LoVe team seminar

Typing Differentiable Programming

Marie Kerjean

CNRS & LIPN, Université Paris 13

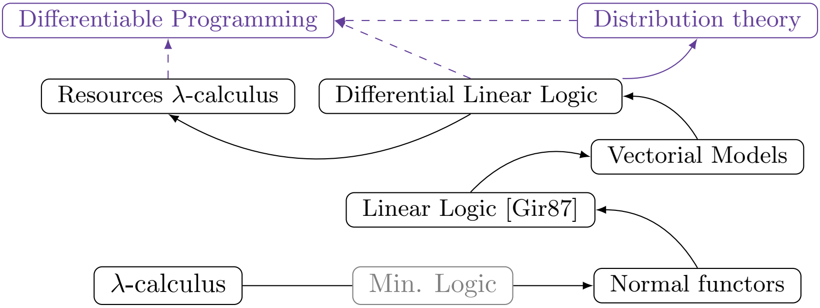
February 2021

Work in Progress with Pierre-Marie Pédrot.

Curry-Howard for semantics

The syntax mirrors the semantics.

Programs	**Logic**	**Semantics**
`fun (x:A)-> (t:B)`	Proof of $A \vdash B$	$f : A \to B$.
Types	Formulas	Objects
Execution	Cut-elimination	Equality



Differentiable programming

A new area triggered by the advances of deep learning algorithms on neural networks, it tries to attach two very old domains:

- Automatic Differentiation.
- λ-calculus.

Goal: Exploring modular way to express reverse differentiation in functional programming languages:

- Abadi & Plotkin, POPL20. (traces and big-step semantics)
- Brunel & Mazza & Pagani, POPL20. More on that latter
- Elliot, ICFP18, (compositional differentiation)
- Wang and al., ICFP 19, (delimited continuations)
- Interactions with probabilistic programming...

Automatic Differentiation

How does one compute the differentiation of an algebraic expression, computed as a sequence of elementary operations ?

E.g. : $z = y + cos(x^2)$
$$\begin{array}{ll} x_1 = x_0^2 & x_1' = 2x_0x_0' \\ x_2 = cos(x_1) & x_2' = -x_0' sin(x_0) \\ z = y + x_2 & z' = y' + 2x_2x_2' \end{array}$$

The computation of the final results requires the computation of the derivative of all partial computation. But in which order ?

Forward Mode differentiation [Wengert, 1964]
$(x_1, x_1') \to (x_2, x_2') \to (z, z')$.
Reverse Mode differentiation: [Speelpenning, Rall, 1980s]
$x_1 \to x_2 \to z \to z' \to x_2' \to x_1'$ *while keeping formal the unknown derivative*.

AD from a higher-order functional point of view

$$\mathrm{D}_u(f \circ g) = D_{g(u)} f \circ D_u(f)$$

- **Forward Mode differentiation :**
  $g(u) \to D_u g \to f(g(u)) \to D_{g(u)} f \to D_{g(u)} f \circ D_u(f)$.
- **Reverse Mode differentiation:**
  $g(u) \to f(g(u)) \to D_{g(u)} f \to D_u g \to D_{g(u)} f \circ D_u(f)$

The choice of an algorithm is due to complexity considerations:

- Forward mode for $f : \mathbb{R} \to \mathbb{R}^n$.
- Reverse mode for $f : \mathbb{R}^n \to \mathbb{R}$

$\rightsquigarrow$ Differentiation is about *linearizing* a function/program. Some people have a very specific idea of what a *linear program* or a *linear type* should be.

1. Reverse-Mode Differentation as a Logical transformation

2. Calculus and differentiation typed by Linear Logic

Linear logic : the type of function

Usual implication

Linear implication

Linear decomposition of the implication

$$A \Rightarrow B = !\, A \multimap B$$
$$\mathcal{C}^{\infty}(A, B) \simeq \mathcal{L}(!A, B)$$

A proof is linear when it uses only once its hypothesis A.

