Thomas Ehrhard’s 60 birthday

\( \partial \) is for Dialectica

Marie Kerjean

CNRS & LIPN, Université Sorbonne Paris Nord

Work in collaboration with Pierre-Marie Pédrot
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?
Thank you Thomas

... For the opportunity to finally understand **differentiation**.

**What’s differentiation?**
Thank you Thomas

... For the opportunity to finally understand differentiation.

What’s differentiation?

That’s differentiation!

\[
\frac{\partial}{\partial x} ((\lambda z.t)u) \cdot s = \left( \frac{\partial(\lambda z.t)}{\partial x} \cdot s \right)u + (D(\lambda z.y) \cdot \left( \frac{\partial u}{\partial x} \cdot s \right))u
\]
What’s this talk is about

Joining Dialectica and Differential $\lambda$-calculus through Reverse Differentiation
What’s this talk is about

Joining Dialectica and Differential Linear Logic through Reverse Differentiation
What’s this talk is about

Joining Dialectica and Differential Categories through Reverse Differentiation
What’s this talk is about

Joining *Dialectica* and *Differential $\lambda$-calculus* through
*Reverse Differentiation*
Gödel’s Dialectica Transformation

1. \((F \land G)' = (∃yv) (zw) [A (y, z, x) \land B (v, w, u)].\)
2. \((F \lor G)' = (∃yvt) (zw) [t=0 \land A (y, z, x) \lor \cdot t=1 \land B (v, w, u)].\)
3. \([s] F)' = (∃Y) (sz) A (Y (s), z, x).
4. \([∃s] F)' = (∃sy) (z) A (y, z, x).
5. \((F \supset G)' = (∃VZ) (yw) [A (y, Z (yw), x) \supset B (V (y), w, u)].\)
6. \((\neg F)' = (∃Z) (y) \neg A (y, Z (y), x).\)

▶ Validates semi-classical axioms:
▶ Markov’s principle: \(\neg\neg∃x A \rightarrow ∃x A\) when \(A\) is decidable.
▶ Numerous applications:
▶ Soudness results
▶ **Proof mining**: applying Dialectica to theorems in analysis extract quantitative information.

”There are infinitely many prime numbers.”

\[
\Downarrow
\]

”For any \(m\) there exists some \(m < p \leq \lceil e^{m-\gamma}\rceil\) such that \(p\) is prime.”
And now for something completely different: 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{align*}
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{align*}

Derivative of a sequence of instruction
\[ \downarrow \]
sequence of instruction $\times$ sequence of derivatives

**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.
I hate graphs

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

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

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

The choice of an algorithm is due to complexity considerations:

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

\[ \sim \] Differentiable programming is a new research area triggered by the advances of deep learning algorithms on neural networks, it tries to attach two very old domains: lambda-calculus and automatic differentiation, with correctness and modularity goals in mind.
Functorial Forward AD

\[ \mathbf{D}_u(f \circ g) = \mathbf{D}_{g(u)} f \circ \mathbf{D}_u(g) \]

Non-functorial !!!

How to make differentiation functorial? Make it act on pairs!

Forward Mode differentiation:

\[ g : E \Rightarrow F \rightsquigarrow \overrightarrow{D} g : E \Rightarrow E \rightarrow F. \]

Functorial forward differentiation:

\[ \overrightarrow{D}(g) : \begin{cases} E \times E \rightarrow F \times F \\ (a, x) \mapsto (f(a), (D_a f \cdot x)) \end{cases} \]
Reverse functorial differentiation

**Linear implication**

\[ A^\perp \equiv A \rightarrow \bot \equiv \mathcal{L}(A, \mathbb{R}) \equiv A' \]
Reverse functorial differentiation

Linear implication
\[ A \perp \equiv A \rightarrow \perp \equiv \mathcal{L}(A, \mathbb{R}) \equiv A' \]

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

\[ D_u(g) : F' \rightarrow E' ; \ell \mapsto \ell \circ D_u g \]

\[ g : E \Rightarrow F \rightsquigarrow \bar{D}g : E \Rightarrow F \perp \rightarrow E \perp. \]

