∂ is for Dialectica

Marie Kerjean[1] and Pierre-Marie Pédrot[2]

[1]CNRS, LIPN, Université Sorbonne Paris Nord, France
kerjean@lipn.univ-paris13.fr
[2] INRIA, France,
pierre-marie.pedrot@inria.fr

Abstract

Automatic Differentiation is the study of the efficient computation of differentials. While the first automatic differentiation algorithms are concomitant with the birth of computer science, the specific reverse differentiation algorithm has been brought to a modern light by its application to neural networks. In this work, we unveil a surprising connection between reverse differentiation and Gödel's Dialectica interpretation, a logical translation that realizes semi-classical axioms. This unexpected correspondence is detailed through the setting of λ-calculus, linear logic, and categories. Thus reverse differentiation, as a logical transformation, extracts intentional information on programs and proofs.

1 Introduction

Dialectica was originally introduced by Gödel in a famous paper [Göd58] as a way to constructively interpret an extension of HA [AF98], but turned out to be a very fertile object of its own. Judged too complex, it was quickly simplified by Kreisel into the well-known realizability interpretation that now bears his name. Soon after the inception of Linear Logic (LL), Dialectica was shown to factorize through Girard's embedding of LJ into LL, purveying an expressive technique to build categorical models of LL [dP89]. In its logical outfit, Dialectica led to numerous applications and was tweaked into an unending array of variations in the proof mining community [Koh08].

The modern way to look at Dialectica is however to consider it as a program translation, or more precisely *two* mutually defined translations of the λ-calculus exposing intensional information [Péd14].

In a different scientific universe, Automatic Differentiation [GW08] (AD) is the field that studies the design and implementation of *efficient* algorithms computing the differentiation of mathematical expressions and numerical programs. Indeed, due to the chain rule, computing the differential of a sequence of expressions involves a choice, namely when to compute the value of a given expression and when to compute the value of its derivative. Two extremal algorithms coexist. On the one hand, forward differentiation [Wen64] computes functions and their derivatives pairwise in the order they are provided, while on the other hand reverse differentiation [Lin76] computes all functions first and then their derivative in reverse order. Depending on the setting, one can behave more efficiently than the other. Notably, reverse differentiation has been critically used in the fashionable context of deep learning.

Differentiable programming is a rather new and lively research domain aiming at expressing automatic differentiation techniques through the prism of the traditional tools of the programming language theory community. As such, it has been studied through big-steps semantics [AP20],

continuations [WZD+19], functoriality [Ell18], and linear types [BMP20]. It led to a myriad of implementation over rich programming languages, proven correct through semantics of higher-order differentiable functions [KPJK+22, Vák21] [MP21]. Surprisingly, these various principled explorations of automatic differentiation are what allows us to draw a link between Dialectica and differentiation in logic.

The simple, albeit fundamental claim of this paper is that, behind its different logical avatars, the Dialectica translation is in fact a reverse differentiation algorithm, where the linearity and involutivity of differentiation have been forgotten. In the domain of proof theory, differentiation has been very much studied from the point of view of *linear logic*. This led to Differential Linear Logic [ER06] (DiLL), differential categories [BCS06], or the differential λ-calculus. To support our thesis with evidence, we will draw a correspondence between each of these objects and the corresponding Dialectica interpretation.

We would like however to expose immediately the kernel of this correspondence between Dialectica and reverse differentiation. In its most traditional form, Dialectica acts as some elaborate form of prenexation form on formulas of intuitionnistic arithmetic:

$$A \rightsquigarrow \exists x \forall y A_D[x, y]$$

Like the differential transformation of differentiable programming which apply to programs of all types, Dialectica applies to all formulas. It is however acknowledged [AF98] that its crucial feature lies in the transformation of implication.

$$(A \Rightarrow B)_D[\vec{\phi}; \vec{\psi}, \vec{u}; \vec{v}] := A_D(\vec{u}, \vec{\psi}\vec{u}\vec{v}) \Rightarrow B_D(\vec{\phi}\vec{u}, \vec{v})$$

The variable $\vec{\psi}$ is functional with two arguments, and in fact it is propagated through a *chain rule* through modus ponens, as detailed in Section 2.2.1. The chain rule is the bread and butter of automatic differentiation. It states that differentiation is non-functorial, meaning that for two composable functions:

$$D_a(g \circ f) = D_{f(a)}(g) \circ D_a f.$$

Reverse automatic differentiation consists in propagating differentials by reversing the order in which the functions were primarily computed. For the composition of two functions f and g, this means that after computing a and $f(a)$, one will compute $D_{f(a)}(g)$ and only after that compute $D_{f(a)}(g) \circ D_a f$. Computationally, this means a *continuation-passing style* transformation on derivative [WZD+19]. The last ingredient to understand why Dialectica is reverse differentiation consists in the notion of *linear substitution*, introduced by Ehrhard and Regnier in their differential λ-calculus [ER03]. While the usual application between an abstraction and a value is β-reduced to a substitution, the application between a differential and a value is β_D-reduced to a linear substitution, summing over all the linear occurrences of a variable and substituting them:

$$D(\lambda x.t)v \longrightarrow \lambda x. \left(\frac{\partial t}{\partial x} \cdot v \right)$$

The linear substitution of the composition of two λ-terms is computed as follows:

$$\frac{\partial (su)}{\partial x} \cdot t = \left(\frac{\partial s}{\partial x} \cdot t \right) u + \left(Ds \cdot \left(\frac{\partial u}{\partial x} \cdot t \right) \right) u \tag{1}$$

The left term of the sum indicates the linear substitution of a variable in head position, while the second is the exact translation of the chain rule.

Now let us finally observe the modern Dialectica transformation [dP89][Péd14], where array of variables are replaced by λ-terms. The computational Dialectica applied to the composition of two λ-terms is then computed as follows:

$$(s\,u)_x := \lambda\pi.\,(s_x\,(u^\bullet, \pi)) \circledast ((s^\bullet.2)\,\pi\,u^\bullet \gg\!= u_x)$$

The $\circledast$ operation denotes a formal sum, and π is to be understood as a continuation. The $\gg\!=$ operation corresponds to the bind of a monad handling properly the sums. The transformation $(\_)^\bullet$ denotes a functorial differentiation on terms, which is defined mutually with $(\_)_x$. We hope the reader sees now that this is nothing but a CPS variant of Equation 1, and as such consists in a reverse differentiation on λ-terms. We acknowledge that this has none of the optimizations that are usually embedded in differentiable programming and detail more of that in Section 2.1. To ease the understanding of the reader, we will tackle the computational content of Dialectica only in Section 5, and we will first explore less specific settings in which the correspondence between Dialectica and reverse differentiation takes place.

Related works As far as we know, this is the first time a formal connection has been drawn between Dialectica and reverse differentiation. However, several works around Dialectica are close to the ones on differentiable programming. Powell [Pow16] formally relates the concept of learning with realizers for the Dialectica translation. His definition of learning algorithm relates to the concept of approximation. Differentiation being just the best linear approximation, our work merely formalizes this relation with linearity. More generally, Dialectica is known for extracting *quantitative* information from proofs [Koh08], and this relates very much with the quantitative point of view that differentiation has brought to λ-calculus [BM20]. Herbelin also notices at the end of its paper realizing Markov's rule through delimited continuations that this axiom has the type of a differentiation operator [Her10]. We explore the possible consequences of formally relating reverse differentiation and Dialectica to proof mining and Herbelin's work in the conclusion. Finally, reverse differentiation has also been explored through the categorical concept of lenses [CGG$^+$22], which are known to be related to Dialectica categories [dP91].

Outline and Contributions

- We begin this paper in Section 2.1 by reviewing the functorial and computational interpretation of differentiation, mainly brought to light by differentiable programming. In particular, we recall Brunel, Mazza and Pagani's result that reverse differentiation is functorial differentiation where differentials are typed by the linear negation.
- Section 2.2 provides a simple explanation of Dialectica in terms of differentiation that we believe to be new. Specifically, Section 2.2.1 shows that unification in Dialectica agrees to a chain rule, while Section 2.2.2 shows that the types of existential variables in the Dialectica interpretation are the one of functorial reverse derivatives.
- In Section 3 we consider a differential storage category (that is, a categorical model of Intuitionnistic DiLL) $\mathcal{L}$, and construct a reverse differentiation functor from the co-Kleisli category of $\mathcal{L}$ to the Dialectica category over it. We show that this reverse differentiation functor is right-adjoint to a forgetful functor between $\mathcal{L}_!$ and $\mathcal{L}$.
- In Section 4 we relate the Dialectica transformation acting on LL and DiLL. We build a Dialectica transformation from LL to DiLL, which preserves provability and factorizes the Dialectica transformation on λ-calculus when only intuitionnistic formulas of LL and DiLL are considered.
- In Section 5 we prove a contravariant logical relation between terms of the differential λ-calculus and the two transformation over λ-terms realizing witness and counter types of the Dialectica transformation. We then define a translation over terms typed by counter types that allows translating directly terms which underwent the Dialectica translation into differential λ-calculus.

2 Background material

2.1 Differentiable programming

We give here an introduction to Automatic Differentiation (AD) oriented towards differentiable programming and higher-order functional programming. Our presentation is free from partial derivatives and Jacobians notations, which are traditionally used for presenting AD. We refer to [BPRS17] for a more comprehensive introduction to automatic differentiation.

Notation 1. We write $D_t(f)$ for the differential of f at t. We denote by $- \cdot -$ the pointwise multiplication of reals or real functions. In the rest of the paper, we will freely make use of the linear logic notation for arrows : $E \longrightarrow F$ is the type of a possibly non-linear map, while $E \multimap F$ is the type of a linear map.

Forward and reverse differentiation Let us recall the chain rule, namely for any two differentiable functions $f : E \longrightarrow F$ and $g : F \longrightarrow G$ and a point : E we have

$$\begin{array}{rcl} D_t(g \circ f) & : & E \multimap G \\ & = & D_{f(t)}(g) \circ D_t(f) \end{array}$$

When computing the value of $D_t(g \circ f)$ at a point $v : E$ one must determine in which order the following computations must be performed: $f(t)$, $D_t(f)(v)$, the function $D_{f(t)}(g)$ and finally $D_{f(t)}(g)(D_t(f)(v))$. The first two computations are independent from the other ones.

In a nutshell, reverse differentiation amounts to computing first $f(t)$, then $g(f(t))$, then $D_{f(t)}(g)(v))$, then computing $D_t(f)$ and lastly the application of $D_{f(t)}(g)$ to $(D_t(f)(v))$. Conversely, forward differentiation computes first $f(t)$, then $D_t(f)$, then $g(f(t))$, then the function $D_{f(t)}(g)$ and lastly applies $D_{f(t)}(g)$ to $D_t(f)$. This explanation naturally fits into our higher-order functional setting. For a diagrammatic interpretation, see for example [BMP20].

These two techniques have different efficiency profiles, depending on the dimension of E and F as vector spaces. Reverse differentiation is more efficient for functions with many variables going into spaces of small dimensions. When applied, they feature important optimizations: in particular, differentials are not propagated through higher-order functionals in the chain rule but they are propagated compositionally. *What we will present in Section 5.1 is devoid of any optimization* and is thus extremely inefficient. Algorithmic efficiency is not the purpose of this paper. Our goal is instead to weave links with Dialectica and as such we do not prove any complexity result.

Functorial Differentiation Consider $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ differentiable. Then for every $a \in \mathbb{R}^n$, one has a linear map $D_a f : \mathbb{R}^n \multimap \mathbb{R}^m$, and the forward differential transformation has type

$$\overrightarrow{D}(f) : \left\{ \begin{array}{ccc} \mathbb{R}^n \times \mathbb{R}^m & \longrightarrow & \mathbb{R}^n \times \mathbb{R}^m \\ (a, x) & \mapsto & (f(a), D_a f \cdot x) \end{array} \right.$$

where $- \cdot -$ stands for the scalar product.

