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.*

<table>
<thead>
<tr>
<th>Programs</th>
<th>Logic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{fun } (x:A) \rightarrow (t:B) )</td>
<td>Proof of ( A \vdash B )</td>
<td>( f : A \rightarrow B ).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Types</th>
<th>Formulas</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution</td>
<td>Cut-elimination</td>
<td>Equality</td>
</tr>
</tbody>
</table>

Differentiable Programming

- Resources λ-calculus
- Differential Linear Logic
- Distribution theory
- Vectorial Models
- Linear Logic [Gir87]
- Normal functors
- Min. Logic
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.
- $\lambda$-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?

\[ x_1 = x_0^2 \quad x_1' = 2x_0x_0' \]

E.g. : \( z = y + \cos(x^2) \quad x_2 = \cos(x_1) \quad x_2' = -x_0' \sin(x_0) \)

\[ z = y + x_2 \quad z' = y' + 2x_2x_2' \]

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]
\( (x_1, x_1') \rightarrow (x_2, x_2') \rightarrow (z, z'). \)

**Reverse Mode differentiation:** [Speelpenning, Rall, 1980s]
\( x_1 \rightarrow x_2 \rightarrow z \rightarrow z' \rightarrow x_2' \rightarrow x_1' \) while keeping formal the unknown derivative.
AD from a higher-order functional point of view

\[ D_u(f \circ g) = D_{g(u)}f \circ D_u(f) \]

- **Forward Mode differentiation**:  
  \[ g(u) \rightarrow D_u g \rightarrow f(g(u)) \rightarrow D_{g(u)}f \rightarrow D_{g(u)}f \circ D_u(f) \]

- **Reverse Mode differentiation**:  
  \[ g(u) \rightarrow f(g(u)) \rightarrow D_{g(u)}f \rightarrow D_u g \rightarrow D_{g(u)}f \circ D_u(f) \]

The choice of an algorithm is due to complexity considerations:

- Forward mode for \( f : \mathbb{R} \rightarrow \mathbb{R}^n \).
- Reverse mode for \( f : \mathbb{R}^n \rightarrow \mathbb{R} \)

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 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 \( A^\perp = A \multimap \bot \): an involutive linear negation interpreted by linear forms.

\[ [A^\perp] = \mathcal{L}([A], \mathbb{R}) \]
Key Idea

Reverse derivatives are typed by linear negation.

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

$$\hat{D}(f) : \begin{cases} \mathbb{R}^n \times \mathbb{R}^\perp \rightarrow \mathbb{R} \times \mathbb{R}^{n\perp} \\
(a, x) \mapsto (f(a), (v \mapsto x \cdot (D_a f \cdot v)) \end{cases}$$

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

$$A, B, C ::= R \mid A \times B \mid A \rightarrow B \mid R^\perp \mid \lambda x.t \mid (t)u \mid t[x()]! := u \mid < t, u > \mid t + u...$$
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.

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

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

- Pedrot [2014] A *computational* Dialectica translation preserving \( \beta \)-equivalence, via the introduction of an ”abstract multiset constructor” on types on the target.
Pédrot’s Dialectica Transformation

\( \mathcal{M} A \) is endowed with a sum (\( \otimes, \emptyset \)) and a monadic structure (\( \{\_\}, \gg\gg \))

Types:
\[
\begin{align*}
\mathcal{W}(\alpha) & := \alpha_{\mathcal{W}} & \mathcal{C}(\alpha) & := \alpha_{\mathcal{C}} \\
\mathcal{W}(A \Rightarrow B) & := (\mathcal{W}(A) \Rightarrow \mathcal{W}(B)) \times (\mathcal{W}(A) \Rightarrow \mathcal{C}(B) \Rightarrow \mathcal{M} \mathcal{C}(A)) \\
\mathcal{C}(A \Rightarrow B) & := \mathcal{W}(A) \times \mathcal{C}(B)
\end{align*}
\]

