∂ is for Dialectica

Marie Kerjean
marie.kerjean@cnrs.fr
CNRS, LIPN, Université Sorbonne Paris Nord
France

Pierre-Marie Pédrot
pierre-marie.pedrot@inria.fr
INRIA
France

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 backpropagation algorithm has been brought to a modern light by its application to neural networks. This work unveils a surprising connection between backpropagation and Gödel's Dialectica interpretation, a logical translation that realizes semi-classical axioms. This unexpected correspondence is exploited through different logical settings. In particular, we show that the computational interpretation of Dialectica translates to the differential λ-calculus and that Differential Linear Logic subsumes the logical interpretation of Dialectica.

CCS CONCEPTS

• **Mathematics of computing** → **Differential calculus**; • **Theory of computation** → **Linear logic**; **Type theory**; *Control primitives.*

KEYWORDS

Dialectica, Automatic Differentiation, Linear Logic, Realizability, Category Theory

ACM Reference Format:
Marie Kerjean and Pierre-Marie Pédrot. 2024. ∂ is for Dialectica. In *39th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS '24), July 8–11, 2024, Tallinn, Estonia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3661814.3662106

1 INTRODUCTION

Dialectica was originally introduced by Gödel in a famous paper [23] as a way to constructively interpret an extension of higher-order arithmetic [4]. It 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 [15]. Yet another way to look at Dialectica is to consider it as a program translation, or more precisely *two* mutually defined translations of the λ-calculus exposing intensional information [34]. Meanwhile, in its logical outfit, Dialectica led to numerous applications and was tweaked into an unending array of variations in the proof mining community [28].

LICS '24, July 8–11, 2024, Tallinn, Estonia

ACM ISBN 979-8-4007-0660-8/24/07
https://doi.org/10.1145/3661814.3662106

In a different scientific universe, Automatic Differentiation [24] (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 [43] computes functions and their derivatives pairwise in the order they are provided, while on the other hand reverse differentiation [30] 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-step semantics [1], continuations [42], functoriality [21, 40], and linear types [10, 37]. 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. 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 [20] (DiLL), differential categories [8], or the differential λ-calculus [19]. To support our thesis with evidence, we will draw a correspondence between each of these objects and the corresponding Dialectica interpretation.

2 SUMMARY OF THE RESULTS

Our main thesis is that the computational content of the Dialectica interpretation *is* reverse differentiation. In particular, the Dialectica interpretation of function composition boils down to the chain rule. We would like however to expose immediately the kernel of this correspondence.

2.1 A Dialectica Primer

In its most traditional form, Dialectica acts as some elaborate prenexation of formulas from intuitionnistic arithmetic HA into higher-order arithmetic HA^ω. The soundness theorem of the interpretation can be stated as follows.

$$\text{If} \quad \vdash_{\mathsf{HA}} A \quad \text{then} \quad \vdash_{\mathsf{HA}^\omega} \exists x : \mathbb{W}(A).\, \forall y : \mathbb{C}(A).\, x \perp_A y. \qquad (1)$$

This should be read as a realizability model, where $\mathbb{W}(A)$ is the simple type of realizers or witnesses of A, $\mathbb{C}(A)$ the simple type of stacks or counters of A and $\perp_A \subseteq \mathbb{W}(A) \times \mathbb{C}(A)$ the realizability condition for A.

The need for higher-order terms is a staple of realizability, where logical implications are interpreted as some flavour of higher-order functions. By the time it was invented, Dialectica was the first of its kind. Although published quite late, it was indeed designed by Gödel even before the inception of the concept of realizability by Kleene. This primitive essence resulted in Dialectica being nicknamed *the functional interpretation*, but retrospectively other realizability models are no less functional.

As another side-effect of its antiqueness, Dialectica is somewhat shrouded in a veil of mystery. A good part of this is due to historical cruft that can be easily scrubbed away. For instance, the original version notoriously relies on sequences of objects for the $\mathbb{W}$ and $\mathbb{C}$ mentioned above. This is just an artifact encoding that can be advantageously replaced by products. Indeed, to interpret positive connectives the sequence-based presentation requires encoding away algebraic types in an ad-hoc way, e.g. the sum type $A + B$ is encoded as $\mathbb{N}; A; B$. This requires both A and B to feature dummy inhabitants to build the injections, and this also destroys the usual equational theory of sum types as the encoding is not bijective. For simplicity we will also ditch sequences.

Since we will be presenting a version of Dialectica that preserves the equational theory of the λ-calculus in Section 4, we will not detail the full historical interpretation here but simply hint at the important facts. While most connectives follow the BHK interpretation in Dialectica, the surprise stems from the interpretation of implication [4].

$$\begin{aligned}
\mathbb{W}(A \Rightarrow B) &:= \begin{cases} (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \\ \times \\ (\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A)) \end{cases} \\
\mathbb{C}(A \Rightarrow B) &:= \mathbb{W}(A) \times \mathbb{C}(B) \\
(f, \phi) \perp_{A \Rightarrow B} (u, \pi) &:= u \perp_A \phi\, u\, \pi \Rightarrow f\, u \perp_B \pi
\end{aligned}$$

Contrarily to the usual BHK interpretation as implemented in e.g. Kreisel realizability, implications $A \Rightarrow B$ are not simply interpreted as maps $\mathbb{W}(A) \Rightarrow \mathbb{W}(B)$ from witnesses to witnesses, but they additionally carry a function of type $\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A)$ mapping counters to counters in the reverse direction.

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 [28, 8.1]. It is also presented as a way to "extract constructive information through a purely local procedure" [4, 3.3].

The historical presentation of Dialectica is merely a *logical* interpretation rather than a *computational* one. By this we mean that it maps formulas provable in HA to properties that hold in the meta, but in general it does not preserve the equational theory of HA seen as a kind of λ-calculus [35]. To say it in a categorical fashion, one can always collapse a category to a preorder by identifying all morphisms. In this light, the historical Dialectica is a functor between the collapsed preorders, but not between the original proof-relevant categories. In spite of this limitation, the historical Dialectica is already good enough to understand our original claim.

2.2 The Chain Rule

The chain rule is the bread and butter of calculus, let alone automatic differentiation. Writing $D_a f$ for the differentiation of f at a, the chain rule is the name of the high-school equation

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

that describes differentation of the composition of two functions in terms of the differentiation of these functions.

