∂ is for Dialectica

Marie Kerjean, Pierre-Marie Pédrot

∂ 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 backpropagation algorithm has been brought to a modern light by its application to neural networks. In this work, we unveil 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. It opens the way for explorations on the logical power of automatic differentiation algorithms.

1 Introduction

Dialectica was originally introduced by Gödel in a famous paper [G$\ddot{5}$8] 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]. 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 [Péd14]. Meanwhile, in its logical outfit, Dialectica led to numerous applications and was tweaked into an unending array of variations in the proof mining community [Koh08].

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 expression 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]. 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 [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.

Related work As far as we know, this is the first time a parallel 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 from 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].

Content We begin this paper in Section 2 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. We begin Section 3 by recalling the traditional definition of Dialectica, acting as a specific form of skolemization on formulas of intuitionistic arithmetic. As a foretaste, we emphasize that the realizers for the Dialectica interpretation of an implication formula already satisfy a kind of chain rule. We recall the types given to the witness and counter variables to a Dialectica interpretation, and highlight that the counter witness for an implication has the type of a reverse differential. Then in Section 4 we recall the definition of a Dialectica category and show that it factorises through $*$-autonomous differential categories, which are exactly the models of DiLL. In Section 5 we show that the factorization of Dialectica through LL refines to DiLL. The most involved section is Section 6.1, devoted to the computational interpretation of Dialectica. There we recall Pédrot's sound computational interpretation [Péd14] and the rules of the differential λ-calculus, to finally show that Pédrot's reverse translation correspond on first-order terms to a reverse version of the differential λ-calculus linear substitution. We conclude by some perspective on the possible outcomes of this correspondence.

2 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 [BPRS17] 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 $f : E \longrightarrow F$ and $g : F \longrightarrow G$ 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 $g(f(t))$, 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 6.1 does not acknowledge these optimizations 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.

Brunel, Mazza and Pagani [BMP20] refined the functional presentation by Wang and al. [WZD+19] 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 [AP20, WZD+19, Ell18], their differential transformation acts on pairs in the linear substitution calculus [Acc18], so as to make it compositional. 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

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

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 $\mathbb{R}$.

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

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 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 function between (topological) vector spaces: $f : E \longrightarrow F$, one has:

$$\begin{aligned}\overleftarrow{D}(f) : E \times F' &\to F \times E' \\ (a, \ell) &\mapsto (f(a), (v \in\mapsto (v \cdot (\mathrm{D}_a f \cdot x)))\end{aligned}$$

Consider a function $g : F \longrightarrow G$. Then $\overleftarrow{D}(f)$ has the type $F \times G' \longrightarrow G \times F'$. If G and F are not self-dual, there is no way to define the composition of $\overleftarrow{D}(f)$ with $\overleftarrow{D}(g)$. 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 6.

3 A functional transformation expressing differentiation

The goal of this section is to 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.

$$
\begin{aligned}
(t = u)_D &:= t = u \\
\bot_D &:= \bot \\
\top_D &:= \top \\
(A \wedge B)_D[\vec{u}; \vec{v}, \vec{x}; \vec{y}] &:= A_D[\vec{u}, \vec{x}] \wedge B_D[\vec{v}, \vec{y}] \\
(A \vee B)_D[b, \vec{u}; \vec{v}, \vec{x}; \vec{y}] &:= (b = 0 \wedge A_D[\vec{u}, \vec{x}]) \vee \\
&\quad\ (b = 1 \wedge B_D[\vec{v}, \vec{y}]) \\
(A \Rightarrow B)_D[\vec{\phi}; \vec{\psi}, \vec{u}; \vec{v}] &:= A_D(\vec{u}, \vec{\psi}\vec{u}\vec{v}) \Rightarrow B_D(\vec{\phi}\vec{u}, \vec{v}) \\
(\forall z.A)_D[\vec{u}, z; \vec{x}] &:= A_D[\vec{u}z, \vec{x}] \\
(\exists z.A)_D[z; \vec{u}, \vec{x}] &:= A_D[\vec{u}, \vec{x}z]
\end{aligned}
$$

Figure 1: Dialectica interpretation of HA

3.1 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[1]. 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 show 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:

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

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

$$
\begin{array}{l}
(A \Rightarrow B)_D[\phi_1; \psi_1, u_1; v_1], \\
(B \Rightarrow C)_D[\phi_2; \psi_2, u_2; v_2]
\end{array}
\quad \vdash (A \Rightarrow C)_D[\phi_3; \psi_3, u_3; v_3]
$$