In backward mode, their transformation also acts on pairs, but with a contravariant second component, encoded via a linear dual $(-)^\perp$. The notation $(-)^\perp$ is borrowed from LL, where the (hence linear) negation is interpreted denotationally as the dual on $\mathbb{R}$-vector spaces:

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

Thus, an element of $A^\perp$ is a map which computes linearly on A to return a scalar in $\mathbb{R}$.

$$\overleftarrow{D}(f) : \left\{ \begin{array}{ccc} \mathbb{R}^n \times \mathbb{R}^{m\prime} & \to & \mathbb{R}^m \times \mathbb{R}^{n\prime} \\ (a, x) & \mapsto & (f(a), (v \mapsto v \cdot (\mathrm{D}_a f \cdot x)) \end{array} \right.$$

This encodes backward differentiation as, during the differentiation of a composition $g \circ f$, the contravariant aspect of the second component will make the derivative of g be computed before the derivative of f. The functorial presentation of differentiation in functional programming was explored by Eliott [Ell18], while Brunel, Mazza and Pagani [BMP20] refined the functional presentation by Wang and al. [WZD$^+$19] using a linear negation on ground types.

Higher-order differentiation Indeed, when one considers more abstractly a function $f : E \longrightarrow F$ between (topological) vector spaces, one has for any point $a : E$ a forward differential:

$$D_a f : E \multimap F$$

and a reverse differential:

$$(D_a f)' : \left\{ \begin{array}{ccc} F' & \multimap & E' \\ \ell \in F' & \mapsto & (\ell \circ D_a f) \end{array} \right.$$

The general type of a reverse differential is thus :

$$(D\_f)' : E \Rightarrow F' \to E'.$$

This is to be related with the type of witness variable in the Dialectica transformation, see Section 2.2.2. What Dialectica does can be understood as a *refinement over the dual type*, to allow the propagation of reverse differentials. Let us insist on the fact that *the type of derivative enforces the order of their evaluation.* For two functions $f : E \longrightarrow F$ and $g : F \longrightarrow G$, differentials $(D_e f)' : F' \multimap E'$ and $(D_{f(e)} g)' : G' \multimap F'$ can only be computed in reverse order $(D_e f)' \circ (D_{f(e)} g)' : G' \multimap E'$ [1].

Composing higher-order differentials On finite dimensional vector spaces, one has $\mathbb{R}^{n\prime} \simeq \mathbb{R}^n$, which gives the impression that one could compose the reverse derivatives of composable functions $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ and $g : \mathbb{R}^m \longrightarrow \mathbb{R}^p$. However, this reasoning does not lift to higher-order.

The fact that the first member is covariant while the second is contravariant makes it impossible to lift this transformation to higher-order. Indeed, when one considers more abstractly a function $f : E \longrightarrow F$ between (topological) vector spaces , one has:

$$\overleftarrow{D}(f) : \left\{ \begin{array}{ccc} E \times F' & \longrightarrow & F \times E' \\ (a, \ell) & \mapsto & (f(a), (D_a f)') \end{array} \right.$$

Consider the functorial reverse differential $\overleftarrow{D}(g) : F \times G' \longrightarrow G \times F'$. If G and F are not self-dual,there is no way to define the composition of $\overleftarrow{D}(f)$ with $\overleftarrow{D}(g)$. Said otherwise, at higher-order, the functorial reverse differentials of two composable programs/functions are not composable. Thus we would like to argue that higher-order differentiation is achieved using *two distinct differential transformations*. This is the case in the differential λ-calculus for forward AD or the Dialectica Transformation for reverse AD, as we show in Section 5.

2.2 Dialectica

In this section we show that already in its historical version, Dialectica is well explained trough the concept of reverse differentiation. This intuitions will be made formal when linearity enters the game in the following sections.

$$(t = u)_D := t = u \qquad (A \Rightarrow B)_D[\vec{\phi}; \vec{\psi}, \vec{u}; \vec{v}] := A_D(\vec{u}, \vec{\psi}\vec{u}\vec{v}) \Rightarrow B_D(\vec{\phi}\vec{u}, \vec{v})$$
$$\bot_D := \bot \qquad (\forall z.A)_D[\vec{u}, z; \vec{x}] := A_D[\vec{u}z, \vec{x}]$$
$$\top_D := \top \qquad (\exists z.A)_D[z; \vec{u}, \vec{x}] := A_D[\vec{u}, \vec{x}z]$$

$$(A \wedge B)_D[\vec{u}; \vec{v}, \vec{x}; \vec{y}] := A_D[\vec{u}, \vec{x}] \wedge B_D[\vec{v}, \vec{y}]$$
$$(A \vee B)_D[b, \vec{u}; \vec{v}, \vec{x}; \vec{y}] := (b = 0 \wedge A_D[\vec{u}, \vec{x}]) \vee (b = 1 \wedge B_D[\vec{v}, \vec{y}])$$

Figure 1: Dialectica interpretation of HA

2.2.1 Dialectica acting on formulas of HA

In this section, we examine Dialectica as a logical transformation acting on intuitionistic arithmetic. Gödel's Dialectica transforms a formula A of Heyting Arithmetic (HA) into a formula $A_D(\vec{u}, \vec{v})$, where A_D is a formula of HA^ω parametrized by a witness sequence $\vec{u}$ and a counter sequence $\vec{v}$ of variables of System T. The need for higher-order terms is a staple of realizability, where logical implications are interpreted as some flavour of higher-order functions[2]. The interpretation of formulas is detailed in Figure 1 where the ";" symbol denotes the concatenation of sequences of variables.

Theorem 1. If $\vdash_{\mathsf{HA}} A$ then $\vdash_{\mathsf{HA}^\omega} \exists \vec{u}. \forall \vec{x}. A_D[\vec{u}, \vec{x}]$.

The most involved case of the above transformation is largely acknowledged to be the one for implication. It can be explained as the *least unconstructive* way to perform a skolemization on the implication of two formulas which already went through the Dialectica transformation [Koh08, 8.1]. It is also presented as a way to "extract constructive information through a purely local procedure" [AF98, 3.3]. This second intuition corresponds to the one of differentiation: *extracting at each point the best local approximation of a function.*

Let us notice that in the Dialectica transformation, the witness sequences for function types verify the chain rule. Consider two implications $(A \Rightarrow B)$ and $(B \Rightarrow C)$ and their composition $(A \Rightarrow C)$, through the Dialectica interpretation:

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

The Dialectica interpretation of the composition provides a solution to the system below in the general case.

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

This solution amounts to the following equations:

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

which can be simplified to:

$$\phi_3\, u_3 = \phi_2\, (\phi_1\, u_3) \qquad\qquad \psi_3\, u_3\, v_3 = \psi_2\, (\phi_1\, u_3)\, (\psi_1\, u_3\, v_3)$$

[1]The only case where we would have an ambiguity would be for functions acting on self-dual space $E' = E$. This is famously the case for finite dimensional vector spaces or Hilbert spaces, but is otherwise very scarce in mathematics and is in particular false on spaces of differentiable functions $\mathcal{C}^1(\mathbb{R}^n, \mathbb{R})$.

[2]Dialectica was the first of its kind, giving rise to its nickname as the *functional interpretation*, but other realizabilities are no less functional.

$$
\begin{array}{llll}
\mathbb{W}(\bot) = \mathbb{C}(\bot) = \emptyset & & \mathbb{W}(t=u) = C(t=u) = \emptyset \\
\mathbb{W}(\top) = \mathbb{C}(\top) = \emptyset & & \mathbb{W}(A \vee B) = \mathbb{N}; \mathbb{W}(A); \mathbb{W}(B) \\
\mathbb{W}(A \wedge B) = \mathbb{W}(A); \mathbb{W}(B) & & \mathbb{C}(A \vee B) = \mathbb{C}(A); \mathbb{C}(B) \\
\mathbb{C}(A \wedge B) = \mathbb{C}(A); \mathbb{C}(B) & & \mathbb{C}(A \Rightarrow B) = \mathbb{W}(A) \times \mathbb{C}(B)
\end{array}
$$

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

Figure 2: Types of Dialectica realizers

While the left equation represents the traditional functoriality of composition, the right equation is exactly the chain rule. Said otherwise, we would like to assert the following.

Thesis 1. *The pair $(\vec{\phi}, \vec{\psi})$ of sequences of variables witnessing the Dialectica interpretation of an implication represents sequences of functions ϕ and their differential ψ.*

However, many functional transformations satisfy the chain rule [AAKM10]. We will strengthen our claim and show that Dialectica is indeed reverse differentiation.

2.2.2 Witness and counter types

The witness and counter sequences $\vec{u}$ and $\vec{v}$ can actually be typed by sequences of types of System T, giving a better understanding of the transformation. The type $\mathbb{W}(A)$ is called the *witness type* of A while the type $\mathbb{C}(A)$ is called its *counter type*. The formula A_D then acts as an orthogonality test between these two types. They are detailed in Figure 2.

Remember that if a function has the type $f : A \longrightarrow B$, its differential is usually presented with the type $Df : A \longrightarrow A \multimap B$, the second arrow in Df representing a linear map from A to B, that is the differential of f at a point. Here, the second projection of the witness type of an arrow is slightly different. Indeed, the second arrow in this projection is contravariant, as the second component of $\mathbb{W}(A \Rightarrow B)$ is $\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A)$

That is, as explained in Section 2.1, if $\mathbb{C}(A)$ were to represent the dual of A, the second projection of the witness of an arrow has the type of a *reverse* differential. The type $\mathbb{W}(A \Rightarrow B) = (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A))$ is thus the type of a pair of *a function and its reverse differential*. We will show that the dynamical theory of the terms typed by witness and counter types also agree with reverse differentiation. This necessitates linear implications, which will be done in Section 4. The interpretation of Dialectica as a reverse differentiable programming language will be performed in Section 5.1.

3 Differential categories results in Dialectica Categories

The Dialectica transformation was studied from a categorical point of view by De Paiva and Hyland [dP89]. They have been used as a way to generate *new models* of LL [dP89, Hed14]. Our point of view is quite orthogonal. We suggest instead that they may characterize *specific models* of LL.

Definition 2 ([dP89])**.** Consider $\mathcal{C}$ a category with finite limits. The Dialectica category $\mathscr{D}(\mathcal{C})$ over $\mathcal{C}$ has as objects subpairs $\alpha \subseteq (A, X)$ of objects of $\mathcal{C}$, and as arrows pairs $(f, F) : \alpha \subseteq (A, X) \longrightarrow \beta \subseteq (B, Y)$ of maps

$$
\begin{cases} f : & A & \longrightarrow & B \\ F : & A \times Y & \longrightarrow & X \end{cases}
$$

such that if $(a; F(a; y)) \in \alpha$ then $(f(a); y) \in \beta$. Consider

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

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

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

The identity on an object $\alpha \subseteq (A, X)$ is the pair $(id_A, (\_).2)$ where $(\_).2$ is the projection on the second component of $A \times X$.

In our point of view, objects α of $\mathscr{D}(\mathcal{C})$ generalize the relation between a space A and its tangent space. Arrows (f, F) represents a function and its reverse map F, according to the typing intuitions developed in Section 2.1. Composition is exactly the chain rule. Therefore, it is natural to investigate the relationship between Dialectica categories and differential categories. The various axiomatizations for differentiation in categories [BCS06, CC14], all encode forward derivatives. To encode reverse derivatives in these structures one must use some notion of duality. Therefore we will restrict to the narrow setting of categorical models of DiLL. Indeed, the linear duality at stake allows making use of the intuitions developed in Section 2.1.