Just by looking at its type, one can already forecast that the second parameter in the Dialectica witness of an implication will validate a chain rule. (Equation 3). Let us make this explicit. Let $(f, \phi) : \mathbb{W}(A \Rightarrow B)$ and $(g, \psi) : \mathbb{W}(B \Rightarrow C)$ two witnesses of the corresponding implication through the Dialectica interpretation. That is, it is the case that for all $u : \mathbb{W}(A), v : \mathbb{W}(B), \pi : \mathbb{C}(B)$ and $\rho : \mathbb{C}(C)$ we have:

$$\begin{aligned}
(f, \phi) \perp_{A \Rightarrow B} (u, \pi) &:= u \perp_A \phi\, u\, \pi \Rightarrow f\, u \perp_B \pi \\
(g, \psi) \perp_{B \Rightarrow C} (v, \rho) &:= v \perp_B \psi\, v\, \rho \Rightarrow g\, v \perp_C \rho.
\end{aligned}$$

The Dialectica interpretation of the composition provides a solution $(h, \chi) : \mathbb{W}(A \Rightarrow C)$ satisfying the system below for any $u : \mathbb{W}(A)$ and $\rho : \mathbb{C}(C)$.

$$(h, \chi) \perp_{A \Rightarrow C} (u, \rho) := u \perp_A \chi\, u\, \rho \Rightarrow h\, u \perp_C \rho$$

This solution amounts to the following equations.

$$h\, u = g\, (f\, u) \tag{2}$$

$$\chi\, u\, \rho = \phi\, u\, (\psi\, (f\, u)\, \rho) \tag{3}$$

While the first equation represents the traditional functoriality of composition, the second equation is almost exactly the chain rule. Indeed, let us write $D_a(f, \phi) := \phi\, a$. Then equation 3 can be written as

$$D_a((g, \psi) \circ_D (f, \phi)) = D_{f(a)}(g, \psi) \bullet D_a(f, \phi) \tag{4}$$

where $\circ_D$ is the Dialectica interpretation of the composition and $f \bullet g := g \circ f$. We discuss this reversal a bit later, but for now, let us assert the main thesis of this paper.

Thesis 1. *The pair (f, ϕ) of Dialectica witness of an implication computes a function f and its differential ϕ.*

In analysis, many functional transformations satisfy the chain rule [3]. Our goal is thus to strengthen our claim and leave no doubt to the reader that Dialectica is indeed differentiation.

2.3 Reverse Automatic Differentiation

Let us now focus on the puzzling reversal of the chain rule. Remember that assuming a function $f : A \to B$, its differential is usually given the type $Df : A \to A \multimap B$. The second arrow actually stands for a linear map from A to B. Indeed, the differential of f at any point can be defined as the best linear approximation of a function at that point.

By contrast and as we have seen, in Dialectica, 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 $\mathbb{W}(A) \Rightarrow \mathbb{C}(B) \Rightarrow \mathbb{C}(A)$. This is what forces us to write function composition as the chain rule the other way around.

Thankfully, this is a well-understood phenomenon, both from the point of view of differentation and realizability. If we were to understand $\mathbb{C}(A)$ as a kind of *dual* of the space $\mathbb{W}(A)$, the second projection of the witness of an arrow would have the type of a

reverse differential. This dualization is not surprising from the realizability standpoint either, as $\mathbb{C}(A)$ materializes the type of *stacks* of A. Stacks appear naturally when in the context of abstract machines, where they are dual to terms. As a matter of fact, stacks play a critical role in classical realizability where they are given a first-class citizenship, and can be typed by types that dualize the types of terms.

More astonishingly, it turns out that this dualization *already* exists in the realm of automatic differentation. It is indeed the core of the process of *reverse automatic differentiation*, which was introduced in machine learning for algorithmic efficiency reasons. 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 that the derivative is computed via a *continuation-passing style* transformation [42]. This is made clear by looking at the types of these operations. While the usual forward differential of a function $f : A \to B$ has type $\overrightarrow{D}f : A \to A \multimap B$, its reverse differential has type $\overleftarrow{D}f : A \to B^\perp \multimap A^\perp$, where $A^\perp$ typically stands for the linear dual of the vector space A. In finite dimensions, those two types are isomorphic but this is not generally the case otherwise.

2.4 Linear Substitution

The last ingredient to understand why Dialectica is reverse differentiation is the notion of *linear substitution*, introduced by Ehrhard and Regnier in their differential λ-calculus [19]. This calculus introduces a new kind of function former $D\,t \cdot u$ called the *differential* of t at point u, which can be typed with the following typing rule.

$$\frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash D\,t \cdot u : A \to B}$$

In addition to the usual β-rule from the λ-calculus, which turns a β-redex into a substitution, the differential λ-calculus contains a rule β_D that turns a differential redex into a *linear* substitution.

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

Linear substitution is a process that sums over all the linear occurrences of a variable, see Section 4.4 for a in-depth definition. Importantly, linear substitution of application enjoys the following equation.

$$\frac{\partial (s\ u)}{\partial x} \cdot t = \left(\frac{\partial s}{\partial x} \cdot t\right) u + \left(D\,s \cdot \left(\frac{\partial u}{\partial x} \cdot t\right)\right) u \tag{5}$$

The left hand side of the sum stands for the linear substitution of the implicit bound variable of s in head position, while the right hand side is the translation of the chain rule.

This definition has a direct equivalent in the Dialectica interpretation of application, as long as we pick the modern variant of Dialectica [15, 34], which is a generalization of the Diller-Nahm flavour [16] and will be formally presented in Section 4. The reverse translation of application is defined as follows.

$$(s\ u)_x\ \pi := \left(s_x\ (u^\bullet, \pi)\right) \circledast \left((s^\bullet.2)\ \pi\ u^\bullet \gg\!= u_x\right) \tag{6}$$

Let us give some intuitions. The transformation $(\_)^\bullet$ corresponds morally to the forward component of a Dialectica arrow, seeing a term in a context as a function from that context. Dually, $(\_)_x$ is the reverse arrow associated to a given variable x from the context. As already explained, π is to be understood as a continuation.

Contrarily to the differential λ-calculus that makes implicit use of multisets as bags, the modern Dialectica relies on an explicit bag-like monad which adds a bit of noise. Here, the $\gg\!=$ operation corresponds to the bind of this monad, while the $\circledast$ operation stands for a formal sum akin to bag union.

Forgetting about this monadic noise and writing $(s)^\bullet.2$ as $D\,s$, the similarities between Equation 5 and Equation 6 are now striking. In fact, the transformation $t_x\{x := v\}$ is to be understood as a continuation passing style version of the linear substitution $\frac{\partial t}{\partial x} \cdot v$. We will also argue that $(\_)^\bullet$ stands for a functorial reverse differentiation transformation.

Thesis 2. *The second component of the Dialectica translation of an arrow computes the linear substitution of the source term.*

This correspondence will be made formal through a logical relation in Section 4.10 and through an ad-hoc translation in Section 4.13.

2.5 Dialectica and Linear Logic

Differential linear logic (DiLL) [20, 32] and differential categories [8, 22] express differentiation directly, without relying on the previously exposed notion of linear substitution. In these systems, differentiation as an operation from non-linear proofs (resp. morphisms) to linear proofs (resp. morphisms).

In the case of DiLL, this is achieved by the introduction of new inferences rules which make the following derivation admissible:

$$\dfrac{!A \vdash B}{\dfrac{}{A \vdash B}}$$

Proofs of $A \vdash B$ are linear proofs, while proofs of $!A \vdash B$ are understood as non-linear proofs of B under the hypothesis A.

In the case of differential categories, differentiation is introduced as a natural transformation $\bar{d} : !A \to A$.

