LICS 2024

∂ is for Dialectica

Marie Kerjean, Pierre-Marie Pédrot

CNRS & Inria

What this talk is about

Joining Dialectica and Differentiation

September 2015 in Paris, about Pédrot's thesis:
"that's (just) differential lambda-calculus"

What this talk is about

Joining Dialectica and Reverse Differentiation

June 2019 in Nantes, about "λ the ultimate backpropagator":
"That's (just) Dialectica"

What this talk is about

Joining Dialectica and Differential λ-calculus
through
Reverse Differentiation

June 2024 in Tallinn

What this talk is not about

- Joining Dialectica and Differential Linear Logic through Reverse Differentiation
- Joining Dialectica and Differential Categories through Reverse Differentiation

Gödel's Dialectica Transformation

1. $(\mathrm{F} \wedge \mathrm{G})' = (\exists yv)(zw)[\mathrm{A}(y, z, x) \wedge \mathrm{B}(v, w, u)]$.
2. $(\mathrm{F} \vee \mathrm{G})' = (\exists yvt)(zw)[t=0 \wedge \mathrm{A}(y, z, x) \cdot\vee\cdot t=1 \wedge \mathrm{B}(v, w, u)]$.
3. $[(s)\mathrm{F}]' = (\exists \mathrm{Y})(sz)\mathrm{A}(\mathrm{Y}(s), z, x)$.
4. $[(\exists s)\mathrm{F}]' = (\exists sy)(z)\mathrm{A}(y, z, x)$.
5. $(\mathrm{F} \supset \mathrm{G})' = (\exists \mathrm{VZ})(yw)[\mathrm{A}(y, \mathrm{Z}(yw), x) \supset \mathrm{B}(\mathrm{V}(y), w, u)]$.
6. $(\neg \mathrm{F})' = (\exists \bar{\mathrm{Z}})(y) \neg \mathrm{A}(y, \bar{\mathrm{Z}}(y), x)$.

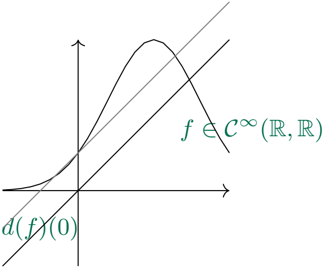
Kurt Gödel (1958). Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. Dialectica.

- Validates semi-classical axioms:
  - Markov's principle : $\neg\neg\exists x A \to \exists x A$ when A is decidable.
- Numerous applications :
  - Soundness results
  - **Proof mining**: applying Dialectica to theorems in analysis extract quantitative information.
- Reformulated through Linear logic, or Dialectica Categories

V. de Paiva. 1989. A Dialectica-like Model of Linear Logic.,

Differentiation

- Differentiation is finding the best linear approximation to a function at a point.



For $f : \mathbb{R} \to \mathbb{R}$, $x \in \mathbb{R}$
$D_x f : v \in \mathbb{R} \mapsto f'(x) \cdot v \in \mathbb{R}$

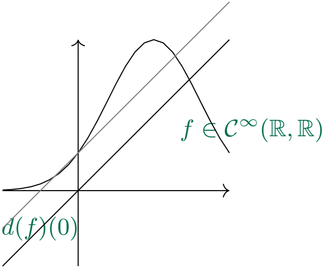
For $f : E \to F$, $x \in E$
$D_x f : v \in E \mapsto D_x f(v) \in F$

$$\text{Chain Rule}: \ D_x(f \circ g) = D_{g(x)} f \circ D_x g$$

- Differentiation is a mathematical operation which needs to be fitted to logical and computer science use.
  - Algorithmic Differentiation : differentiating sequences of many-valued functions efficiently.
  - Differential Linear Logic and Differential λ-calculus : Differentiating proofs and λ-terms.

Differentiation

- Differentiation is finding the best linear approximation to a function at a point.



For $f : \mathbb{R} \to \mathbb{R}$, $x \in \mathbb{R}$
$D_x f : v \in \mathbb{R} \mapsto f'(x) \cdot v \in \mathbb{R}$

For $f : E \to F$, $x \in E$
$D_x f : v \in E \mapsto D_x f(v) \in F$

Chain Rule : $D_x(f \circ g) = D_{g(x)} f \circ D_x g$

- Differentiation is a mathematical operation which needs to be fitted to logical and computer science use.
  - Algorithmic Differentiation : differentiating sequences of many-valued functions efficiently.
  - Differential Linear Logic and Differential λ-calculus : Differentiating proofs and λ-terms.