[Mazza, Pagani, POPL2020]
Reverse functorial differentiation

Linear implication
\[ A^\perp \equiv A \rightarrow \perp \equiv \mathcal{L}(A, \mathbb{R}) \equiv A' \]

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

  \[ D_u(g) : F' \rightarrow E'; \ell \mapsto \ell \circ D_u g \]

  \[ g : E \Rightarrow F \rightsquigarrow \overleftarrow{D}g : E \Rightarrow F^\perp \rightarrow E^\perp. \]

  [Mazza, Pagani, POPL2020]

- **Reverse functorial differentiation:**
  \[ (f, \overleftarrow{D}(f)) : (E \Rightarrow F) \times (E \Rightarrow F^\perp \rightarrow E^\perp) \]
Outline of the talk

• Reverse differentiation and differentiable programming.

• Dialectica acting on formulas.

• Dialectica acting on $\lambda$-terms.

• Factorizing Dialectica through differential linear logic.

• Applications and related work.
A Dialectica Transformation

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

\[ A \leadsto \exists u : W(A), \forall x : C(A), A^D[u, x] \]

- De Paiva [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.
1. \((F \land G)' = (\exists yv) (zw) [A(y, z, x) \land B(v, w, u)]\).
2. \((F \lor G)' = (\exists yvl) (zw) [t=0 \land A(y, z, x) \lor t=1 \land B(v, w, u)]\).
3. \([(s) F]' = (\exists Y) (sz) A(Y(s), z, x)\).
4. \([(\exists s) F]' = (\exists sy) (z) A(y, z, x)\).
5. \((F \supset G)' = (\exists VZ) (yw) [A(y, Z(yw), x) \supset B(V(y), w, u)]\).
6. \((\neg F)' = (\exists \tilde{Z}) (y) \neg A(y, \tilde{Z}(y), x)\).

Gödel’s Dialectica

- Validates semi-classical axioms:
  - Markov’s principle: $\neg\neg\exists x A \rightarrow \exists x A$ when $A$ is decidable.
  - Independant of premises: $(A \rightarrow \exists x B) \rightarrow (\exists x.(A \rightarrow B))$

- Numerous applications:
  - Soundness results
  - Proof mining

A further distinguishing feature of the D-interpretation is its nice behavior with respect to modus ponens. In contrast to cut-elimination, which entails a global (and computationally infeasible) transformation of proofs, the D-interpretation extracts constructive information through a purely local procedure: when proofs of $\varphi$ and $\varphi \rightarrow \psi$ are combined to yield a proof of $\psi$, witnessing terms for the antecedents of this last inference are combined to yield a witnessing term for the conclusion. As a result of this modularity, the interpretation of a theorem can be readily obtained from the interpretations of the lemmata used in its proof.

A peek into Dialectica interpretation of functions

\[(A \rightarrow B)_D = \exists f g \forall xy (A_D(x, gxy) \rightarrow B_D(fx, y))\]

**Usual explanation** : least unconstructive prenexation.

- Start from \(\exists x, \forall u, A_D[x, u] \rightarrow \exists y, \forall v, B_D[y, v]\).
- Obvious prenexation : \(\forall x (\forall u, A_D[x, u] \rightarrow \exists y, \forall v, B_D[y, v])\)
- Weak form of IP : \(\forall x \exists y, \forall v, \forall \neg \exists u (A_D[x, u] \rightarrow B_D[y, v])\)
- Prenexation : \(\forall x \exists y, \forall v, \forall \neg \exists u (A_D[x, u] \rightarrow B_D[y, v])\).
- Markov : \(\forall x, \exists y, \forall v, \exists u (A_D[x, u] \rightarrow B_D[y, v])\)
- Axiom of choice : \(\exists f, \exists g, \forall u, \forall v, (A_D(u, guv) \rightarrow B_D[fu, v])\).