On the one hand, the Dialectica translation is already known to factor through LL. In this paper, we show that, in fact, it factors through DiLL (Proposition 5.4). More importantly, Dialectica does nothing more than using DiLL rules on LL implications (Proposition 5.1). For this result to go through, the classical nature of DiLL is essential so as to interpret the reverse part of Dialectica by duality.

On the other hand, we show that Dialectica categories over co-Kleisli categories of differential categories readily embed the reverse differentiation functor (Proposition 5.7). This contrasts with traditional approaches where Dialectica categories are seen as an instance of the Chu translation, and used to construct new categorical models of LL [26].

2.6 Related work

As far as we know, this is the first time a parallel has been drawn between Dialectica and reverse differentiation. However, in hindsight several lines of work around Dialectica are ominously reminiscent of differentiable programming. Powell [36] formally relates the concept of learning with Dialectica realizers. His definition

of learning algorithm is tied to the notion of approximation. Differentiation being just the best linear approximation, our work merely formalizes this relation with linearity. From the categorical point of view, Dialectica is related to lenses [25], which provide themselves a sound categorical interpretation for gradient based learning [13]. More generally, Dialectica is also known to extract *quantitative* information from proofs [28], which relates very much with the quantitative point of view that differentiation has brought to λ-calculus [5].

2.7 Outline

We begin this paper in Section 3 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. The most involved section is Section 4, devoted to the computational interpretation of Dialectica. There we recall Pédrot's sound computational interpretation [34] and the rules of the differential λ-calculus, to finally show that Pédrot's reverse translation corresponds on first-order terms to a reverse version of the differential λ-calculus linear substitution. In Section 5 we show that the factorization of Dialectica through LL refines to DiLL. Then in Section 5.2 we recall the definition of a Dialectica category and show that it factors through $*$-autonomous differential categories, which are exactly the models of DiLL. We conclude by some perspective on the possible outcomes of this correspondence.

3 DIFFERENTIABLE PROGRAMMING

We give here an introduction to Automatic Differentiation (AD) oriented towards differential calculus 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 [6] for a more comprehensive introduction to automatic differentiation. We write $D_t(f)$ for the differential of f at t. We denote by $- \cdot -$ the pointwise multiplication of reals or real functions.

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

$$D_t(g \circ f) = D_{f(t)}(g) \circ D_t(f).$$

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 the function $D_{f(t)}(g)$, then computing $D_t(f)$ and lastly the application of $D_{f(t)}(g)$ to $D_t(f)$. Conversely, forward differentiation computes first $f(t)$, then $D_t(f)$, then only $g(f(t))$, 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 [10].

These two techniques have different efficiency profiles, depending on the dimensions 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. The systems we will present in Section 4.1 are not designed with efficiency in mind, and will in particular be completely oblivious to this kind of optimizations. To the risk of repeating ourselves, algorithmic efficiency is not the purpose of this paper, rather we wish to weave equational links between differentiation and Dialectica. As such we do not prove any complexity result.

Brunel, Mazza and Pagani [10] refined the functional presentation by Wang and al. [42] using a linear negation on ground types, and provided complexity results. What we present now is very close in spirit, although their work relies mainly on computational graphs while ours is directed towards type systems and functional analysis. At the core, and as in most of the literature [1, 21, 42], their differential transformation acts on pairs in the linear substitution calculus [2], so as to make it compositional. Consider $f : \mathbb{R}^n \to \mathbb{R}^m$ 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

$$\overrightarrow{D}(f) : (a, x) \in \mathbb{R}^n \times \mathbb{R}^n \mapsto (f(a), D_a f \cdot x) \in \mathbb{R}^m \times \mathbb{R}^m$$

where $- \cdot -$ stands for the application of the matrix $D_a f$ to the vector x.

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 $\mathbb{R}$.

$$\overleftarrow{D}(f) : \mathbb{R}^n \times \mathbb{R}^{m\perp} \to \mathbb{R}^m \times \mathbb{R}^{n\perp}$$
$$(a, x) \mapsto (f(a), (v \mapsto v \cdot (\mathrm{D}_a f \cdot x)))$$

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.

Remark 1. 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 $f : E \to F$ between (topological) vector spaces, one has:

$$\overleftarrow{D}(f) : E \times F' \to F \times E'$$
$$(a, \ell) \mapsto (f(a), \ell \circ \mathrm{D}_a f)$$

Consider a function $g : F \to G$. Then $\overleftarrow{D}(g)$ has the type $F \times G' \to G \times F'$. If G and F are not self-dual, there is no way to define the composition of $\overleftarrow{D}(f)$ with $\overleftarrow{D}(g)$. Thus 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 4.

However, many functional transformations satisfy the chain rule [3]. Our goal is thus to strengthen our claim and leave no doubt to the reader that Dialectica is indeed differentiation.

4 THE COMPUTATIONAL DIALECTICA AND BACKPROPAGATION

In this section, we tackle what we believe to be the most solid link between Dialectica and Reverse Differentiation. We show that the computational content of Dialectica embeds into a continuation-passing-style differential λ-calculus. This is done via logical relations in Section 4.5 and via an ad-hoc translation in Section 4.6. We first recall the computational presentation of Dialectica in Section 4.1, before exhibiting the chain rule in this context in Section 4.2 and insisting on its rich type theory in Section 4.3.

4.1 An account of the modern Dialectica transformation

After its original presentation by Gödel, Dialectica has been refined as a logical transformation acting from LL to the simply-typed λ-calculus with pairs and sums, by looking at the witness and counter types [15].

This presentation allows removing a lot of historical complexity, including the need for sequences of variables.

In modern terms, one would call it a realizability interpretation over an extended λ-calculus, whose effect is to export intensional content from the underlying terms, i.e. the way variables are used. In its first version 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 decidability condition and makes Dialectica preserve β-equivalence, leading to a kind of Diller-Nahm variant [16].

We recall below the Dialectica translation of the simply-typed λ-calculus. 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 4.1. An abstract multiset structure is a parameterized type $\mathfrak{M}\,(-)$ equipped with the following primitives.

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

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

We furthemore expect that abstract multisets satisfy the following equational theory. Formally, this means that $\mathfrak{M}\,A$ is a monad with a semimodule structure over $\mathbb{N}$.

Monadic laws.

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

$$A, B ::= \alpha \mid A \Rightarrow B \mid A \times B$$
$$t, u ::= x \mid \lambda x.\, t \mid t\ u \mid (t, u) \mid t.1 \mid t.2$$
$$\begin{array}{rcll} (\lambda x.\, t)\ u & \to_\beta & t[x \leftarrow u] & \\ t & \equiv_\eta & \lambda x.\, t\ x & x \notin t \\ (t_1, t_2).i & \to_\beta & t_i & \\ t & \equiv_\eta & (t.1, t.2) & \end{array}$$

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

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.\, t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B}$$

$$\frac{\Gamma \vdash t : A_1 \times A_2}{\Gamma \vdash t.i : A_i}$$

Figure 1: Target $\lambda^\times$-calculus

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.

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

We now turn to the Dialectica interpretation itself, which is defined at Figure 2, 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, and $(-)_x$ the *reverse transformation*.