Consider $\mathcal{L}$ a categorical model of DiLL as formalized by Blute, Cockett, Seely and Fiore. We refer to [BCLS20] for an overview of the different variations within definitions of differential categories. We have a monoidal closed category $(\mathcal{L}, \otimes, 1)$ endowed with a biproduct $\Diamond$, a strongly monoidal comonad ! and a natural transformation ∂ satisfying the appropriate commutative diagrams [Fio07]:

$$(!, d, \mu) : (\mathcal{L}, \otimes) \longrightarrow (\mathcal{L}, \Diamond) \qquad\qquad \partial : Id \otimes ! \longrightarrow !.$$

Let us denote $(\_)^\perp$ the hom-functor: $A \mapsto \mathcal{L}(A, 1)$. Consider $f \in \mathcal{L}(!A, B)$ a morphism of the coKleisli category $\mathcal{L}_!$. The morphism $f \circ \partial \in \mathcal{L}(A \otimes !A, B)$ traditionally interprets the differential of the function f. Through the embedding $B \longrightarrow B^{\perp\perp}$ and the monoidal closedness of $\mathcal{L}$ one constructs a morphism:

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

Composing with the dereliction $d_{B^\perp} \in \mathcal{L}(!(B^\perp), B^\perp)$ and the strong monoidality of !, one gets a morphism:

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

Proposition 3. Suppose that the category $\mathcal{L}_!$ has finite limits. Then one has a functor from the co-Kleisli $\mathcal{L}_!$ to the Dialectica category over it $\mathscr{D}(\mathcal{L}_!)$:

$$\mathcal{R} : \left\{ \begin{array}{ccc} \mathcal{L}_! & \longrightarrow & \mathscr{D}(\mathcal{L}_!) \\ A & \mapsto & A \times A^\perp \\ f & \mapsto & (f, \overleftarrow{D}(f)) \end{array} \right.$$

Proof. If f is a morphism from A to B in $\mathcal{L}_!$, then $f \in \mathcal{L}(!A, B)$ and $\overleftarrow{D}(f)$ is a morphism from $A \times B^\perp$ to $A^\perp$ in $\mathcal{L}_!$, so $(f, \overleftarrow{D}(f))$ is indeed a morphism from $A \times A^\perp$ to $B \times B^\perp$ in $\mathscr{D}(\mathcal{L}_!)$. If f is the identity on A in $\mathcal{L}_!$, that is $f = d_A \in \mathcal{L}(!A, A)$, then the comonad equation for ∂ [Fio07, Definition 4.2.2] ensures that $\overleftarrow{D}(f)$ is indeed the projection on the second component. Finally, if $f \in \mathcal{L}(!A, B)$ and $g \in \mathcal{L}(!B, C)$, then the second monad rules guarantees that

$$g \circ !f \circ \mu \circ \partial = g \circ \partial \circ (f \circ \partial \otimes !f) \circ (1 \otimes \bar{m})$$

where $\bar{m}$ is the composition of the biproduct diagonal and the comonad strong monoidality. See the litterature [Fio07] for explicit handling of annihilation operators and coproducts in this formula, which is nothing but the categorical restatement of the chaine rule. The above formula then exactly corresponds to the composition in $\mathcal{L}_!$ of $\overleftarrow{D}(g)$ and $(f \circ \pi.1, \overleftarrow{D}(f))$, modulo the strong monoidality of !. □

We have an obvious forgetful functor:

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

which is left adjoint to $\mathcal{R}$, both forming a co-reflection on $\mathcal{L}_!$. However, the Dialectica category over $\mathcal{L}_!$ is in no way equivalent to $\mathcal{L}_!$, as if no linearity condition is imposed several operators other than differentiation can verify the chain rule.

$$\mathscr{D}(\mathcal{L}_!) \underset{\mathcal{R}}{\overset{\mathcal{U}}{\rightleftarrows}} \mathcal{L}_! \quad \dashv$$

Remark 1. The fact that objects of Dialectica categories generalize the relation between a manifold and its tangent space is noticed in by de Paiva and da Silva in a recent paper [dPdS21, remark 4.1].

Example 4. For the co-kleisli of a differential storage categories to admit finite limits, it is enough that is admits all equalizers, as it always has finite products and a terminal objects. This include any category of complete spaces where functions are continuous. This exclude models with only bounded non-linear maps [KT16] but include Köthe spaces [Ehr02](equalizers are double orthogonal of the equalizer in **Set**) or convenient models [BET12].

This setting can surely be relaxed, and there might be broader relations between Dialectica categories and differential categories, for example with reverse derivative categories [CCG+20].

4 Dialectica is reverse differentiation in linear logic

We now study the differential connection between Dialectica and LL from a syntactic point of view. After its original presentation by Gödel, Dialectica has been refined as a logical transformation acting from MELL to the simply-typed λ-calculus with pairs and sums, by looking at the witness and counter types [dP89].

This presentation allows removing a lot of historical accidental complexity, including the need for sequences of variables. We will not detail here the action of this translation on terms of λ-calculus, as we will give in the next section the refined computational version of the Linear Dialectica by Pédrot. These works are type-oriented, working directly on witness types and terms of λ-calculus. They are distinct from works applying Dialectica directly on formulas of LL [FO11]. Formulas of LL are constructed according to the following grammar.

$$A, B := 0 \mid 1 \mid \bot \mid \top \mid A \oplus B \mid A \,\&\, B \mid A \parr B \mid A \otimes B \mid !A \mid ?A$$

We define as usual the involutive negation $(-)^\bot$, & being the dual of $\oplus$, $\otimes$ the dual of $\parr$ and ! the dual of ?. As per the standard practice, we define the *linear implication* $A \multimap B := A^\bot \parr B$, from which the usual non-linear implication can be derived through the call-by-name encoding $A \Rightarrow B := !A \multimap B$, where the exponential formula $!A$ represents the possibility to use A an arbitrary number of times.

Figure 3 defines the witness and counter interpretations of LL connectives into intuitionistic types. While this refinement was introduced by de Paiva [dP89], we incorporate to the translation one of the tweaks made by Pédrot [Péd14], namely the fact that $\mathbb{W}(0) := 1$. This is justified by the irrelevance of dummy terms. We refer the reader to the literature for more details on dummy terms and computability conditions in the Dialectica translation.

As expected, the interpretation of the intuitionistic arrow factorizes through the call-by-name translation of LJ into LL, i.e. we have

$$\mathbb{W}(!A \multimap B) = \mathbb{W}((!A \otimes B^\bot)^\bot) = (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{C}(B) \Rightarrow \mathbb{W}(A) \Rightarrow \mathbb{C}(A))$$

which through cartesian closedeness is isomorphic to $\mathbb{W}(A \Rightarrow B)$ as defined in Figure 2.

$$
\begin{array}{lcllcl}
\mathbb{W}(0) & := & 1 & \mathbb{C}(0) & := & 1 \\
\mathbb{W}(1) & := & 1 & \mathbb{C}(1) & := & 1 \\
\mathbb{W}(A^\perp) & := & \mathbb{C}(A) & \mathbb{C}(A^\perp) & := & \mathbb{W}(A) \\
\mathbb{W}(A \oplus B) & := & \mathbb{W}(A) + \mathbb{W}(B) & \mathbb{C}(A \oplus B) & := & \mathbb{C}(A) \times \mathbb{C}(B) \\
\mathbb{W}(!A) & := & \mathbb{W}(A) & \mathbb{C}(!A) & := & \mathbb{W}(A) \Rightarrow \mathbb{C}(A)
\end{array}
$$

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

Figure 3: Witness and counter types for LL formulas into $\lambda^{+,\times}$-calculus [dP89, Péd14].

$$
\frac{\Gamma \vdash B}{\Gamma, !A \vdash B}\, w \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma !A \vdash B}\, c \qquad \frac{\Gamma, A \vdash B}{\Gamma, !A \vdash B}\, d
$$

$$
\frac{}{\vdash !A}\, \bar{w} \qquad \frac{\Gamma \vdash !A \qquad \Delta \vdash !A}{\Gamma, \Delta, \vdash !A}\, \bar{c} \qquad \frac{\Gamma \vdash A}{\Gamma \vdash !A}\, \bar{d}
$$

$$
\frac{!\Gamma \vdash A}{!\Gamma \vdash !A}\, p
$$

Figure 4: Exponential rules of DiLL

Remark 2. The fact that cartesian closedeness is needed to make the Dialectica Translation on LJ and the one on LL correspond might be related to the operational semantics of the differential λ-calculus. Indeed, while semantically they are equivalent, dynamically the linear logic correspondence means that the linear argument $\mathbb{C}(B)$ is evaluated before the non-linear argument $\mathbb{W}(A)$. This is also what happens for the linear substitution of the differential λ-calculus, where the linear variable is to be substituted before the non-linear variable.

4.1 Differential Linear Logic

Figure 4 recalls the exponential rules of Differential Linear Logic [ER06] (DiLL), presented here in their intuitionistic version for reasons explained below. DiLL adds to LL rules to differentiate proofs which are recalled together with the usual exponential rules in Figure 4. While LL already features a dereliction rule, that forgets linear proofs into non-linear ones, DiLL adds in particular a co-dereliction rules, which linearize a non-linear proof into a linear one. To sum it up, to the traditional weakening w, contraction c, dereliction d and promotion p rules DiLL adds:

- a co-weakening rule $\bar{w}$, accounting for the introduction of constant functions,
- a co-contraction rule $\bar{c}$, accounting for the possibility to sum in the function domains,
- a co-dereliction $\bar{d}$, accounting for the possibility to differentiate functions
- sums of proofs, generated by the cut-elimination procedure,
- cut-elimination rules account for the basic rules of differential calculus.

A previously, we argue that *witness variables for implications are pairs of a function and its reverse differential*. The description of A_D as an orthogonality relation between witness and counter types acts as a test for this relation. However, the semantics of DiLL is neither forward nor backward. Indeed, as DiLL is classical, one can go from forward mode to reverse mode and to reverse mode to forward mode, as detailled in Section 2.1. Syntactically, this is due to the associativity of $⅋$,

$$
\begin{array}{lcl@{\qquad}lcl}
\mathbb{W}(0) & := & 0 & \mathbb{C}(0) & := & \top \\
\mathbb{W}(1) & := & \top & \mathbb{C}(1) & := & 0 \\
\mathbb{W}(\top) & := & \top & \mathbb{C}(\top) & := & 0 \\
\mathbb{W}(A \otimes B) & := & \mathbb{W}(A) \otimes \mathbb{W}(B) & \mathbb{C}(A \otimes B) & := & (\mathbb{W}(A) \multimap \mathbb{C}(B)) \\
\mathbb{W}(A \multimap B) & := & (\mathbb{W}(A) \multimap \mathbb{W}(B)) & & & \oplus (\mathbb{W}(B) \multimap \mathbb{C}(A)) \\
 & & \&(\mathbb{C}(B) \multimap \mathbb{C}(A)) & \mathbb{C}(A \multimap B) & := & \mathbb{W}(A) \otimes \mathbb{C}(B) \\
\mathbb{W}(A \,\&\, B) & := & \mathbb{W}(A) \,\&\, \mathbb{W}(B) & \mathbb{C}(A \,\&\, B) & := & \mathbb{C}(A) \oplus \mathbb{C}(B) \\
\mathbb{W}(A \oplus B) & := & \mathbb{W}(A) \oplus \mathbb{W}(B) & \mathbb{C}(A \oplus B) & := & \mathbb{C}(A) \,\&\, \mathbb{C}(B) \\
\mathbb{W}(!A) & := & !\mathbb{W}(A) & \mathbb{C}(!A) & := & !\mathbb{W}(A) \multimap \mathbb{C}(A)
\end{array}
$$

Figure 5: Witness and counter types for ILL formulas into DiLL

the multiplicative disjunction behind the the linear implication. Hence, we need to encode LL in DiLL through a *contravariant translation on arrows* and we want to make DiLL act on intuitionistic formulas in order to *force the reverse translation*.

