A Functional Functional Interpretation

Pierre-Marie Pédrot

Laboratoire PPS, CNRS, UMR 7126, Univ Paris Diderot,
Sorbonne Paris Cité, PiR2, INRIA Paris Rocquencourt, F-75205
Paris, France
pierre-marie.pedrot@inria.fr

Abstract

In this paper, we present a modern reformulation of the Dialectica interpretation based on the linearized version of de Paiva. Contrarily to Gödel's original translation which translated $\mathbf{HA}$ into system $\mathbf{T}$, our presentation applies on untyped λ-terms and features nicer proof-theoretical properties.

In the Curry-Howard perspective, we show that the computational behaviour of this translation can be accurately described by the explicit stack manipulation of the Krivine abstract machine, thus giving it a direct-style operational explanation.

Finally, we give direct evidence that supports the fact our presentation is quite relevant, by showing that we can apply it to the dependently-typed calculus of constructions with universes $\mathbf{CC}_\omega$ almost without any adaptation. This answers the question of the validity of Dialectica-like constructions in a dependent setting.

Categories and Subject Descriptors Theory of computation [*Logic*]: Type theory

General Terms Theory

Keywords Dialectica translation, Linear logic, Dependent types, Abstract machine, Curry-Howard

Introduction

The Dialectica transformation, also known as the functional interpretation, was introduced by Gödel in the eponymous journal in 1958 [7], although he had been designing it since the 30's [1]. It turns out it was a tentative workaround to the incompleteness theorem, then perceived as great catastrophe. As classical logic could not be considered a trustful tool to justify itself anymore, one had to solve the problem of foundations by using *constructive* means.

Similarly to its predecessor, the double-negation translation, Dialectica aimed at providing classical logic with a reliable basis rooted in computation, through a transformation of $\mathbf{HA}$ into system $\mathbf{T}$ [1]. Unlike the double-negation translation, Dialectica is quite finer-grained. Indeed, Dialectica realizes two non-intuitionistic principles, namely Markov's principle (MP) and independence of premises (IP), while retaining the disjunction and existence properties. These two semi-classical principles are usually stated as follows:

$$\text{MP}\ \frac{\neg(\forall n^{\mathbb{N}}.\, \neg P\, n)}{\exists n^{\mathbb{N}}.\, P\, n} \qquad \frac{(\forall m^{\mathbb{N}}.\, P\, m) \to \exists n^{\mathbb{N}}.\, R\, n}{\exists n^{\mathbb{N}}.\, (\forall m^{\mathbb{N}}.\, P\, m) \to R\, n}\ \text{IP}$$

where P is a decidable proposition on natural numbers. Markov's principle can be practically implemented by a simple unbounded loop, checking if P is true at each n, starting from zero. By a classical reasoning, this algorithm must terminate [8]. Independence of premises is more challenging to justify on constructivist grounds, and does conflict with the usual Curry-Howard interpretation [1].

In the wake of modern proof theory, it is not until the end of the 80's that De Paiva provided a radical categorical cleanup of the underlying processes at work in the Dialectica transformation [3]. Even better, it resulted that Dialectica factored nicely through linear logic decompositions of intuitionistic calculi. De Paiva's works are actually the essential firm ground upon which this article is built.

We recall that Dialectica associates to any formula A from $\mathbf{HA}$ a first-order formula A_D of the following shape:

$$A_D := \exists \vec{u}.\, \forall \vec{x}.\, A^D[\vec{u}, \vec{x}]$$

According to de Paiva's presentation, each part of the Dialectica translation result can be understood on its own:

- The $\exists \vec{u}$ part can be given a unique "witness" type
- The $\forall \vec{x}$ part can be given a unique "counter" type
- The formula A^D acts as a dynamic test and can be seen as an orthogonality relation stating who the winner is, in a game semantics player-versus-opponent fashion.

As shown by Hyland and Schalk [9, 10], it happens coincidently that many models of linear logic are based upon this kind of three-part decomposition, and *de facto*, the derived *double-glueing* construction is often used as a decomposition tool for such models.

Curiously enough, the Dialectica translation by itself did not profit from this nice categorical presentation, probably because of a community mismatch. More suprisingly, the computational behaviour of this translation was never explained in a Curry-Howard manner. The main goal of this article is to remedy this unfair situation, by proposing a modern syntactic presentation of the Dialectica transform, based on de Paiva's decomposition, as well as a computational account for it, inspired by classical realizability.

The first part of the article consists in defining and manipulating quite a few subtly distinct translations. In order not to confuse the reader, let us present a quick overview of those translations:

- The historical Dialectica translation will only be alluded to. This is Gödel's original one [7], which is described in [1] for instance.

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

http://dx.doi.org/10.1145/2603088.2603094

- The linear Dialectica is essentially the translation from **LL** into **LJ** described by de Paiva [3]. It is briefly recalled at Section 1. Its direct application, the linearized Dialectica, is a syntactic translation acting on the λ-calculus, described at Section 2.
- Finally, the revised Dialectica is a variant of the previous one solving semantical well-behavedness issues. Section 3 is dedicated to its presentation.

Following the general guidance principle of [13], we present the linearized and revised translations through λ-terms instead of sequents, in a purely proof-as-program paradigm. The similarity of exposition culminates in Section 4 where we provide a realizability explanation for the Dialectica translation by means of the Krivine abstract machine.

Finally, we show that the revised Dialectica can be applied to more complicated settings without pain. We show at Section 5 how we can straightforwardly adapt our syntactic presentation to the dependently-typed $\mathbf{CC}_\omega$ system.

1. Rephrasing Linear Dialectica

The historical Dialectica transformation acted on intuitionistic arithmetic, and resulted essentially in another intuitionistic logic, save for a few additional semi-classical axioms. As shown by de Paiva [3], the Dialectica transformation can also be seen as a genuine way to design new models of linear logic from old ones. Indeed, the historical Dialectica happens to factor through the usual call-by-name translation from **LJ** to **LL**. This decomposition will be discussed afterwards. For now, we are going to focus on linear logic alone.

Both Dialectica and its linearized counterpart are type-directed, in the sense that they operate on sequents rather than on untyped terms. We recall in this section the translation on linear types, which are defined by the following inductive grammar.

$$A, B ::= \alpha \mid A \otimes B \mid !A \mid A^\perp$$

This presentation reduces the required number of connectives, as they shall be defined by duality. In particular, we will conveniently write the linear arrow $A \multimap B := (A \otimes B^\perp)^\perp$. As we will be interested in translations of the λ-calculus, we choose to forget about the additive connectives, even though they could be properly described in this setting.

Each linear type A is mapped to two distinct intuitionistic types: the type $\mathbb{W}[A]$ of witnesses of A, and the type $\mathbb{C}[A]$ of counters of A. While witnesses can be understood as a straightforward intuitionistic interpretation of types, counters are their duals, thought as opponents in game semantics, or stacks in abstract machines. We will tend to write witnesses with roman letters $t, u, \ldots$ and counters with Greek letters $\pi, \varphi, \ldots$ even though their rôle may be swapped.

We define inductively at Figure 1 the witness and counter interpretations. The interpretation of atoms is not fixed, but rather parameterized by two valuations $\mathbb{W}_\alpha$ and $\mathbb{C}_\alpha$, and we assume the following to hold.

Assumption 1. For all α, the types $\mathbb{W}_\alpha$ and $\mathbb{C}_\alpha$ are inhabited.

As the target types contain products, we need a λ-calculus equipped with products on the level of terms. We formalize it at Figure 2.