Theorem 4.2 (Soundness [34]). *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 $t_x \equiv u_x$.

From [34], 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.

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

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

Figure 2: The computational Dialectica

4.2 A 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 [33]. As such, it allows extracting derivatives in this very general setting.

Lemma 4.3 (Generalized chain rule). *Assuming t is a source function, let us evocatively and locally write $t' := t^\bullet.2$. 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*

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

One can recognize this formula as a generalization of the derivative chain rule where the scalar multiplication, or the composition between functions, has been replaced by the monad multiplication.

The abstract multiset is here to formalize the notion of types endowed with a *sum*, i.e. a commutative monoid structure. By picking a specific instance of abstract multisets, we can formally show that the Dialectica interpretation computes program differentiation.

Definition 4.4. 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 equations 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

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

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

Figure 3: Dialectica Derivative Extension

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 3.

The soundness theorem is then adapted trivially. As a direct consequence of Lemma 4.3 and the observation that for any two $\alpha, \beta : \mathbb{R}$ we have

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

and thus the following theorem.

Theorem 4.5. *The following equation holds in the target.*

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

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.

4.3 Higher dimensions

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

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

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

Pairs and projections are translated in the obvious way, and their equational theory is preserved, assuming a few commutation lemmas in the target [35].

Writing $\mathbb{R}^n := \mathbb{R} \times \ldots \times \mathbb{R}$ 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 4.5 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.

Theorem 4.6. *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 [15], the source language can be equipped with inductive types. It can also be adapted to second-order quantification and even dependent types [31, 34]. We sketch the type interpretation for sum types and second-order quantification in Figure 4.

This is in stark contrast with other approaches to the problem, that are limited to weak languages, like the simply-typed λ-calculus.

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

Figure 4: Extensions of Dialectica (types only)

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.

4.4 The differential λ-calculus

In this section, we give a quick recap of the syntax of the differential λ-calculus. This will allow us to relate Dialectica with differentials on λ-terms in Sections 4.5 and 4.6.

The differential λ-calculus was introduced by Ehrhard and Regnier [19] as a syntactic account for the mathematical theory of differential calculus. It extends the λ-calculus with a *differential application* $D\,s\cdot u$ 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 once. It also follows the traditional mathematical intuition, that is *head* variables—acting as functions—are linear: by definition, one always has

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

while

$$f(x+y)=f(x)+f(y)$$

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

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

The newly introduced $\frac{\partial t}{\partial z}\cdot u$ is called the linear substitution of z by u in t. Just like the usual substitution, linear substitution is a meta-theoretical operation defined by induction on t. Contrarily to the former, it only replaces a single linear occurrence of z in t. As a result, not all occurrences of z are replaced at the same time, hence z may still be free in $\frac{\partial t}{\partial z}\cdot u$, and thus $D\,(\lambda z.t)\cdot u$ reduces to a λ-abstraction that binds z. We now detail the syntax and operational semantics of this calculus.

Terms of the differential λ-calculus. 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.

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

$$\frac{\partial y}{\partial x}\cdot T=\begin{cases}T & \text{if } x=y\\ 0 & \text{otherwise}\end{cases}\tag{7}$$

$$\frac{\partial(\lambda y.s)}{\partial x}\cdot T=\lambda y.\left(\frac{\partial s}{\partial x}\cdot T\right)\tag{8}$$

$$\frac{\partial\,0}{\partial x}\cdot T=0\tag{9}$$

$$\frac{\partial(s\;U)}{\partial x}\cdot T=\left(\frac{\partial s}{\partial x}\cdot T\right)U+\left(D\,s\cdot\left(\frac{\partial U}{\partial x}\cdot T\right)\right)U\tag{10}$$

$$\frac{\partial(D\,s\cdot u)}{\partial x}\cdot T=D\left(\frac{\partial s}{\partial x}\cdot T\right)\cdot u+D\,s\cdot\left(\frac{\partial u}{\partial x}\cdot T\right)\tag{11}$$

$$\frac{\partial(S+U)}{\partial x}\cdot T=\frac{\partial S}{\partial x}\cdot T+\frac{\partial U}{\partial x}\cdot T\tag{12}$$

Figure 5: Linear substitution

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

Reduction in the differential λ-calculus is the smallest relation generated by the two rules:

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

and 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, η and associativity axioms for $+$.

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.

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

Linear substitution. We recall the rules of linear substitution in Figure 5. The central and most intricate of them is the one defining linear substitution on an application. It follows the simple fact that a linear variable should be used exactly once. This is illustrated for example in the rule for linearly substituting in an application 10, which we present here in a simpler form.

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

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 the computational interpretation for the chain rule in differential calculus.

Types of terms that are used in linear substitution are simpler than in Dialectica.

Lemma 4.7. *[11, 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 a λ abstraction. While $(\lambda y.s)_x$ is destined 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.

4.5 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 recursively. We will implicitly cast pure λ-terms into the differential λ-calculus.

Definition 4.8. Given two simple types A and X, we mutually define by induction on A two binary relations

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

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 $\bowtie_\alpha^X$ satisfying further properties specified below.
- The recursive case for arrow types is defined at Figure 6.

In the remainder of this section, we assume that the atomic logical relations satisfy the closure conditions of Figure 7. 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 4.9. *The closure properties of Figure 7 generalize to any simple type.*

Theorem 4.10. *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 $x : X \in \Gamma$,*

$$t_x[\Gamma \leftarrow \vec{r}] \bowtie_A^X \lambda z. \left(\frac{\partial t}{\partial x} \cdot z\right)[\Gamma \leftarrow \vec{R}].$$

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 $\phi \bowtie_X^Y k$ then

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

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 4.9. 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. 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 2. This distinction in Pédrot's Dialectica between summable and non-summable terms strongly relates with Ehrhard's recent work on deterministic probabilistic coherent spaces [18].

4.6 Translating Dialectica into differential λ-calculus

In this section, we show that the two transformation acting on λ-terms in Dialectica are a CPS version of the ones in differential λ-calculus. We define 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.

Definition 4.11. 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, then $[\![s]\!] := \lambda k.\, ([\![s.2]\!](k(s.1)))$,
- Otherwise $[\![s]\!] := \lambda k.ks$.

Lemma 4.12. *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 $\mathbb{W}(A)^\perp := \mathbb{W}(A) \multimap \perp$, through the rules of Linear Logic (see Section 5 later). Taking into account the involutivity of duals in Linear Logic, we have thus for a term s of type $\mathbb{C}(A \Rightarrow B) = \mathbb{W}(A) \times \mathbb{C}(B)$:

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

For s 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 4.13. *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*

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

The proof goes by induction on the typing derivation of t, for any variable x and counter witness u. It makes a heavy use of the monadic and monoidal laws on $\mathfrak{M}\,(\_)$ recalled in Section 4.1, and of the fact that the linearity of witnesses is now enforced.