A peek into Dialectica interpretation of functions

$$(A \to B)_D = \exists f g \forall x y (A_D(x, gxy) \to B_D(fx, y))$$

Question: $(A \Rightarrow B)_D; (B \Rightarrow C)_D \rightsquigarrow (A \Rightarrow C)_D$?
Usual explanation : least unconstructive prenexation.

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

- Variables f agree to the usual composition rule.
- Variables g agree to a chain rule: $g_3(x, y) = g_1(x, g_2(f_1 x, y))$

Mathematical meaning : it's some kind of approximation.

Algorithmic Differentiation

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

E.g. : $z = y + cos(x^2)$

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

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*.

Typing Algorithmic Differentiation

Algorithmic differentiation:
making a choice when computing the chain rule: $\mathrm{D}_u(f \circ g) = \mathrm{D}_{g(u)} f \circ \mathrm{D}_u(g)$

Typing Forward Mode differentiation :

$$g : E \Rightarrow F \rightsquigarrow \overrightarrow{D} g : E \Rightarrow E \multimap F.$$

The Linear negation: $A^{\perp} \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R}) \equiv A'$

Typing Reverse Mode differentiation

$$g(u) \to f(g(u)) \to D_{g(u)} f \to D_{g(u)} f \circ D_u(g)$$

$$g : E \Rightarrow F \rightsquigarrow \overleftarrow{D} g : E \Rightarrow F^{\perp} \multimap E^{\perp}; \ell \mapsto \ell \circ D_u g$$

Brunel, Mazza, Pagani. Backpropagation in the simply typed λ-calc. with linear negation.

Reverse differentiation : $(g, \overleftarrow{D}(g)) : (E \Rightarrow F) \times (E \Rightarrow F^{\perp} \multimap E^{\perp})$

Typing Algorithmic Differentiation

Algorithmic differentiation:
making a choice when computing the chain rule: $\mathrm{D}_u(f \circ g) = \mathrm{D}_{g(u)} f \circ \mathrm{D}_u(g)$

Typing Forward Mode differentiation :

$$g : E \Rightarrow F \rightsquigarrow \overrightarrow{D} g : E \Rightarrow E \multimap F.$$

The Linear negation: $A^\perp \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R}) \equiv A'$

Typing Reverse Mode differentiation

$$g(u) \to f(g(u)) \to D_{g(u)} f \to D_{g(u)} f \circ D_u(g)$$

$$g : E \Rightarrow F \rightsquigarrow \overleftarrow{D} g : E \Rightarrow F^\perp \multimap E^\perp ; \ell \mapsto \ell \circ D_u g$$

Brunel, Mazza, Pagani. Backpropagation in the simply typed λ-calc. with linear negation.

Reverse differentiation : $(g, \overleftarrow{D}(g)) : (E \Rightarrow F) \times (E \Rightarrow F^\perp \multimap E^\perp)$

Typing Algorithmic Differentiation

Algorithmic differentiation:
making a choice when computing the chain rule: $\mathrm{D}_u(f \circ g) = \mathrm{D}_{g(u)} f \circ \mathrm{D}_u(g)$

Typing Forward Mode differentiation :

$$g : E \Rightarrow F \rightsquigarrow \overrightarrow{D} g : E \Rightarrow E \multimap F.$$

The Linear negation: $A^{\perp} \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R}) \equiv A'$

Typing Reverse Mode differentiation

$$g(u) \to f(g(u)) \to D_{g(u)} f \to D_{g(u)} f \circ D_u(g)$$

$$g : E \Rightarrow F \rightsquigarrow \overleftarrow{D} g : E \Rightarrow F^{\perp} \multimap E^{\perp}; \ell \mapsto \ell \circ D_u g$$

Brunel, Mazza, Pagani. Backpropagation in the simply typed λ-calc. with linear negation.

Reverse differentiation : $(g, \overleftarrow{D}(g)) : (E \Rightarrow F) \times (E \Rightarrow F^{\perp} \multimap E^{\perp})$

Types !

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

$$A \rightsquigarrow \exists \overbrace{x : \mathbb{W}(A)}^{\text{witness}}, \forall \underbrace{u : \mathbb{C}(A)}_{\text{opponent}}, A_D[x, u]$$

Witness and counter types :

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

Reverse Mode differentiation:

$$\text{Functorial : } (h, \overleftarrow{D}h) : (A \Rightarrow B) \times (A \Rightarrow B^{\perp} \multimap A^{\perp})$$

However:

- Having the same type does not mean you're the same program.
- Some french (linear) logicians have a strong opinion on what proof/program differentiation should be.