This solution amounts to the following equations:

$$
\begin{aligned}
u_3 &= u_1 & \psi_3\, u_3\, v_3 &= \psi_1\, u_1\, v_1 \\
v_3 &= v_2 & \phi_2\, u_2 &= \phi_1\, u_1 \\
u_2 &= \phi_1\, u_1 & v_2 &= \phi_1\, u_1\, v_1
\end{aligned}
$$

[1] 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

which can be simplified to:

$$
\phi_3\, u_3 = \phi_2\, (\phi_1\, u_3) \tag{1}
$$

$$
\psi_3\, u_3\, v_3 = \psi_2\, (\phi_1\, u_3)\, (\psi_1\, u_3\, v_3) \tag{2}
$$

While the first Equation 1 represents the traditional functoriality of composition, the second 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]. Our goal is thus to strengthen our claim and leave no doubt to the reader that Dialectica is indeed differentiation.

3.2 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 $Df : A \longrightarrow A \longrightarrow B$, the second arrow in Df representing a linear map from A to B, that is the differenial 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, 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. Deepening this connection between Dialectica and differentiation neccesitates linear implications, which will be done in Section 5. The interpretation of Dialectica as a reverse differentiable programming language will be performed in Section 6.1.

4 Dialectica Categories are differential 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

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

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. 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, such as differential, cartesian differential or tangent categories [BCS06, BCS09, 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 to make use of the intuitions developed in Section 2.

Consider $\mathcal{L}$ a categorical model of DiLL as formalized by Blute, Cockett, Seely and Fiore [Fio07, BCS06]. We have a monoidal closed category $(\mathcal{L}, \otimes, 1)$ endowed with a biproduct $\Diamond$, a strongly monoidal comonad

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

and a natural transformation

$$\partial : Id \otimes ! \longrightarrow !$$

satisfying the appropriate commutative diagrams [Fio07].

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 $(\_)^\perp : \mathcal{L}^{op} \longrightarrow \mathcal{L}$ such that there is a natural isomorphism $\chi : \mathcal{L}(B \otimes A, C^\perp) \simeq (A, (B \otimes C)^\perp)$. 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 3. In the setting described above, one has a functor from the co-Kleisli $\mathcal{L}_!$ to the Dialectica category over it $\mathscr{D}(\mathcal{L}_!)$:

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

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 literature [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 !. □

Reverse derivative categories This setting can surely be relaxed, and there might be more broad relations between Dialectica categories and differential categories. In particular, if $\mathcal{C}$ is a reverse derivative category [CCG+20], one should without difficulties construct a functor

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

where $R[f]$ represents the reverse derivative of an arrow f as described in the paper. We do not study further this connection here.

5 Dialectica from LL to DiLL

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 types of witness and terms of λ-calculus. They are distinct from works applying Dialectica directly on formulas of LL, which we don't investigate here [FO11]. 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 $(-)^\bot$, & 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

$$\begin{aligned} & \mathbb{W}(!A \multimap B) \\ = \; & \mathbb{W}((!A \otimes B^\bot)^\bot) \\ = \; & \mathbb{C}(!A \otimes B^\bot) \\ = \; & (\mathbb{W}(A) \Rightarrow \mathbb{W}(B)) \times (\mathbb{C}(B) \Rightarrow \mathbb{W}(A) \Rightarrow \mathbb{C}(A)) \end{aligned}$$

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 1. 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.

5.1 Differential Linear Logic

Figure 4 recalls the exponential rules of Differential Linear Logic [ER06], presented here in their intuitionistic version for reasons explained below. DiLL adds to LL rules to differentiate proofs. That is, 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.

The cut-elimination rules follows the real-analysis intuitions on differentiation, and are not detailled here.

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 has equivalently

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

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

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

That is, due to the associativity of $⅋$, or equivalently due to the involutive linear negation, reverse and forward derivative are equivalent. 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 backward translation.

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

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

We present the Dialectica translation from LL to DiLL in Figure 5, and prove a basic soundness result on witness in Proposition 4. 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 $A \multimap B := A^{\perp} ⅋ B$, 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.

When we go back to classical DiLL, we are able to prove that the Dialectica translation of Figure 5 is sound. DiLL is fundamentally forward, and linear negation is needed to express the backward derivative present in the witness type for an implication.