**Dynamic behaviour** : agrees to a chain rule.

Mathematical meaning : it’s some kind of approximation.
Dialectica verifies the chain rules

\[(A \Rightarrow B)_{D}[\phi_1; \psi_1, u_1; v_1] := A_{D}(u_1, \psi_1 u_1 v_1) \Rightarrow B_{D}(\phi_1 u_1, v_1)\]
\[(B \Rightarrow C)_{D}[\phi_2; \psi_2, u_2; v_2] := B_{D}(u_2, \psi_2 u_2 v_2) \Rightarrow C_{D}(\phi_2 u_2, v_2)\]
\[(A \Rightarrow C)_{D}[\phi_3; \psi_3, u_3; v_3] := A_{D}(u_3, \psi_3 u_3 v_3) \Rightarrow C_{D}(\phi_3 u_3, v_3)\]

The Dialectica interpretation amounts to the following equations:

\[u_3 = u_1\]
\[v_3 = v_2\]
\[u_2 = \phi_1 u_1\]
\[\psi_3 u_3 v_3 = \psi_1 u_1 v_1\]
\[\phi_2 u_2 = \phi_1 u_1\]
\[v_2 = \phi_1 u_1 v_1\]

which can be simplified to:

\[\phi_3(u_3) = \phi_2 (\phi_1 u_3) \text{ composition of functions}\]
\[\psi_3 * (u_3 v_3) = \psi_2 (\phi_1 u_3) (\psi_1 u_3 v_3) \text{ composition of their differentials}\]
Types!

Programs and variable are typed by logical formulas which describe their behavior

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

Witness and counter types:

\[ \mathbb{C}(A \Rightarrow B) = \mathbb{C}(A) \times \mathbb{C}(B) \]

\[ \mathbb{W}(A \Rightarrow B) = (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A)) \]
Types!

Programs and variable are **typed**

by logical formulas which describe their behavior

\[
A \leadsto \exists x : \mathbb{W}(A), \forall u : \mathbb{C}(A), A_D[x, u]
\]

**Witness and counter for implication types**:

\[
\mathbb{C}(A \Rightarrow B) = \mathbb{W}(A) \times \mathbb{C}(B)
\]

\[
\mathbb{W}(A \Rightarrow B) = (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times \left( \mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A) \right)
\]

**Reverse Mode differentiation**:

Functorial: \((h, \overset{\text{̄}}{D} h) : (A \Rightarrow B) \times (A \Rightarrow B^\perp \rightarrow A^\perp)\)

**However**:

- Having the same type does not mean you’re the same program.
- We (linear logicians) know what program differentiation is.
The computational Dialectica: a reverse Differential $\lambda$-calculus
A computational Dialectica

Making Dialectica act on λ-terms instead of formulas:

An abstract multiset \( \mathcal{M}(\_ ) \)

\[
\begin{align*}
\Gamma \vdash \emptyset : \mathcal{M} A & \quad \frac{\Gamma \vdash m_1 : \mathcal{M} A \quad \Gamma \vdash m_2 : \mathcal{M} A}{\Gamma \vdash m_1 \odot m_2 : \mathcal{M} A} \\
\Gamma \vdash t : A & \quad \frac{\Gamma \vdash m : \mathcal{M} A \quad \Gamma \vdash f : A \Rightarrow \mathcal{M} B}{\Gamma \vdash m \triangleright= f : \mathcal{M} B}
\end{align*}
\]

\[
\begin{align*}
\mathcal{W}(A \Rightarrow B) & := (\mathcal{W}(A) \Rightarrow \mathcal{W}(B)) \\
& \quad \times (\mathcal{C}(B) \Rightarrow \mathcal{W}(A) \Rightarrow \mathcal{M} \mathcal{C}(A)) \\
\mathcal{C}(A \Rightarrow B) & := \mathcal{W}(A) \times \mathcal{C}(B)
\end{align*}
\]
Pédrot’s Dialectica Transformation