A linear negation

From $\neg A = A \Rightarrow \bot$ to $A^{\bot} = A \multimap \bot$: an involutive linear negation interpreted by linear forms.

$$[\![A^{\bot}]\!] = \mathcal{L}([\![A]\!], \mathbb{R})$$

Mazza and Pagani [POPL2020]

Key Idea

Reverse derivatives are typed by linear negation.

Consider $f : \mathbb{R}^n \to \mathbb{R}$ a function variable.

$$\overleftarrow{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^\perp \to\in \mathbb{R} \times \mathbb{R}^{n\perp} \\ \quad (a, x) \mapsto (f(a), (v \mapsto x \cdot (\mathrm{D}_a f \cdot v)) \end{cases}$$

This leads to a **compositional reverse derivative** transformation over the *linear substitution calculus*, and proven complexity results.

$$A, B, C ::= R \mid A \times B \mid A \to B | R^{\perp_d}$$

$$t, u := x \mid x^! \mid \lambda x.t \mid (t)u \mid t[x^{()!} := u] \mid < t, u > \mid t + u ...$$

The real inventor of deep learning

(I'm joking)

A Dialectica Transformation

- Gödel Dialectica transformation [1958] : a translation from intuitionistic arithmetic to a finite type extension of primitive recursive arithmetic.

$$A \rightsquigarrow \exists u : \mathbb{W}(A), \forall x : \mathbb{C}(A), A^D[u, x]$$

- DePaiva [1991]: the linearized Dialectica translation operates on Linear Logic (types) and λ-calculus (terms).

- Pedrot [2014] A *computational* Dialectica translation preserving β-equivalence, via the introduction of an "abstract multiset constructor" on types on the target.

Pédrot's Dialectica Transformation

$\mathfrak{M}\, A$ is endowed with a sum $(\circledast, \emptyset)$ and a monadic structure $(\{\_\}, \gg\!=)$.

Types:

$$\begin{array}{lll} \mathbb{W}(\alpha) & := & \alpha_{\mathbb{W}} \qquad\qquad \mathbb{C}(\alpha) := \alpha_{\mathbb{C}} \\ \mathbb{W}(A \Rightarrow B) & := & (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathfrak{M}\,\mathbb{C}(A)) \\ \mathbb{C}(A \Rightarrow B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \end{array}$$

Terms:

$$\begin{array}{lllllll} x_x & := & \lambda\pi.\,\{\pi\} & & x^\bullet & := & x \\ x_y & := & \lambda\pi.\,\varnothing \;\; \text{if } x \neq y & & (\lambda x.\,t)^\bullet & := & (\lambda x.\,t^\bullet, \lambda x\pi.\,t_x\;\pi) \\ (\lambda x.\,t)_y & := & \lambda\pi.\,(\lambda x.\,t_y)\;\pi.1\;\pi.2 & & (t\;u)^\bullet & := & (t^\bullet.1)\;u^\bullet \end{array}$$

$$(t\;u)_y := \lambda\pi.\,(t_y\,(u^\bullet, \pi)) \circledast ((t^\bullet.2)\,u^\bullet\,\pi \gg\!= u_y)$$

Flashback: Differential λ-calculus [Ehrhard, Regnier 04]

Inspired by denotational models of Linear Logic in vector spaces of sequences, it introduces a differentiation of λ-terms.

*$D(\lambda x.t)$ is the **linearization** of $\lambda x.t$, it substitute x linearly, and then it remains a term t' where x is free.*

Syntax:

$$\Lambda^d : S, T, U, V ::= 0 \mid s \mid s + T$$
$$\Lambda^s : s, t, u, v ::= x \mid \lambda x.s \mid sT \mid \mathrm{D}s \cdot t$$

Operational Semantics:

$$(\lambda x.s)T \rightarrow_\beta s[T/x]$$
$$\mathrm{D}(\lambda x.s) \cdot t \rightarrow_{\beta_D} \lambda x.\frac{\partial s}{\partial x} \cdot t$$

where $\frac{\partial s}{\partial x} \cdot t$ is the **linear substitution** of x by t in s.