We now make the Dialectica translation act on *formulas of intuitionistic* LL:

$$A, B := 0 \mid 1 \mid \top \mid A \oplus B \mid A \,\&\, B \mid A \multimap B \mid A \otimes B \mid !A$$

While in classical LL the linear implication is encoded through $A \multimap B := A^{\perp} \parr B$, it is a primitive connective of the intuitionnistic version. This has a major influence in terms of the execution order of programs, or equivalently on the composition order of functions [CFMM16]. Accordingly, we will use intuitionistic LL to enforce the reverse differentiation of functions.

We present the Dialectica translation from LL to DiLL in Figure 5, and prove a basic soundness result in Proposition 7. This translation hardwires the fact that an implication must be accompanied by its reverse differential. If the implication depends on an exponential, then some real differentiation will happen, otherwise the differential part will just hardwire the classicality of implications.

With the translation defined in Figure 5 and through the usual encoding $A \multimap B := A^{\perp} \parr B$, one has

$$\mathbb{W}(!A \multimap B) = (\mathbb{C}(B) \multimap !\mathbb{W}(A) \multimap \mathbb{C}(A)).$$

As such, the functional translation from LL to DiLL only encodes the differential part of Dialectica. It corresponds to our running intuition that the second component of $\mathbb{W}(!A \multimap B)$ types a reverse differential, linear in the dual of B and non-linear in A.

Remark 3. While differential, the translation presented in Figure 5 is not specifically reverse. Indeed, as Differential Linear Logic is classical, one has equivalently:

$$(!\mathbb{W}(A) \multimap \mathbb{C}(B) \multimap \mathbb{C}(A)) \equiv (!\mathbb{W}(A) \multimap \mathbb{W}(A) \multimap \mathbb{W}(B)).$$

That is, due to the presence and associativity of the $\parr$, or equivalently due to the involutive linear negation reverse and forward derivative are equivalent. It is reverse when one considers only the $\multimap$ connective, as in Proposition 10. However, we need DiLL to be classical to make proposition 6 work. To avoid the confusion above, one could design a reverse intuitionistic DiLL.

Lemma 5. For A a formula of intuitionnistic LL, we have a proof of $\mathbb{W}(A) \vdash \mathbb{C}(A) \multimap \perp$ in intuitionnistic LL.

Proof. The proof is done by induction on the formula A, thanks to the elimination and construction rules of its connectives[3]. The only non-trivial case is the one for $!A$, which uses dereliction and contraction. □

[3]We refer to the wiki on Linear Logic for a concise presentation of these rules `http://llwiki.ens-lyon.fr/mediawiki/index.php/Intuitionistic_linear_logic`

Thanks to the co-structural rule of DiLL, which encode the possibility to differentiate proofs, we obtain another duality relation between counter and witness types.

Lemma 6. For A a formula of intuitionnistic LL, we have a proof of $\mathbb{W}(A) \multimap \bot \vdash \mathbb{C}(A)$ in classical DiLL.

Proof. The proof is done by induction on the formula A. The case for $\otimes$, & and $\oplus$ are straightforward. The case for $\multimap$ however needs to hard-encode the fact that for any formula C and D, $C \,\&\, D \multimap \bot = (C \multimap \bot) \oplus (D \multimap \bot)$ and $(C \multimap D) \multimap \bot = C \otimes (D \multimap \bot)$. Hence the classicality requirement in this lemma. The exponential case is the one where differentiation happens, using codereliction and cocontraction were dereliction and contraction were necessary for Lemma 5. We detail the proof, keeping use of formulas of Intuitionnistic Linear Logic intentionally.

$$\dfrac{\dfrac{\dfrac{\pi}{!\mathbb{W}(A) \multimap \bot, !\mathbb{W}(A) \vdash \mathbb{W}(A) \multimap \bot}\multimap_R \qquad \mathbb{W}(A) \multimap \bot \vdash \mathbb{C}(A)}{!\mathbb{W}(A) \multimap \bot, !\mathbb{W}(A) \vdash \mathbb{C}(A)}\,\text{cut}}{!\mathbb{W}(A) \multimap \bot \vdash !\mathbb{W}(A) \multimap \mathbb{C}(A)}\multimap_R$$

where π accounts for the following proof:

$$\dfrac{\dfrac{!\mathbb{W}(A) \multimap \bot, !\mathbb{W}(A) \vdash \bot \qquad \dfrac{!\mathbb{W}(A) \vdash !\mathbb{W}(A) \qquad !\mathbb{W}(A) \vdash !\mathbb{W}(A)}{!\mathbb{W}(A), !\mathbb{W}(A) \vdash !A}\,\bar{c}}{!\mathbb{W}(A) \multimap \bot, !\mathbb{W}(A), !\mathbb{W}(A) \vdash \bot}\,\text{cut} \qquad \dfrac{\mathbb{W}(A) \vdash \mathbb{W}(A)}{\mathbb{W}(A) \vdash !\mathbb{W}(A)}\,\bar{d}}{!\mathbb{W}(A) \multimap \bot, !\mathbb{W}(A), \mathbb{W}(A) \vdash \bot}\,\text{cut}$$

□

The soundness statement proposition makes use of the previous lemmas but does not need more of the decomposition of the arrow via a $\parr$, and only uses the fundamental codereliction and cocontraction rules.

Proposition 7. If $\Gamma \vdash A$ in ILL, then $\mathbb{W}(\Gamma) \vdash \mathbb{W}(A)$ in classical DiLL.

Proof. The proof is then a straightforward induction on the formula A for any context Γ. The only subtle case concern the case for the implication $A = C \multimap B$ which makes use of the lemmas 5 and 6. Suppose we have a proof of $\Gamma \vdash C \multimap D$ in intuitionnistic LL.

$$\dfrac{\dfrac{\dfrac{\text{induction hypothesis}}{\mathbb{W}(\Gamma), \mathbb{W}(C) \vdash \mathbb{W}(D)} \qquad \dfrac{\text{Lemma 5}}{\mathbb{W}(D) \vdash \mathbb{C}(D) \multimap \bot}}{\mathbb{W}(\Gamma), \mathbb{W}(C) \vdash \mathbb{C}(D) \multimap \bot}\,\text{cut} \qquad \dfrac{\text{Lemma 6}}{\mathbb{W}(C) \multimap \bot \vdash \mathbb{C}(C)}}{\mathbb{W}(\Gamma), \mathbb{C}(D) \vdash \mathbb{C}(C)}\,\text{cut} \qquad \pi}{\mathbb{W}(\Gamma) \vdash (\mathbb{W}(C) \multimap \mathbb{W}(D)) \,\&\, (\mathbb{C}(D)) \multimap \mathbb{C}(C))}$$

where π accounts for the left introduction $\multimap$ and the use of the induction hypothesis. □

4.2 Factorization

The first Dialectica translation we presented in Figure 2 can in fact be modernized as a translation to and from types of $\lambda^{+,\times}$-calculus: this is recalled in Figure 6. It factorizes through the linear Dialectica by injecting LJ into LL via the economical translation.

$$\begin{array}{ccc} & \mathsf{LL} & \\ {\scriptstyle [\![-]\!]_e}\nearrow & & \searrow{\scriptstyle \mathbb{W}\ \mathbb{C}} \\ \lambda^{+,\times} & \xrightarrow[\mathbb{W}\quad\mathbb{C}]{} & \lambda^{+,\times} \end{array} \qquad (2)$$

$$\begin{array}{lcl} \mathbb{W}(0) := 1 & & \mathbb{C}(0) := 1 \\ \mathbb{W}(1) := 1 & & \mathbb{C}(1) := 1 \\ \mathbb{W}(A \times B) & := & \mathbb{W}(A) \times \mathbb{W}(B) \\ \mathbb{C}(A \times B) & := & \mathbb{C}(A) + \mathbb{C}(B) \end{array} \qquad \begin{array}{lcl} \mathbb{W}(A + B) & := & \mathbb{W}(A) + \mathbb{W}(B) \\ \mathbb{C}(A + B) & := & \mathbb{C}(A) \times \mathbb{C}(B) \\ \mathbb{C}(A \Rightarrow B) & := & \mathbb{W}(A) \times \mathbb{C}(B) \end{array} \qquad \mathbb{W}(A \Rightarrow B) :=$$

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

Figure 6: A modernized Dialectica [Péd15, 8.3.1]

Definition 8. [Cos92][Péd15, 8.4.2] The following is called the economical translation.

$$\begin{array}{lcllcl} [\![A \Rightarrow B]\!]_e & := & !A \multimap B & & & \\ [\![A + B]\!]_e & := & A \oplus B & [\![0]\!]_e & := & 0 \\ [\![A \times B]\!]_e & := & A \,\&\, B & [\![1]\!]_e & := & 1 \end{array}$$

This translation has some surprising features. While arrows are interpreted in call-by-name, products and sums are respectively interpreted in call-by-value. Figure 4.2 presents the factorization of the translation of of $\lambda^{+,\times}$-calculus through the economical translation. We would like now to recover it through its differential refinement (Figure 5), as in Figure 3.

$$\begin{array}{ccc} \mathsf{ILL} & \xrightarrow[\mathbb{C}]{\mathbb{W}} & \mathsf{IDiLL} \\ \Big\uparrow{\scriptstyle ...} & & \Big\downarrow{\scriptstyle ...} \\ \lambda^{+,\times} & \xrightarrow[\mathbb{C}]{\mathbb{W}} & \lambda^{+,\times} \end{array} \qquad (3)$$

Definition 9. The translation from intuitionistic DiLL to types of $\lambda^{+,\times}$-calculus is defined as follows:

$$\begin{array}{lll} \mathcal{U}(!A) := A & & \mathcal{U}(\top) = \mathcal{U}(1) = \mathcal{U}(0) := 1 \\ \mathcal{U}(A \,\&\, B) := \mathcal{U}(A) \times \mathcal{U}(B) & & \mathcal{U}(A \oplus B) = \mathcal{U}(A) + \mathcal{U}(B) \\ \mathcal{U}(A \otimes B) := \mathcal{U}(A) \times \mathcal{U}(B) & & \mathcal{U}(A \multimap B) := \mathcal{U}(A) \Rightarrow \mathcal{U}(B) \end{array}$$

Proposition 10. The Dialectica transformation on types factorizes through LL and DiLL as follows:

$$\begin{array}{ccc} \mathsf{LL} & \xrightarrow[\mathbb{C}]{\mathbb{W}} & \mathsf{DiLL} \\ \Big\uparrow{\scriptstyle [\![\_]\!]_e} & & \Big\downarrow{\scriptstyle \mathcal{U}} \\ \lambda^{+,\times} & \xrightarrow[\mathbb{C}]{\mathbb{W}} & \lambda^{+,\times} \end{array}$$

Proof. The proof proceeds by an immediate induction on the syntax of formulas. Note that we used the same notation for witness and counter types of LL and $\lambda^{\times +}$, but they can easily be discriminated from the context. Let us show that for any type A of $\lambda^{+,\times}$-calculus, $\mathcal{U}(\mathbb{W}([\![A]\!]_e)) = \mathbb{W}(A)$ and $\mathcal{U}(\mathbb{C}([\![A]\!]_e)) = \mathbb{C}(A)$.