Terms:
\[
\begin{align*}
x_x & := \lambda \pi. \{\pi\} & x^\bullet & := x \\
x_y & := \lambda \pi. \emptyset \text{ if } x \neq y & (\lambda x. t)^\bullet & := (\lambda x^\bullet, \lambda x \pi. t_x \pi) \\
(\lambda x. t)_y & := \lambda \pi. (\lambda x. t_y) \pi.1 \pi.2 & (t \ u)^\bullet & := (t^\bullet.1) u^\bullet \\
(t \ u)_y & := \lambda \pi. (t_y (u^\bullet, \pi)) \otimes ((t^\bullet.2) u^\bullet \pi \gg\gg u_y)
\end{align*}
\]
Flashback: Differential $\lambda$-calculus [Ehrhard, Regnier 04]

Inspired by denotational models of Linear Logic in vector spaces of sequences, it introduces a differentiation of $\lambda$-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:

$\Lambda^d : S, T, U, V ::= 0 \mid s \mid s+T$

$\Lambda^s : s, t, u, v ::= x \mid \lambda x.s \mid sT \mid Ds.t$

Operational Semantics:

$(\lambda x.s)T \rightarrow_\beta s[T/x]$

$D(\lambda x.s) \cdot t \rightarrow_\beta_D \lambda x.\frac{\partial s}{\partial x} \cdot t$

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

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial}{\partial x} (sU) \cdot T = (\frac{\partial s}{\partial x} \cdot T)U + (Ds \cdot (\frac{\partial U}{\partial x} \cdot T))U$$

$$\frac{\partial}{\partial x} (\lambda y.s) \cdot T = \lambda y. \frac{\partial s}{\partial x} \cdot T$$

$$\frac{\partial}{\partial x} (Ds \cdot u) \cdot T = D(\frac{\partial s}{\partial x} \cdot T) \cdot u + Ds \cdot (\frac{\partial u}{\partial x} \cdot T)$$

$$\frac{\partial 0}{\partial x} \cdot T = 0$$

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

$$\frac{\partial s}{\partial x} \cdot t$$ 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

$\mathcal{W}(\Gamma) \vdash t^\bullet : \mathcal{W}(A)$

$\mathcal{W}(\Gamma) \vdash t_x : \mathcal{C}(A) \Rightarrow \mathcal{M}\mathcal{C}(X)$ provided $x : X \in \Gamma$.

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

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

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

$(t \ u)_y := \lambda \pi. (t_y (u^\bullet, \pi)) \odot ((t^\bullet.2) u^\bullet \pi \ggg u_y)$
Tracking differentiation in Dialectica

Soundness [Ped14]

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

\[
\begin{align*}
\text{1.} & \quad W(\Gamma) \vdash t^\bullet : W(A) \\
\text{2.} & \quad W(\Gamma) \vdash t_x : C(A) \Rightarrow M_C(X) \text{ provided } x : X \in \Gamma.
\end{align*}
\]

5 years ago: ”That’s Differential \( \lambda \)-calculus”

\[
\begin{align*}
xx & := \lambda \pi. \{ \pi \} & x^\bullet & := x \\
x_y & := \lambda \pi. \emptyset \quad \text{if } x \neq y & (\lambda x. t)^\bullet & := (\lambda x. t^\bullet, \lambda x^\bullet. t_x \pi) \\
(\lambda x. t)_y & := \lambda \pi. (\lambda x. t_y) \pi.1 \pi.2 & (t \ u)^\bullet & := (t^\bullet.1) u^\bullet \\
(t \ u)_y & := \lambda \pi. (t_y (u^\bullet, \pi)) \uplus ((t^\bullet.2) u^\bullet \pi \ggg u_y)
\end{align*}
\]
Tracking differentiation in Dialectica

Soundness [Ped14]

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

\[ \begin{align*}
\text{\( W(\Gamma) \vdash t^\bullet : W(A) \)} \\
\text{\( W(\Gamma) \vdash t_x : C(A) \Rightarrow M C(X) \) provided } x : X \in \Gamma.
\end{align*} \]