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot T = \{ \begin{matrix} T \ if \ x = y \\ 0 \ otherwise \end{matrix} \qquad \frac{\partial}{\partial x}(sU) \cdot T = (\frac{\partial s}{\partial x} \cdot T)U + (\mathrm{D}s \cdot (\frac{\partial U}{\partial x} \cdot T))U$$

$$\frac{\partial}{\partial x}(\lambda y.s) \cdot T = \lambda y.\frac{\partial s}{\partial x} \cdot T \qquad \frac{\partial}{\partial x}(\mathrm{D}s \cdot u) \cdot T = \mathrm{D}(\frac{\partial s}{\partial x} \cdot T) \cdot u + \mathrm{D}s \cdot (\frac{\partial u}{\partial x} \cdot T)$$

$$\frac{\partial 0}{\partial x} \cdot T = 0 \qquad \frac{\partial}{\partial x}(s + U) \cdot T = \frac{\partial s}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T$$

$\frac{\partial s}{\partial x} \cdot t$ represents s where x is linearly (i.e. one time) substituted by t.

Tracking differentiation in Dialectica

Soundness [Ped14]

If $\Gamma \vdash t : A$ in the source then we have in the target

- $\mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A)$
- $\mathbb{W}(\Gamma) \vdash t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X)$ provided $x : X \in \Gamma$.

$$
\begin{array}{lll lll}
x_x & := & \lambda \pi.\, \{\pi\} & x^\bullet & := & x \\
x_y & := & \lambda \pi.\, \varnothing \quad \text{if } x \neq y & (\lambda x.\, t)^\bullet & := & (\lambda x.\, t^\bullet, \lambda x \pi.\, t_x\ \pi) \\
(\lambda x.\, t)_y & := & \lambda \pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2 & (t\ u)^\bullet & := & (t^\bullet.1)\ u^\bullet
\end{array}
$$

$$
(t\ u)_y := \lambda \pi.\, (t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg= u_y)
$$

Tracking differentiation in Dialectica

Soundness [Ped14]

If $\Gamma \vdash t : A$ in the source then we have in the target

- $\mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A)$
- $\mathbb{W}(\Gamma) \vdash t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X)$ provided $x : X \in \Gamma$.

5 years ago : "That's Differential λ-calculus"

$$x_x := \lambda\pi.\,\{\pi\} \qquad x^\bullet := x$$

$$x_y := \lambda\pi.\,\varnothing \quad \text{if } x \neq y \qquad (\lambda x.\,t)^\bullet := (\lambda x.\,t^\bullet, \lambda x\pi.\,t_x\ \pi)$$

$$(\lambda x.\,t)_y := \lambda\pi.\,(\lambda x.\,t_y)\ \pi.1\ \pi.2 \qquad (t\ u)^\bullet := (t^\bullet.1)\ u^\bullet$$

$$(t\ u)_y := \lambda\pi.\,(t_y\,(u^\bullet, \pi)) \circledast ((t^\bullet.2)\,u^\bullet\,\pi \gg\!= u_y)$$

Tracking differentiation in Dialectica

Soundness [Ped14]

If $\Gamma \vdash t : A$ in the source then we have in the target

- $\mathbb{W}(\Gamma) \vdash t^\bullet : \mathbb{W}(A)$
- $\mathbb{W}(\Gamma) \vdash t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X)$ provided $x : X \in \Gamma$.

