Computability in Europe 2023
Special session on Proof Theory

∂ is for Dialectica

Marie Kerjean

CNRS & LIPN, Université Sorbonne Paris Nord

Work in collaboration with Pierre-Marie Pédrot

Gödel's Dialectica Transformation

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

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

- De Paiva [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.

Gödel's Dialectica

1. $(F \wedge G)' = (\exists yv)(zw)[A(y,z,x) \wedge B(v,w,u)]$.
2. $(F \vee G)' = (\exists yvt)(zw)[t=0 \wedge A(y,z,x) \cdot \vee \cdot t=1 \wedge B(v,w,u)]$.
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. $(\neg F)' = (\exists \bar{Z})(y) \neg A(y,\bar{Z}(y),x)$.

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

Gödel's Dialectica

- Validates semi-classical axioms:
  - Markov's principle : $\neg\neg\exists x A \to \exists x A$ when A is decidable.
  - Independant of premises : $(A \to \exists x B) \to (\exists x.(A \to B))$
- Numerous applications :
  - Soudness 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 φ and $\varphi \to \psi$ are combined to yield a proof of ψ, 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.

Jeremy Avigad and Solomon Feferman (1999). Gödel's functional ("Dialectica") interpretation

A peek into Dialectica interpretation of functions

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

Usual explanation : least unconstructive prenexation.

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

Dynamic behaviour : agrees to a chain rule.

Mathematical meaning : it's some kind of approximation.

Ulrich Kohlenbach, Applied Proof Theory: Proof Interpretations and their Use in Mathematics, 2008

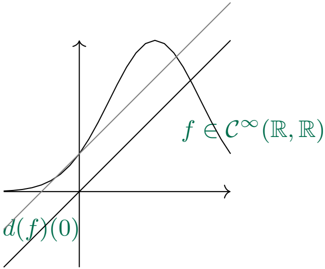
Outline of the talk

- The Historical Dialectica
- Differentiation and Differentiable Programming.
- Factorizing Dialectica through differential linear logic.
- Dialectica acting on λ-terms.
- Applications and related work.

Differentiable Programming

Differentiation

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

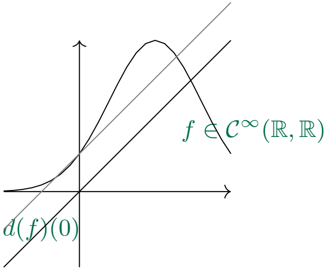


$$\text{Chain Rule}: \; D_0(f \circ g) = D_{g(0)} f \circ D_0 g$$

- 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 : Differentiating proofs and λ-terms.

Differentiation

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



$$\text{Chain Rule} : D_0(f \circ g) = D_{g(0)} f \circ D_0 g$$

- 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 : Differentiating proofs and λ-terms.

Dialectica verifies the chain rule

Composing the Dialectica interpretation of arrows:

$$(A \Rightarrow B)_D[\phi_1; \psi_1, u_1; v_1] := A_D(u_1, \psi_1\, u_1\, v_1) \Rightarrow B_D(\phi_1\, u_1, v_1)$$
$$(B \Rightarrow C)_D[\phi_2; \psi_2, u_2; v_2] := B_D(u_2, \psi_2\, u_2\, v_2) \Rightarrow C_D(\phi_2\, u_2, v_2)$$
$$(A \Rightarrow C)_D[\phi_3; \psi_3, u_3; v_3] := A_D(u_3, \psi_3\, u_3\, v_3) \Rightarrow C_D(\phi_3\, u_3, v_3)$$

The Dialectica interpretation amounts to the following equations:

$$u_3 = u_1 \qquad \psi_3, u_3, v_3 = \psi_1, u_1, v_1$$
$$v_3 = v_2 \qquad \phi_2\, u_2 = \phi_1, u_1$$
$$u_2 = \phi_1\, u_1 \qquad v_1 = \psi_2(u_2, v_2)$$

which can be simplified to:

$$\phi_3(u_3) = \phi_2\,(\phi_1\,(u_3)) \textit{ composition of functions}$$
$$\psi_3(u_3, v_3) = \psi_1\,(u_3, \psi_2(\phi_1 u_3, v_3)) \textit{ composition of their differentials}$$

Thanks to T. Powell for noticing typos here.