Defining the interpretation of proofs requires some additional refinements. The linear Dialectica inherited from the historical one the necessary use of *paraproofs*, that is, terms inhabiting a given type $\mathbb{W}[A]$, except they may not be a valid proof. To discriminate between valid and invalid paraproofs, one has to use an orthogonality relation, relating witnesses and counters at a given type. We will write $t \perp_A \pi$ whenever $t : \mathbb{W}[A]$ and $\pi : \mathbb{C}[A]$ are orthogonal

$$\begin{array}{rcl}
\mathbb{W}[\alpha] & := & \mathbb{W}_\alpha \\
\mathbb{C}[\alpha] & := & \mathbb{C}_\alpha \\
\mathbb{W}[A \otimes B] & := & \mathbb{W}[A] \times \mathbb{W}[B] \\
\mathbb{C}[A \otimes B] & := & (\mathbb{W}[A] \Rightarrow \mathbb{C}[B]) \times (\mathbb{W}[B] \Rightarrow \mathbb{C}[A]) \\
\mathbb{W}[!A] & := & \mathbb{W}[A] \\
\mathbb{C}[!A] & := & \mathbb{W}[A] \Rightarrow \mathbb{C}[A] \\
\mathbb{W}[A^\perp] & := & \mathbb{C}[A] \\
\mathbb{C}[A^\perp] & := & \mathbb{W}[A]
\end{array}$$

Figure 1. Translation on types

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

$$t, u ::= x \mid \lambda x.\, t \mid t\, u \mid (t, u) \mid \mathtt{match}\ t\ \mathtt{with}\ (x, y) \mapsto u$$

$$\begin{array}{rcl}
(\lambda x.\, t)\, u & \to_\beta & t\{x \leftarrow u\} \\
\mathtt{match}\ (t, u)\ \mathtt{with}\ (x, y) \mapsto r & \to_\beta & r\{x, y \leftarrow t, u\}
\end{array}$$

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

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

$$\frac{\Gamma \vdash t : A \times B \qquad \Gamma, x : A, y : B \vdash r : C}{\Gamma \vdash \mathtt{match}\ t\ \mathtt{with}\ (x, y) \mapsto r : C}$$

Figure 2. Target $\lambda^\times$-calculus

at type A. Being orthogonal can roughly be understood in terms of game semantics: $t \perp_A \pi$ means that the player t wins against the opponent π in the arena A.

The precise use of the orthogonality will be made explicit at Section 2. For now, let us state it: given a family of atomic relations $\perp_\alpha$ as parameters, it is the relation generated by the rules of Figure 3 and closed by β-equivalence. We require additionally the following condition on the atomic relations.

Assumption 2. The orthogonality on atomic types must be decidable, i.e. for all α, there is a λ-term $\mathtt{decide}_\alpha$ such that:

$$\mathtt{decide}_\alpha\, t\, \pi\, u_1\, u_2 \to^* \begin{cases} u_1 & \text{if } t \perp_\alpha \pi \\ u_2 & \text{otherwise} \end{cases}$$

$$\frac{\pi \not\perp_A t}{t \perp_{A^\perp} \pi} \qquad \frac{t \perp_A \psi\, u \qquad u \perp_B \varphi\, t}{(t, u) \perp_{A \otimes B} (\varphi, \psi)} \qquad \frac{t \perp_A \varphi\, t}{t \perp_{!A} \varphi}$$

Figure 3. Orthogonality relation

Definition 1 (Proofs)**.** A proof of A is a term $\vdash t : \mathbb{W}[A]$ s.t. for all $\vdash \pi : \mathbb{C}[A]$, $t \perp_A \pi$. We may alternatively say that t is valid.

Proposition 1 (Soundness)**.** *If A is provable in linear logic, then there is a proof of A.*

Proposition 2 (Non-degeneracy)**.** *There exist a formula A such that there is no proof of A for all possible instances of $\mathbb{W}_\alpha$, $\mathbb{C}_\alpha$ and $\perp_\alpha$.*

We are not really interested in linear logic *per se*, but rather in its application to λ-calculus, hence we immediatly carry on. We will give the particular content of the translation in the case of the λ-calculus in Section 2.2.

2. Translating call-by-name λ-calculus

In this part, we will be translating λ-calculus through two distinct transformations, according to the following scheme:

$$\lambda \xrightarrow{[\![\cdot]\!]} \mathbf{LL} \xrightarrow{D} \lambda^{\times}$$

where D is the linear Dialectica transform exposed previously. The advantage of applying such a decomposition lies in the more elegant presentation of the whole Dialectica process we recover at the end.

Our source λ-calculus is endowed with the following simple types:

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

with the usual typing rules. It may have been more uniform, and indeed possible, to start from $\lambda^{\times}$ instead of the usual λ-calculus, but it would have been at the expense of clarity and technical simplicity.

Note that in this section, we still need to work with typed terms as in the historical Dialectica, but we will adapt this presentation to the untyped setting later on.

2.1 Call-by-name linear translation on types

There are several ways to translate λ-calculus into linear logic, depending on the calling convention one wants to give to the translation. The two most known translations are the historical decomposition, which is call-by-name, and the "boring" translation, which is call-by-value, both described by Girard in [5].

We will focus on call-by-name, because this calling convention is slightly simpler to present through linear logic, and, moreover, the historical Dialectica is actually call-by-name itself, at least for the arrow. This way, we will be able to understand the computational content of a cleaned-up version of Dialectica. Yet, this does not forbid us to design a call-by-value Dialectica, or even a classical variant. One of these classical decompositions happens to have been treated in [17], albeit not explicitly through the prism of linear logic.

We recall now the call-by-name translation from **LJ** to **LL**. The call-by-name linear translation amounts to the decomposition of the intuitionistic arrow into a linear arrow together with the exponential modality, according to the famous equation:

$$[\![A \Rightarrow B]\!] := ![\![A]\!] \multimap [\![B]\!]$$

The remaining of the translation is straightforward. We only have to define the interpretation of a sequent, which is akin to the arrow case, where Γ is made of the formulae $\Gamma_1, \dots, \Gamma_n$:

$$[\![\Gamma \vdash A]\!] := ![\![\Gamma_1]\!] \otimes \dots \otimes ![\![\Gamma_n]\!] \multimap [\![A]\!]$$

For the sake of readability, we will keep implicit the $[\![\cdot]\!]$ interpretation and freely write intuitionistic types instead of their linear translation when the context is clear.

2.2 Translating sequents

We now make explicit the composition of $[\![\cdot]\!]$ and D on the untyped λ-calculus. First, we will tweak the straightforward translation of sequents to obtain a nice-looking translation on open terms, by carefully looking at the global result.

Proposition 3. *We have the following isomorphism:*

$$\mathbb{W}[\Gamma_1, \dots, \Gamma_n \vdash A] \cong \begin{cases} \overrightarrow{\mathbb{W}[\Gamma]} \Rightarrow \mathbb{W}[A] \\ \quad\times \\ \overrightarrow{\mathbb{W}[\Gamma]} \Rightarrow \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_1] \\ \quad\times \\ \quad\vdots \\ \quad\times \\ \overrightarrow{\mathbb{W}[\Gamma]} \Rightarrow \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_n] \end{cases}$$

We will therefore see all our sequents through this isomorphism. As one can observe, the call-by-name translation of an open term $\Gamma \vdash t : A$ produces two sorts of objects:

- On the one hand, the $\overrightarrow{\mathbb{W}[\Gamma]} \Rightarrow \mathbb{W}[A]$ component. We will call this part of the product the *intuitionistic component* of the translation of t, written $t^{\bullet}$.
- On the other hand, for each free variable $x_i : \Gamma_i$ of t, a term $t_{x_i} : \overrightarrow{\mathbb{W}[\Gamma]} \Rightarrow \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_i]$. We will call this one the x_i-*collector* of t. The choice for this name will be made clearer later on.

For the sake of simplicity, because all of these translated terms share a common $\overrightarrow{\mathbb{W}[\Gamma]}$ prefix corresponding to the free variables of the original term, we will simply consider that the translation preserves free variables, up to type lifting.