**Soundness [Ped14]**

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

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

**A global and a local transformation**

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

\[
\frac{\partial y}{\partial x} \cdot t = \begin{cases} 
  t & \text{if } x = y \\
  0 & \text{otherwise}
\end{cases}
\]

\[
\frac{\partial}{\partial x}(tu) \cdot s = \left( \frac{\partial t}{\partial x} \cdot s \right)u + (Dt \cdot \left( \frac{\partial u}{\partial x} \cdot s \right))u
\]

\[
\frac{\partial}{\partial x}(\lambda y \cdot s) \cdot t = \lambda y \cdot \frac{\partial s}{\partial x} \cdot t
\]

\[
\frac{\partial}{\partial x}(Ds \cdot u) \cdot t = D\left( \frac{\partial s}{\partial x} \cdot t \right) \cdot u + Ds \cdot \left( \frac{\partial u}{\partial x} \cdot t \right)
\]

\[
\frac{\partial 0}{\partial x} \cdot t = 0
\]

\[
\frac{\partial}{\partial x}(s + u) \cdot t = \frac{\partial s}{\partial x} \cdot t + \frac{\partial u}{\partial x} \cdot t
\]

\[
\frac{\partial s}{\partial x} \cdot t
\]
represents \( s \) where \( x \) is linearly (i.e. one time) substituted by \( t \).
The linear substitution ...

The computational Dialectica

\[
\frac{\partial y}{\partial x} \cdot t = \begin{cases} t & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \quad \frac{\partial}{\partial x} (t u) \cdot s = \left( \frac{\partial t}{\partial x} \cdot s \right) u + \left( D t \cdot \left( \frac{\partial u}{\partial x} \cdot s \right) \right) u
\]

\[
x_y \cdot \pi = \begin{cases} \pi & \text{if } x = y \\ \emptyset & \text{otherwise} \end{cases} \quad (t \ u)_y := \lambda \pi. (t_y (u^\bullet, \pi)) \otimes ((t^\bullet.2) \pi \ u^\bullet \gg u_y)
\]

\[
\frac{\partial}{\partial x} (\lambda y.s) \cdot t = \lambda y. \frac{\partial s}{\partial x} \cdot t \quad \frac{\partial}{\partial x} (D s \cdot u) \cdot t = D (\frac{\partial s}{\partial x} \cdot t) \cdot u + D s \cdot \left( \frac{\partial u}{\partial x} \cdot t \right)
\]

\[
\frac{\partial 0}{\partial x} \cdot t = 0 \quad \frac{\partial}{\partial x} (s + u) \cdot t = \frac{\partial s}{\partial x} \cdot t + \frac{\partial u}{\partial x} \cdot t
\]
Soundness [Ped14]

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

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

Tracking differentiation in Dialectica
Soundness [Ped14]

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

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

That’s reverse differentiation

- $(\_)^\bullet.2$ obeys the chain rule, $(\_)^\bullet$ is the functorial differentiation.
- $t_x$ is contravariant in $x$, representing a reverse linear substitution.
Tracking differentiation in Dialectica

Soundness [Ped14]

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

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

That’s reverse differentiation

- $(\_)^\bullet.2$ obeys the chain rule, $(\_)^\bullet$ is the functorial differentiation.
- $t_x$ is contravariant in $x$, representing a reverse linear substitution.

Theorem [K. Pédrot 22]

$$[u \gg= t_x[\Gamma \leftarrow \overrightarrow{r^\bullet}]] \equiv_{\beta, \eta} \lambda z. ([u] ((\partial x.t[\Gamma \leftarrow \overrightarrow{r^\bullet}])z))$$
A Linear Logic Refinement
Differential Linear Logic

\[ \vdash \ell : A \multimap B \quad d \]
\[ \vdash \ell : !A \multimap B \quad d \]

A linear proof

is in particular non-linear.

\[ \vdash f : !A \multimap B \quad \bar{d} \]
\[ \vdash D_0f : A \multimap B \quad \bar{d} \]

From a non-linear proof

we can extract a linear proof

\[ f \in C^\infty(\mathbb{R}, \mathbb{R}) \]