But verifying the chain rule does not make you differentiation!

- More modern presentations of Dialectica.
- More Computer Science Friendly presentations of Differentiation.
- Linearity must enter the game.

Curry-Howard for 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
	Dialectica	
Differential λ-calculus	Differential Linear Logic	Differential Categories

Dialectica is Backward Differentiation in Logic

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

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*.

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

- Programs acts on programs.
  - Functions are higher-order: they act not only on $\mathbb{R}^n$, but also on $\mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$.
- Programs are typed.
  - $Add : \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}) \times \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R}) \to \mathcal{C}^\infty(\mathbb{R}^n, \mathbb{R})$
- Everything is interpreted in Categories.
  - Objects are Data
  - Functions are Programs
  - Transformations are functorial:

$$\mathcal{F}(p_1; p_2) = \mathcal{F}(p_1); \mathcal{F}(p_2)$$

$$\mathcal{F}(f_2 \circ f_1) = \mathcal{F}(f_2) \circ \mathcal{F}(f_1)$$

Back to AD: I hate graphs

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

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

The choice of an algorithm is due to complexity considerations:

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

$\leadsto$ *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.

AD from a functorial point of view

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

Non-functorial !!!

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

$$f : E \Rightarrow F$$

Forward Mode differentiation :

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

$$\overrightarrow{D}(f) : \begin{cases} E \Rightarrow E \multimap F \\ u \mapsto v \mapsto D_u(f)(v) \end{cases}$$

Functorial forward differentiation :

$$(f, \overrightarrow{D}(f)) : \begin{cases} E \times E \to F \times F \\ (a, x) \mapsto (f(a), (\mathrm{D}_a f \cdot x)) \end{cases}$$

Reverse AD from a functorial point of view

How to make **reverse** differentiation functorial ?

Make it act on pairs with linear duals !

Reverse functorial differentiation

Linear Dual

$$A^{\perp} \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R})$$

- **Reverse Mode differentiation:**

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

$$f : E \Rightarrow F \rightsquigarrow \overleftarrow{D}f : E \Rightarrow F^{\perp} \Rightarrow E^{\perp}.$$

$$\overleftarrow{D}(f) : \begin{cases} E \Rightarrow F^{\perp} \multimap E^{\perp} \\ u \mapsto \ell \mapsto \ell \circ D_u(f) \end{cases}$$

[Mazza, Pagani, POPL2020]

- **Reverse functorial differentiation :**

$$(f, \overleftarrow{D}(f)) : (E \Rightarrow F) \times (E \Rightarrow F^{\perp} \Rightarrow E^{\perp})$$

Reverse functorial differentiation

Linear Dual

$$A^{\perp} \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R})$$

- **Reverse Mode differentiation:**

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

$$f : E \Rightarrow F \rightsquigarrow \overleftarrow{D} f : E \Rightarrow F^{\perp} \Rightarrow E^{\perp}.$$

$$\overleftarrow{D}(f) : \begin{cases} E \Rightarrow F^{\perp} \multimap E^{\perp} \\ u \mapsto \ell \mapsto \ell \circ D_u(f) \end{cases}$$

[Mazza, Pagani, POPL2020]

- **Reverse functorial differentiation :**

$$(f, \overleftarrow{D}(f)) : (E \Rightarrow F) \times (E \Rightarrow F^{\perp} \Rightarrow E^{\perp})$$

Reverse functorial differentiation

Linear Dual

$$A^{\perp} \equiv A \multimap \perp \equiv \mathcal{L}(A, \mathbb{R})$$

- **Reverse Mode differentiation:**

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

$$f : E \Rightarrow F \rightsquigarrow \overleftarrow{D}f : E \Rightarrow F^{\perp} \Rightarrow E^{\perp}.$$

$$\overleftarrow{D}(f) : \begin{cases} E \Rightarrow F^{\perp} \multimap E^{\perp} \\ u \mapsto \ell \mapsto \ell \circ D_u(f) \end{cases}$$

[Mazza, Pagani, POPL2020]

- **Reverse functorial differentiation :**

$$(f, \overleftarrow{D}(f)) : (E \Rightarrow F) \times (E \Rightarrow F^{\perp} \Rightarrow E^{\perp})$$

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 differentiation should.