Definition 2 (Sequent translation)**.** The Dialectica translation maps a sequent, that is, an open term

$$x_1 : \Gamma_1, \dots, x_n : \Gamma_n \vdash t : A$$

to an intuitionistic proof:

$$x_1 : \mathbb{W}[\Gamma_1], \dots, x_n : \mathbb{W}[\Gamma_n] \vdash t^{\bullet} : \mathbb{W}[A]$$

together with, for each free variable x_i, its x_i-collector:

$$x_1 : \mathbb{W}[\Gamma_1], \dots, x_n : \mathbb{W}[\Gamma_n] \vdash t_{x_i} : \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_i]$$

The orthogonality relation is lifted to sequents seamlessly and results in the following definition.

Definition 3 (Sequent validity)**.** A sequent $x_1 : \Gamma_1, \dots, x_n : \Gamma_n \vdash t : A$ is valid whenever for all $\vec{u} : \mathbb{W}[\vec{\Gamma}]$ and $\pi : \mathbb{C}[A]$, if for all i, $u_i \perp_{\Gamma_i} (t_{x_i}\{\vec{x} \leftarrow \vec{u}\})\,\pi$ then $t^{\bullet}\{\vec{x} \leftarrow \vec{u}\} \perp_A \pi$.

2.3 The actual translation

In order to present the term translation, we need to introduce some more material. We recall that we required the atomic types $\mathbb{W}_{\alpha}$ and $\mathbb{C}_{\alpha}$ to be non-empty. Then, for every type A we can construct a *dummy term* $\varnothing_A : \mathbb{W}[A]$.

Definition 4 (Dummy term)**.** Dummy terms are defined by induction on A:

- $\varnothing_{\alpha}$ (resp. $\varnothing_{\alpha^{\perp}}$) is a term inhabiting $\mathbb{W}_{\alpha}$ (resp. $\mathbb{C}_{\alpha}$);
- $\varnothing_{!A} := \varnothing_A$ and $\varnothing_{(!A)^{\perp}} := \lambda x.\, \varnothing_{A^{\perp}}$
- $\varnothing_{A \otimes B} := (\varnothing_A, \varnothing_B)$ and $\varnothing_{(A \otimes B)^{\perp}} := (\lambda x.\, \varnothing_{A^{\perp}}, \lambda y.\, \varnothing_{B^{\perp}})$

As we shall see, the dummy term serves an encoding purpose, and for the translation of the λ-calculus, we will only need it at the level of stacks, not at the level of terms.

Definition 5 (Merge)**.** Similarly to the dummy term, we can use the decidability of the orthogonality on atomic types to lift it to any type. In particular, we can easily write a λ-term

$$\mathtt{merge}_A : \mathbb{C}[A] \Rightarrow \mathbb{C}[A] \Rightarrow \mathbb{W}[A] \Rightarrow \mathbb{C}[A]$$

with the following property:

$$\mathtt{merge}_A\, \pi_1\, \pi_2\, t \to_{\beta}^{*} \begin{cases} \pi_2 & \text{if } t \perp_A \pi_1 \\ \pi_1 & \text{otherwise} \end{cases}$$

We also use some syntactic sugar to ease the reading. In particular, we will write

$$\lambda(x, y).\, t := \lambda p.\, \mathtt{match}\ p\ \mathtt{with}\ (x, y) \mapsto t$$

for some fresh variable p, and $\mathtt{p}_1$, $\mathtt{p}_2$ for the projections.

Definition 6 (Linearized Dialectica)**.** The linearized Dialectica transformation is defined at Figure 4.

Variable case: $\Gamma, x : A \vdash x : A.$

$x^\bullet := x$

$x_x := \lambda\pi.\, \pi$

$x_y := \lambda\pi.\, \varnothing_{\Gamma_i^\perp}$ when $x \neq y$ and $(y : \Gamma_i) \in \Gamma$

Abstraction case: $\Gamma \vdash \lambda x.\, t : A \Rightarrow B.$

$(\lambda x.\, t)^\bullet := (\lambda x.\, t^\bullet, \lambda\pi x.\, t_x\, \pi)$

$(\lambda x.\, t)_y := \lambda(x, \pi).\, t_y\, \pi$

Application case: $\Gamma \vdash t\, u : B.$

$(t\, u)^\bullet := (\mathtt{p}_1\, t^\bullet)\, u^\bullet$

$(t\, u)_y := \lambda\pi.\, \mathtt{merge}_{\Gamma_i}\, (t_y\, (u^\bullet, \pi))\, (u_y\, (\mathtt{p}_2\, t^\bullet\, \pi\, u^\bullet))\, y$

Figure 4. The linearized Dialectica transformation

Proposition 4 (Soundness)**.** *The Dialectica translation preserves sequent typing, i.e. for all* $\Gamma \vdash t : A$ *and* $(x : \Gamma_i) \in \Gamma$*,*

$$\mathbb{W}[\Gamma] \vdash t^\bullet : \mathbb{W}[A]$$

$$\mathbb{W}[\Gamma] \vdash t_x : \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_i].$$

Proposition 5 (Validity)**.** *If* $\Gamma \vdash t : A$*, then it is a valid sequent.*

3. A cleaner Dialectica translation

3.1 Stating the problem

The translation presented in the previous sections suffers from several defects. First, it still only applies in a typed setting, because we need the hypothesis type to build the variable and application collectors. Indeed, `merge` and $\varnothing$ are only defined with respect to a given type.

Second, it does not comply with one of the most fundamental properties expected for a λ-calculus translation, that is, compatibility with reduction.

Fact 1. In general, when $t \to^* u$, we do not have $t^\bullet \equiv_\beta u^\bullet$.

From where does the problem stem? If one tried to prove that such a property did hold, she would get stuck because of the lack of algebraic equalities during reduction of the `merge` operator. For example, we do not have the following desirable algebraic equality in general:

$$\mathtt{merge}_A\, \varnothing_{A^\perp}\, \pi\, t \equiv_\beta \pi$$

Indeed, because it uses an ad hoc definition parameterized with the $\mathtt{decide}_\alpha$ user-provided functions, `merge` may have a highly non-uniform behaviour in its return value, resulting in an unpredictable choice between $\varnothing_{A^\perp}$ and π.

Let us step back and inspect what are our requirements in the construction of the translation as well as in its soundness proof.

Fact 2. The following properties are essential in the Dialectica transformation:

1. The existence of the $\varnothing$ family of terms, as well as of the `merge` operator;
2. the orthogonality preservation of the `merge`:

$$t \perp_A \mathtt{merge}_A\, \pi_1\, \pi_2\, t \quad \leftrightarrow \quad t \perp_A \pi_1 \wedge t \perp_A \pi_2.$$

This clearly looks like the encoding of a cleaner object. We can abstract away these properties using the following data structure.

Definition 7 (Abstract multisets)**.** An abstract multiset data structure is the conjunction of the following data:

- A parameterized type $\mathfrak{M}\,(-)$
- The following constants[1]: $\varnothing$, $\circledast$, $\{\cdot\}$ and $\gg\!=$ admitting the following typing rules:

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

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

For now, we do not require any β-equivalence on these terms. The reader can foresee that we will want to obtain a monad structure obeying some monoidal rules, though. The natural candidate will obviously be the finite multiset monad.

The $\circledast$ operator will be used as a rewriting-friendly `merge`, and $\varnothing$ will be used as a uniform dummy term.

Note also that the typing here is essentially for readability purposes. Indeed, we will be working with untyped λ-calculus soon, where such a datastructure could be written uniformly without caring about the types at all.

3.2 Towards a déjà vu

Equipped with our axiomatic structure, let us head to an adaptation of the linear Dialectica transformation to try to fix our problems. As we remarked, we used the dummy terms to play the role of a generic placeholder. Because we are about to get rid of dummy terms, we can also remove the non-emptyness condition on atomic types upon which they were defined.

Assumption 3. From now on, we do not require anymore $\mathbb{W}_\alpha$ and $\mathbb{C}_\alpha$ to be non-empty.

It turns out that we do not really need the orthogonality relation either in this new paradigm. Actually, the orthogonality was solely used to discriminate dummy terms. Now that atomic types may be empty, there is no way to construct such terms and paraproofs automatically become valid proofs.