Differential interaction nets, Ehrhard and Regnier, TCS (2006)
Exponential rules of Differential Linear Logic

\[ \vdash \Gamma, \text{cst}_1 : ?A \]
\[ \vdash \Gamma, \delta_0 : !A \]
\[ \vdash \Gamma \]
\[ \vdash \Gamma \]
\[ \vdash \Gamma, f : ?A, g : ?A \]
\[ \vdash \Gamma, f.g : ?A \]
\[ \vdash \Gamma, \phi : !A \]
\[ \vdash \Delta, \psi : !A \]
\[ \vdash \Gamma, \Delta, \psi \ast \phi : !A \]
\[ \vdash \Gamma, x : A \]
\[ \vdash \Gamma, \ell : ?A \]
\[ \vdash \Gamma, \ell : ?A \]
\[ \vdash \Gamma, \ell : ?A \]
\[ \vdash \Gamma, D_0(\cdot)(x) : !A \]
\[ \vdash \Gamma, x : A \]
\[ \vdash \Gamma, x : A \]
\[ ?\Gamma \vdash \delta_x : !A \]

\[ \vdash \Gamma \]

\[ ?\Gamma \vdash x : A \]

\[ \vdash \Gamma \]

\[ \vdash \Gamma \]

\[ \vdash \Gamma \]

\[ \vdash \Gamma \]

\[ \vdash \Gamma \]

\[ \vdash \Gamma \]
Dialectica factorizes through Linear Logic

The call by name arrow

\[ A \Rightarrow B := !A \rightarrow B := (!A) \perp \otimes B \]

\[
\begin{align*}
\mathsf{W}(A \perp) & := \mathsf{C}(A) & \mathsf{C}(A \perp) & := \mathsf{W}(A) \\
\mathsf{W}(A \oplus B) & := \mathsf{W}(A) + \mathsf{W}(B) & \mathsf{C}(A \oplus B) & := \mathsf{C}(A) \times \mathsf{C}(B) \\
\mathsf{W}(!A) & := \mathsf{W}(A) & \mathsf{C}(!A) & := \mathsf{W}(A) \Rightarrow \mathsf{C}(A) \\
\mathsf{W}(A \otimes B) & := \mathsf{W}(A) \times \mathsf{W}(B) & \mathsf{C}(A \otimes B) & := (\mathsf{W}(A) \Rightarrow \mathsf{C}(B)) \times (\mathsf{W}(B) \Rightarrow \mathsf{C}(A))
\end{align*}
\]

Valeria de Paiva, 1989, A dialectica-like model of linear logic.
Dialectica factorizes through Differential Linear Logic

Witnesses are functorial reverse derivative

\[ W(A \Rightarrow B) = (W(A) \Rightarrow W(B)) \times (W(A) \Rightarrow C(B) \Rightarrow C(A)) \]

\[
\begin{align*}
W(!A) & := \! W(A) & C(!A) & := \! W(A) \rightarrow C(A) \\
W(A \otimes B) & := W(A) \otimes W(B) \\
C(A \otimes B) & := (W(A) \rightarrow C(B)) \oplus (W(B) \rightarrow C(A)) \\
W(A \multimap B) & := (W(A) \rightarrow W(B)) \& (C(B) \rightarrow C(A)) \\
C(A \multimap B) & := W(A) \otimes C(B)
\end{align*}
\]

If \( \Gamma \vdash A \) in LL, then \( W(\Gamma) \vdash W(A) \) in classical DiLL.

\[
\begin{array}{c}
\Gamma \vdash A, A \perp \\
\frac{\text{ax}}{\Gamma \vdash A, !A \perp} \\
\frac{\text{ax}}{\Gamma \vdash ?A, !A \perp} \\
\frac{\text{ax}}{\Gamma \vdash ?A, A, !A \perp} \\
\frac{\pi}{\Gamma \vdash ?A} \\
\frac{\text{cut}}{\Gamma \vdash ?A, A}
\end{array}
\]
Dialectica factorizes through Differential Linear Logic

The economical translation

\[
\begin{align*}
[A \Rightarrow B]_e & := !A \multimap B \\
[A \times B]_e & := A \& B \\
[A + B]_e & := A \oplus B
\end{align*}
\]

\[\text{ILL} \xrightarrow{W} \xrightarrow{C} \text{IDiLL}\]

\[\lambda^{+,\times} \xrightarrow{W} \xrightarrow{C} \lambda^{+,\times}\]

IDiLL : Intuitionistic Differential Linear Logic? Oh no ...
Dialectica is differentiation in categories
What’s categorical differentiation?