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

Figure 6: Logical relations for the arrow type

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

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

Figure 7: Atomic closure conditions

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

Figure 8: A modernized Dialectica [35, 8.3.1]

5 DIALECTICA FROM LL TO DiLL

The close relationship between Dialectica and Linear Logic was identified as the second was found. Dialectica factorizes through Linear Logic and transports its structure to new models for it. In this section, we show that, more precisely, Dialectica merely adds rules from Differential Linear Logic to Linear Logic, and embeds faithfully reverse functorial differentiation in its arrows. We do that in sequent calculus in Section 5.1 and in categorical models of Dialectica in 5.2.

5.1 The linear Dialectica is differential

In this section, we show that Dialectica factorizes through DiLL, and that rules of DiLL prove Dialectica.

Linear Logic. Formulas of LL are constructed according to the following grammar.

$$
\begin{array}{rcl}
A, B & := & 0 \mid 1 \mid \bot \mid \top \mid A \oplus B \mid A \,\&\, B \mid \\
 & & A \parr B \mid A \otimes B \mid\ !A \mid\ ?A
\end{array}
$$

We define as usual the involutive negation $(-)^\perp$, & being the dual of $\oplus$, $\otimes$ the dual of $\parr$ and ! the dual of ?. As per the standard practice, we define the *linear implication* $A \multimap B := A^\perp \parr B$, 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.

Dialectica through Linear Logic. The Dialectica translation can in fact be modernized as a translation to and from types of $\lambda^{+,\times}$-calculus: this was already described on implications in Figure 2 and is fully recalled in Figure 8.

$$
\begin{array}{llllll}
\mathbb{W}(0) & := & 1 & \mathbb{W}(A\oplus B) & := & \mathbb{W}(A) + \mathbb{W}(B) \\
\mathbb{C}(0) & := & 1 & \mathbb{C}(A\oplus B) & := & \mathbb{C}(A)\times\mathbb{C}(B) \\
\mathbb{W}(1) & := & 1 & \mathbb{W}(A\otimes B) & := & \mathbb{W}(A)\times\mathbb{W}(B) \\
\mathbb{C}(1) & := & 1 & \mathbb{W}(!A) & := & \mathbb{W}(A) \\
\mathbb{W}(A^\perp) & := & \mathbb{C}(A) & \mathbb{C}(!A) & := & \mathbb{W}(A) \Rightarrow \mathbb{C}(A) \\
\mathbb{C}(A^\perp) & := & \mathbb{W}(A) & & & \\
\multicolumn{6}{c}{\mathbb{C}(A\otimes B) \quad := \quad (\mathbb{W}(A) \Rightarrow \mathbb{C}(B)) \times (\mathbb{W}(B) \Rightarrow \mathbb{C}(A))}
\end{array}
$$

Figure 9: Witness and counter types for LL formulas into $\lambda^{+,\times}$-calculus [15] [34].

This translation factorizes as a translation from LL connectives into intuitionistic types [15], described in Figure 9[1] It factorizes through the linear Dialectica by injecting LJ into LL via the economical translation $[\![\_]\!]_e$, call-by-name on arrows and call-by-value on products.

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

$$
\begin{array}{ccc}
 & \mathsf{LL} & \\
{\scriptstyle [\![\_]\!]_e}\nearrow & & \searrow{\scriptstyle \mathbb{W}\ \mathbb{C}} \\
\lambda^{+,\times} & \xrightarrow{\quad\mathbb{W}\quad\mathbb{C}\quad} & \lambda^{+,\times}
\end{array}
\tag{13}
$$

Differential Linear Logic. Figure 10 recalls the exponential rules of Differential Linear Logic (DiLL)[20]. DiLL allows the *differentiation* of Linear Logic proofs. This is done by adding rules to the exponential rules of LL (weakening w, contraction c, dereliction d and promotion p). DiLL features:

- a co-weakening rule $\overline{w}$, accounting for the introduction of constant functions,
- a co-contraction rule $\overline{c}$, accounting for the possibility to sum in the function domains,
- a co-dereliction $\overline{d}$, 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.

We now present the Dialectica translation from LL to DiLL in Figure 11. 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 translation is straightforward.

Through the usual encoding $A \multimap B := A^\perp \parr B$, one has

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

[1] While this refinement was introduced by de Paiva, we incorporate to the translation one of the tweaks made by Pédrot [34], namely the fact that $\mathbb{W}(0) := 1$, justified by the irrelevance of dummy terms.

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

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

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

Figure 10: Exponential rules of DiLL

As such, the functional translation from LL to DiLL only encodes the differential part of Dialectica.

Proposition 5.1. *For any formula A, one has $A \vdash \mathbb{W}(A)$ and $\mathbb{C}(A)^\perp \vdash A$. The proof themselves are functorial, only in the case $\mathbb{C}(!A)^\perp \vdash !A$, which uses the codereliction and co-contraction rule of* DiLL.

Proof. We use the fact that, when it is defined, $\mathbb{W}(A^\perp) = \mathbb{C}(A)$ and show that the statement holds for any connective of LL including $\parr$ and ?. The proof is then a straightforward induction on the formula A for any context Γ. The only interesting case is the one for the witness to the exponential ?, namely that when $\Gamma \vdash ?A$ then $\Gamma \vdash \mathbb{W}(?A) = \mathbb{C}(!A^\perp) = ?\mathbb{W}(A) \parr \mathbb{W}(A)$. The fact that when $\Gamma \vdash ?A$ in LL then $\Gamma \vdash ?A \parr A$ in DiLL constitute the very heart of Differential Linear Logic, and uses the newly introduced codereliction and cocontraction rules. As LL is a subsystem of DiLL, from a proof π of $\Gamma \vdash ?A$ one easily constructs a proof of $\Gamma \vdash ?A \parr A$ from a dereliction on $A^\perp$ (corresponding to the reverse argument) and a co-contraction on $!A^\perp$ with an axiom introducing the non-linear argument.

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

□

Factorisation of Dialectica. While differential, the translation presented in Figure 11 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.

Hence, to recover a factorization of Dialectica through LL and DiLL we must distinguish forward and reverse differentiation in order to force the backward translation. We now make the Dialectica translation act on *formulas of intuitionistic* LL:

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

Figure 12 presents an intuitionnistic variant the Dialectica translation from LL to DiLL, varying from figure 12 only on multiplicative connectives.

To recover the translations from and to types of $\lambda^{+,\times}$-calculus through the types of λ-calculus, we refine the economical translation. We interpret the arrow by both a call-by-name arrow, which will be differentiated, and a linear arrow, which will be translated to itslef after going through DiLL.