Definition 8 (Revised proofs)**.** A proof of A is now any term $t : \mathbb{W}[A]$.

The careful scrutiny of the translation indicates that the only type using the dummy terms and merging features was the exponential $!A$. Indeed, those constructions only appear in collectors, and with types from the context of the sequent, which are precisely the types wearing an exponential modality in the translation. We can parallel the use of these terms with structural rules introduced by the call-by-name translation:

- Weakening at axioms for an unused variable, matched by the use of a $\varnothing$ in our translation;
- Contraction at applications, matched by the `merge` operator;
- Promotion and dereliction, transparent in Dialectica.

Therefore, we only have to adapt the translation of the bang connective as follows. All other type interpretations remain unchanged.

Definition 9 (Revised bang)**.** The counter interpretation of the bang connective is now interpreted as:

$$\mathbb{C}[!A] := \mathbb{W}[A] \Rightarrow \mathfrak{M}\, (\mathbb{C}[A])$$

This also affects the typing of sequents, as collectors' typing is turned from:

$$\mathbb{W}[\Gamma] \vdash t_{x_i} : \mathbb{C}[A] \Rightarrow \mathbb{C}[\Gamma_i]$$

into:

$$\mathbb{W}[\Gamma] \vdash t_{x_i} : \mathbb{C}[A] \Rightarrow \mathfrak{M}\, (\mathbb{C}[\Gamma_i])$$

[1] We use infix notation to ease readability, but they should be treated as plain λ-terms.

Fact 3. If $\mathfrak{M}\ A$ is taken to be the type of finite sets over A, then the translation we recover is the linearization of the so-called Diller-Nahm variant of the Dialectica translation [4].

Rediscovering a known transformation through a bypath is quite a reassuring thing. As we shall see, it is nonetheless more natural to consider finite multisets instead of finite sets here, even though the finite set version would comply with the requirements stated in the next section. This is not unrelated to the similar choice found in coherent spaces [10].

3.3 The revised Dialectica

The translation is essentially the same. The intuitionistic part remains unchanged. We only add abstract multiset related structure in the variable and application rules.

Definition 10 (Revised Dialectica)**.** The revised Dialectica transformation is defined at Figure 5.

Variable case:

$$\begin{array}{lcll} x^\bullet & := & x & \\ x_x & := & \lambda\pi.\,\{\pi\} & \\ x_y & := & \lambda\pi.\,\varnothing & \text{when } x \neq y \end{array}$$

Abstraction case:

$$\begin{array}{lcl} (\lambda x.\,t)^\bullet & := & (\lambda x.\,t^\bullet, \lambda\pi x.\,t_x\ \pi) \\ (\lambda x.\,t)_y & := & \lambda(x,\pi).\,t_y\ \pi \end{array}$$

Application case:

$$\begin{array}{lcl} (t\,u)^\bullet & := & (\mathtt{p}_1\ t^\bullet)\ u^\bullet \\ (t\,u)_y & := & \lambda\pi.\,(t_y\ (u^\bullet, \pi)) \circledast (\mathtt{p}_2\ t^\bullet\ \pi\ u^\bullet \gg\!= (\lambda\rho.\,u_y\ \rho)) \end{array}$$

Figure 5. The revised Dialectica transformation

Preservation of typability is obtained as expected.

Theorem 1 (Soundness)**.** *For all* $\Gamma \vdash t : A$ *and* $(x : \Gamma_i) \in \Gamma$*, we have:*

$$\mathbb{W}[\Gamma] \vdash t^\bullet : \mathbb{W}[A]$$
$$\mathbb{W}[\Gamma] \vdash t_x : \mathbb{C}[A] \Rightarrow \mathfrak{M}\,(\mathbb{C}[\Gamma_i])$$

The real gain of this variant is that, given the right operational semantics on the abstract multiset structure, we recover a true proof-theoretic translation that preserves β-equivalence.

Assumption 4. We assume that the abstract multiset complies with the following rewriting rules.

Monadic laws

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

Monoidal laws

$$t \circledast u \equiv_\beta u \circledast t$$
$$\varnothing \circledast t \equiv_\beta t \circledast \varnothing \equiv_\beta t$$
$$(t \circledast u) \circledast v \equiv_\beta t \circledast (u \circledast v)$$

Distributivity laws

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

Commutative cuts

$$\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto \varnothing \equiv_\beta \varnothing$$
$$\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto m_1 \circledast m_2 \equiv_\beta$$
$$(\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto m_1) \circledast (\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto m_2)$$
$$(\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto u) \gg\!= f \equiv_\beta$$
$$\mathtt{match}\ t\ \mathtt{with}\ (x,y) \mapsto (u \gg\!= f) \quad (x, y \text{ not free in } f)$$

There are some comments to be made about such a structure. The first three groups of rules are natural and can be thought of as the expected dynamical behaviour of a (finite) multiset. Nonetheless, note that it may be degenerated, as the unit monad $\mathfrak{M}\ A := \mathbb{1}$ does fit the bill with the trivial operations.

The existence of the free version of an abstract multiset in the pure λ-calculus should be questioned, but the usual list structure is not far from being a candidate. Actually, the only real property absent in lists is the commutativity of the union. As we never escape from the multiset monad, one solution could consist in working with up-to-reordering lists by natively putting built-in equivalence rules in the rewriting system of an extended λ-calculus. Or we could just see it from the programming language side, by assuming an abstract primitive datastructure coming as an extension of the λ-calculus.

The commutative cut rules are more of a technical problem. We need them to ensure the good properties of the translation. In our case, they are an elegant way to palliate more serious issues of positive connectives in λ-calculus. This particular problem has been around for a long time, but seems to have been considered as a technical hassle introduced by inductive types. For more details, see for instance [6, 14].

Assuming those rewriting rules, one can derive the following set of equivalences on Dialectica.

Proposition 6 (Emptyness)**.** *If* x *is not a free variable of* t*, then* $t_x \equiv_\beta \lambda\pi.\,\varnothing$*.*

Proposition 7 (Substitution lemma)**.** *We have:*

$$(t\{x \leftarrow r\})^\bullet \equiv_\beta t^\bullet\{x \leftarrow r^\bullet\}$$

when x *is not free in* r*, and:*

$$(t\{x \leftarrow r\})_y \equiv_\beta$$
$$\lambda\pi.\,(t_y\{x \leftarrow r^\bullet\}\ \pi) \circledast (t_x\{x \leftarrow r^\bullet\}\ \pi \gg\!= \lambda\rho.\,r_y\ \rho)$$

when x *is not free in* r *and* $x \neq y$*.*

Remark 1. Note that by combining the emptyness lemma with the substitution lemma, we obtain:

$$(t\{x \leftarrow r\})_y \equiv_\beta t_y\{x \leftarrow r^\bullet\}$$

when both x and y are not free in r and $x \neq y$.

We can finally conclude that β-equivalence is preserved through the revised Dialectica transformation.

Theorem 2. *If* $t \equiv_\beta u$*, then* $t^\bullet \equiv_\beta u^\bullet$*.*

4. Through the Abstract Machine

So far, we have only been interested in the typing and soundness properties of the Dialectica transformation. This is not the end of the story, though. We can show that the Dialectica transformation is not only one of the first Curry-Howard-like logical transformation, but that it *is* actually one of those. As we will show in this section, the content of the translation can be explained by the operational semantics of an abstract machine, and not just any. Dialectica can indeed be described thanks to the famous Krivine abstract machine.

4.1 The Krivine abstract machine

The Krivine abstract machine (KAM for short) shall be the key to the unveiling of the computational content hidden into Dialectica. This is a call-by-name abstract machine that serves many purposes, one of its most prolific uses being the classical realizability [11].