To cook a good differential category, one needs:

- A category of regular/continuous/non-linear functions
  \[ \mathcal{C}(A, B) = !A \to B. \]

- A category of linear functions, in which differentiation embeds
  \[ \mathcal{L}(A, B) = A \to B. \]

- Something which linearizes:
  \[ \bar{d} : A \to !A \]

- A notion of duality, if one wants to encode reverse differentiation.

\[ \sim \] Basically, one wants a categorical model of DILL.
Dialectica categories

Categories representing specific relations

Consider a category $\mathcal{C}$. $\text{Dial}(\mathcal{C})$ is constructed as follows:

- **Objects**: relations $\alpha \subseteq U \times X$, $\beta \subseteq V \times Y$.
- **Maps from $\alpha$ to $\beta$**:

  $$(f : U \to V, F : U \times Y \to X)$$

- **Composition**: the chain rule!

Consider $$(f, F) : \alpha \subseteq (A, X) \to \beta \subseteq (B, Y)$$ and $$(g, G) : \beta \subseteq (B, Y) \to \gamma \subseteq (C, Z)$$ two arrows of the Dialectica category. Then their composition is defined as $$(g, G) \circ (f, F) := (g \circ f, (a, z) \mapsto G(f(a), F(a, z))).$$
Dialectica categories through Differential Categories

In a \(*\)-autonomous differential category: from \(f : !A \to B\) one constructs:

\[
\vec{D}(f) \in \mathcal{L}(!A \otimes B^\perp, A^\perp).
\]

Dialectica categories factorize through differential categories

If \(\mathcal{L}\) is a model of DiLL such that \(\mathcal{L}!\) has finite limits:

\[
\begin{align*}
\mathcal{L}! &\to \mathcal{D}(\mathcal{L}!) \\
A &\mapsto A \times A^\perp \\
f &\mapsto (f, \vec{D}(f))
\end{align*}
\]

We have an obvious forgetful functor:

\[
\mathcal{U} : \begin{cases}
\mathcal{D}(\mathcal{L}!) &\to \mathcal{L}! \\
\alpha \subseteq A \times X &\mapsto A \\
(f, F) &\mapsto f
\end{cases}
\]

which is left adjoint to \(\mathcal{R}\), forming a reflection on \(\mathcal{L}_{\text{loc}}\).

To be declined in reverse/cartesian differential categories...
Recap

<table>
<thead>
<tr>
<th>Programs</th>
<th>Logic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>fun (x:A)-&gt; (t:B)</td>
<td>Proof of $A \vdash B$</td>
<td>$f : A \rightarrow B$.</td>
</tr>
<tr>
<td>Types</td>
<td>Formulas</td>
<td>Objects</td>
</tr>
<tr>
<td>Execution</td>
<td>Cut-elimination</td>
<td>Equality</td>
</tr>
</tbody>
</table>
Recap

<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>( \text{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>

Differential \( \lambda \)-calculus [Ehr04]

Differential Linear Logic [Ehrhard06]

Vectorial Models

Linear Logic [Gir87]

\( \lambda \)-calculus

Min. Logic

Normal functors
Recap

<table>
<thead>
<tr>
<th>Programs</th>
<th>Logic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fun (x:A)-&gt; (t:B)</code></td>
<td>Proof of $A \vdash B$</td>
<td>$f : A \rightarrow B$.</td>
</tr>
<tr>
<td>Types</td>
<td>Formulas</td>
<td>Objects</td>
</tr>
<tr>
<td>Execution</td>
<td>Cut-elimination</td>
<td>Equality</td>
</tr>
</tbody>
</table>