- **Units**. All units are mapped to units via the $[\![\_]\!]_e$ and witness and counter types on LL. As $\mathcal{U}$ maps all units except $\bot$to 1, and as witness and counter on types of $\lambda^{+,\times}$ are to unit which are never $\bot$, we have $\mathcal{U}(\mathbb{W}([\![A]\!]_D)) = \mathbb{W}(A)$ and $\mathcal{U}(\mathbb{C}([\![A]\!]_D)) = \mathbb{C}(A)$ for $A = 1$ or $A = 0$.
- **Product ans sum** Through $[\![\_]\!]_e$, $\times$ and $+$ are mapped to the additive connectives of LL, which are translated as themselves through $\mathbb{W}$, or as their dual via $\mathbb{C}$, and then mapped to to $\times$ and $+$ again via $\mathcal{U}$. Thus one finds back the interpretation of Figure 6.

- **Implication**. We will detail the computations for the witness case, the counter case being straightforward. Consider A and B types of the $\lambda^{+,\times}$-calculus.

$$\begin{aligned}
&\mathcal{U}(\mathbb{W}([\![A \Rightarrow B]\!]_e)) = \mathcal{U}\,(\mathbb{W}(!{[\![A]\!]_e} \multimap [\![B]\!]_e)) \\
&= \mathcal{U}\,(\mathbb{W}(!{[\![A]\!]_e}) \multimap \mathbb{W}([\![B]\!]_e)) \,\&\, (\mathbb{C}([\![B]\!]_e) \multimap \mathbb{C}(!{[\![A]\!]_e}))) \\
&= \mathcal{U}\,(!\mathbb{W}([\![A]\!]_e) \multimap \mathbb{W}([\![B]\!]_e)) \,\&\, (\mathbb{C}([\![B]\!]_e) \multimap !\mathbb{W}([\![A]\!]_e) \multimap \mathbb{C}([\![A]\!]_e))) \\
&= (\mathcal{U}(\mathbb{W}([\![A]\!]_e) \Rightarrow \mathcal{U}(\mathbb{W}([\![B]\!]_e))) \times (\mathcal{U}(\mathbb{W}([\![A]\!]_e)) \Rightarrow \mathcal{U}(\mathbb{C}([\![B]\!]_e)) \Rightarrow \mathcal{U}(\mathbb{C}([\![A]\!]_e)))
\end{aligned}$$

□

Remark 4. Our translation seems to be quite different from Dialectica interpretations applied directly of formulas of second order Intuitionistic Linear Logic [FO11], and not on witness and counter types with linear types. Indeed, the factorization does not happen in the same order, as formulas of intuitionistic both in the domain and co-domain of the Dialectica translation are embedded into LL. Here, in the contrary, we strive to put LL and DiLL as intermediate steps of a factorization between formulas of intuitionistic logic.

5 Dialectica is reverse differential λ-calculus

The previous sections have shown that both on the categorical side (Section 3) on the typing side (Section 4), the Dialectica transformation corresponds to a reverse implementation of differentiation. In this section, we tackle the computational side, which was at the center of Pédrot's work [Péd14].

5.1 An account of the modern Dialectica transformation

As hinted in the previous section, Dialectica can be applied to typed λ-terms instead of HA derivations. In modern terms, one would call it a realizability interpretation into an extended λ-calculus that introduces the ability to observe intensional content from the underlying terms, i.e. the way variables are used. In its first version [Göd58, dP89] however, it relied on the existence of dummy terms at each type and on decidability of the orthogonality condition. The introduction of "abstract multisets" allows getting rid of the decidablity condition and makes Dialectica preserve β-equivalence, generalizing the Diller-Nahm variant [Dil74].

We recall the Dialectica translation of the simply-typed λ-calculus below. Types of the source language are inductively defined as

$$A, B := \alpha \mid A \Rightarrow B$$

where α ranges over a fixed set of atomic types. Terms are the usual λ-terms endowed with the standard β, η-equational theory.

The target language is a bit more involved, as it needs to feature negative pairs and *abstract multisets*.

Definition 11. An abstract multiset structure is a parameterized type $\mathfrak{M}\,(-)$ equipped with the following primitives.

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

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

$$\begin{array}{llll} \text{Types} & A, B & ::= & \alpha \mid A \Rightarrow B \mid A \times B \\ \text{Terms} & t, u & ::= & x \mid \lambda x.\, t \mid t\, u \mid (t, u) \mid t.1 \mid t.2 \end{array}$$

Reduction rules:

$$(\lambda x.\, t)\, u \to_\beta t\{x \leftarrow u\} \qquad (t_1, t_2).i \to_\beta t_i \qquad t \equiv_\eta (t_1, t_2)$$

Typing rules:

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B}$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.\, t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B} \qquad \frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash t.i : A_i}$$

Figure 7: Target $\lambda^\times$-calculus

Monadic laws

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

Monoidal laws

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

Distributivity laws

$$\varnothing \gg\!= f \equiv \varnothing \qquad t \gg\!= \lambda x.\, \varnothing \equiv \varnothing$$
$$(t \circledast u) \gg\!= f \equiv (t \gg\!= f) \circledast (u \gg\!= f)$$
$$t \gg\!= \lambda x.\, (f\ x \circledast g\ x) \equiv (t \gg\!= f) \circledast (t \gg\!= g)$$

Figure 8: Equational theory of abstract multisets

We furthermore expect that abstract multisets satisfy the equational theory from Figure 8. Formally, this means that $\mathfrak{M}\, A$ is a monad with a semimodule structure over $\mathbb{N}$, or alternatively that it is the oidification of a semiring.

We now turn to the Dialectica interpretation itself, which is defined at Figure 9, and that we comment hereafter. We need to define the translation for types and terms. For types, we have two translations $\mathbb{W}(-)$ and $\mathbb{C}(-)$, which correspond to the types of translated terms and stacks respectively. For terms, we also have two translations $(-)^\bullet$ and $(-)_x$, where x is a λ-calculus variable from the source language. According to the thesis defended in this paper, we call $(-)^\bullet$ the *forward transformation*, corresponding to the AD forward pass on functions while accumulating differentials, and $(-)_x$ the *reverse transformation*, which computes differentials.

Theorem 12 (Soundness [Péd14])**.** 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 $x : X \in \Gamma$.

Furthermore, if $t \equiv u$ then $t^\bullet \equiv u^\bullet$ and $t_x \equiv u_x$.

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

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

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

Figure 9: The computational Dialectica

From [Péd14], it follows that the $(-)_x$ translation allows observing the uses of x by the underlying term. Namely, if $t : A$ depends on some variable $x : X$, then $t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X)$ applied to some stack $\pi : \mathbb{C}(A)$ produces the multiset of stacks against which x appears in head position in the Krivine machine when t is evaluated against π.

5.2 A higher-order differential account of the modern Dialectica transformation

In particular, every function in the interpretation comes with the intensional contents of its bound variable as the second component of a pair. We claim that this additional data is essentially the same as the one provided in the Pearlmutter-Siskind untyped translation implementing reverse AD [PS08]. As such, it allows extracting derivatives in this very general setting.

Lemma 13 (Generalized chain rule)**.** Assuming t is a source function, let us evocatively and locally write $t' := t^\bullet.\mathtt{2}$. Let f and g be two terms from the source language and x a fresh variable. Then, writing $f \circ g := \lambda x.\, f\ (g\ x)$, we have

$$(f \circ g)'\ x \equiv \lambda \pi.\, (f'\ (g\ x)^\bullet\ \pi) \gg\!= (g'\ x).$$

It is not hard to recognize this formula as a generalization of the derivative chain rule where the field multiplication has been replaced by the monad multiplication. We do not even need a field structure to express this, as this construction is manipulating free structures, in a categorical sense.

It should be clear by now that the abstract multiset is here to formalize the notion of types endowed with a *sum*. By picking a specific instance of abstract multisets, we can formally show that the Dialectica intepretation computes program differentiation.

Definition 14. We will instantiate $\mathfrak{M}\,(-)$ with the free vector space over $\mathbb{R}$, i.e. inhabitants of $\mathfrak{M}\,A$ are formal finite sums of pairs of terms of type A and values of type $\mathbb{R}$, quotiented by the standard equations. We will write

$$\{t_1 \mapsto \alpha_1, \ldots, t_n \mapsto \alpha_n\}$$

for the formal sum $\Sigma_{0<i\leq n}\ (\alpha_i \cdot t_i)$ where $\alpha_i : \mathbb{R}$ and $t_i : A$.

It is easy to check that this data structure satisfies the expected requirements for abstract multisets, and that ordinary multisets inject into this type by restricting to positive integer coefficients.

We now enrich both our source and target λ-calculi with a type of reals $\mathbb{R}$. We assume furthermore that the source contains functions symbols $\varphi, \psi, \ldots : \mathbb{R} \to \mathbb{R}$ whose semantics is given by some derivable function, whose derivative will be written $\varphi', \psi', \ldots$ The Dialectica translation is then extended at Figure 10. The soundness theorem is then adapted trivially.

$$\mathbb{W}(\mathbb{R}) := \mathbb{R} \qquad \mathbb{C}(\mathbb{R}) := 1$$

$$\varphi^\bullet := (\varphi, \lambda\alpha\,\pi.\,\{() \mapsto \varphi'(\alpha)\}) \qquad \varphi_x := \lambda\pi.\,\varnothing$$

Figure 10: Dialectica Derivative Extension

$$\begin{array}{lcl}
\mathbb{W}(A+B) & := & \mathbb{W}(A) + \mathbb{W}(B) \\
\mathbb{C}(A+B) & := & (\mathbb{W}(A) \to \mathfrak{M}\,\mathbb{C}(A)) \times (\mathbb{W}(B) \to \mathfrak{M}\,\mathbb{C}(B)) \\
\mathbb{W}(\forall\alpha.\,A) & := & \forall\alpha_{\mathbb{W}}.\,\forall\alpha_{\mathbb{C}}.\,\mathbb{W}(A) \\
\mathbb{C}(\forall\alpha.\,A) & := & \exists\alpha_{\mathbb{W}}.\,\exists\alpha_{\mathbb{C}}.\,\mathbb{C}(A)
\end{array}$$

Figure 11: Extensions of Dialectica (types only)

Proposition 15. The following equation holds in the target.

$$(\varphi_1 \circ \ldots \circ \varphi_n)^\bullet.2\ \alpha\ ()\ \equiv \{() \mapsto (\varphi_1 \circ \ldots \circ \varphi_n)'(\alpha)\}$$

Proof. By Lemma 13 and the observation that for any two $\alpha, \beta : \mathbb{R}$ we have

$$\{() \mapsto \alpha \times \beta\} \equiv \{() \mapsto \alpha\} \gg\!= \lambda\pi.\,\{() \mapsto \beta\}.$$

□

We insist that the theory is closed by conversion, so in practice any program composed of arbitrary λ-terms that evaluates to a composition of primitive real-valued functions also satisfies this equation. Thus, Dialectica systematically computes derivatives in a higher-order language.

It is well-known that Dialectica also interprets negative pairs, whose translation will be recalled here. Quite amazingly, they allow to straigthforwardly provide differentials for arbitrary functions $\mathbb{R}^n \to \mathbb{R}^m$.

Let us write $A \times B$ for the negative product in the source language. It is interpreted as

$$\mathbb{W}(A \times B) := \mathbb{W}(A) \times \mathbb{W}(B), \quad \mathbb{C}(A \times B) := \mathbb{C}(A) + \mathbb{C}(B).$$

Pairs and projections are translated in the obvious way, and their equational theory is preserved, assuming a few commutation lemmas in the target [Péd15].

Writing $\mathbb{R}^n := \mathbb{R} \times ... \times \mathbb{R}$ n times, we have the isomorphism

$$\mathbb{C}(\mathbb{R}^n) \to \mathfrak{M}\,\mathbb{C}(\mathbb{R}^m) \cong \mathbb{R}^{nm}.$$

In particular, up to this isomorphism, Theorem 15 can be generalized to arbitrary differentiable functions $\varphi : \mathbb{R}^n \to \mathbb{R}^m$, and the second component of a such function can be understood as an (n, m)-matrix, which is no more than the Jacobian of that function.