Definition 5.2. The following defines a translation from types of $\lambda^{+,\times}$ to LL:

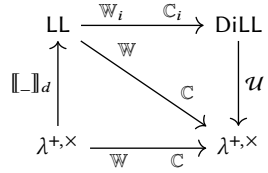
$$[\![A \times B]\!]_d := A \,\&\, B \qquad [\![A + B]\!]_d := A \oplus B$$
$$[\![A \Rightarrow B]\!]_d := (!A \multimap B) \,\&\, (A \multimap B) \qquad [\![0]\!]_d := 0 \qquad [\![1]\!]_d := 1$$

Definition 5.3. The translation from intuitionistic DiLL to types of $\lambda^{+,\times}$-calculus is defined as follows:

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

We then obtain the expected commutative diagram. 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. The counter witness for implication is only recovered up to logical equivalence, as $\mathcal{U}(\mathbb{C}_i([\![A \Rightarrow B]\!])) = (\mathcal{U}(\mathbb{W}_i(A)) \times \mathcal{U}(\mathbb{C}_i(B))) \oplus (\mathcal{U}(\mathbb{W}_i(A)) \times \mathcal{U}(\mathbb{C}_i(B))) \equiv (\mathcal{U}(\mathbb{W}_i(A)) \times \mathcal{U}(\mathbb{C}_i(B)))$.

Proposition 5.4. *The Dialectica transformation on types factorizes through* LL *and* DiLL *as follows:*



The advantage of this translation is that it isolate the duplication of function to the $[\![\_]\!]_d$ translation, while the purely differential part is embedded into a translation from LL to DiLL. Note that the translation of $\mathbb{C}_i(A \otimes B)$ as $(\mathbb{W}_i(A) \multimap \mathbb{C}_i(B)) \,\&\, (\mathbb{W}_i(B) \multimap \mathbb{C}_i(A)$ is necessary only for the decomposition of witness and counter types from LL to $\lambda^{+,\times}$ through DiLL. If one does not try to decompose this arrow and insert it as a diagonal in the diagram above, then one could have interpreted $\mathbb{C}_i(A \otimes B)$ as $(\mathbb{W}_i(A) \multimap \mathbb{C}_i(B))$.

5.2 Dialectica and Differential Categories

This section finally tackles the semantical side of the correspondance between Dialectica and Differentation. We show that Dialectica constructions over the co-Kleisli of differential categories faithfully embed functorial reverse differentials.

The Dialectica transformation was studied from a categorical point of view by De Paiva and Hyland [15]. They have been used as a way to generate *new models* of LL [15, 25]. Our point of view is quite orthogonal. We prove that they also characterize *specific models* of LL: if $\mathcal{C}$ is a model of LL, then the Dialectica Category constructed on $\mathcal{C}$ inherits its monoidal and exponential structure, to make it a new model of LL.

Definition 5.5 ([15]). Consider $\mathcal{C}$ a category with finite limits. The Dialectica category $\mathscr{D}(\mathcal{C})$ over $\mathcal{C}$ has as objects relations $\alpha \subseteq$

$$\begin{array}{rclrclrclrcl}
\mathbb{W}(1) & := & 1 & \mathbb{C}(1) & := & \top & \mathbb{W}(A \otimes B) & := & \mathbb{W}(A) \otimes \mathbb{W}(B) & \mathbb{C}(A \otimes B) & := & \mathbb{C}(A) \parr \mathbb{C}(B) \\
\mathbb{W}(\top) & := & \top & \mathbb{C}(\top) & := & 0 & \mathbb{W}(A \oplus B) & := & \mathbb{W}(A) \oplus \mathbb{W}(B) & \mathbb{C}(A \oplus B) & := & \mathbb{C}(A) \,\&\, \mathbb{C}(B) \\
\mathbb{W}(A^\perp) & := & \mathbb{C}(A) & \mathbb{C}(A^\perp) & := & \mathbb{W}(A) & \mathbb{W}(!A) & := & !\mathbb{W}(A) & \mathbb{C}(!A) & := & ?\mathbb{C}(A) \parr \mathbb{C}(A)
\end{array}$$

Figure 11: Witness and counter types of formulas of Linear Logic in classical Differential Linear Logic

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

$$\mathbb{W}_i(A \multimap B) := (\mathbb{W}_i(A) \multimap \mathbb{W}_i(B)) \qquad \mathbb{C}_i(A \otimes B) := (\mathbb{W}_i(A) \multimap \mathbb{C}_i(B)) \,\&\, (\mathbb{W}_i(B) \multimap \mathbb{C}_i(A))$$

Figure 12: Witness and counter types for ILL formulas into intuitionnistic DiLL

(A, X) on objects of $\mathcal{C}$, and as arrows pairs $(f, F) : \alpha \subseteq (A, X) \to \beta \subseteq (B, Y)$ of maps

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

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

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

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

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

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 $\mathscr{D}(C)$ generalize the relation between a space A and its tangent space. Arrows (f, F) represents a function and its reverse differential F, according to the typing intuitions developed in Section 3. Composition is exactly the chain rule.

Categorical axiomatizations of differentiation has been widely studied as stemming from models of Differential Linear Logic. The various axiomatizations such as differential, cartesian differential or tangent categories [8**?** , 9], 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 3.

The differential structure is defined over Seely categories [39], which provide a categorical interpretation of LL. These are monoidal closed categories $(\mathcal{L}, \otimes, 1)$ with finite products and a comonad $(!, \mathsf{d}, \nu)$ such that ! is a (strong) monoidal functor

$$(\mathcal{L}, \times) \to (\mathcal{L}, \otimes)$$

such that the so-called "Seely isomorphisms" hold:

$$!A \otimes !B \xrightarrow{\simeq} !(A \times B) \quad \text{and} \quad 1 \xrightarrow{\simeq} !\top$$

where $\top$ denotes a terminal object.

Definition 5.6. A **differential category**[8, 22], called a "differential storage category" in [7], is a Seely category $(\mathcal{L}, \otimes, 1)$ in which:

- products coincide with coproducts as a biproduct which we now denote by $(\diamond, 0)$, and
- the comonad $(!, \mathsf{d}, \nu)$ is equipped with a natural transformation $\overline{\mathsf{d}} : \mathrm{id} \Rightarrow !$ satisfying:
  - Invariance of linear maps under differentiation: $\overline{\mathsf{d}}; \mathsf{d} = \mathrm{id}$, and
  - The chain rule:

$$\mathrm{id}_{!A} \otimes \overline{\mathsf{d}}_A; \overline{\mathsf{c}}_A; \nu_A = \mathsf{c}_A \otimes \overline{\mathsf{d}}_A; \mathrm{id}_{!A} \otimes \overline{\mathsf{c}}_A; \nu_A \otimes \overline{\mathsf{d}}_{!A}; \overline{\mathsf{c}}_{!A}.$$

Let us suppose moreover that $\mathcal{L}$ is a model of *classical* DiLL. That is, $\mathcal{L}$ is a $*$-autonomous category endowed with a full and faithful functor $(\_)^\perp : \mathcal{L}^{op} \to \mathcal{L}$ such that there is a natural isomorphism $\chi : \mathcal{L}(B \otimes A, C^\perp) \simeq \mathcal{L}(A, (B \otimes C)^\perp)$. 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 involution of $(\_)^\perp$ 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 5.7. *In the setting described above, one has a functor from the co-Kleisli $\mathcal{L}_!$ to the Dialectica category over it $\mathscr{D}(\mathcal{L}_!)$:*

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

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 ∂ [22, 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 rule 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 literature [22] for explicit handling of annihilation operators and coproducts in this formula, which is nothing but the categorical restatement of the chain 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 immediate forgetful functor:

$$\Pi_1 : \begin{cases} \mathscr{D}(\mathcal{L}_!) & \to & \mathcal{L}_! \\ \alpha \subset A \times X & \mapsto & A \\ (f, F) & \mapsto & f \end{cases}$$

However, it does not result in an adjunction between $\mathcal{L}_!$ and $\mathscr{D}(\mathcal{L}_!)$ as without any linearity condition on F, it might not be equal to its own reverse derivative. With a linearity condition however unicity of differentiation holds [29], and an adjunction between Π_1 and $\overleftarrow{D}$ should be ensured.