- Differential $\lambda$-calculus [Ehr04]
- Differential Linear Logic [Ehrhard06]
- Vectorial Models
- Linear Logic [Gir87]
- Dialectica [Göd58]
- Automatic Differentiation [80s]
- $\lambda$-calculus
- Min. Logic
- Normal functors

Linear Logic [Gir87]

32 / 38
Recap

<table>
<thead>
<tr>
<th>Programs</th>
<th>Logic</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{fun (x:A)-&gt; (t:B)}$</td>
<td>Proof of $A \vdash B$</td>
<td>$f : A \rightarrow B$.</td>
</tr>
<tr>
<td>Types</td>
<td>Formulas</td>
<td>Objects</td>
</tr>
<tr>
<td>Execution</td>
<td>Cut-elimination</td>
<td>Equality</td>
</tr>
</tbody>
</table>

Differentiable Programming

Differential $\lambda$-calculus [Ehr04]

Differential Linear Logic [Ehrhard06]

Linear Logic [Gir87]

Dialectica [Göd58]

Automatic Differentiation [80s]

$\lambda$-calculus

Min. Logic

Vectorial Models

Normal functors

A good point for logicians: Gödel invented Dialectica 40 years before reverse differentiation was put to light
## Recap

<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>

### Diagram

- **Differentiable Programming**
  - **Differential \( \lambda \)-calculus** [Ehr04]
  - **Differential Linear Logic** [Ehrhard06]
  - **Vectorial Models**
  - **Linear Logic** [Gir87]

- **Automatic Differentiation** [80s]
- **Dialectica** [Göd58]
- **Min. Logic**
- **Normal functors**
- **\( \lambda \)-calculus**
Conclusion and applications
Take home message:

**Dialectica is functorial reverse differentiation,**
extracting intensional local content from proofs.

A new semantical correspondance between computations and mathematics: *intentional meaning* of program is *local behaviour* of functions.

<table>
<thead>
<tr>
<th>Program</th>
<th>Proof</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantitative Resources</td>
<td>Classical Principles</td>
<td>Linearity</td>
</tr>
<tr>
<td>Control</td>
<td>Intentional</td>
<td>Differentiation</td>
</tr>
<tr>
<td>Extentional</td>
<td></td>
<td>Local</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Global</td>
</tr>
</tbody>
</table>

Related work and applications:

- Markov’s principle and delimited continuations on positive formulas.
- Proof mining and backpropagation.
Dialectica is differentiation ...

... We knew it already!

The codereliction of differential proof nets: In terms of polarity in linear logic [23], the \( \forall \rightarrow \)-free constraint characterizes the formulas of intuitionistic logic that can be built only from positive connectives (\( \oplus, \otimes, 0, 1, ! \)) and the why-not connective ("?"). In this framework, Markov’s principle expresses that from such a \( \forall \rightarrow \)-free formula \( A \) (e.g. \( ? \oplus_x (?A(x) \otimes ?B(x)) \)) where the presence of "?" indicates that the proof possibly used weakening (\texttt{efq} or \texttt{throw}) or contraction (\texttt{catch}), a linear proof of \( A \) purged from the occurrences of its "?" connective can be extracted (meaning for the example above a proof of \( \oplus_x (A(x) \otimes B(x)) \)).

Interestingly, the removal of the "?", i.e. the steps from \( ?P \) to \( P \), correspond to applying the codereliction rule of differential proof nets [24].