Proposition 4. Let us translate $A \multimap B$ as $A^{\perp} ⅋ B$. Then if $\Gamma \vdash A$ in LL, then $\Gamma \vdash \mathbb{W}(A)$ in classical 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 $⅋$ 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 $\Gamma \vdash \mathbb{W}(?A) = \mathbb{C}(!A^{\perp}) = ?\mathbb{W}(A) \otimes \mathbb{W}(A)$. The fact that when $\Gamma \vdash A$ in LL then $\Gamma \vdash ?A \otimes A$ 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 $\Gamma \vdash !A ⅋ A$ 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.

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

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

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

Figure 6: A modernized Dialectica [Péd15, 8.3.1]

□

5.2 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.

Definition 5. [Péd15, 8.4.2] The following is called the economical translation.

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

$$\begin{array}{ccc} & \mathsf{LL} & \\ {\scriptstyle [\![-]\!]_e}\nearrow & & \searrow{\scriptstyle \mathbb{W}\ \mathbb{C}} \\ \lambda^{+,\times} & \xrightarrow[\mathbb{W}\quad\mathbb{C}]{} & \lambda^{+,\times} \end{array} \tag{3}$$

This translation has some surprising features. While arrows are interpreted in call-by-name, products and sums are respectively interpreted in call-by-value.

We would like now to recover the translations from and to types of $\lambda^{+,\times}$-calculus through its differential refinement (Figure 5).

$$\begin{array}{ccc} \mathsf{ILL} & \xrightarrow{\mathbb{W}\quad\mathbb{C}} & \mathsf{IDiLL} \\ \vdots\uparrow & & \downarrow\vdots \\ \lambda^{+,\times} & \xrightarrow[\mathbb{W}\quad\mathbb{C}]{} & \lambda^{+,\times} \end{array} \tag{4}$$

The tricky part is to refine the economical translation from $\lambda^{+,\times}$-calculus into LL. As the translation from LL to DiLL already encodes the duplication of functions, and the last translation $\mathcal{U}$ will encode the call-by-value interpretation of $\times$ and $+$, we can define a simple call-by-name translation.

Definition 6. We define the differential translation of types of $\lambda^{+,\times}$-calculus into LL below.

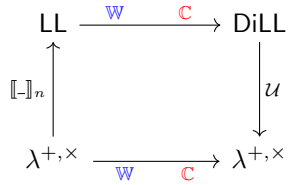
$$\begin{aligned}
[\![A \Rightarrow B]\!]_n &:= (!A \multimap B) \\
[\![A \times B]\!]_n &:= A \otimes B \\
[\![A + B]\!]_n &:= A \oplus B \\
[\![0]\!]_n &:= 0 \\
[\![1]\!]_n &:= 1
\end{aligned}$$

Definition 7. The translation from intuitionistic DiLL to types of $\lambda^{+,\times}$-calculus is defined as follows:

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

We then obtain the following expected commutative diagram.

Proposition 8. The Dialectica transformation on types factorizes through LL and DiLL as follows:



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.

6 The computational Dialectica and backpropagation

The previous sections have shown that both on the logical side (Section 3), on the categorical side (Section 4) on the typing side (Section 5), 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].

6.1 An account of the modern Dialectica transformation

As hinted in the previous section, the Dialectica transformation can be applied to typed λ-terms of instead of HA derivations. 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 to get rid of the decidablity condition and makes Dialectica preserve β-equivalence, leading to a kind of 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*.

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

$$t, u ::= x \mid \lambda x.\, t \mid t\, u \mid (t, u) \mid t.1 \mid t.2$$

$$\begin{array}{ccc} (\lambda x.\, t)\, u & \to_\beta & t\{x \leftarrow u\} \\ (t_1, t_2).i & \to_\beta & t_i \\ t & \equiv_\eta & (t_1, t_2) \end{array}$$

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

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

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

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

Definition 9. An abstract multiset 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

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

Monoidal laws

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

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

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

Figure 8: The computational Dialectica

Distributivity laws

$$
\begin{array}{c}
\varnothing \gg\!= f \equiv \varnothing \qquad\qquad t \gg\!= \lambda x.\,\varnothing \equiv \varnothing \\
(t \circledast u) \gg\!= f \equiv (t \gg\!= f) \circledast (u \gg\!= f) \\
t \gg\!= \lambda x.\,(f\ x \circledast g\ x) \equiv (t \gg\!= f) \circledast (t \gg\!= g)
\end{array}
$$

We now turn to the Dialectica interpretation itself, which is defined at Figure 8, 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 10 (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 $t_x \equiv u_x$.

From [Péd14], it follows that the $(-)_x$ translation allows to observe 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.