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 differentiation should.

A Linear Logic Refinement

Curry-Howard for semantics

The syntax mirrors the semantics.

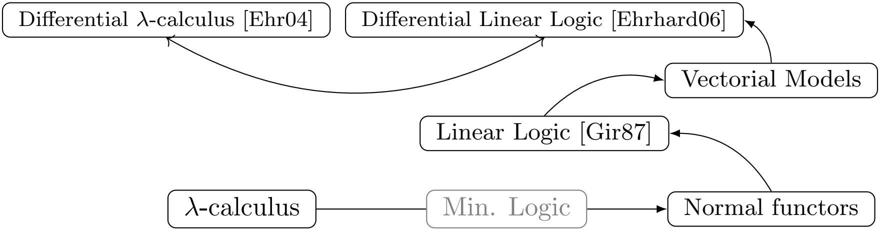
Programs	**Logic**	**Semantics**
`fun (x:A)-> (t:B)`	Proof of $A \vdash B$	$f : A \to B$.
Types	Formulas	Objects
Execution	Cut-elimination	Equality

Doing to proofs everything we do to functions

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



Doing to proofs everything we do to functions

Linear Logic

Usual Implication

Linear and Non Linear Arrows

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

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

- Notions of ressources
  which have made their way into programmation through linear types.
- The dynamics of linearity gets encoded through the rules of the $!$ connective, and its dual $?$.

$$A, B := A \otimes B | A \parr B | A \oplus B | A \,\&\, B | !A | ?A$$

Linear Logic

Usual implication

Linear and Non Linear Arrows

Linear 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.

- Notions of ressources which have made their way into programmation through linear types.
- The dynamics of linearity gets encoded through the rules of the ! connective, and its dual ?.

$$A, B := A \otimes B | A \parr B | A \oplus B | A \,\&\, B | !A | ?A$$

Linear Logic

Usual implication

Linear and Non Linear Arrows

Linear Implication

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

Exponential

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

- Notions of ressources
  which have made their way into programmation through linear types.
- The dynamics of linearity gets encoded through the rules of the ! connective, and its dual ?.

$$A, B := A \otimes B | A \parr B | A \oplus B | A \,\&\, B | !A | ?A$$

Dialectica factorizes through Linear Logic

The call by name arrow

$$A \Rightarrow B := {!A} \multimap B := (({!A}) \otimes B^{\perp})^{\perp}$$

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\mathbb{W}(A^{\perp}) & := & \mathbb{C}(A) & \mathbb{C}(A^{\perp}) & := & \mathbb{W}(A) \\
\mathbb{W}(!A) & := & \mathbb{W}(A) & \mathbb{C}(!A) & := & \mathbb{W}(A) \Rightarrow \mathbb{C}(A)
\end{array}$$

$$\begin{array}{lcl}
\mathbb{W}(A \otimes B) & := & \mathbb{W}(A) \times \mathbb{W}(B) \\
\mathbb{C}(A \otimes B) & := & (\mathbb{W}(A) \Rightarrow \mathbb{C}(B)) \times (\mathbb{W}(B) \Rightarrow \mathbb{C}(A))
\end{array}$$

$$\begin{tikzcd}
 & \mathrm{LL} \arrow[dr, "\mathbb{W}\ \mathbb{C}"] & \\
\lambda^{+,\times} \arrow[ur, "{[\![-]\!]_e}"] \arrow[rr, "\mathbb{W}\qquad\mathbb{C}"'] & & \lambda^{+,\times}
\end{tikzcd}$$

Valeria de Paiva, 1989, A dialectica-like model of linear logic.