Differentiation : \( (?P = (P \rightarrow \bot) \Rightarrow \bot) \rightarrow ((P \rightarrow \bot) \rightarrow \bot) \equiv P \)
Markov’s principle is proved by allowing catch and throw operations on hereditary positive formulas.

\[
\begin{align*}
\frac{a : \neg
\neg T \vdash \alpha : T \quad a : \neg
\neg T}{\text{AXIOM}} & \quad \frac{b : T \vdash \alpha : T \quad b : T}{\text{AXIOM}} \\
\frac{b : T \vdash \alpha : T \quad \text{throw}_\alpha b : \bot}{\text{THROW}} & \quad \frac{\vdash \alpha : T \quad \lambda b . \text{throw}_\alpha b : \neg T}{\rightarrow_I} \\
\frac{a : \neg \neg T \vdash \alpha : T \quad a (\lambda b . \text{throw}_\alpha b) : \bot}{\vdash E} & \quad \frac{\vdash E}{\rightarrow_E} \\
\frac{a : \neg \neg T \vdash \alpha : T \quad \text{efq} a (\lambda b . \text{throw}_\alpha b) : T}{\rightarrow_I} & \quad \frac{\rightarrow_I}{\downarrow E} \\
\frac{a : \neg \neg T \vdash \text{catch}_\alpha \text{efq} a (\lambda b . \text{throw}_\alpha b) : T}{\rightarrow_I} & \quad \frac{\rightarrow_I}{\rightarrow_I} \\
\vdash \lambda a . \text{catch}_\alpha \text{efq} a (\lambda b . \text{throw}_\alpha b) : \neg \neg T \rightarrow T
\end{align*}
\]

Figure 3. Proof of MP
Proof Mining

Extracting quantitative information from proofs.

Effective moduli from ineffective uniqueness proofs. An unwinding of
de La Vallée Poussin’s proof for Chebycheff approximation∗

Ulrich Kohlenbach
Fachbereich Mathematik, J.W. Goethe Universität
Robert–Mayer–Str. 6/10, 60000 Frankfurt am Main, FRG

Abstract
We consider uniqueness theorems in classical analysis having the form

\[(+) \forall u \in U, v_1, v_2 \in V_u \left( G(u, v_1) = 0 = G(u, v_2) \rightarrow v_1 = v_2 \right),\]

where \(U, V\) are complete separable metric spaces, \(V_u\) is compact in \(V\) and \(G : U \times V \rightarrow \mathbb{R}\) is a
constructive function.

If \((+\) is proved by arithmetical means from analytical assumptions

\[(++ \forall x \in X \exists y \in Y \forall z \in Z \left( F(x, y, z) = 0 \right)\]

only (where \(X, Y, Z\) are complete separable metric spaces, \(Y_x \subset Y\) is compact and
\(F : X \times Y \times Z \rightarrow \mathbb{R}\) constructive), then we can extract from the proof of \((++ \rightarrow (+) an
effective modulus of uniqueness, i.e.

\[(+++ \forall u \in U, v_1, v_2 \in V_u, k \in \mathbb{N} \left( |G(u, v_1)|, |G(u, v_2)| \leq 2 \overset{\text{a.k.}}{\rightarrow} d_V(v_1, v_2) \leq 2^{-k} \right).\]
Proof Mining

Extracting quantitative information from proofs.

∀u, v₁v₂, Pol(u, v₁) = Pol(u, v₂) → v₁ = v₂

⇓

∀u, v₁v₂, ∀ε > 0, ∃η > 0, ∥G(u, v₁) − G(u, v₂)∥ < η → dᵥ(v₁, v₂) < ε

⇓

∃ϕ, ∀u, k, v₁v₂, ∥G(u, v₁) − G(u, v₂)∥ < 2⁻ϕ(u,k) → dᵥ(v₁, v₂) < 2⁻k.
Proof Mining

Markov’s principle and the independence of premises are necessary for most of mathematical analysis proofs:

Proof mining allows to refine these proofs by taking away these principles as guaranteed by (some variant of) Dialectica’s transformation.

Conjecture

Does it differentiate the function \((\epsilon \rightarrow \eta)\) in:

\[
\forall u, v_1 v_2, \forall \epsilon > 0, \exists \eta > 0, \|G(u, v_1) - G(u, v_2)\| < \eta \rightarrow d_{V}(v_1, v_2) < \epsilon
\]

Is proof mining (based on) reverse differentiation applied to proofs?

What else can we explain by differentiation?