6.2 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 [PS08]. As such, it allows to extract derivatives in this very general setting.

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

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

Figure 9: Dialectica Derivative Extension

Lemma 11 (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

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

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 12. 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 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 9.

The soundness theorem is then adapted trivially.

Theorem 13. 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. Direct consequence of Lemma 11 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 satisfy this equation. Thus, Dialectica systematically computes derivatives in a higher-order language.

6.3 Higher dimensions

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 $\mathbb{R}^n \to \mathbb{R}^m$.

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

Figure 10: Extensions of Dialectica (types only)

Let us write $A \times B$ for the negative product in the source language. It is interpreted directly 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 $\mathbb{R}^n := \mathbb{R}\times\ldots\times\mathbb{R}$ 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 13 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 14. 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 10.

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 to handle 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.

6.4 A reverse differential λ-calculus

In this section, we relate the two transformation acting on λ-terms in Dialectica with those at stake in differential λ-calculus [ER03].

Denotationaly speaking, this means that forward and reverse differentiation corresponds on first-order functions $f\in\mathcal{C}^\infty(\mathbb{R}^n,\mathbb{R})$, and that the situation on higher-order functions is more intricate.

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 operator. Thus differential λ-calculus 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.

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

$$\frac{\partial y}{\partial x} \cdot T = \begin{cases} T & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\frac{\partial(\lambda y.s)}{\partial x} \cdot T = \lambda y. \left(\frac{\partial s}{\partial x} \cdot T\right) \tag{6}$$

$$\frac{\partial\, 0}{\partial x} \cdot T = 0 \tag{7}$$

$$\frac{\partial(s\ U)}{\partial x} \cdot T = \left(\frac{\partial s}{\partial x} \cdot T\right) U + \left(Ds \cdot \left(\frac{\partial U}{\partial x} \cdot T\right)\right) U \tag{8}$$

$$\frac{\partial(Ds \cdot u)}{\partial x} \cdot T = D\left(\frac{\partial s}{\partial x} \cdot T\right) \cdot u + Ds \cdot \left(\frac{\partial u}{\partial x} \cdot T\right) \tag{9}$$

$$\frac{\partial(S+U)}{\partial x} \cdot T = \frac{\partial S}{\partial x} \cdot T + \frac{\partial U}{\partial x} \cdot T \tag{10}$$

Figure 11: Linear substitution

We write $\lambda x. \sum_i s_i$ for $\sum_i \lambda x.s_i$, $(\sum_i s_i)T$ for $\sum_i s_i T$, 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.

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 11. 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 8, which we present here in a simpler form.

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

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.

Comparing Types Types of terms that are used in linear substitution are more simple than in Dialectica.

Lemma 15. [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 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.

6.5 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.

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

Figure 12: Logical relations for the arrow type

Definition 16. 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 12.

In the remainder of this section, we assume that the atomic logical relations satisfy the closure conditions of Figure 13. 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 17. The closure properties of Figure 13 generalize to any simple 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 13: Atomic closure conditions

Theorem 18. 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 17. 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 terms which are to be summed and other ones strongly relates with Ehrhard's recent work on deterministic probabilistic coherent spaces [Ehr21].

7 Perspectives

In this paper we related the different interpretations of Gödel's Dialectica with logical differentiation. We draw two possible outcomes from this.

7.1 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 [Sch54] [Ker18]. These comes as traditional arguments, encoded through diracs:

$$\tilde{f}(\delta_a) \longrightarrow\!\!\!\!\!\rightarrow f(a),$$

or they act on differentiated arguments

$$\tilde{f}(D_0(\_)a) \longrightarrow\!\!\!\!\!\rightarrow D_0(f)a.$$

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. An exploration to the L-calculus [CMM10] adapted to Differential Linear Logic and linear context could allow to express such principles.

7.2 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 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.

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.

[Acc18] Beniamino Accattoli. 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*, pages 37–61, 2018.

[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] Martin Abadi and Gordon D. Plotkin. A simple differentiable programming language, 2020.

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

[BCS09] R. F. Blute, J. R. B. Cockett, and R. A. G. Seely. Cartesian differential categories. *Theory Appl. Categ.*, 2009.

[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.

[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.

[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.

[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.

[G$\ddot{\text{o}}$58] 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.

[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.

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

[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.

[Sch54] Laurent Schwartz. Sur l'impossibilité de la multiplication des distributions. *C. R. Acad. Sci. Paris*, 239:847–8, 1954.

[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.