{\partial}{\partial x_i}{P_0}^2 + \frac{\partial}{\partial x_i}e \quad \text{or} \quad \frac{\partial}{\partial x_i}(X^2 + e) \to \frac{\partial}{\partial x_i}X^2 + \frac{\partial}{\partial x_i}e$$

Evaluation .
During the evaluation the variable index in a differentiation operator is bound to a number. An EIN term such as $\frac{\partial}{\partial x_i}{P_c}^n$ is evaluated.
When i and c are both 0:

$$\frac{\partial}{\partial x_0}X \to 1 \quad \frac{\partial}{\partial x_0}X^2 \to 2 * X \quad \frac{\partial}{\partial x_0}X^3 \to 3 * X^2$$

When i and c are both 1:

$$\frac{\partial}{\partial x_1}Y \to 1 \quad \frac{\partial}{\partial x_1}Y^2 \to 2 * Y \quad \frac{\partial}{\partial x_1}Y^3 \to 3 * Y^2$$

When i and c are not the same

$$\frac{\partial}{\partial x_1}X^n \to 0 \quad \frac{\partial}{\partial x_1}Y^n \to 0$$