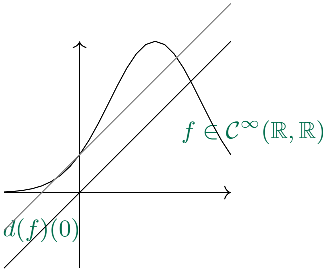
Differential Linear Logic

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

A linear proof
is in particular non-linear.

$$\frac{\vdash f : !A \multimap B}{\vdash D_0 f : A \multimap B}\ \bar{d}$$

From a non-linear proof
we can extract a linear proof



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

Exponential rules of Differential Linear Logic

Exponential connectives:

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

$$\frac{\vdash \Gamma}{\vdash \Gamma, cst_1 : ?A}\, w \qquad \frac{\vdash \Gamma, f : ?A, g : ?A}{\vdash \Gamma, f.g : ?A}\, c \qquad \frac{\vdash \Gamma, \ell : A}{\vdash \Gamma, \ell : ?A}\, d$$

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

$$\frac{?\Gamma \vdash x : A}{?\Gamma \vdash \delta_x : !A}\, p$$

Differentiation in Differential Linear Logic

The only thing you need to know:

$$\dfrac{\vdash \Gamma, \delta_u : !A \qquad \dfrac{\vdash \Gamma, v : A}{\vdash \Gamma, D_0(\_)(v) : !A}\,\bar{d}}{\vdash \Gamma, \Delta, D_u(\_)(v) : !A}\,\bar{c}$$

Dialectica factorizes through Differential Linear Logic

Witnesses are functorial reverse derivative

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

$$\begin{array}{lcl}
\mathbb{W}(A \otimes B) & := & \mathbb{W}(A) \otimes \mathbb{W}(B) \\
\mathbb{W}(A \multimap B) & := & (\mathbb{W}(A) \multimap \mathbb{W}(B)) \\
 & & \&(\mathbb{C}(B) \multimap \mathbb{C}(A)) \\
\mathbb{W}(A \,\&\, B) & := & \mathbb{W}(A) \,\&\, \mathbb{W}(B) \\
\mathbb{W}(A \oplus B) & := & \mathbb{W}(A) \oplus \mathbb{W}(B) \\
\mathbb{W}(!A) & := & !\mathbb{W}(A)
\end{array}
\qquad
\begin{array}{lcl}
\mathbb{C}(A \otimes B) & := & (\mathbb{W}(A) \multimap \mathbb{C}(B)) \\
 & & \oplus(\mathbb{W}(B) \multimap \mathbb{C}(A)) \\
\mathbb{C}(A \multimap B) & := & \mathbb{W}(A) \otimes \mathbb{C}(B) \\
\mathbb{C}(A \,\&\, B) & := & \mathbb{C}(A) \oplus \mathbb{C}(B) \\
\mathbb{C}(A \oplus B) & := & \mathbb{C}(A) \,\&\, \mathbb{C}(B) \\
\mathbb{C}(!A) & := & !\mathbb{W}(A) \multimap \mathbb{C}(A)
\end{array}$$

If $\Gamma \vdash A$ in LL, then $\mathbb{W}(\Gamma) \vdash \mathbb{W}(A)$ in classical DiLL.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash A, A^{\perp}}\,\mathrm{ax}}{\vdash A, !A^{\perp}}\,\bar{d} \qquad \dfrac{}{\vdash ?A, !A^{\perp}}\,\mathrm{ax}}{\vdash ?A, A, !A^{\perp}}\,\bar{c} \qquad \dfrac{\pi}{\Gamma \vdash ?A}}{\Gamma \vdash ?A, A}\,\mathrm{cut}$$

Dialectica factorizes through Differential Linear Logic

The economical translation

$$\llbracket A \Rightarrow B \rrbracket_e := !A \multimap B$$
$$\llbracket A \times B \rrbracket_e := A \,\&\, B$$
$$\llbracket A + B \rrbracket_e := A \oplus B$$

$$\begin{tikzcd}
\mathsf{ILL} \arrow[r, "\mathbb{W} \quad \mathbb{C}"] & \mathsf{IDiLL} \arrow[d, "\cdots"] \\
\lambda^{+,\times} \arrow[u, "\llbracket - \rrbracket_e"] \arrow[r, "\mathbb{W} \quad \mathbb{C}"'] & \lambda^{+,\times}
\end{tikzcd}$$

IDILL : Intuitionnistic Differential Linear Logic ? Oh no ...

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

Let's say x, u, f, g are λ-terms.