Reverse derivative categories. This setting can surely be relaxed, and there might be broader relations between Dialectica categories and differential categories. In particular, if $\mathcal{C}$ is a reverse derivative category [12], one could construct a functor

$$\begin{cases} C & \to & \mathscr{D}(C) \\ A & \mapsto & A \times A \\ f & \mapsto & (f, R[f]) \end{cases}$$

where $R[f]$ represents the reverse derivative of an arrow f as described reverse derivative categories.

6 CONCLUSION AND PERSPECTIVES

In this paper we related the different interpretations of Gödel's Dialectica with logical differentiation. We first studied Dialectica as a transformation on λ-terms, and showed that it corresponds to a reverse differential λ-calulus. We then explored how Dialectica consists in adding rules of DiLL to connectives of LL, and embedded differential categories into Dialectica Categories. The absence of formal link between Section 4.1 and 5.1 could be worrying to the reader: why is one not the direct translation of the other? The reason however is that differential λ-calculus is not typed by DiLL. While differential λ-calculus has the same models and originate from the same structure [17] as DiLL, they are not related with the Curry-Howard correspondence. We are missing a proof-term language for DiLL: below we suggest a few automatic differentiation features that could come from it. Finally, this paper overlooked an essential feature of Dialectica: its main use today is in so-called Applied Proof Theory, as it allows extracting quantitative statements from existence theorem in mathematics. Deepening the connection between differentiation and refinement of mathematical theorems seems an exciting task to us.

Automatic differentiation and reduction strategies. The Dialectica interpretation explored in this paper is fundamentally call-by-name on the arrow, as recalled in Section 5 or in its categorical semantics. This points out that the call-by-name intepretation of functions and their derivative might implement some kind of reverse derivative. The consequences of this could be interesting in a language typed by Differential Linear Logic. Indeed, in the semantics of Differential Linear Logic, non-linear functions f are are seen as functions $\tilde{f}$ that act on distributions [27, 38]. These comes as traditional arguments, encoded through diracs:

$$\tilde{f}(\delta_a) \to f(a),$$

or they act on differentiated arguments

$$\tilde{f}(D_0(\_)a) \to D_0(f)a.$$

Giving the priority to the evaluation of f (call-by-name) relate to backward differentiation, while giving the priority to $D_0(\_)a$ (call-by-value) relates to forward differentiation. Exploring a L-calculus [14] adapted to Differential Linear Logic and linear context, and refining work by Vaux [41], is work in progress that would allow to express such principles.

Proof mining and differentiation. Proof mining [28] 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 are able to transform existential proofs into quantitative proofs. For instance, from unicity proofs in approximation theory one gets an effective moduli of uniqueness, that is 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 consists in functional analysis in differentiating the "ε" function. For example, if a unicity statement

$$\forall \varepsilon, \exists \eta, |G_u(a,b)| < \eta \Rightarrow |a - b| < \varepsilon$$

is established, extracting a quantitative rate of convergence would consists in differentiating the function $\varepsilon \mapsto \eta$. Exploring the consequences of metatheorems in proof mining over logical differentiation seems like an interesting perspective.

ACKNOWLEDGMENTS

We thanks Thomas Powell for enriching discussions on the subject. We are grateful to the generous reviewers for many suggestions and remarks, leading to a better version of the paper.

REFERENCES

[1] Martín Abadi and Gordon D. Plotkin. 2020. A simple differentiable programming language. *Proc. ACM Program. Lang.* 4, POPL (2020), 38:1–38:28. https://doi.org/10.1145/3371106

[2] Beniamino Accattoli. 2018. Proof Nets and the Linear Substitution Calculus. In *Theoretical Aspects of Computing - ICTAC 2018 - 15th International Colloquium, Stellenbosch, South Africa, October 16-19, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11187)*, Bernd Fischer and Tarmo Uustalu (Eds.). Springer, 37–61. https://doi.org/10.1007/978-3-030-02508-3_3

[3] Shiri Artstein-Avidan, Hermann König, and Vitali Milman. 2010. The chain rule as a functional equation. *Journal of Functional Analysis* 259, 11 (2010), 2999–3024. https://doi.org/10.1016/j.jfa.2010.07.002

[4] Jeremy Avigad and Solomon Feferman. 1998. Gödel's Functional ('Dialectica') Interpretation. In *Handbook of Proof Theory*, Samuel R. Buss (Ed.). Elsevier Science Publishers, Amsterdam, 337–405.

[5] Davide Barbarossa and Giulio Manzonetto. 2020. Taylor subsumes Scott, Berry, Kahn and Plotkin. *Proc. ACM Program. Lang.* 4, POPL (2020), 1:1–1:23. https://doi.org/10.1145/3371069

[6] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. 2017. Automatic Differentiation in Machine Learning: a Survey. *J. Mach. Learn. Res.* 18 (2017), 153:1–153:43. http://jmlr.org/papers/v18/17-468.html

[7] Richard Blute, J. Robin B. Cockett, Jean-Simon Pacaud Lemay, and Robert A. G. Seely. 2020. Differential Categories Revisited. *Appl. Categorical Struct.* 28, 2 (2020), 171–235. https://doi.org/10.1007/S10485-019-09572-Y

[8] Richard Blute, J. Robin B. Cockett, and Robert A. G. Seely. 2006. Differential categories. *Math. Struct. Comput. Sci.* 16, 6 (2006), 1049–1083. https://doi.org/10.1017/S0960129506005676

[9] Richard Blute, J. Robin B. Cockett, and Robert A. G. Seely. 2009. Cartesian differential categories. *Theory Appl. Categ.* (2009).