Proposition 16. The Dialectica interpretation systematically computes the total derivative in a higher-order language.

The main strength of our approach lies in the expressivity of the Dialectica interpretation. Due to the modularity of our translation, it can be extended to any construction handled by Dialectica, provided the target language is rich enough. For instance, via the linear decomposition [dP89], the source language can be equipped with inductive types. It can also be adapted to second-order quantification and even dependent types [Péd14]. We sketch the type interpretation for sum types and second-order in Figure 11.

This is in stark contrast with other approaches to the problem, that are limited to weak languages, like the simply-typed λ-calculus. The key ingredient of this expressivity is the generalization of scalars to free vector spaces, as $\mathbb{R} \cong \mathfrak{M}\,1$. The monadic structure of the latter allows handling arbitrary type generalizations. The downside of this approach is that one cannot apply the transformation over itself, in apparent contradiction with what happens for differentiable functions.

5.3 Relating Dialectica and the differential λ-calculus

In this section, we relate the two transformations acting on λ-terms in Dialectica with those at stake in differential λ-calculus [ER03]. It was introduced by Ehrhard and Regnier as a syntactic account for the mathematical theory of differential calculus. To the terms of of λ-calculus is added a *differential application* $Ds \cdot u$ which represents the term s linearly applied to u. Linearity is understood through the intuition of call-by-name LL: a linear variable is a variable which is going to be computed exactly one time. It also follows the traditional mathematical intuition, that is *head* variables - acting as functions- are linear: one always have

$$(f+g)(x) = f(x) + g(x)$$

while $f(x+y) = f(x) + f(y)$ asks for a special requirement on f.

That is, during the whole computation, a linear argument u should be used only *once* in $Ds \cdot u$. This is why the authors introduced a new reduction rule for this differential application:

$$(D\lambda z.t) \cdot u \to_{\beta_D} \lambda z. \frac{\partial t}{\partial z} \cdot u.$$

The newly introduced $\frac{\partial t}{\partial z} \cdot u$ is the linear substitution of z by u in t. It is an inductively defined substitution where one chooses to replace a unique linear occurrences of z in t. As this supposes that not all occurrences of z are replaced at the same time, z is still free in the reduced $\frac{\partial t}{\partial z} \cdot u$, and thus $(D\lambda z.t).u$ reduces to a function of z.

The differential λ-calculus We recall the syntax and operational semantics of the differential λ-calculus. This extension of λ-calculus is enriched with a differential application of a term to a value. This differential application will reduce to a linear application. *Linearity in λ-terms is understood in terms of resources*: a variable is linear if and only if it is used exactly once during the reduction of the term. Thus a variable in head-position is always linear, and linear substitution will track down the usage of resources in the λ-term.

Differential λ-calculus also needs to deal with sums of terms. We write simple terms as s, t, u, v, w while sums of terms are denoted with capital letters S, T, U. The set of simple terms is denoted Λ^s and the set of sums of terms is denoted Λ^d. They are constructed according to the following quotient-inductive syntax.

$$\begin{array}{llll} s,t,u,v & \in & \Lambda^s & ::= \quad x \mid \lambda x.s \mid s\,T \mid Ds \cdot t \\ S,T,U,V & \in & \Lambda^d & ::= \quad 0 \mid s \mid S+T \end{array} \qquad 0+T \equiv T \quad T+0 \equiv T \quad S+T \equiv T+S$$

We write $\lambda x. \sum_i s_i$ for $\sum_i \lambda x.s_i$, $(\sum_i s_i)T$ for $\sum_i s_i T$, and $D(\sum_i s_i) \cdot (\sum_j t_j)$ for $\sum_{i,j} Ds_i \cdot t_j$.

The reduction process in differential λ-calculus is the smallest reduction relation following the two rules:

$$\begin{array}{lll} (\lambda x.\, s)\; T & \to_\beta & s\{x \leftarrow T\} \\ D(\lambda x.\, s) \cdot t & \to_{\beta_D} & \lambda x.\, \left(\frac{\partial s}{\partial x} \cdot t\right) \end{array}$$

which is closed by the usual contextual rules.

We consider moreover the simple terms of differential λ-calculus up to η-reduction: in the abstraction $\lambda x.s$, x is supposed to be free in s. We denote $\equiv$ the equivalence relation generated by β, β_D and η.

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ \text{otherwise} \end{cases} \quad (4)$$

$$\frac{\partial(\lambda y.s)}{\partial x} \cdot T = \lambda y. \left(\frac{\partial s}{\partial x} \cdot T \right) \quad (5)$$

$$\frac{\partial 0}{\partial x} \cdot T = 0 \quad (6)$$

$$\frac{\partial(s\ U)}{\partial x} \cdot T = \left(\frac{\partial s}{\partial x} \cdot T \right) U + \left(Ds \cdot \left(\frac{\partial U}{\partial x} \cdot T \right) \right) U \quad (7)$$

$$\frac{\partial(Ds \cdot u)}{\partial x} \cdot T = D \left(\frac{\partial s}{\partial x} \cdot T \right) \cdot u + Ds \cdot \left(\frac{\partial u}{\partial x} \cdot T \right) \quad (8)$$

$$\frac{\partial(S + U)}{\partial x} \cdot T = \frac{\partial S}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T \quad (9)$$

Figure 12: Linear substitution [ER03]

The simply-typed λ-calculus can be extended straightforwardly to handle this generalized syntax, in a way which preserves properties such as subject reduction. In particular the differential can be typed by the rule below.

$$\frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash Ds \cdot t : A \to B}$$

Linear substitution We recall the rules of linear substitution in Figure 12. The central and most intricate of them is the one defining linear substitution on an application. This is illustrated for example in Rule 7 defining linear substitution in an application, which we present here in a simpler form.

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

Let us break the intuitions of this formula for the reader unaccustomed to differential λ-calculus. If z is linear in s, then so it is in $s\ v$. To substitute linearly z by u in $s\ v$, we can then substitute it linearly in s and then apply the result to v. But we can also look for a linear occurrence of z in v. In that case, for v to remain linear in $\frac{\partial v}{\partial z} \cdot u$, we should *linearize* s before applying it to $\frac{\partial v}{\partial z} \cdot u$. Then s will be fed by a linear copy of $\frac{\partial v}{\partial z} \cdot u$, and then it will be fed by u as usual. This last case is exactly the computational interpretation for the chain rule in differential calculus.

Comparing Types Let us compare the types of linearly substituted terms of differential λ-calculus with those of terms which underwent a reverse translation in Dialectica.

Lemma 17. [Buc10, 3.1] Let $\Gamma, x : X \vdash t : A$ and $\Gamma \vdash u : X$. Then $\Gamma, x : X \vdash \frac{\partial s}{\partial x} \cdot u : A$.

In contrast, in Dialectica, one would have:

$$\mathbb{W}(\Gamma), x : X \vdash t_x : \mathbb{C}(A) \Rightarrow \mathfrak{M}\,\mathbb{C}(X).$$

This is particularly obvious when t is an abstraction. While $(\lambda y.s)_x$ is bound to compute on y before computing on x, $\lambda z. \frac{\partial(\lambda y.s)}{\partial x} \cdot z$ does the reverse and first waits for y to be substituted before computing on x.

Relating Dialectica and the differential λ-calculus In what follows, we show that Dialectica and the differential λ-calculus behave essentially the same by defining a logical relation between those two languages. Actually, since we have two classes of objects, witnesses and counters, we need to define not one but two relations mutually. We will implicitly cast pure λ-terms into the differential λ-calculus.

$$
\begin{array}{lcl}
t \sim_{A\to B} T & := & \forall u \sim_A U.\quad (t.1\ u) \sim_B (T\ U) \quad\wedge\quad (t.2\ u) \bowtie^A_B (\lambda z.\, (DT \cdot z)\ U) \\
t \bowtie^X_{A\to B} T & := & \forall u \sim_A U.\quad \lambda\pi.\, t\ (u,\pi) \bowtie^X_B (\lambda z.\, T\ z\ U)
\end{array}
$$

Figure 13: Logical relations for the arrow type

$$
\frac{}{(\lambda\pi.\,\varnothing) \bowtie^X_\alpha 0}
\qquad
\frac{t \bowtie^X_\alpha T \qquad u \bowtie^X_\alpha U}{(\lambda\pi.\, t\ \pi \circledast u\ \pi) \bowtie^X_\alpha T + U}
$$

$$
\frac{\vec{t} \sim_\Gamma \vec{T}}{(\lambda\pi.\,\{\vec{t},\pi\}) \bowtie^{\Gamma\to\alpha}_\alpha (\lambda z.\, z\ \vec{T})}
$$

Figure 14: Atomic closure conditions

Definition 18. Given two simple types A and X, we mutually define by induction on A two binary relations

$$
\begin{array}{lcl}
\sim_A & \subseteq & \{t : \lambda^\times \mid\vdash t : \mathbb{W}(A)\} \times \{T : \Lambda^d \mid\vdash T : A\} \\
\bowtie^X_A & \subseteq & \{\phi : \lambda^\times \mid\vdash \phi : \mathbb{C}(A) \to \mathfrak{M}\,\mathbb{C}(X)\} \times \\
 & & \{K : \Lambda^d \mid\vdash K : X \to A\}.
\end{array}
$$

As is usual, we implicitly close the relation by the equational theory of the corresponding calculi.

- For any atomic type α, we assume given base relations $\sim_\alpha$ and $\bowtie^X_\alpha$ satisfying further properties specified below.
- The recursive case for arrow types is defined at Figure 13.

In the remainder of this section, we assume that the atomic logical relations satisfy the closure conditions of Figure 14. The first two rules ask for the relation to be compatible with the additive structure of $\mathfrak{M}\,(-)$ on the one hand and Λ^d on the other. In the third rule, Γ stands a list of types and all notations are intepreted pointwise. This rule is asking for the compatibility of the return operation of the multiset monad. We do not need an explicit compatibility with $\gg\!=$ because it will end up being provable in the soundness theorem.

Lemma 19. The closure properties of Figure 14 generalize to any simple type.

Theorem 20. If $\Gamma \vdash t : A$ is a simply-typed λ-term, then

- for all $\vec{r} \sim_\Gamma \vec{R}$, $t^\bullet\{\Gamma \leftarrow \vec{r}\} \sim_A t\{\Gamma \leftarrow \vec{R}\}$,
- and for all $\vec{r} \sim_\Gamma \vec{R}$ and $x : X \in \Gamma$,

$$
t_x\{\Gamma \leftarrow \vec{r}\} \bowtie^X_A \lambda z.\, \left(\frac{\partial t}{\partial x} \cdot z\right)\{\Gamma \leftarrow \vec{R}\}.
$$

Proof. As usual, the proof goes by induction over the typing derivation. We need to slightly strengthen the induction hypothesis by proving a generalized form of substitution lemma relating $\gg\!=$ on the left with composition on the right, i.e. for any $\phi \bowtie^Y_X k$ then

$$
(\lambda\pi.\, t_x\{\Gamma \leftarrow \vec{r}\}\ \pi \gg\!= \phi) \bowtie^Y_A \lambda z.\, \left(\frac{\partial t}{\partial x} \cdot (k\ z)\right)\{\Gamma \leftarrow \vec{R}\}
$$

from which the second statement of the theorem is obtained by picking $\phi := \lambda\pi.\,\{\pi\}$ and $k := \lambda z.\, z$, which are always in relation by Lemma 19. The proof is otherwise straightforwardly achieved by equational reasoning. □

This theorem is a formal way to state that the Dialectica interpretation and the differential λ-calculus are computing the same thing without having to embed them in the same language. It makes obvious the relationship between the $(-)_x$ interpretation and the $\frac{\partial -}{\partial x} \cdot -$ operation. As usual, the result depends on the intepretation of atomic types and may be degenerate, e.g. if all atomic types are interpreted as the full relation. Yet, this shows that implication preserves the intended relation regardless of its subcomponents.