Types !

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

$$A \rightsquigarrow \exists \overbrace{x : \mathbb{W}(A)}^{\text{global witness}}, \forall \underbrace{u : \mathbb{C}(A)}_{\text{local opponent}}, A_D[x, u]$$

Witness and counter for implication types :

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

$$\mathbb{W}(A \Rightarrow B) = \overbrace{(\mathbb{W}(A) \Rightarrow \mathbb{W}(B))}^{\text{function}} \times \left(\mathbb{W}(A) \Rightarrow \underbrace{\mathbb{C}(B) \Rightarrow \mathbb{C}(A)}_{\text{reverse derivative}} \right)$$

Reverse Mode differentiation:

$$\text{Functorial : } (h, \overleftarrow{D}h) : (A \Rightarrow B) \times (A \Rightarrow B^{\perp} \multimap A^{\perp})$$

However:

- Having the same type does not mean you're the same program.
- Some french (linear) logicians have a strong opinion on what proof/program differentiation should be.

$$A \rightsquigarrow \exists \overbrace{x : \mathbb{W}(A)}^{\text{witness}}, \forall \underbrace{u : \mathbb{C}(A)}_{\text{opponent}}, A_D[x, u]$$

Let's say x, u, f, g are λ-terms.

The computational Dialectica : a reverse Differential λ-calculus

"Behind every successful proof there is an exhausted program"

Pédrot's Dialectica Transformation

Making Dialectica act on λ-terms instead of formulas.

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 $x : X \in \Gamma$.

A global and a local transformation

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

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

Differential λ-calculus

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:

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

Operational Semantics:

$$\begin{array}{c} (\lambda x.s)T \to_\beta s[T/x] \\ \mathrm{D}(\lambda x.s) \cdot t \to_{\beta_D} \lambda x.\frac{\partial s}{\partial x} \cdot t \end{array}$$

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

T. Ehrhard, L. Regnier. The differential lambda-calculus. TCS, 2004

See the Alonzo Church award' talk on Wednesday !

The linear substitution ...

... which is not exactly a substitution

$$\frac{\partial y}{\partial x} \cdot t = \begin{cases} t \; if \; x = y \\ 0 \; otherwise \end{cases} \qquad \frac{\partial}{\partial x}(tu) \cdot s = (\frac{\partial t}{\partial x} \cdot s)u + (\mathrm{D}t \cdot (\frac{\partial u}{\partial x} \cdot s))u$$

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

$$\frac{\partial 0}{\partial x} \cdot t = 0 \qquad \frac{\partial}{\partial x}(s + u) \cdot t = \frac{\partial s}{\partial x} \cdot t + \frac{\partial u}{\partial x} \cdot t$$

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

The linear substitution ...

The computational Dialectica

$$\frac{\partial y}{\partial x} \cdot t = \begin{cases} t \; if \; x = y \\ 0 \; otherwise \end{cases} \qquad \frac{\partial}{\partial x}(tu) \cdot s = (\frac{\partial t}{\partial x} \cdot s)u + (\mathrm{D}t \cdot (\frac{\partial u}{\partial x} \cdot s))u$$

$$x_y \cdot \pi = \begin{cases} \pi \; if \; x = y \\ \emptyset \; otherwise \end{cases} \qquad (t\ u)_y := \lambda\pi.\, (t_y\, (u^\bullet, \pi)) \circledast ((t^\bullet.2)\, \pi\, u^\bullet \gg\!= u_y)$$

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

$$\frac{\partial 0}{\partial x} \cdot t = 0 \qquad \frac{\partial}{\partial x}(s + u) \cdot t = \frac{\partial s}{\partial x} \cdot t + \frac{\partial u}{\partial x} \cdot t$$

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 $x : X \in \Gamma$.

That's reverse differentiation [KP24]

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

Other formulations:

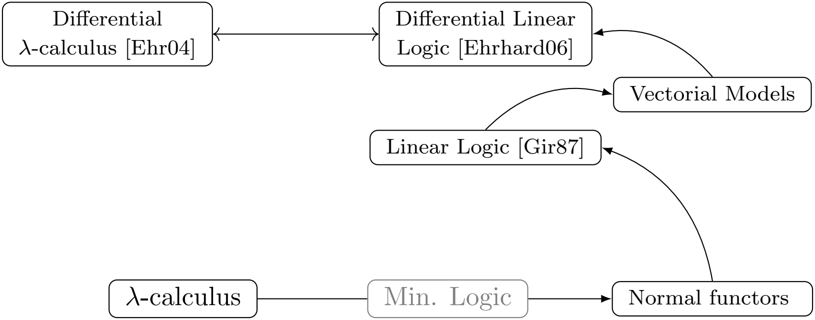
- The Linear Dialectica and Differential Linear Logic
- Dialectica Categories and Differential Categories