The computational Dialectica : a reverse Differential λ-calculus

"Behind every successful proof there is a program", *Gödel's wife*

A computational Dialectica

Making Dialectica act on λ-terms instead of formulas.

λ-terms with an extra type allowing for sums

$$\frac{}{\Gamma \vdash \varnothing : \mathfrak{M}\,A} \qquad \frac{\Gamma \vdash m_1 : \mathfrak{M}\,A \qquad \Gamma \vdash m_2 : \mathfrak{M}\,A}{\Gamma \vdash m_1 \circledast m_2 : \mathfrak{M}\,A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \{t\} : \mathfrak{M}\,A} \qquad \frac{\Gamma \vdash m : \mathfrak{M}\,A \qquad \Gamma \vdash f : A \Rightarrow \mathfrak{M}\,B}{\Gamma \vdash m \gg= f : \mathfrak{M}\,B}$$

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

Pédrot's Dialectica Transformation

Soundness [Ped14]

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

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

A global and a local transformation

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

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

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:

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

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

Linearity in Linear Logic

Linearity is about resources: A proof/program is *linear* iff it uses only once its hypotheses/argument.

Linear	**Non-linear**
$A \vdash A \vee B$	$A \vdash A \wedge A$
$\lambda f \lambda x. f x x$	$\lambda x. \lambda f. f x x$

Differentiation is about making a λ-term linear :

$\rightsquigarrow$ about making a λ-term have a linear usage of its arguments.

$$\lambda x \lambda f. f x x \rightsquigarrow ?$$

Linearity in Linear Logic

Linearity is about resources: A proof/program is *linear* iff it uses only once its hypotheses/argument.

Linear	**Non-linear**
$A \vdash A \vee B$	$A \vdash A \wedge A$
$\lambda f \lambda x. f x x$	$\lambda x. \lambda f. f x x$

Differentiation is about making a λ-term linear :

$\rightsquigarrow$ about making a λ-term have a linear usage of its arguments.

$$D(\lambda x \lambda f. f x x) \cdot v := \lambda x. \lambda f. v x + ?$$

Linearity in Linear Logic

Linearity is about resources: A proof/program is *linear* iff it uses only once its hypotheses/argument.

Linear	**Non-linear**
$A \vdash A \vee B$	$A \vdash A \wedge A$
$\lambda f \lambda x. f x x$	$\lambda x. \lambda f. f x x$

Differentiation is about making a λ-term linear :

$\rightsquigarrow$ about making a λ-term have a linear usage of its arguments.

$$D(\lambda x \lambda f. f x x) \cdot v := \lambda x. \lambda f. v x + \lambda x. \lambda f. D x v$$

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

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

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

Tracking differentiation in Dialectica

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

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

Tracking differentiation in Dialectica

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

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

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]

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

Tracking differentiation in Dialectica

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

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]

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

Dialectica is differentiation in categories

That's already known through lenses !

What's categorical differentiation ?

To cook a good differential category, one needs :

- A category of regular/continuous/non-linear functions
$$\mathbb{C}(A, B) = \,!A \multimap B \; .$$
- A category of linear functions, in which differentiation embeds
$$\mathscr{L}(A, B) = A \multimap B.$$
- Something which linearizes :
$$\bar{d} : A \to \,!A$$
- A notion of duality, if one wants to encode reverse. differentiation.

$\rightsquigarrow$ Basically, one wants a categorical model of DiLL.

Dialectica categories

Categories representing specific relations

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

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

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

- Composition : the chain rule !

Consider

$$\begin{array}{rrrcl} & (f, F) : & \alpha \subseteq (A, X) & \to & \beta \subseteq (B, Y) \\ \text{and} & (g, G) : & \beta \subseteq (B, Y) & \to & \gamma \subseteq (C, Z) \end{array}$$

two arrows of the Dialectica category. Then their composition is defined as

$$(g, G) \circ (f, F) := (g \circ f, (a, z) \mapsto F(a, G(f(a), z))).$$

Dialectica categories through Differential Categories

In a $*$-autonomous differential category :

$$\partial : Id \otimes ! \to !$$

$$\mathcal{L}(B \otimes A, C^{\perp}) \simeq \mathcal{L}(A, (B \otimes C)^{\perp})$$

from $f : !A \to B$ one constructs :

$$\overleftarrow{D}(f) \in \mathcal{L}(!A \otimes B^{\perp}, A^{\perp}).$$

Dialectica categories factorize through differential categories

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

$$\begin{cases} \mathcal{L}_! & \to & \mathscr{D}(\mathcal{L}_!) \\ A & \mapsto & A \times A^{\perp} \\ f & \mapsto & (f, \overleftarrow{D}(f)) \end{cases}$$