Interestingly, $\bowtie_A^X$ relates two functions going in the opposite direction. While the left-hand side has type $\mathbb{C}(A) \to \mathfrak{M}\,\mathbb{C}(X)$ in $\lambda^\times$, the right-hand side has type $X \to A$ in the differential λ-calculus. We believe that this is a reflection of the isomorphism between a linear arrow and its linear contrapositive, since both sides of the relation are actually linear functions.

Remark 5. This distinction in Pédrot's Dialectica between terms which are to be summed and other ones strongly relates with Ehrhard's recent work on deterministic probabilistic coherent spaces [Ehr21].

5.4 Translating Dialectica into differential λ-calculus

In this section, we suggest a translation on counter terms to make the previous logical relation a translation from Dialectica terms to differential λ-calculus. This translation works basically as the logical relation before, but with an enforced CPS translation to retrieve forward differentiation. We have been claiming that Dialectica is categorically and logically a reverse differentiation algorithm where the linearity of arrows has been forgotten. Let us show that when the linearity of counter types is enforced, then it is exactly a reverse differential calculus. To do so, we define a CPS translation $[\![\_]\!]$ defined on terms typed by counter witnesses.

Definition 21. Consider a term s of the $\lambda^\times$-calculus, typed in some context Γ by $\Gamma \vdash s : S$. Define $[\![s]\!]$ as follows:

- If $S = A \times B$ is pair, them then $[\![s]\!] := \lambda k.\,([\![s.2]\!](k(s.1)))$,
- Otherwise $[\![s]\!] := \lambda k.ks$.

Lemma 22. If $s \equiv s'$ then $[\![s]\!] \equiv [\![s']\!]$.

This translation is directed by the intuition that a term typed by a counter type $s : \mathbb{C}(A)$ can be translated to a term typed by a linear dual to a witness type $\mathbb{W}(A) \multimap \bot$, through the rules of classical DiLL (see Lemmas 5 and 6). Taking into account the involutivity of duals in DiLL, we have thus dor $s : \mathbb{C}(A \Rightarrow B) = \mathbb{W}(A) \times \mathbb{C}(B)$:

$$\begin{aligned}[\![s]\!] : \mathbb{W}(A) \times (\mathbb{W}(B) \multimap \bot) &= ((\mathbb{W}(A) \times (\mathbb{W}(B) \multimap \bot)) \multimap \bot) \multimap \bot) \\ &\equiv (\mathbb{W}(A) \multimap \mathbb{W}(B)) \multimap \bot \\ &\equiv (\mathbb{W}(A \Rightarrow B)).1 \multimap \bot.\end{aligned}$$

Consider a pair of a function and its reverse differential $(k, k') : \mathbb{W}(A \Rightarrow B)$. The projection on the first argument of (k, k') interrupts the storage of differentials to compute the function k and the differentials already stored.

For s is a term of the $\lambda^{\times,+}$-calculus we make $[\![\_]\!]$ distribute over sums and translate s into a term of the differential λ-calculus:

$$[\![\emptyset]\!] := \lambda k.k0 \qquad [\![t \circledast u]\!] := [\![t]\!] + [\![u]\!] \qquad [\![\{t\}]\!] := [\![t]\!].$$

Theorem 23. Consider t a simply-typed λ-term, a term of the $\lambda^{\times,+}$-calculus u, and a variable x such that in some context Γ we have $\Gamma; x : X \vdash t : B$ and $\Gamma \vdash u : \mathfrak{M}\,\mathbb{C}(B)$. Then

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

Proof. Let us prove the result by induction on the typing derivation of t, for any variable x and counter witness u. We will make a heavy use of the monadic and monoidal laws on $\mathfrak{M}(\_)$ recalled in Section 5.1. As the operator $\circledast$ is associative and commutative, we will denote $\circledast_i u^i$ the term $u^1 \circledast ... \circledast u^n$ for $n \in \mathbb{N}^*$. For readability issues, when the substitution $[\Gamma \leftarrow \overrightarrow{r}]$ does not play a role in the proof we will omit it.

- The variable case is straightforward. If $t := x$, then as u is a term of pure λ-calculus we have by the monadic laws:
$$[\![u \gg\!= x_x]\!] = [\![u \gg\!= \lambda\pi\{\pi\}]\!] \equiv_\beta [\![u]\!].$$
On the other hand, by η-equivalence we have
$$\lambda z.[\![u]\!](\frac{\partial x}{\partial x} \cdot z) = (\lambda z.[\![u]\!]z) \equiv [\![u]\!].$$
If t equals another variable $y \neq x$, then similarly through the distributivity laws:
$$[\![u \gg\!= y_x]\!] \equiv_\beta [\![u \gg\!= \lambda\pi.\varnothing]\!] = 0$$
and through the laws of differential λ-calculus [ER03]
$$\lambda z.[\![u]\!]\left(\frac{\partial y}{\partial x} \cdot z\right) = \lambda z.[\![u]\!]0 \equiv_\beta \lambda z.(0u) \equiv \lambda z.0 \equiv 0.$$

- Let us tackle the abstraction case and suppose
$$t = \lambda y.s : A \Rightarrow B$$
for some variable $y \neq x$ and some λ-calculus term s. This case is more intricate as, while $(\lambda y.s)_x$ is destined to compute on y before computing on x, $\frac{\partial(\lambda y.s)}{\partial x} \cdot t$ first linearly substitutes x with t before computing on y.
As u has type $\mathfrak{M}(\mathbb{C}(A \Rightarrow B)) = \mathfrak{M}(\mathbb{W}(A) \times \mathbb{C}(B))$, u reduces to a possibly empty sum $\circledast_i\{u^i\}$ with $u^i : \mathbb{W}(A) \times \mathbb{C}(B)$ in the context Γ. Indeed, $\mathfrak{M}$ is a monad over the types of the $\lambda^\times$-calculus, and as such one does not construct pairs of $\mathfrak{M}$ types. By the definition of $(\lambda y.s)_x$, by application of monadic laws and as x is free in u:
$$\begin{aligned}[\![\circledast_i\{u^i\} \gg\!= (\lambda y.s)_x]\!] &\equiv [\![\circledast_i (\lambda y.s)_x u^i]\!] \equiv [\![\circledast_i (\lambda y.s_x) u^i.1 u^i.2]\!] \\ &\equiv \sum_i [\![(s_x[u^i.1/y])u^i.2]\!] \equiv \sum_i [\![\{u^i.2\} \gg\!= (s[u^i.1)/y])_x)]\!].\end{aligned}$$
On the differential λ-calculus side:
$$\begin{aligned}\lambda z.[\![u]\!]\left(\frac{\partial \lambda y.s}{\partial x} \cdot z\right) &\equiv \lambda z.\left(\sum_i [\![u^i]\!]\right)\left(\frac{\partial \lambda y.s}{\partial x} \cdot z\right) \\ &\equiv \sum_i \lambda z.(\lambda k.[\![u^i.2]\!](k(u^i.1))\left(\frac{\partial \lambda y.s}{\partial x} \cdot z\right).\end{aligned}$$
As $\frac{\partial \lambda y.s}{\partial x} \cdot z = \lambda y.\frac{\partial s}{\partial x} \cdot z$, we have:
$$\begin{aligned}\lambda z.[\![u]\!]\left(\frac{\partial \lambda y.s}{\partial x} \cdot z\right) &\equiv \sum_i \lambda z.[\![u^i.2]\!](\lambda y.\left(\frac{\partial s}{\partial x} \cdot z\right) u^i.1) \\ &\equiv \sum_i \lambda z.[\![u^i.2]\!](\left(\frac{\partial s}{\partial x} \cdot z\right)[u^i.1/y]).\end{aligned}$$

As x is free in u, one has

$$\left(\frac{\partial s}{\partial x}\cdot z\right)[u_1^i/y] = \left(\frac{\partial s[u_1^i/y])}{\partial x}\cdot z\right)$$

and the induction hypothesis concludes the case:

$$\begin{aligned}\sum_i \lambda z.[\![u^i.2]\!](\left(\frac{\partial s}{\partial x}\cdot z\right)[u^i.1/y]) &= \sum_i \lambda z.[\![u^i.2]\!]\left(\frac{\partial [u^i.1/y]}{\partial x}\cdot z\right)\\ &\equiv \sum_i [\![u^i.2]\!] \gg\!= (s[u^i.1/y])_x).\end{aligned}$$

- Let us study the application case for the reverse transformation. Suppose that $t = (s)v$, where $s : A \Rightarrow B$ and $v : A$. Then

$$[\![u \gg\!= ((s)v)_x[\Gamma \leftarrow \overrightarrow{r}]]\!] = [\![u \gg\!= (\lambda\pi.(s_x(v^\bullet, \pi) \circledast ((s^\bullet 2)\pi v^\bullet \gg\!= v_x)))]\!]$$

We have $s \equiv \lambda y.s'$ for some term s' of the pure λ-calculus. As such, $s^\bullet 2 \equiv \lambda\pi\lambda y.s'_y\pi$ and $s_x \equiv \lambda\pi.(\lambda y.s'_x)\pi.1\pi.2$ by the soundness theorem 12. Thus, thanks to the distributivity laws (Definition 11):

$$\begin{aligned}&[\![u \gg\!= ((s)v)_x[\Gamma \leftarrow \overrightarrow{r^\bullet}]]\!]\\ &= [\![u \gg\!= (\lambda\pi.((\lambda y.s'_x)v^\bullet\pi) \circledast ((\lambda y.(s'_y\pi)v^\bullet) \gg\!= v_x)[\Gamma \leftarrow \overrightarrow{r^\bullet}]]\!]\\ &= [\![u \gg\!= (\lambda\pi.((\lambda y.s'_x)v^\bullet\pi)[\Gamma \leftarrow \overrightarrow{r^\bullet}]]\!] + [\![u \gg\!= \lambda\pi.(((\lambda y.(s'_y\pi))v^\bullet) \gg\!= v_x)[\Gamma \leftarrow \overrightarrow{r^\bullet}]]\!]\\ &\equiv [\![u \gg\!= (\lambda\pi.(s'_x\pi)[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet]]\!] + [\![u \gg\!= \lambda\pi.((s'_y\pi)[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet]) \gg\!= v_x[\Gamma \leftarrow \overrightarrow{r^\bullet}])]\!]\end{aligned}$$

As x is free in u, and by induction hypothesis on s':

$$\begin{aligned}[\![u \gg\!= (\lambda\pi.(s'_x\pi)[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet]]\!] &\equiv \lambda z.[\![u]\!]\left(\frac{\partial s'[\Gamma, y \leftarrow \overrightarrow{r}, v]}{\partial x}\cdot z\right)\\ &\equiv \lambda z.[\![u]\!](\left(\lambda y.\frac{\partial s'[\Gamma, \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right) v).\end{aligned}$$

By the monadic laws on $\gg\!=$ and by the induction hypothesis on v, we have:

$$\begin{aligned}&[\![u \gg\!= \lambda\pi.((s'_y\pi)[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet]) \gg\!= v_x[\Gamma \leftarrow \overrightarrow{r^\bullet}])]\!]\\ &= [\![(u \gg\!= \lambda\pi.((s'_y\pi)[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet])) \gg\!= v_x[\Gamma \leftarrow \overrightarrow{r^\bullet}])]\!]\\ &\equiv \lambda z.[\![(u \gg\!= \lambda\pi.((s'_y\pi))[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet]]\!]\left(\frac{\partial v[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right)\end{aligned}$$