We recall at Figure 6 the basic definitions of the KAM. A KAM state, also called process, is a pair of a head closure c and a stack π. The head closure is responsible for the transition rules to apply to the process, according to the rules of Figure 6.

$$\begin{array}{llll} \text{Closures} & c & ::= & (t,\sigma) \\ \text{Environments} & \sigma & ::= & \emptyset \mid \sigma + (x := c) \\ \text{Stacks} & \pi & ::= & \varepsilon \mid c \cdot \pi \\ \text{Processes} & p & ::= & \langle c \mid \pi \rangle \end{array}$$

$$\begin{array}{lllll} \textsc{Push} & \langle (t\,u,\sigma) \mid \pi \rangle & & \to & \langle (t,\sigma) \mid (u,\sigma)\cdot\pi \rangle \\ \textsc{Pop} & \langle (\lambda x.\,t,\sigma) \mid c\cdot\pi \rangle & & \to & \langle (t,\sigma + (x := c)) \mid \pi \rangle \\ \textsc{Grab} & \langle (x,\sigma + (x := c)) \mid \pi \rangle & & \to & \langle c \mid \pi \rangle \\ \textsc{Garbage} & \langle (x,\sigma + (y := c)) \mid \pi \rangle & & \to & \langle (x,\sigma) \mid \pi \rangle \end{array}$$

Figure 6. The Krivine Abstract Machine (KAM)

We shall emphasize that there are several variants of the KAM in the wild. Note in particular that in the present variant, we use closures instead of direct substitution. This feature is an absolute requirement to understand how Dialectica works under the hood, in a fashion similar to [13]. It turns out that the GRAB rule will be our quintessential point of attention.

4.2 Lifting Dialectica

Working with a direct-style operational semantics like the KAM is lighter, but this forces us to relate its first-class objects to the encoded structures found in the translation. There are essentially two straightforward but cumbersome additional translations to define, one for each newly introduced object: environments and stacks.

We will rule out the issue of environments quickly by translating closures to plain terms. We will do so by flattening them recursively.

Definition 11 (Closure flattening)**.** Let (t,σ) be a closure. The closure flattening of (t,σ), written $t \ltimes \sigma$, is defined inductively as:

- $t \ltimes \emptyset := t$
- $t \ltimes \sigma + (x := (u,\tau)) := (t \ltimes \sigma)\{x \leftarrow u \ltimes \tau\}$

If $\sigma = (x_1 := (r_1,\tau_1)) + \ldots + (x_n := (r_n,\tau_n))$, we will write by an abuse of notation:

$$t\{\vec{x} \leftarrow \sigma^\bullet\} := t\{\vec{x} \leftarrow (\vec{r} \ltimes \vec{\tau})^\bullet\}$$

Proposition 8. *The following equivalence holds:*

$$t^\bullet\{\vec{x} \leftarrow \sigma^\bullet\} \equiv_\beta (t \ltimes \sigma)^\bullet$$

More interestingly, KAM stacks are also given a first-class citizenship in the Dialectica transform, albeit not being formally separated from the target of λ-terms. Thanks to the existence of pairs in $\lambda^\times$, we can define the following simple translation of stacks.

Definition 12 (Stack translation)**.** Let π be a KAM stack. We define its Dialectica translation $\pi^\bullet$ by induction over π:

- $\varepsilon^\bullet := \varepsilon$, for some fresh variable ε
- $((t,\sigma)\cdot\pi)^\bullet := ((t \ltimes \sigma)^\bullet, \pi^\bullet)$.

Actually, we could even design a typing judgment for KAM stacks, that would be preserved through the stack translation, where a stack $\pi \vdash A$ would be mapped to a counter $\vdash \pi^\bullet : \mathbb{C}[A]$.

4.3 The main result

We can now answer the issue of the computational content of the Dialectica translation.

Theorem 3 (KAM simulation)**.** *Let t be a term, σ an environment and π a stack. Assume x a free variable of t such that x appears neither in any hereditary binding of σ nor in π. Suppose now that there exists an environment τ and a stack ρ s.t.:*

$$\langle (t,\sigma) \mid \pi \rangle \to^* \langle (x,\tau) \mid \rho \rangle$$

Then there exists a term m such that:

$$t_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi^\bullet \equiv_\beta \{\rho^\bullet\} \circledast m$$

Proof. By induction on the length of the machine reduction.

- Rule GRAB: there are two cases.
  If $\langle (x,\sigma + (x := c)) \mid \pi \rangle \to \langle c \mid \pi \rangle$. We have:
  $$x_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi^\bullet \to_\beta \{\pi^\bullet\} \equiv_\beta \{\pi^\bullet\} \circledast \varnothing$$
  If $\langle (y,\sigma + (y := c)) \mid \pi \rangle \to \langle c \mid \pi \rangle$ where $y \neq x$. We assumed that x was not in σ nor in π, therefore it cannot reduce to a state $\langle (x,\tau) \mid \rho \rangle$. Note that in this case:
  $$y_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi^\bullet \to_\beta \varnothing$$
- Rule GARBAGE. This rule is transparent for the translation. We conclude by induction hypothesis.
- Rule POP. Assume, for some $y \neq x$:
  $$\langle (\lambda y.\,t,\sigma) \mid (u,\tau)\cdot\pi \rangle \to \langle (t,\sigma + (y := (u,\tau)) \mid \pi \rangle$$
  Then we have:
  $$\begin{array}{ll} & (\lambda y.\,t)_x\{\vec{x} \leftarrow \sigma^\bullet\}\,((u \ltimes \tau)^\bullet, \pi^\bullet) \\ \equiv_\beta & t_x\{\vec{x} \leftarrow \sigma^\bullet, y \leftarrow (u \ltimes \tau)^\bullet\}\,\pi^\bullet \end{array}$$
  We conclude by induction hypothesis.
- Rule PUSH. This is the interesting rule. Assume:
  $$\langle (t\,u,\sigma) \mid \pi \rangle \to \langle (t,\sigma) \mid (u,\sigma)\cdot\pi \rangle$$
  The translation gives:
  $$\begin{array}{ll} & (t\,u)_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi^\bullet \\ \to_\beta & t_x\{\vec{x} \leftarrow \sigma^\bullet\}\,(u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\}, \pi^\bullet) \circledast \\ & \mathtt{p}_2\,(t^\bullet\{\vec{x} \leftarrow \sigma^\bullet\})\,\pi^\bullet\,u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\} \gg\!= \\ & \lambda\pi.\,u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi \end{array}$$
  There are two ways the right hand side may evaluate to some state $\langle (x,\tau) \mid \rho \rangle$. The variable x is encountered either in the reduction of t or in the reduction of u.

  If x is encountered in the reduction of t, we can α-rename the occurences of x in the closure (u,σ) as $(\hat{u},\hat{\sigma})$ without affecting the considered reduction path. We get the following α-converted reduction:
  $$\langle (t,\sigma) \mid (\hat{u},\hat{\sigma})\cdot\pi \rangle \to^* \langle (x,\tau) \mid \rho \rangle$$
  The altered reduction has the same length as it original counterpart, so we recover by the induction hypothesis:

Actually, we could even design a typing judgment for KAM stacks, that would be preserved through the stack translation, where a stack $\pi \vdash A$ would be mapped to a counter $\vdash \pi^\bullet : \mathbb{C}[A]$.

$$
\begin{array}{ll}
 & t_x\{\vec{x} \leftarrow \sigma^\bullet\}\,(u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\}, \pi^\bullet) \\
\equiv_\beta & t_x\{\vec{x} \leftarrow \sigma^\bullet\}\,((\hat{u} \ltimes \hat{\sigma})^\bullet, \pi^\bullet) \\
\equiv_\beta & \{\rho^\bullet\} \circledast m
\end{array}
$$

from which we can conclude using rewriting steps.