Recap

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

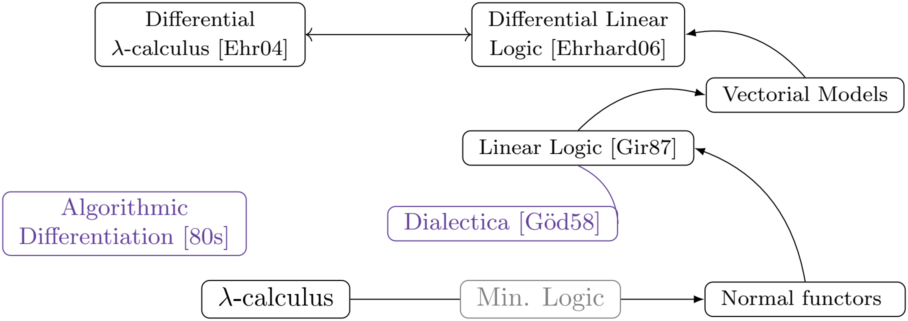
Recap

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



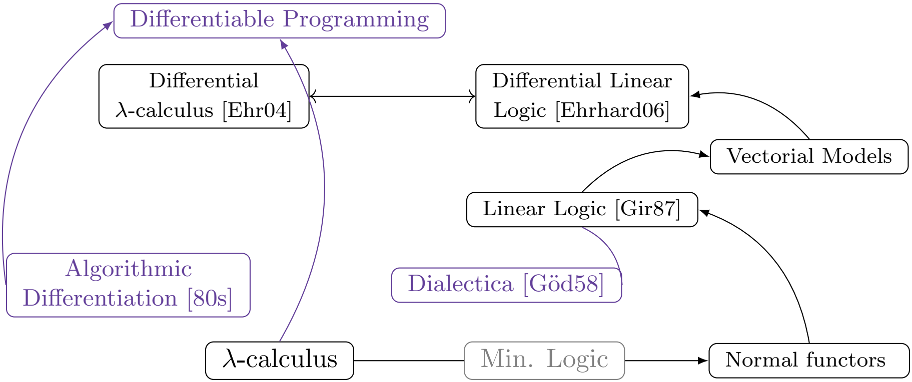
Recap

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



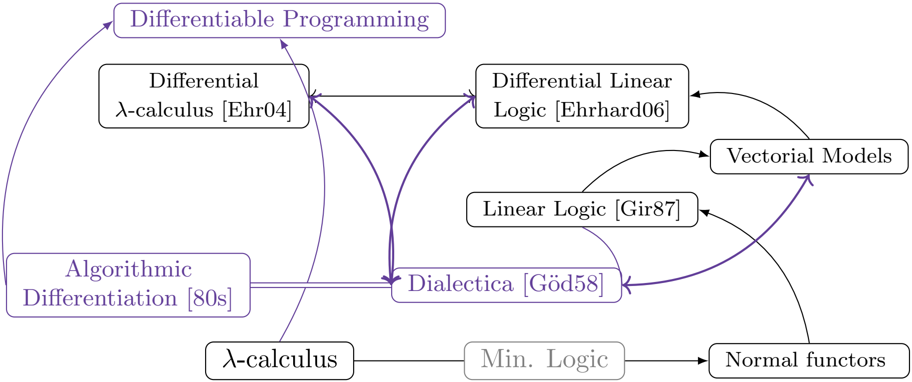
Recap

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



Recap

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



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

Conclusion and applications

Take home message:

Dialectica computes higher-order functorial reverse differentiation,
extracting ~~intensional~~ local content from proofs.

A new semantical correspondence between computations and mathematics :
intentional meaning of program is local behavior of functions.

Is this result obvious? Maybe, and I'm happy if it is.

Related work and potential applications:

- Markov's principle and delimited continuations on positive formulas.
- Proof mining and backpropagation.
- Bar Induction and Taylor Exponentiation.

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 (`efq` or `throw`) or contraction (`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 : $(?P = (P \multimap \bot) \Rightarrow \bot) \rightarrow ((P \multimap \bot) \multimap \bot) \equiv P)$

Hugo Herbelin, "An intuitionistic logic that proves Markov's principle", LICS '10 .

Are mathematical transformation realizing the axioms they need ?