5 years ago : ”That’s Differential \( \lambda \)-calculus”

\[
\begin{align*}
x_x & := \lambda \pi. \frac{\partial x}{\partial x} \cdot \pi & x^\bullet & := x \\
x_y & := \lambda \pi. \frac{\partial x}{\partial y} \cdot \pi \text{ if } x \neq y & (\lambda x. t)^\bullet & := (\lambda x. t^\bullet, \lambda x \pi. t_x \pi) \\
(\lambda x. t)_y & := \lambda \pi. (\lambda x. t_y) \pi.1 \pi.2 & (t u)^\bullet & := \equiv (\lambda x. (tx)^\bullet) u^\bullet \\
(t u)_y & := \lambda \pi. (t_y (u^\bullet, \pi)) \oplus (t^\bullet.2) u^\bullet \pi \gg\gg u_y)
\end{align*}
\]
Tracking differentiation in Dialectica

5 years ago: ”That’s Differential $\lambda$-calculus”

\[
\begin{align*}
xx & := \lambda\pi. \frac{\partial x}{\partial x} \cdot \pi & x^\bullet & := x \\
xy & := \lambda\pi. \frac{\partial x}{\partial y} \cdot \pi \quad \text{if } x \neq y & (\lambda x. t)^\bullet & := (\lambda x. t^\bullet, \lambda x\pi. \lambda\pi. \frac{\partial t}{\partial x} \cdot \pi) \\
(\lambda x. t)_y & := \lambda\pi. (\lambda x. t)_y \pi \cdot 1 \pi \cdot 2 & (t u)^\bullet & \equiv (\lambda x. (tx)^\bullet) u^\bullet
\end{align*}
\]

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. $\text{Dial}(\mathcal{C})$ is a new category:

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

show that $\text{Dial} \mathcal{C}$ is a category. Given two maps $(f, F) : \alpha \to \beta$ and $(g, G) : \beta \to \gamma$ their composition $(g, G) \circ (f, F)$ is $gf : U \to W$ in the first coordinate and $G \circ F : U \times Z \to X$ given by:

$$
\begin{align*}
U \times Z & \xrightarrow{f \times Z} U \times U \times Z \\
& \xrightarrow{U \times F \times Z} U \times V \times Z \\
& \xrightarrow{U \times G} U \times Y \\
& \xrightarrow{F} X
\end{align*}
$$
Consider $\mathcal{C}$ a $\ast$-autonomous differential category. One has a functor from $\mathcal{C}$ to $\text{Dial}(\mathcal{C})$

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

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

linear proof  \( \frac{A \vdash B}{\ell : !A \vdash B} \)
Linear Logic  \( \frac{\ell : !A \vdash B}{A \vdash B} \)

non-linear proof

\( f : !A \vdash B \)

\( D_0(f) : A \vdash B \)

linear \(\rightarrow\) non-linear. non-linear \(\rightarrow\) linear

\(\rightsquigarrow\) A specific point of view on differentiation induced by duality:

\( A^\bot^\bot \simeq A \)

Normal functors, power series and \(\lambda\)-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:

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

\[ \leadsto \textbf{Insight:} \text{ a language typed by linear logic, } u : !A \text{ is a primitive object representing a program transformation.} \]

Consider \( t : A \Rightarrow B \equiv !A \rightarrow B \):

\[ D_0 t \cdot a \simeq t(D_0 \cdot a : !A) \]
Exponentials are distributions

\[
\begin{align*}
\lbrack ?A \rbrack & := C^\infty([A]', \mathbb{R})' \\
\lbrack !A \rbrack & := C^\infty([A], \mathbb{R})'
\end{align*}
\]

functions distributions

A typical distribution is the dirac operator:

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

Exponential rules of DiLL\(_0\)

\[
\frac{\vdash \Gamma, f : ?A, g : ?A}{\vdash \Gamma, f \cdot g : ?A} \quad w
\]

\[
\frac{\vdash \Gamma}{\vdash \Gamma, cst_0 : ?A} \quad c
\]

\[
\frac{\vdash \Gamma, \ell : A}{\vdash \Gamma, \ell : ?A} \quad d
\]

\[
\frac{\vdash \Gamma, \phi : !A, \vdash \Delta, \psi : !A}{\vdash \Gamma, \Delta, \phi \ast \psi : !A} \quad \bar{c}
\]

\[
\frac{\vdash \delta_0 : !A}{\vdash \delta_0 : !A} \quad \bar{w}
\]

\[
\frac{\vdash \Gamma, v : A}{\vdash \Gamma, D_0(-)(v) : !A} \quad \bar{d}
\]

\[
\frac{\vdash ?\Gamma, v : A}{\vdash ?\Gamma, \delta_v : !A} \quad p
\]
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.

\[ !A \leftrightarrow !\{0\} \xrightarrow{w} !A \]

\[ !A \leftarrow \bar{c} !(A \diamond A) \simeq !A \otimes !A \xrightarrow{c} !A \]

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

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

\[ \wp(A^\perp) \cong \wp(B^\perp) \hookrightarrow \wp(A^\perp \times B^\perp) \]

Yes, the \( \wp \) 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).

\[
\phi \ast \psi := f \mapsto \phi(x \mapsto \psi(y \mapsto f(x + y)) \\
\delta_u \ast \delta_v \rightarrow \delta_{u \ast v}
\]

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

\[
f \cdot g := x \mapsto f(x) \cdot g(x) \\
(\lambda y.t) \cdot (\lambda z.s) \rightarrow \lambda x.(t[x/y]) \cdot (s[x/z])
\]
A few operations typed by DiLL

The composition of linear functions:
\[
\frac{\Gamma \vdash f : A \rightarrow B \quad \Delta \vdash g : B \rightarrow C}{\Gamma, \Delta \vdash g \circ f : A \rightarrow C}\] cut

The composition of non-linear functions:
\[
\frac{\Gamma \vdash f : !A \rightarrow B}{\Delta \vdash (x \mapsto \delta_f(x)) : !A \rightarrow !B} \quad \frac{\Delta \vdash g : !B \rightarrow C}{\Gamma, \Delta \vdash g \circ f = (x \mapsto \delta_f(x)g) : !A \rightarrow C}\) cut

The Differentiation of non-linear functions:
\[
\frac{\Gamma \vdash f : !A \rightarrow B}{\Gamma, \Delta \vdash \frac{\partial}{\partial v} (f(v)) : !A} \quad \frac{\vdash \Delta, v : A}{\Gamma, \Delta \vdash D_0(f)(v) : B}\] 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 $\delta_x$ with differential arguments $D_u t$.

\[
\begin{align*}
\Gamma & \vdash f : !A \to B \\
\Gamma & \vdash (x \mapsto \delta_f(x)) : !A \to !B & \text{p} \\
\Delta & \vdash g : !B \to C & \text{cut} \\
\Gamma, \Delta & \vdash g \circ \delta_f : !A \to C & \text{cut} \\
\Gamma, \Delta, \Delta' & \vdash D_0(g \circ f)(v) : c & \text{\textmd{\textnumero\textmd{\textnumero}}} \\
\end{align*}
\]

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:

\[ u, v := x \mid t^\perp \mid u \times v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta_u \mid D_u(t) \mid \downarrow t \]

\[ t, s := u^\perp \mid t \cdot s \mid w_1 : N \mid \lambda x.t \mid dx.t \mid \uparrow u \]

A function \( \lambda x.t \) can be matched with two kind of arguments: diracs \( \delta_u \) or differential operators \( D_u t \).

\[
\begin{align*}
(\lambda x.t) \delta_u & \rightarrow t[u/x] \\
(\lambda x.t) D_w u & \rightarrow \cdots
\end{align*}
\]

Ideas:

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

\[ u, v := x \mid t \uparrow \mid u \ast v \mid \emptyset \mid u \otimes v \mid 1 \mid \delta_u \mid D_u(t) \mid \downarrow t \]
\[ t, s := u \uparrow \mid t \cdot s \mid w_1 : N \mid \lambda x. t \mid dx. t \mid \uparrow u \]

An inductively defined differentiation:

\[ (\lambda x. t)D_w u \rightarrow \cdots \]

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

\[ (\lambda x. (t)u)D_w s \rightarrow \uparrow(\downarrow((\lambda x. t)D_w s)u \ast \downarrow(t((\lambda x. u)D_w s))) \]

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

\[ (\lambda x. \uparrow \delta_t)D_u s \rightarrow (\lambda z. \uparrow(D_z((\lambda x. t)D_u s))((\lambda x. t)(u))) \]

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

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

\[
(\lambda x.((\lambda y.s)\delta_t))D_u r \rightarrow (\lambda x.(\lambda y.s))D_u r)\delta_t * ((\lambda y.s)((\lambda x.\delta_t)D_u r)))
\]

\[
\rightarrow^* \emptyset * (\lambda y.s)((\lambda x.\delta_t)D_u r))) as x is free in s
\]

\[
\rightarrow^* (\lambda y.s)((\lambda x.\delta_t)D_u r))
\]

\[
\rightarrow (\lambda y.s)(\lambda z.(D_z((\lambda x.t)D_u r)))((\lambda x.t)(u)))
\]

\[
\rightarrow ((\lambda y.s)(\lambda z.(D_z((\lambda x.t)D_u r))))((t[w/x])) as u = \delta_w
\]

\[
\rightarrow^* (\lambda y.s)D_v((\lambda x.t)D_u r) if (t[w/x] \rightarrow^* \delta_v)
\]

▶ 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).

\[ \mathbb{L}(\alpha_W) := \alpha \]
\[ \mathbb{L}(\mathbb{M} A) := \uparrow!\mathbb{L}(A) \]
\[ \mathbb{L}(A \Rightarrow B) := \downarrow\mathbb{L}(A) \Rightarrow \mathbb{L}(B) \]
\[ \mathbb{L}(A \times B) := \mathbb{L}(A) \times \mathbb{L}(B) \]
\[ [\mathbb{M} A] := ![A] \]

\[ [x] := x \]
\[ [\lambda x.t] := \lambda x.[t] \]
\[ [\emptyset] := \uparrow\emptyset \]
\[ [u \odot v] := [u] \ast [v] \]
\[ [(t, u)] := ([t], [u]) \]
\[ [{t}] := (D_\emptyset t) \]
\[ [m \gg f] := (dx. [f]x)[m] \]

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

\[ \{t\} >\!\!\!\!\!\!= f \equiv f \cdot t \quad \text{and} \quad t >\!\!\!\!\!\!= (\lambda x. \{x\}) \equiv t \]

\[ (t >\!\!\!\!\!\!= f) >\!\!\!\!\!\!= g \equiv t >\!\!\!\!\!\!= (\lambda x. f \cdot x >\!\!\!\!\!\!= g) \]

Monoidal laws

\[ t \odot u \equiv u \odot t \quad \emptyset \odot t \equiv t \odot \emptyset \equiv t \]

\[ (t \odot u) \odot v \equiv t \odot (u \odot v) \]

Distributivity laws

\[ \emptyset >\!\!\!\!\!\!= f \equiv \emptyset \quad \text{and} \quad t >\!\!\!\!\!\!= \lambda x. \emptyset \equiv \emptyset \]

\[ (t \odot u) >\!\!\!\!\!\!= f \equiv (t >\!\!\!\!\!\!= f) \odot (u >\!\!\!\!\!\!= f) \]

\[ t >\!\!\!\!\!\!= \lambda x. (f \cdot x \odot g \cdot x) \equiv (t >\!\!\!\!\!\!= f) \odot (t >\!\!\!\!\!\!= g) \]