If x is encountered in the reduction of u, then we have a reduction of the following shape where x_0 is fresh:

$$
\begin{array}{ll}
 & \langle (t, \sigma) \mid (u, \sigma) \cdot \pi \rangle \\
\to^* & \langle (\lambda x_0.\, t_0, \sigma) \mid (u, \sigma) \cdot \pi \rangle \\
\to & \langle (t_0, \sigma + x_0 := (u, \sigma)) \mid \pi \rangle \\
\to^* & \langle (x_0, \sigma + x_0 := (u, \sigma)) \mid \hat{\pi} \rangle \\
\to & \langle (u, \sigma) \mid \hat{\pi} \rangle \\
\to^* & \langle (x, \tau) \mid \rho \rangle
\end{array}
$$

By applying the induction hypothesis on the variable x_0 to $\langle (t_0, \sigma + x_0 := (u, \sigma)) \mid \pi \rangle$, we obtain

$$t_{0x_0}\{\vec{x} \leftarrow \sigma^\bullet, x_0 \leftarrow (u \ltimes \sigma)^\bullet\}\,\pi^\bullet \equiv_\beta \{\hat{\pi}^\bullet\} \circledast m$$

for some m. By the KAM reduction, we also have:

$$t\{\vec{x} \leftarrow \sigma\} \to_\beta^* \lambda x_0.\, t_0\{\vec{x} \leftarrow \sigma\}$$

from which we can rewrite:

$$
\begin{array}{ll}
 & \mathrm{p}_2\,(t^\bullet\{\vec{x} \leftarrow \sigma^\bullet\})\,\pi^\bullet\,u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\} \gg\!= \\
 & \quad \lambda\pi.\, u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi \\
\to_\beta^* & \mathrm{p}_2\,(\lambda x_0.\, t_0\{\vec{x} \leftarrow \sigma\})^\bullet\,\pi^\bullet\,u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\} \gg\!= \\
 & \quad \lambda\pi.\, u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi \\
\to_\beta^+ & t_{0x_0}\{\vec{x} \leftarrow \sigma, x_0 \leftarrow u^\bullet\{\vec{x} \leftarrow \sigma^\bullet\}\}\,\pi^\bullet \gg\!= \\
 & \quad \lambda\pi.\, u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi \\
\equiv_\beta & \{\hat{\pi}^\bullet\} \circledast m \gg\!= \lambda\pi.\, u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi \\
\equiv_\beta & (u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\hat{\pi}^\bullet) \circledast (m \gg\!= \lambda\pi.\, u_x\{\vec{x} \leftarrow \sigma^\bullet\}\,\pi)
\end{array}
$$

We can conclude by applying the induction hypothesis to $\langle (u, \sigma) \mid \hat{\pi} \rangle$.

□

We could refine this result at the cost of a more technical presentation. Actually, it is easy to check that the collectors also preserve multiplicity of the encountered stacks, as witnessed by the rules for variables and application. This is why the choice of finite multisets is more natural than the choice of finite sets from an operational point of view.

Note also that what the we obtain is an equivalence rather than some reduction. This is chiefly due to the fact we had to present the commutative cuts as equivalences. Recovering a reduction instead is far from from obvious, if achievable at all.

4.4 Is Dialectica broken?

We can rephrase this result and comment it a bit. What Dialectica does is no more than a bookkeeping of the states of the abstract machine that triggered a GRAB rule. Each time a variable of a closure is accessed, Dialectica remembers the current stack to return it at the end. Hence the name of x-collectors we gave to the $(\cdot)_x$ transformations.

In particular, the second component of the translation of λ-abstractions can be simply seen as the collector for the bound variable hidden in the λ-term by η-expansion. And indeed, it happens that we have such a correspondance.

Fact 4 (Collector η-expansion). For all term t and for all variables x and π free in t, we have:

$$\mathrm{p}_2\,t^\bullet\,\pi\,x \equiv_\beta (t\,x)_x\,\pi$$

Therefore, it seems that we should be satisfied with the KAM description. Unluckily, there is a catch. The simulation result actually raises more legitimate questions than it solves. Indeed, we can make the following experimental observation.

Fact 5. The stacks are produced regardless of the evaluation order, i.e. if we implemented $\mathfrak{M}\,A$ as a list, then the produced list would not respect the order in which each stack is encountered in the KAM.

This is actually a critical problem. The Dialectica transform is computing something which is partly wrong: it does produce the encountered stacks with the right multiplicity, but the result does not agree with the intrinsic sequentiality of the KAM. One could argue that the KAM is the faulty one here. But it does not sound right, for Dialectica has the following defects:

1. List-using Dialectica does not preserve β-reduction, as the commutation rule for $\circledast$ is essential in the proof of theorem 2.
2. The output order is totally irrelevant with respect to any sensible reduction strategy we can think of anyway.

The issue arises in the translation of the application rule. We recall that the translation of applications is of the following two-part form.

$$(t\,u)_y\,\pi \equiv_\beta (t_y\,(u^\bullet, \pi)) \circledast (\mathrm{p}_2\,t^\bullet\,\pi\,u^\bullet \gg\!= (\lambda\rho.\, u_y\,\rho))$$

Under our interpretation, the left part corresponds to accesses to y in t, while the right part corresponds to repeated accesses in u after its substitution in $t \equiv_\beta \lambda x.\, t_0$. The problem is, these two parts should be interleaved. Indeed, t may access y before *and* after reducing to $\lambda x.\, t_0$. In the latter case, accesses to y may occur *in between* accesses to x. There is therefore no reason to split y-accesses in two parts as done in our translation. In the light of this evidence, the commutativity requirement for the abstract multisets is a barely a workaround to this deep design issue.

Alas, there is no obvious way to fix Dialectica such that it agrees with the sequential semantics of the KAM. The translation of the linear arrow seems to be the culprit: it is totally oblivious to the relative order by which events happen, and there is no way to recover causality dependence between its two components.

5. Towards $\mathbf{CC}_\omega$

There have been ongoing efforts to lift Dialectica-like translations to dependent types in a categorical system through the double-glueing construction, to no avail. This failure seems to originate in the *ad hoc* encodings of the historical translation. Our formulation renders the problem tractable, if not trivial, in a syntactic setting.

We will indeed show now, as a proof-of-concept, how our presentation naturally applies to the quite expressive $\mathbf{CC}_\omega$. Indeed, $\mathbf{CC}_\omega$ is a pure type system with a denumerable quantity of universes featuring amongst others dependent types. It is a variant of Luo's ECC [12] without any impredicative universes.

5.1 Presenting $\mathbf{CC}_\omega$

In this section, we provide the reader with one of the formalization of the Curry-style $\mathbf{CC}_\omega$ system. The whole set of rules is described at Figure 7 for the record, but for the moment, let us discuss a bit about this theory.

$$A, B, t, u ::= x \mid \lambda x.\, t \mid t\, u \mid \Box_{i \in \mathbb{N}} \mid \Pi x : A.\, B$$
$$(\lambda x.\, t)\, u \quad \to_\beta \quad t\{x \leftarrow u\}$$

$$\textsc{Wf-Empty}\ \frac{}{\vdash \cdot} \qquad \frac{\Gamma \vdash A : \Box_i}{\vdash \Gamma, x : A}\ \textsc{Wf-Cons}$$

$$\textsc{Ax}\ \frac{\Gamma \vdash A : \Box_i}{\Gamma, x : A \vdash x : A} \qquad \frac{i < j \qquad \vdash \Gamma}{\Gamma \vdash \Box_i : \Box_j}\ \textsc{Type}$$

$$\textsc{Wkn}\ \frac{\Gamma \vdash t : B \qquad \Gamma \vdash A : \Box_i \qquad x \text{ not free in } t}{\Gamma, x : A \vdash t : B}$$

$$\textsc{Abs}\ \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash \Pi x : A.\, B : \Box_i}{\Gamma \vdash \lambda x.\, t : \Pi x : A.\, B}$$

$$\textsc{ForAll}\ \frac{\Gamma \vdash A : \Box_i \qquad \Gamma, x : A \vdash B : \Box_j}{\Gamma \vdash \Pi x : A.\, B : \Box_{\max(i,j)}}$$

$$\textsc{App}\ \frac{\Gamma \vdash t : \Pi x : A.\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B\{x \leftarrow u\}}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash B : \Box_i \qquad A \equiv_\beta B}{\Gamma \vdash t : B}$$

Figure 7. The $\mathbf{CC}_\omega$ system

As for the calculus of constructions from where it stems, the usual λ-calculus arrow is refined as a *dependent arrow* $\Pi x : A.\, B$ where B may depend on A. This allows would-be types to contain plain terms, hence the adjective *dependent*.