5 years ago : "That's Differential λ-calculus"

$$
\begin{array}{llll}
x_x & := \lambda\pi.\, \frac{\partial x}{\partial x} \cdot \pi & x^\bullet & := x \\
x_y & := \lambda\pi.\, \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y & (\lambda x.\, t)^\bullet & := (\lambda x.\, t^\bullet, \lambda x\pi.\, t_x\ \pi) \\
(\lambda x.\, t)_y & := \lambda\pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2 & (t\ u)^\bullet & := \equiv (\lambda x.\, (tx)^\bullet)\ u^\bullet
\end{array}
$$

$$(t\ u)_y := \lambda\pi.\, (t_y\ (u^\bullet, \pi)) \circledast ((t^\bullet.2)\ u^\bullet\ \pi \gg\!= u_y)$$

Tracking differentiation in Dialectica

5 years ago : "That's Differential λ-calculus"

$$
\begin{array}{lll}
x_x & := & \lambda\pi.\, \frac{\partial x}{\partial x} \cdot \pi \\
x_y & := & \lambda\pi.\, \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y \\
(\lambda x.\, t)_y & := & \lambda\pi.\, (\lambda x.\, t_y)\ \pi.1\ \pi.2
\end{array}
\qquad
\begin{array}{lll}
x^\bullet & := & x \\
(\lambda x.\, t)^\bullet & := & (\lambda x.\, t^\bullet, \lambda x\pi.\, \lambda\pi.\, \frac{\partial t}{\partial x} \cdot \pi) \\
(t\ u)^\bullet & \equiv & (\lambda x.\, (tx)^\bullet)\ u^\bullet
\end{array}
$$

Theorem

- $(\_)^\bullet.2$ obeys the chain rule.
- t_x is contravariant in x.

Dialectica :

- Higher-Order and fine-grained reverse differential transformation.
- Agrees with a call-by-name point of view on execution of programs.
- Which operates on function variables and a few operations.

Differential categories are Dialectica categories

[De Paiva & Hyland [87,89]]

Consider a category $\mathcal{C}$ with finite product. **Dial(C)** is a new category:

- Objects: relations $\alpha \subseteq U \times X$, $\beta \subseteq V \times Y$.
- Maps from α to β: $(f : U \to V, F : U \times Y \to X)$ such that if $u \alpha F(u, y)$ then $f(u) \beta y$. *tangent spaces*
- Composition: *That's the chain law!*

show that DC is a category. Given two maps (f,F):α→β and (g,G):β→γ their composition (g,G)∘(f,F) is gf:U→W in the first coordinate and G∘F:U×Z→X given by:

$$U\times Z \xrightarrow{\delta\times Z} U\times U\times Z \xrightarrow{U\times f\times Z} U\times V\times Z \xrightarrow{U\times G} U\times Y \xrightarrow{F} X$$

Consider $\mathcal{C}$ a $*$-autonomous differential category. One has a functor from $\mathcal{C}$ to **Dial(C)**

- $A \mapsto (A, A^{\perp})$
- $f \mapsto (f, (u, \ell) \mapsto \ell \circ D_u f)$

This should be an equivalence

This relates to several other results, e.g : "Gödel's functional interpretation and the concept of learning" T. Powell, Lics 2017

Automatic Differentiation as a choice of reduction strategy

Refining λ-calculus with operations from distribution theory.

Juste a glimpse at Differential Linear Logic

Differentiation in the proofs

Differential Linear Logic

$$\frac{\ell : A \vdash B}{\ell : !A \vdash B}$$

linear $\hookrightarrow$ *non-linear.*

$$\frac{f : !A \vdash B}{D_0(f) : A \vdash B}$$

non-linear $\hookrightarrow$ *linear*

$\rightsquigarrow$ *A specific point of view on differentiation induced by duality:*

$$A^{\perp\perp} \simeq A$$

Normal functors, power series and λ-calculus. Girard, APAL(1988)

Differential interaction nets, Ehrhard and Regnier, TCS (2006)

Smooth models

Historically: discrete models and *quantitative semantics*.

$$!A := \sum_n A^{\otimes^n}$$

Exponentials as distributions [K., LICS18]