[10] Aloïs Brunel, Damiano Mazza, and Michele Pagani. 2020. Backpropagation in the Simply Typed Lambda-calculus with Linear Negation. *POPL* (2020). http://arxiv.org/abs/1909.13768

[11] Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. 2010. Categorical Models for Simply Typed Resource Calculi. In *Proceedings of the 26th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2010, Ottawa, Ontario, Canada, May 6-10, 2010 (Electronic Notes in Theoretical Computer Science, Vol. 265)*, Michael W. Mislove and Peter Selinger (Eds.). Elsevier, 213–230. https://doi.org/10.1016/J.ENTCS.2010.08.013

[12] J. Robin B. Cockett, Geoff S. H. Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon D. Plotkin, and Dorette Pronk. 2020. Reverse Derivative Categories. In *28th EACSL Annual Conference on Computer Science Logic, CSL 2020, January 13-16, 2020, Barcelona, Spain (LIPIcs, Vol. 152)*, Maribel Fernández and Anca Muscholl (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 18:1–18:16. https://doi.org/10.4230/LIPICS.CSL.2020.18

[13] Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. 2022. Categorical Foundations of Gradient-Based Learning. Springer-Verlag, Berlin, Heidelberg, 1–28. https://doi.org/10.1007/978-3-030-99336-8_1

[14] Pierre-Louis Curien and Guillaume Munch-Maccagnoni. 2010. The Duality of Computation under Focus. In *Theoretical Computer Science - 6th IFIP TC 1/WG 2.2 International Conference, TCS 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings (IFIP Advances in Information and Communication Technology, Vol. 323)*, Cristian S. Calude and Vladimiro Sassone (Eds.). Springer, 165–181. https://doi.org/10.1007/978-3-642-15240-5_13

[15] Valeria de Paiva. 1989. A Dialectica-like Model of Linear Logic. In *Category Theory and Computer Science, Manchester, UK, September 5-8, 1989, Proceedings (Lecture Notes in Computer Science, Vol. 389)*, David H. Pitt, David E. Rydeheard, Peter Dybjer, Andrew M. Pitts, and Axel Poigné (Eds.). Springer, 341–356. https://doi.org/10.1007/BFB0018360

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

[17] Thomas Ehrhard. 2002. On Köthe Sequence Spaces and Linear Logic. *Math. Struct. Comput. Sci.* 12, 5 (2002), 579–623. https://doi.org/10.1017/S0960129502003729

[18] Thomas Ehrhard. 2023. Coherent differentiation. *Math. Struct. Comput. Sci.* 33, 4-5 (2023), 259–310. https://doi.org/10.1017/S0960129523000129

[19] Thomas Ehrhard and Laurent Regnier. 2003. The differential lambda-calculus. *Theor. Comput. Sci.* 309, 1-3 (2003), 1–41. https://doi.org/10.1016/S0304-3975(03)00392-X

[20] Thomas Ehrhard and Laurent Regnier. 2006. Differential interaction nets. *Theor. Comput. Sci.* 364, 2 (2006), 166–195. https://doi.org/10.1016/J.TCS.2006.08.003

[21] Conal Elliott. 2018. The simple essence of automatic differentiation. In *Proceedings of the ACM on Programming Languages (ICFP)*. http://conal.net/papers/essence-of-ad/

[22] Marcelo P. Fiore. 2007. Differential Structure in Models of Multiplicative Biadditive Intuitionistic Linear Logic. 4583 (2007), 163–177. https://doi.org/10.1007/978-3-540-73228-0_13

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

[24] Andreas Griewank and Andrea Walther. 2008. *Evaluating derivatives - principles and techniques of algorithmic differentiation, Second Edition*. SIAM. https://doi.org/10.1137/1.9780898717761

[25] Jules Hedges. 2014. Dialectica Categories and Games with Bidding. In *20th International Conference on Types for Proofs and Programs, TYPES 2014, May 12-15, 2014, Paris, France (LIPIcs, Vol. 39)*, Hugo Herbelin, Pierre Letouzey, and Matthieu Sozeau (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 89–110.

[26] Martin Hyland and Andrea Schalk. 2003. Glueing and orthogonality for models of linear logic. *Theoretical Computer Science* 294 (2003). https://doi.org/10.1016/S0304-3975(01)00241-9 Category Theory and Computer Science.

[27] Marie Kerjean. 2018. 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*, Anuj Dawar and Erich Grädel (Eds.). ACM, 589–598. https://doi.org/10.1145/3209108.3209192

[28] Ulrich Kohlenbach. 2008. *Applied Proof Theory - Proof Interpretations and their Use in Mathematics*. Springer. https://doi.org/10.1007/978-3-540-77533-1

[29] Jean-Simon Pacaud Lemay. 2022. Uniqueness of Differentiation in Differential Categories. *Category Theory Octoberfest* (2022). https://richardblute.files.wordpress.com/2022/10/lemay-ofest.pdf Slides to a talk.

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

[31] Sean K. Moss and Tamara von Glehn. 2018. Dialectica models of type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 739–748. https://doi.org/10.1145/3209108.3209207

[32] Michele Pagani. 2009. The Cut-Elimination Theorem for Differential Nets with Promotion. In *Typed Lambda Calculi and Applications, 9th International Conference, TLCA 2009, Brasilia, Brazil, July 1-3, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5608)*, Pierre-Louis Curien (Ed.). Springer, 219–233. https://doi.org/10.1007/978-3-642-02273-9_17

[33] Barak A. Pearlmutter and Jeffrey Mark Siskind. 2008. Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Trans. Program. Lang. Syst.* 30, 2 (2008), 7:1–7:36. https://doi.org/10.1145/1330017.1330018

[34] Pierre-Marie Pédrot. 2014. 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*, Thomas A. Henzinger and Dale Miller (Eds.). ACM, 77:1–77:10. https://doi.org/10.1145/2603088.2603094

[35] Pierre-Marie Pédrot. 2015. *A Materialist Dialectica. (Une Dialectica matérialiste)*. Ph. D. Dissertation. Paris Diderot University, France. https://tel.archives-ouvertes.fr/tel-01247085

[36] Thomas Powell. 2016. 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, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 136–145. https://doi.org/10.1145/2933575.2933605

[37] Alexey Radul, Adam Paszke, Roy Frostig, Matthew J. Johnson, and Dougal Maclaurin. 2023. You Only Linearize Once: Tangents Transpose to Gradients. 7, POPL (2023). https://doi.org/10.1145/3571236

[38] Laurent Schwartz. 1954. Sur l'impossibilité de la multiplication des distributions. *C. R. Acad. Sci. Paris* 239 (1954), 847–8.

[39] R. A. G. Seely. 1989. Linear Logic, $*$-Autonomous Categories and Cofree Coalgebras. In *Categories in Computer Science and Logic (Contemporary Mathematics, Vol. 92)*. American Mathematical Society, Boulder, Colorado, 371–382.

[40] Matthijs Vákár. 2021. Reverse AD at Higher Types: Pure, Principled and Denotationally Correct. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*, Nobuko Yoshida (Ed.). Springer, 607–634. https://doi.org/10.1007/978-3-030-72019-3_22

[41] Lionel Vaux. 2007. Convolution lambda-mu-Calculus. In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4583)*, Simona Ronchi Della Rocca (Ed.). Springer, 381–395. https://doi.org/10.1007/978-3-540-73228-0_27

[42] Fei Wang, Daniel Zheng, James M. Decker, Xilun Wu, Grégory M. Essertel, and Tiark Rompf. 2019. Demystifying differentiable programming: shift/reset the penultimate backpropagator. *Proc. ACM Program. Lang.* 3, ICFP (2019), 96:1–96:31. https://doi.org/10.1145/3341700

[43] R. E. Wengert. 1964. A Simple Automatic Derivative Evaluation Program. *Commun. ACM* 7, 8 (aug 1964), 463–464. https://doi.org/10.1145/355586.364791