Actually, there is no formal distinction in $\mathbf{CC}_\omega$ between terms and types, which are merged in the same syntactic class generically described as terms. In order to be able to type terms that would correspond to usual types, it features a denumerable hierarchy of universes $\Box_{i \in \mathbb{N}}$ which are the *types of types*, where each type at a level i inhabits in a higher universe at a level $j > i$. This strict constraint ensures that the type system can rule out unsoundness based on variants of Russel's paradox [2].

We made the choice to use a Curry-style $\mathbf{CC}_\omega$ for simplicity of this particular exposition. For the implementer, this is not innocuous, as the Curry-style presentation suffers from practical defects. One of the most salient issues is the indecidability of type inference. We made this choice for the sake of clarity, as we could readily adapt the Dialectica transformation to the usual Church-style presentation. It is just easier to manipulate the Dialectica transform in Curry-style, and it fits more naturally in the course of this article.

5.2 The issues of a type-dependent Dialectica

There is only a small amount of modifications to do in order to lift the translation of Figure 5 to $\mathbf{CC}_\omega$. This essentially amounts to the following:

- Because we introduced dependent types, we need to switch from plain arrows $A \Rightarrow B$ and products $A \times B$ to their dependent counterparts $\Pi x : A.\, B$ and $\Sigma x : A.\, B$.
- As we gave a first-class citizenship to types, we also need to provide them with an interpretation.

Actually, the first point is rather simple, and the second point is almost already solved.

5.2.1 Handling type-dependency

First, the introduction of dependency imposes us to switch from products to dependent products in the target system. We present at Figure 8 the rules to add to $\mathbf{CC}_\omega$ to recover the adapted system $\mathbf{CC}_\omega^\times$.

$$A, B, t, u ::= \ldots \mid (t, u) \mid \texttt{match}\ t\ \texttt{with}\ (x, y) \mapsto u$$
$$\texttt{match}\ (t, u)\ \texttt{with}\ (x, y) \mapsto r \quad \to_\beta \quad r\{x, y \leftarrow t, u\}$$

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B\{x \leftarrow t\} \qquad \Gamma \vdash \Sigma x : A.\, B : \Box_i}{\Gamma \vdash (t, u) : \Sigma x : A.\, B}$$

$$\frac{\Gamma \vdash t : \Sigma x : A.\, B \qquad \Gamma, x : A, y : B \vdash u : C}{\Gamma \vdash \texttt{match}\ t\ \texttt{with}\ (x, y) \mapsto u : C}$$

$$\frac{\Gamma \vdash A : \Box_i \qquad \Gamma, x : A \vdash B : \Box_j}{\Gamma \vdash \Sigma x : A.\, B : \Box_{\max(i,j)}}$$

Figure 8. Target $\mathbf{CC}_\omega^\times$

Note in particular that in $\mathbf{CC}_\omega^\times$, we can recover a non-dependent arrow $A \Rightarrow B$ and a non-dependent product $A \times B$ by the forgetful encoding:

$$A \Rightarrow B := \Pi x : A.\, B \quad A \times B := \Sigma x : A.\, B$$

where x is not free in B.

Additionally, the dependency issue requires some isomorphism reordering to lift the revised Dialectica. This unfortunately breaks the linear decomposition of the arrow. Recall that we had in the revised transformation:

$$\begin{aligned} \mathbb{W}[A \Rightarrow B] &= \mathbb{W}[!A \multimap B] \\ &= \begin{cases} \mathbb{W}[A] \Rightarrow \mathbb{W}[B] \\ \quad \times \\ \mathbb{C}[B] \Rightarrow \mathbb{W}[A] \Rightarrow \mathfrak{M}\,(\mathbb{C}[A]) \end{cases} \end{aligned}$$

Yet, because of the introduction of the dependency, B may now depend on an element of A. Thus the $\mathbb{W}[A]$ argument in the second component should be placed before the $\mathbb{C}[B]$ argument, resulting in the following translation:

$$\mathbb{W}[\Pi x : A.\, B] = \begin{cases} \Pi x : \mathbb{W}[A].\, \mathbb{W}[B] \\ \quad \times \\ \Pi x : \mathbb{W}[A].\, \mathbb{C}[B] \Rightarrow \mathfrak{M}\,(\mathbb{C}[A]) \end{cases}$$

As a side remark, this seems to be a hint at the fact that there is something broken either in the linear decomposition or in the current presentation of dependent types. We do not know which one is the actual culprit though.

5.2.2 Handling higher-order

As $\mathbf{CC}_\omega$ does not distinguish between plain terms and types, we must translate both at the same time. This raises two issues.

First, we must adapt abstract multisets to live well in this higher-order setting. This is not much of a problem, and we tweak them as follows.

Definition 13. In this section, we change the typing rule of $\varnothing$ by:

$$\frac{\Gamma \vdash A : \square_i}{\Gamma \vdash \varnothing : \mathfrak{M}\, A}$$

and we require additional well-behavedness of $\mathfrak{M}$:

$$\frac{\Gamma \vdash A : \square_i}{\Gamma \vdash \mathfrak{M}\, A : \square_i} \qquad \frac{A \equiv_\beta B}{\mathfrak{M}\, A \equiv_\beta \mathfrak{M}\, B}$$

The two typing rules are motivated by coherence conditions, and the compatibility with β-equivalence is natural as soon as we will be computing in types. Remark that we won't be using $\gg\!=$ in a dependent setting, hence there is no need to change its typing rule to a dependent arrow.

We must also fix the problem of the translation of types. There is little room for design choice here. We must be able to access the witness and the counter types of any translated type at any moment. The only sensible way to do this is by setting the intuitionistic translation of a type to be a pair of types, composed of the witness and counter types. Therefore, the $\mathbb{W}[\cdot]$ and $\mathbb{C}[\cdot]$ constructions we have been using since the beginning are going to be turned into simple macros.

Definition 14 (Witness & counter macros)**.** Given any term A, we will write:

$$\mathbb{W}[A] := \mathtt{p}_1\, A^\bullet \qquad \text{and} \qquad \mathbb{C}[A] := \mathtt{p}_2\, A^\bullet$$

Likewise, we must answer the question of the proper collector translation for types. The abstract machine intuition could be a fruitful one, but unfortunately, as of today, there is no clear abstract machine presentation of a typing system for $\mathbf{CC}_\omega$. In particular, it is unclear when we must compute things and in which order. The most elegant way we came up with was to deny any computational contents to types, that is, given any closed type A, we will enforce $A_x \equiv_\beta \lambda\pi.\, \varnothing$.

5.3 Final translation

It is now simple to provide a Dialectica translation to $\mathbf{CC}_\omega$. The λ-calculus part is taken almost unchanged from Figure 5, except for the commutation described at paragraph 5.2.1 which impacts intuitionistic proof of the abstraction and collectors of the application.

The translation on types was already essentially defined at paragraph 5.2.2, so we just stick to these design choices.

All the rewriting lemmas from the simply-type case still hold. It is indeed sufficient to check that they are still valid for the two new constructs $\square_i$ and $\Pi x : A.\, B$. We recover in particular the β-reduction preservation lemma as before.

Proposition 9. *If $t \equiv_\beta u$, then $t^\bullet \equiv_\beta u^\bullet$.*

Thanks to this property, we can tediously check that all typing rules are preserved by the translation, and in particular the Conv inference rule which actively makes use of β-equivalence. We finally recover the preservation of typability in the following form.