A *smooth* and classical model of Differential Linear Logic where:

$$!A = \mathcal{C}^{\infty}(A, \mathbb{R})'.$$

$\rightsquigarrow$ **Insight**: a language typed by linear logic, $u : !A$ is *a primitive object representing a program transformation.*

Consider $t : A \Rightarrow B \equiv !A \multimap B$:

$$D_0 t \cdot a \simeq t(D_{0\_} \cdot a : !A)$$

Exponentials are distributions

$$[\![?A]\!] := \mathcal{C}^\infty([\![A]\!]', \mathbb{R})' \qquad [\![!A]\!] := \mathcal{C}^\infty([\![A]\!], \mathbb{R})'$$

functions *distributions*

A typical distribution is the dirac operator:

$$\delta : \begin{cases} E \to \mathcal{C}^\infty(E, \mathbb{R})' \\ x \mapsto (\phi \mapsto \phi(x)) \end{cases}$$

Exponential rules of DiLL_0

$$\frac{\vdash \Gamma, f : ?A, g : ?A}{\vdash \Gamma, f.g : ?A}\, c \qquad \frac{\vdash \Gamma}{\vdash \Gamma, cst_0 : ?A}\, w \qquad \frac{\vdash \Gamma, \ell : A}{\vdash \Gamma, \ell : ?A}\, d$$

$$\frac{\vdash \Gamma, \phi : !A, \qquad \vdash \Delta, \psi : !A}{\vdash \Gamma, \Delta, \phi * \psi : !A}\, \bar{c} \qquad \frac{}{\vdash \delta_0 : !A}\, \bar{w} \qquad \frac{\vdash \Gamma, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\, \bar{d}$$

$$\frac{\vdash ?\Gamma, v : A}{\vdash ?\Gamma, \delta_v : !A}\, p$$

What can we get from Seely's isomorphisms

(Co)-weakenings and (co)-contractions are interpreted from the presence of a biproduct and seely's isomorphisms.

$$!A \xleftarrow{\bar{w}} !\{0\} \xrightarrow{w} !A$$

$$!A \xleftarrow{\bar{c}} !(A \diamond A) \simeq !A \otimes !A \xrightarrow{c} !A$$

Seely's isomorphism = kernel theorems, ie surjectivity of:

$$\mathcal{C}^\infty(A, \mathbb{R}) \otimes \mathcal{C}^\infty(B, \mathbb{R}) \hookrightarrow \mathcal{C}^\infty(A \times B)$$

$$?(A^\perp) \parr ?(B^\perp) \hookrightarrow ?(A^\perp \times B^\perp)$$

Yes, the $\parr$ is a tensor, completed, just with a different topology. Yes, & and $\oplus$ are the same, on different objects though

Thus: contraction is multiplication (s.calar),
co-contraction is sum (convolution).

Higher-order addition and Higher-order multiplication

Additions are done on the domain, through convolution (ie higher order addition).

$$\phi * \psi := f \mapsto \phi(x \mapsto \psi(y \mapsto f(x+y))$$
$$\delta_u * \delta_v \to \delta_{u*v}$$

Multiplications are done one the codomain, through contractions (ie higher order multiplication).

$$f \cdot g := x \mapsto f(x) \cdot g(x)$$
$$(\lambda y.t) \cdot (\lambda z.s) \to \lambda x.(t[x/y]) \cdot (s[x/z])$$

A few operations typed by DILL

The composition of linear functions:

$$\frac{\Gamma \vdash f : A \multimap B \qquad \Delta \vdash g : B \multimap C}{\Gamma, \Delta \vdash g \circ f : A \multimap C}\ \text{cut}$$

The composition of non-linear functions:

$$\frac{\dfrac{\Gamma \vdash f : !A \multimap B}{\Delta \vdash (x \mapsto \delta_{f(x)}) : !A \multimap !B}\ \text{p} \qquad \Delta \vdash g : !B \multimap C}{\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta_{f(x)} g) : !A \multimap C}\ \text{cut}$$

The Differentiation of non-linear functions:

$$\frac{\Gamma \vdash f : !A \multimap B \qquad \dfrac{\vdash \Delta, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\ \bar{d}}{\Gamma, \Delta \vdash D_0(f)(v) : B}\ \text{cut}$$

Let's translate this into a term language typed by DILL.