We have an obvious forgetful functor:

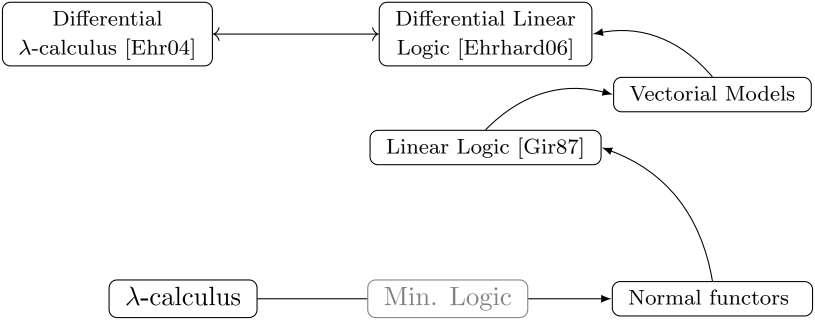
$$\mathcal{U} : \begin{cases} \mathscr{D}(\mathscr{L}_!) & \to & \mathscr{L}_! \\ \alpha \subseteq A \times X & \mapsto & A \\ (f, F) & \mapsto & f \end{cases}$$

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

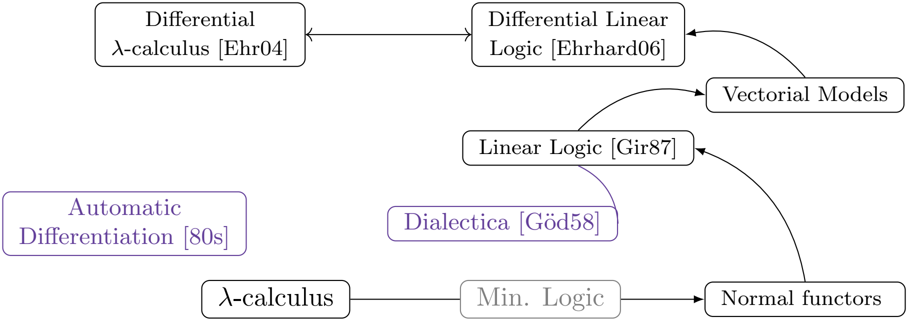
Recap

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



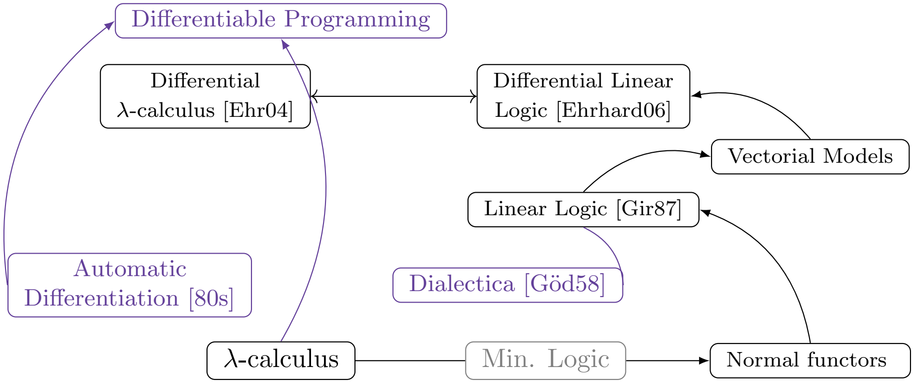
Recap

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



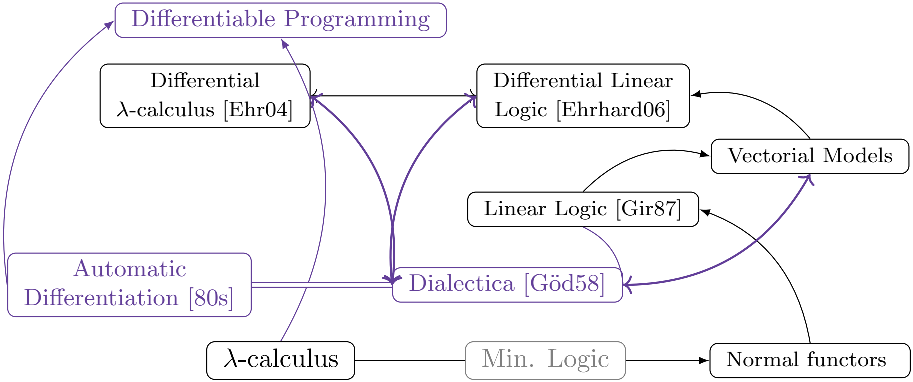
Recap

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 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.

Program	Proof	Function
Quantitative	Resources	Linearity
Control	**Classical Principles**	**Differentiation**

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 .

Differentiation and delimited continuations

Herbelin Lics'10

Markov's principle is proved by allowing `catch` and `throw` operations on hereditary positive formulas.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a:\neg\neg T\vdash_{\alpha:T} a:\neg\neg T}\,\text{AXIOM}\qquad \dfrac{\dfrac{\dfrac{}{b:T\vdash_{\alpha:T} b:T}\,\text{AXIOM}}{b:T\vdash_{\alpha:T}\mathtt{throw}_{\alpha}\,\mathtt{b}:\bot}\,\text{THROW}}{\vdash_{\alpha:T}\lambda b.\mathtt{throw}_{\alpha}\,\mathtt{b}:\neg\mathtt{T}}\,\to_I}{a:\neg\neg T\vdash_{\alpha:T} a\,(\lambda b.\mathtt{throw}_{\alpha}\,\mathtt{b}):\bot}\,\to_E}{a:\neg\neg T\vdash_{\alpha:T}\mathtt{efq}\;\mathtt{a}\,(\lambda\mathtt{b}.