Theorem 4 (Soundness)**.** *For all $\Gamma \vdash t : A$ and $(x : \Gamma_i) \in \Gamma$,*

$$\mathbb{W}[\Gamma] \vdash t^\bullet : \mathbb{W}[A]$$

$$\mathbb{W}[\Gamma] \vdash t_x : \mathbb{C}[A] \Rightarrow \mathfrak{M}\, (\mathbb{C}[\Gamma_i]).$$

To insist on the beauty of the dependent Dialectica translation, it is noteworthy that now, $\mathbb{W}[\cdot]$ and $\mathbb{C}[\cdot]$ are not a preexistent type translation, but actually a macro using the $(\cdot)^\bullet$ inductive term translation.

Theorem 5 (Non-degeneracy)**.** *Assuming $\mathbf{CC}^\times_\omega + \mathfrak{M}$ is coherent, there is no proof of $\mathtt{False}_i := \Pi A : \square_i.\, A$.*

Variable case:

$$\begin{aligned}
x^\bullet &:= x \\
x_x &:= \lambda\pi.\, \{\pi\} \\
x_y &:= \lambda\pi.\, \varnothing \qquad \text{when } x \neq y
\end{aligned}$$

Abstraction case:

$$\begin{aligned}
(\lambda x.\, t)^\bullet &:= (\lambda x.\, t^\bullet, \lambda x\pi.\, t_x\, \pi) \\
(\lambda x.\, t)_y &:= \lambda(x, \pi).\, t_y\, \pi
\end{aligned}$$

Application case:

$$\begin{aligned}
(t\, u)^\bullet &:= (\mathtt{p}_1\, t^\bullet)\, u^\bullet \\
(t\, u)_y &:= \lambda\pi.(t_y\, (u^\bullet, \pi)) \circledast (\mathtt{p}_2\, t^\bullet\, u^\bullet\, \pi \gg\!= (\lambda\rho.u_y\rho))
\end{aligned}$$

Type case:

$$\begin{aligned}
(\square_i)^\bullet &:= (\square_i \times \square_i, \square_i) \\
(\square_i)_y &:= \lambda\pi.\, \varnothing
\end{aligned}$$

Product case:

$$\begin{aligned}
(\Pi x : A.\, B)^+ &:= \begin{cases} \Pi x : \mathbb{W}[A].\, \mathbb{W}[B] \\ \qquad \times \\ \Pi x : \mathbb{W}[A].\, \mathbb{C}[B] \Rightarrow \mathfrak{M}\, \mathbb{C}[A] \end{cases} \\
(\Pi x : A.\, B)^- &:= \Sigma x : \mathbb{W}[A].\, \mathbb{C}[B] \\
(\Pi x : A.\, B)^\bullet &:= ((\Pi x : A.\, B)^+, (\Pi x : A.\, B)^-) \\
(\Pi x : A.\, B)_y &:= \lambda\pi.\, \varnothing
\end{aligned}$$

Figure 9. A type-dependent Dialectica transformation

Proof. Indeed, we have

$$\mathbb{W}[\mathtt{False}_i] \equiv_\beta \begin{cases} \Pi A : (\square_i \times \square_i).\, \mathtt{p}_1\, A \\ \qquad \times \\ \Pi A : (\square_i \times \square_i).\, \Pi\pi : (\mathtt{p}_2\, A).\, \mathfrak{M}\, \square_i \end{cases}$$

and there is no proof of the first component if the target system is coherent. □

Conclusion

In this paper, we have presented a modern formulation of the Dialectica transformation, based on the merging of a categorical presentation together with a syntactical Curry-Howard approach, thus providing it with a operational content. We also showed that a richer type system such as $\mathbf{CC}_\omega$ could accomodate a direct Dialectica transformation.

Yet, we also discovered that there was a subtle mismatch between Dialectica and its underlying computational model, which do not agree on such an important thing as sequentiality. This design issue is hardwired in the translation and cannot be solved trivially. This has some unpleasant consequences. For instance, if one wants to create a practical system not using the theoretical workaround of abstract multisets, she would have first to find a way to present a sequentialized Dialectica.

This argument can also be reversed in favour of Dialectica, and may be justified by the linear decomposition. Actually, models of linear logic are often partially forgetful of sequentialization. The commutativity of the exponential seems indeed a bit at odds with the sequentiality of abstract machines.

On a more positive note, tweaking the Dialectica translation of $\mathbf{CC}_\omega$ could be a fruitful way to achieve the comprehension of linear dependent types, or at least a better insight. In particular, the idea of types as computational-free objects is at stake if we want to bring the abstract machine intuition with dependent types. One

would require a better interleaving of typing and computation to solve this particular problem, a feature that usual PTS-based systems do not provide. Studying other PTS extensions like inductives with dependent elimination may also prove helpful.

From the point of view of side-effects, Dialectica can be considered as a weak form of delimited control. Contrarily to usual undelimited continuations, as featured by control operators like `callcc`, stacks returned by the collectors are delimited by the current stack, i.e. the argument of these collectors. But the continuations produced by Dialectica also have a *polarized* flavor, in the sense that instead of being an abstract continuation, they can be inspected just like a first-order *positive* datatype. Even more strikingly, this property is a requirement of the translation, as shown by the collector of the λ-abstraction, which does pattern-matching on the current stack to extract its argument. Such so-called *inspectable* continuations surprisingly arise in a somewhat related setting, described in an article from the same volume [15], which attempts to give a computational content to the involutive negation through polarization and delimited continuations. It seems that Dialectica features a well-behaved translation for a fragment of those calculi, and the relation between them should be worked out.

As a final constatation, and in the lineage of modern proof theory, we have shown once again that a famous logical translation had a well-defined computational content easily expressed in direct style with an abstract machine. This underlines once more the robustness of this technique.

Acknowledgements I would like to warmly thank Paul-André Melliès, Martin Hyland and Hugo Herbelin for their insightful discussions, Alexis Saurin for his continuous versatile support and Colin Riba for having provided me with the original motivation for this article. I would also like to thank the reviewers for their precious comments and remarks.

References

[1] 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.

[2] Thierry Coquand. An analysis of Girard's paradox. In *LICS*, pages 227–236. IEEE Computer Society, 1986.

[3] Valeria de Paiva. A dialectica-like model of linear logic. In David H. Pitt, David E. Rydeheard, Peter Dybjer, Andrew M. Pitts, and Axel Poigné, editors, *Category Theory and Computer Science*, volume 389 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 1989.

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

[5] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[6] Jean-Yves Girard. *The Blind Spot: Lectures on Logic*. European Mathematical Society, 2011.

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

[8] Hugo Herbelin. An intuitionistic logic that proves Markov's principle. *Logic in Computer Science, Symposium on*, 0:50–56, 2010.

[9] J. M. E. Hyland. Proof theory in the abstract. *Ann. Pure Appl. Logic*, 114(1-3):43–78, 2002.

[10] Martin Hyland and Andrea Schalk. Glueing and orthogonality for models of linear logic. *Theor. Comput. Sci.*, 294(1/2):183–231, 2003.

[11] Jean-Louis Krivine. Dependent choice, 'quote' and the clock. *Theor. Comput. Sci.*, 308(1-3):259–276, 2003.

[12] Zhaohui Luo. ECC, an extended calculus of constructions. In *LICS*, pages 386–395. IEEE Computer Society, 1989.

[13] Alexandre Miquel. Forcing as a program transformation. In *LICS*, pages 197–206. IEEE Computer Society, 2011.

[14] Guillaume Munch-Maccagnoni. *Syntax and Models of a non-Associative Composition of Programs and Proofs*. PhD thesis, Univ. Paris Diderot, December 2013.

[15] Guillaume Munch-Maccagnoni. Formulae-as-types for an involutive negation. In *Proceedings of the joint meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS)*, 2014.

[16] Paulo Oliva. Unifying functional interpretations. *Notre Dame Journal of Formal Logic*, 47(2):263–290, 04 2006.

[17] Thomas Streicher and Ulrich Kohlenbach. Shoenfield is Gödel after Krivine. *Math. Log. Q.*, 53(2):176–179, 2007.