# Implementing Tensor Calc functions in Diderot: Composition

Charisee Chiw

March 28, 2018

**Shorthand** *Diderot_Dev* = https://github.com/cchiw/Diderot-Dev
*Exs* = https://github.com/cchiw/latte/
*Doc* = *Exs*/writeup/paper.pdf
*dissertation* = Chiw's dissertation

## 1 Overview

Functionality: **Field Composition**
Syntax: "compose" and "∘"
field#k($d_1$)[$\alpha$]× field#k($d_0$)[$d_1$] → field#k($d_1$)[$\alpha$]
Branch: *Diderot_Dev*
Text: EIN IR design, rewriting rules, and resolved bugs listed in *Doc*
Issues: none
Examples: Check out programs in *Exs*/composition
Notes:

- *function name syntax* can use "compose" (*Path*/B*/observ.diderot)
- *unicode syntax* can use "∘"(*Path*/X1/observ.diderot,*Path*/X2/t.diderot)
- *chains composition operator* can apply composition operator to other operators and to itself (*Path*/X2/*.diderot)
- *tested with DATm* command: python3 cte.py 1 36
- *solved bugs* Copies of the programs with solved bugs are in *Path*/B*. You can recreate a solved bug with DATm command "python3 cte.py 4 36 17 10 13"

## 2 Design and Implementation

```
field#k(d0)[σ] F0;
field#k(d1)[d0] F1;
field#k(d1)[σ] H = F0 ∘ F1;
tensor[d1] pos;
tensor[σ] out = H(pos);
```

**Representation** We represent the field composition operator with the generic EIN operator.

$$\xrightarrow[init]{} H = \lambda(F,G)\Big\langle F_\alpha \circ [\langle G_\beta\rangle_{\hat{\beta}}]\Big\rangle_{\hat{\alpha}}(F0,F1) \qquad \text{where } \hat{\alpha}=\sigma \text{ and } \hat{\beta}=[d0]$$

The field terms F and G represent fields in the composition. F and G have separate index spaces. $F$ is bound by $\alpha$ and $G$ is bound by $\beta$.

**Normalization** The probe of a composition $(e_1 \circ [\langle e_2\rangle_{\hat{\beta}}])(x)$ is rewritten depending on the structure of the outer term $e_1$. When the outer term is a constant the result does not depend on the composition operation.

$$(c \circ e_c)(x) \xrightarrow[rule]{} c$$

Similarily, when the outer term is a non-field:

$$\begin{array}{llll} (\delta_\alpha \circ e_c)(x) & \xrightarrow[rule]{} \delta_\alpha & (\mathcal{E}_\alpha \circ e_c)(x) & \xrightarrow[rule]{} \mathcal{E}_\alpha \\ (Z_\alpha \circ e_c)(x) & \xrightarrow[rule]{} Z_\alpha & (\mathbf{lift}_d(e) \circ e_c)(x) & \xrightarrow[rule]{} e \end{array}$$

The probe operator is pushed past arithmetic operators:

$$\begin{array}{lll} (\odot_1 e \circ e_c)(x) & \xrightarrow[rule]{} & \odot_1\,(e \circ e_c)(x) \\ (\sum_{\hat{\alpha}} e \circ e_c)(x) & \xrightarrow[rule]{} & \sum_{\hat{\alpha}}(e \circ e_c)(x) \end{array}$$

The probe is distributed:

$$\begin{array}{lll} ((e_a - e_b) \circ e_c)(x) & \xrightarrow[rule]{} & (e_a \circ e_c)(x) - (e_b \circ e_c)(x) \\ ((e_a * e_b * e_s) \circ e_c)(x) & \xrightarrow[rule]{} & (e_a \circ e_c)(x) * (e_b \circ e_c)(x) * (e_s \circ e_c)(x) \end{array}$$

The derivative of a field composition is applied by using the chain rule.

$$\nabla(F \circ G) \longrightarrow_{direct-style} (\nabla F \circ G) \bullet (\nabla G)$$