A few operations typed by DiLL

The chain rule is encoded in the interaction of diracs δ_x with differential arguments $D_u t$.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash f : !A \multimap B}{\Gamma \vdash (x \mapsto \delta_{f(x)}) : !A \multimap !B}\,\mathrm{p} \qquad \Delta \vdash g : !B \multimap C}{\Gamma, \Delta \vdash g \circ \delta_f : !A \multimap C}\,\mathrm{cut} \qquad \dfrac{\vdash \Delta', v : A}{\vdash \Delta', D_0(\_)(v) : !A}\,\bar{d}}{\Gamma, \Delta, \Delta' \vdash D_0(g \circ f)(v) : c}\,\mathrm{cut}$$

$$\rightsquigarrow$$

$$\dfrac{\vdash g : !B \multimap C \qquad \dfrac{\dfrac{\dfrac{\dots}{\vdash D_0(f)(v) : B}\,\bar{d}; f; cut}{\vdash D_0(\_)(D_0 f(v)) : !B}\,\bar{d} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \delta_0 : !A}\,\bar{w} \qquad \vdash f : !A \multimap B}{\vdash f(0) : B}\,\mathrm{cut}}{\vdash \delta_{f(0)} : !B}\,\mathrm{p}}{\vdash D_{f(0)}(\_)(D_0 f(v)) : !B}\,\bar{c}}{\vdash D_{f(0)} g(D_0 f(v)) : c}\,\mathrm{cut}$$

Let's translate this into a term language typed by DiLL.

From two reductions to two arguments

A minimal language allowing to express automatic differentiation, with two class of terms:

$$u, v := x \mid t^{\perp} \mid u * v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta_u \mid D_u(t) \mid \downarrow t$$
$$t, s := u^{\perp} \mid t \cdot s \mid w_1 : N \mid \lambda x.t \mid dx.t \mid \uparrow u$$

A function $\lambda x.t$ can be matched with two kind of arguments: diracs δ_u or differential operators $\mathrm{D}_u t$.

$$(\lambda x.t)\delta_u \to t[u/x]$$
$$(\lambda x.t) D_w u \to \cdots$$

Ideas:

- Differentiation, as an argument, propagates according to reduction strategies.
- Algebraic operations are constructed through specific type rules.

Inductively defined linear substitution

$$u, v := x \mid t^{\perp} \mid u * v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta_u \mid D_u(t) \mid \downarrow t$$
$$t, s := u^{\perp} \mid t \cdot s \mid w_1 : N \mid \lambda x.t \mid dx.t \mid \uparrow u$$

An inductively defined differentiation:

$$(\lambda x.t) D_w u \to \cdots$$

The differentiation $\lambda x.t$ of must be inductively defined on t:

$$(\lambda x.(t)u) D_w s \to \uparrow(\downarrow((\lambda x.t) D_w s) u * \downarrow(t((\lambda x.u) D_w s)))$$

Differentiating an application $(t)u$ is symmetric in t and u.

$$(\lambda x.\uparrow\delta_t) D_u s \to (\lambda z.\uparrow(D_z((\lambda x.t) D_u s)))((\lambda x.t)(u)))$$

The abstraction $\lambda x.\uparrow\delta_t$ will be composed with another abstraction and differentiation must take that into account.