\mathtt{throw}_{\alpha}\,\mathtt{b}):\mathtt{T}}\,\bot_E}{a:\neg\neg T\vdash\mathtt{catch}_{\alpha}\,\mathtt{efq}\;\mathtt{a}\,(\lambda\mathtt{b}.\mathtt{throw}_{\alpha}\,\mathtt{b}):\mathtt{T}}\,\text{CATCH}}{\vdash\lambda a.\mathtt{catch}_{\alpha}\,\mathtt{efq}\;\mathtt{a}\,(\lambda\mathtt{b}.\mathtt{throw}_{\alpha}\,\mathtt{b}):\neg\neg\mathtt{T}\to\mathtt{T}}\,\to_I$$

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, 6000 Frankfurt am Main, FRG

Abstract

We consider uniqueness theorems in classical analysis having the form

$$(+)\ \forall u \in U, v_1, v_2 \in V_u \big(G(u,v_1) = 0 = G(u,v_2) \to v_1 = v_2\big),$$

where U, V are complete separable metric spaces, V_u is compact in V and $G : U \times V \to \mathbb{R}$ is a constructive function.
If $(+)$ is proved by arithmetical means from analytical assumptions

$$(++)\ \forall x \in X \exists y \in Y_x \forall z \in Z \big(F(x,y,z) = 0\big)$$

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

$$(+++)\ \forall u \in U, v_1, v_2 \in V_u, k \in \mathbb{N}\big(|G(u,v_1)|, |G(u,v_2)| \le 2^{-\Phi uk} \to d_V(v_1,v_2) \le 2^{-k}\big).$$

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 thes principles as guaranteed by (some variant of) Dialectica's transformation.

Conjecture

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

$$\forall u, v_1 v_2, \forall \epsilon > 0, \exists \eta > 0, \|G(u, v_1) - G(u, v_2)\| < \eta \to d_V(v_1, v_2) < \epsilon$$

?

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

What else can we explain by differentiation ?

Thank you for Listening !