The derivative of a field composition of two fields is represented in the EIN IR as

$$\nabla_j (F_\alpha \circ [\langle G_{i\beta} \rangle_{\hat{i\beta}}]) \xrightarrow[rule]{} \sum_{\hat{k}} ((\nabla_k F_\alpha \circ [\langle G_{i\beta} \rangle_{\hat{i\beta}}]) * (\nabla_j G_{k\beta}))$$

Generally we use the rewrite rule to apply the rewrite between two EIN expressions:

$$\nabla_j (e_1 \circ [\langle e_2 \rangle_{\hat{i\beta}}]) \xrightarrow[rule]{} \sum_{\hat{k}} ((\nabla_k e_1 \circ [\langle e_2 \rangle_{\hat{i\beta}}]) * (\nabla_j e_{2[i/k]}))$$

Flatten composition operator

$$(a \circ [\langle b \rangle_{\hat{m}}]) \circ e_c \xrightarrow[rule]{} a \circ [\langle b \rangle_{\hat{m}}, e_c]$$

$$a \circ [\langle b \circ e_c \rangle_{\hat{m}}] \xrightarrow[rule]{} a \circ [\langle b \rangle_{\hat{m}}, e_c]$$

**Split** After being normalized the probed composition operator is split into several probes.

$$\text{out} = \lambda F, G, x \Big\langle F_\alpha \circ [\langle G_\beta \rangle_{\hat{\beta}}](x) \Big\rangle_\alpha (F0, F1, x) \xRightarrow[split]{} \begin{array}{rl} \text{t}_0 = & \lambda G, x \langle G_\beta(x) \rangle_{\hat{\beta}} (F1, x) \\ \text{out} = & \lambda F, x \langle F_\alpha(x) \rangle_\alpha (F0, \text{t}_0) \end{array}$$

# 3 Documented (solved) bugs

**Comp-B1** Mistake in index scope when using substitution.

```
field#k(2)[2,2] F0;
field#k(2)[2] F1;
field#k(2)[2] F2;
field#k(2)[2,2] G = (F0 ∘ F1) • F2;
```

There was a mistake in the substitution method. The scope of the composition indices were handled incorrectly. The following is the expected and observed representation of the computation in the EIN IR.
Expected: $e = \sum_{\hat{j}} A_{ij} \circ [\langle B_i \rangle_{\hat{\beta}}] * C_j$
Observed: $e = \sum_{\hat{k}} A_{ik} \circ [\langle B_i \rangle_{\hat{\beta}}] * C_j$
in $\lambda(A, B, C)\langle e \rangle_\beta$(F0,F1,F2).

DATm Command: python3 cte.py 4 36 17 10 13

**Comp-B2** Missing cases in split method.
Probes of a composition are handled differently before reconstruction.
$\sum F(x)$ and $\sum(\text{Comp}(F, G, -))(x)$.
Missing case in method leads to a compile time error. Additionally (Comp(Comp -)-)
label:-

**Comp-B3** Differentiate a composition
The jacobian of a field composition:

```
field#k(d1)[d] F0;
field#k(d)[d1] F1;
field#k(d1)[d,d1] G = ∇⊗ (F0 ∘ F1);
```

is represented as $\langle \nabla_j(\text{Comp}(A_i, B_i, i)) \rangle_{ij}$
In accordance with the chain rule ( (f $\circ$ g)' = (f' $\circ$ g) $\cdot$ g') the rewriting system multiplies the inner derivative (g') with a new composition operation (f' $\circ$ g). In practice, the implementation does a point-wise multiplication when it should do an inner product.
Expected: $\sum_{\hat{k}}(\nabla_k A_i \circ [\langle B_i \rangle_{\hat{\beta}}] * \nabla_j G_k)$
Observed: $\nabla_j A_\beta \circ [\langle B_i \rangle_{\hat{i}}] * (\nabla_j G_i)$
in $\lambda(A, B)\langle e \rangle_{ij}$(F0,F1).