Forward / Backward Differentiation as CBV/CBN

$$D_u((\lambda y.s) \circ (\lambda x.t))r?$$

$$\begin{aligned}
(\lambda x.((\lambda y.s)\delta_t))D_u r &\to (\lambda x.(\lambda y.s))D_u r)\delta_t * ((\lambda y.s)((\lambda x.\delta_t)D_u r))) \\
&\to^* \emptyset * (\lambda y.s)((\lambda x.\delta_t)D_u r)))\text{as } x \text{ is free in } s \\
&\to^* (\lambda y.s)((\lambda x.\delta_t)D_u r)) \\
&\to (\lambda y.s)(\lambda z.(D_z((\lambda x.t)D_u r)))((\lambda x.t)(u))) \\
&\to ((\lambda y.s)(\lambda z.(D_z((\lambda x.t)D_u r))))((t[w/x]))\text{as } u = \delta_w \\
&\to^* (\lambda y.s)D_v((\lambda x.t)D_u r) \text{ if } (t[w/x] \to^* \delta_v)
\end{aligned}$$

- The value of $t[w/x]$ is computed first-hand.
- CBN : $((\lambda x.t)D_u r)$ or CBV : $((\lambda y.s)D_v((\lambda x.t)D_u r))$

And complexity?

$$D_u(\ell \circ f)(v) = (\ell \circ D_u f)(v) = (D_u \ell \circ D_u f)(v)$$

Our differentiation takes into account the linearity of higher-order operations :

$$D_u((\lambda y.s) \circ (\lambda x.t))r?$$

when $\lambda y.s$ is linear.

$$D_0((\lambda y.s) \circ (\lambda x.t))r \equiv (D_0(\lambda y.s) \circ D_0(\lambda x.t))r?$$

when $\lambda y.s$ is linear.

work in progress

Conclusion

Logic acts as a bridge between **programming languages** and **analysis**.

Take-away message:

- Constructs new types (safety).
- Constructs new terms (modularity).

Perspectives:

- (Basic) computer algebra algorithms arising unexpectedly in logical transformation.

And Dialectica ??

Make a monad of the exponential (WIP).

$$\mathbb{L}(\alpha_{\mathbb{W}}) := \alpha \qquad \mathbb{L}(\alpha_{\mathbb{C}}) := \uparrow\alpha^{\perp}$$
$$\mathbb{L}(\mathfrak{M}\, A) := \uparrow!\mathbb{L}(A) \qquad \mathbb{L}(A \times B) := \mathbb{L}(A) \times \mathbb{L}(B)$$
$$\mathbb{L}(A \Rightarrow B) := \downarrow\mathbb{L}(A) \Rightarrow \mathbb{L}(B) \qquad [\mathfrak{M}\, A] := ![A]$$

$$[x] := x$$
$$[\lambda x.t] := \lambda x.[t] \qquad [(t,u)] := ([t],[u])$$
$$[\emptyset] := \uparrow\emptyset \qquad [\{t\}] := (D_{\emptyset} t)$$
$$[u \circledast v] := [u] * [v] \qquad [m \gg\!= f] := (dx.[f]x)[m]$$

A translation on top of Dialectica

If $\Gamma \vdash t : A$ in the target of Dialectica, then $\mathbb{L}(\Gamma) \vdash [t] : \mathbb{L}(A)$ and if $t \equiv u$ in the target of Dialectica then $[t] \equiv [u]$ in our calculus.

More on Dialectica

Monadic laws

$$\{t\} \gg= f \equiv f\ t \qquad t \gg= (\lambda x.\{x\}) \equiv t$$

$$(t \gg= f) \gg= g \equiv t \gg= (\lambda x.\ f\ x \gg= g)$$

Monoidal laws

$$t \circledast u \equiv u \circledast t \qquad \varnothing \circledast t \equiv t \circledast \varnothing \equiv t$$

$$(t \circledast u) \circledast v \equiv t \circledast (u \circledast v)$$

Distributivity laws

$$\varnothing \gg= f \equiv \varnothing \qquad t \gg= \lambda x.\varnothing \equiv \varnothing$$

$$(t \circledast u) \gg= f \equiv (t \gg= f) \circledast (u \gg= f)$$

$$t \gg= \lambda x.(f\ x \circledast g\ x) \equiv (t \gg= f) \circledast (t \gg= g)$$