By induction hypothesis on s' we then have:

$$\begin{aligned}&[\![u \gg\!= \lambda\pi.((s'_y\pi))) \gg\!= v_x[\Gamma, y \leftarrow \overrightarrow{r^\bullet}, v^\bullet])]\!]\\ &\equiv \lambda z.\lambda z'.[\![u]\!]\left(\frac{\partial s'[\Gamma, y \leftarrow \overrightarrow{r}, v]}{\partial y}\cdot z'\right)[v/y]\left(\frac{\partial v[\Gamma, y \leftarrow \overrightarrow{r}, v]}{\partial x}\cdot z\right)\\ &\equiv \lambda z.[\![u]\!]\left((\lambda y.\frac{\partial s'[\Gamma \leftarrow \overrightarrow{r}]}{\partial y}\cdot\frac{\partial v[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z)v)\right)\end{aligned}$$

We conclude the case by computing the the other hand of the equation:

$$\begin{aligned}
&\lambda z.[\![u]\!]\left(\frac{\partial(s)v[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right)\\
&= \lambda z.[\![u]\!](\left(\frac{\partial s[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right)v) + (Ds\cdot\left(\frac{\partial v}{\partial x}\cdot z[\Gamma \leftarrow \overrightarrow{r}]\right))v))\\
&\equiv \lambda z.[\![u]\!]((\left(\frac{\partial s[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right)v) + (\left(\lambda y.\frac{\partial s[\Gamma \leftarrow \overrightarrow{r}]}{\partial y}\cdot\frac{\partial v[\Gamma \leftarrow \overrightarrow{r}]}{\partial x}\cdot z\right)v))\\
&\equiv \lambda z.[\![u]\!]\left((\left(\frac{\partial s}{\partial x}\cdot z\right)v) + (\left(\lambda y.\frac{\partial s}{\partial y}\cdot\frac{\partial v}{\partial x}\cdot z\right)v)\right)[\Gamma \leftarrow \overrightarrow{r}]
\end{aligned}$$

□

6 Conclusion and perspectives

In this paper we related the different interpretations of Gödel's Dialectica with logical differentiation. We demonstrated that Dialectica is basically reverse differentiation without the linear types, and can be refined with them through differential linear logic. We draw three possible outcomes from this.

6.1 Logical axioms and differentiation

Gödel's Dialectica interpretation is known for giving a realizer to Markov's principle

$$\neg\neg\exists x P \longrightarrow \exists x \neg\neg P \quad (\mathsf{MP}).$$

Herbelin proposed another way to realize MP based on statically bound exceptions [Her10]. At the end of his paper, he recalls that MP amounts to double-negation elimination on positive formulas. On these, double-negation is in fact a linear negation followed by a non-linear negation, i.e. MP can be reduced to $?P \longrightarrow P$ for any positive formula P.

Herbelin makes the striking remark that the above type is exactly the one of differentiation. Semantically $?P$ is indeed interpreted as the space of non-linear functions on $P^\perp$, and, in a classical setting, $P \equiv P^\perp \multimap \perp$ is interpreted as the space of linear functions on the space $P^\perp$. Hence MP turns non-linear maps on $P^\perp$ into linear ones.

Herbelin's paper thus breaks down differentiation into smaller computational steps with control operators which could have a mathematical meaning in terms of approximation. Delimited continuations are also used in differentiable programming for propagating reverse differentials of function variables [WZD+19]. Exploring the realizability relation between semi-classical axioms and functional mathematical transformations is an exciting perspective.

6.2 Automatic differentiation and reduction strategies

The Dialectica interpretation explored in this paper is fundamentally call-by-name on the arrow, as we can see in its factorization through LL or in its categorical semantics. This suggests that the call-by-name interpretation of functions and their derivative might implement some kind of reverse derivative. Indeed, in some denotational semantics of DiLL, non-linear functions f can be seen as functions $\tilde{f}$ acting on distributions [Sch66, Ker18]. These come as traditional arguments, encoded through diracs, or as differentiated arguments:

$$\tilde{f}(\delta_a) \longrightarrow f(a) \qquad \tilde{f}(D_0(\_)a) \longrightarrow D_0(f)a.$$

Giving the priority to the evaluation of f (call-by-name) relates to backward differentiation, while giving the priority to $D_0(\_)a$ (call-by-value) relates to forward differentiation. An exploration of the L-calculus [CMM10] adapted to DiLL could lead to the formalization of such principles.

6.3 Proof mining and differentiation

Proof mining [Koh08] consists in applying logical transformations to mathematical proofs, in order to extract more information from these proofs and refine the theorem they prove. This has been particularly effective in functional analysis, where logicians were able to transform existential proofs into quantitative proofs. For instance, from unicity proofs in approximation theory one gets an effective modulus of uniqueness, i.e. a characterization of the rate of convergence of approximants towards the best approximation.

While metatheorems in proof mining guarantee the existence of constructive proofs, applying the Dialectica transformation to proofs might boil down to applying a reverse differentiation procedure to the "ε" function. For example, extracting a quantitative rate of convergence from a unicity statement $\forall\varepsilon\,\exists\eta\,|G_u(a,b)| < \eta \Rightarrow |a-b| < \varepsilon$ would correspond to differentiating the function $\varepsilon \mapsto \eta$. Exploring the consequences of metatheorems in proof mining over logical differentiation could lead to fruitful results.

References

[AAKM10] Shiri Artstein-Avidan, Hermann König, and Vitali Milman. The chain rule as a functional equation. *Journal of Functional Analysis*, 259(11):2999–3024, 2010.

[AF98] Jeremy Avigad and Solomon Feferman. Gödel's functional ('dialectica') interpretation. In Samuel R. Buss, editor, *Handbook of Proof Theory*, pages 337–405. Elsevier Science Publishers, Amsterdam, 1998.

[AP20] Martín Abadi and Gordon D. Plotkin. A simple differentiable programming language. *Proc. ACM Program. Lang.*, 4(POPL):38:1–38:28, 2020.

[BCLS20] Richard Blute, J. Robin B. Cockett, Jean-Simon Pacaud Lemay, and Robert A. G. Seely. Differential categories revisited. *Appl. Categorical Struct.*, 28(2):171–235, 2020.

[BCS06] R. F. Blute, J. R. B. Cockett, and R. A. G. Seely. Differential categories. *Math. Structures Comput. Sci.*, 16(6), 2006.

[BET12] R. Blute, T. Ehrhard, and C. Tasson. A convenient differential category. *Cah. Topol. Géom. Différ. Catég.*, 2012.

[BM20] Davide Barbarossa and Giulio Manzonetto. Taylor subsumes scott, berry, kahn and plotkin. *Proc. ACM Program. Lang.*, 4(POPL):1:1–1:23, 2020.

[BMP20] Aloïs Brunel, Damiano Mazza, and Michele Pagani. Backpropagation in the simply typed lambda-calculus with linear negation. *POPL*, 2020.

[BPRS17] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18:153:1–153:43, 2017.

[Buc10] Manzonetto Bucciarelli, Ehrhard. Categorical models for simply typed resource calculi. *MFPS*, 2010.

[CC14] J. Robin B. Cockett and Geoff S. H. Cruttwell. Differential structure, tangent structure, and SDG. *Appl. Categorical Struct.*, 22(2):331–417, 2014.

[CCG+20] J. Robin B. Cockett, Geoff S. H. Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon D. Plotkin, and Dorette Pronk. Reverse derivative categories. In Maribel Fernández and Anca Muscholl, editors, *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain*, volume 152 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[CFMM16] Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proc. POPL*, 2016.

[CGG+22] Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. page 1–28, Berlin, Heidelberg, 2022. Springer-Verlag.

[CMM10] Pierre-Louis Curien and Guillaume Munch-Maccagnoni. The duality of computation under focus. In Christian S. Calude and Vladimiro Sassone, editors, *IFIP TCS*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 165–181. Springer, 2010.

[Cos92] Roberto Di Cosmo. The linear logic primer, 1992.

[Dil74] Justus Diller. Eine Variante zur Dialectica-Interpretation der Heyting-Arithmetik endlicher Typen. *Archiv für mathematische Logik und Grundlagenforschung*, 16(1-2):49–66, 1974.

[dP89] Valeria de Paiva. A dialectica-like model of linear logic. In *Category Theory and Computer Science, Manchester, UK, September 5-8, 1989, Proceedings*, pages 341–356, 1989.

[dP91] Valeria de Paiva. The dialectica categories. Technical report, University of Cambridge, 1991.

[dPdS21] Valeria de Paiva and Samuel G. da Silva. Kolmogorov-veloso problems and dialectica categories, 2021.

[Ehr02] Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12(5):579–623, 2002.

[Ehr21] Thomas Ehrhard. Coherent differentiation. *CoRR*, abs/2107.05261, 2021.

[Ell18] Conal Elliott. The simple essence of automatic differentiation. In *Proceedings of the ACM on Programming Languages (ICFP)*, 2018.

[ER03] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theoretical Computer Science*, 309(1-3), 2003.

[ER06] T. Ehrhard and L. Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2), 2006.

[Fio07] M. Fiore. Differential structure in models of multiplicative biadditive intuitionistic linear logic. *TLCA*, 2007.

[FO11] Gilda Ferreira and Paulo Oliva. Functional interpretations of intuitionistic linear logic. *Log. Methods Comput. Sci.*, 7(1), 2011.

[Göd58] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12:280–287, 1958.

[GW08] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, USA, second edition, 2008.

[Hed14] Jules Hedges. Dialectica categories and games with bidding. In Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau, editors, *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France*, volume 39 of *LIPIcs*, pages 89–110. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014.

[Her10] Hugo Herbelin. An intuitionistic logic that proves markov's principle. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 50–56. IEEE Computer Society, 2010.

[Ker18] Marie Kerjean. A logical account for linear partial differential equations. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 589–598, 2018.

[Koh08] Ulrich Kohlenbach. *Applied Proof Theory - Proof Interpretations and their Use in Mathematics*. Springer Monographs in Mathematics. Springer, 2008.

[KPJK+22] Faustyna Krawiec, Simon Peyton Jones, Neel Krishnaswami, Tom Ellis, Richard A. Eisenberg, and Andrew Fitzgibbon. Provably correct, asymptotically efficient, higher-order reverse-mode automatic differentiation. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022.

[KT16] M. Kerjean and C. Tasson. Mackey-complete spaces and power series. *MSCS*, 2016. Publisher: Cambridge University Press.

[Lin76] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16:146–160, 1976.

[MP21] Damiano Mazza and Michele Pagani. Automatic differentiation in PCF. *Proc. ACM Program. Lang.*, 5(POPL):1–27, 2021.

[Péd14] Pierre-Marie Pédrot. A functional functional interpretation. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 77:1–77:10, 2014.

[Péd15] Pierre-Marie Pédrot. *A Materialist Dialectica. (Une Dialectica matérialiste).* PhD thesis, Paris Diderot University, France, 2015.

[Pow16] Thomas Powell. Gödel's functional interpretation and the concept of learning. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '16, page 136–145, New York, NY, USA, 2016. Association for Computing Machinery.

[PS08] Barak A. Pearlmutter and Jeffrey Mark Siskind. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.*, 30(2):7:1–7:36, 2008.

[Sch66] L. Schwartz. *Théorie des distributions.* Publications de l'Institut de Mathématique de l'Université de Strasbourg, No. IX-X. Nouvelle édition, entiérement corrigée, refondue et augmentée. Hermann, Paris, 1966.

[Vák21] Matthijs Vákár. CHAD: combinatory homomorphic automatic differentiation. *CoRR*, abs/2103.15776, 2021.

[Wen64] R. E. Wengert. A simple automatic derivative evaluation program. *Commun. ACM*, 7(8):463–464, aug 1964.

[WZD+19] Fei Wang, Daniel Zheng, James Decker, Xilun Wu, Grégory M. Essertel, and Tiark Rompf. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *Proc. ACM Program. Lang.*, 3(ICFP):96:1–96:31, July 2019.