A Materialist Dialectica

1 Introduction

At the very beginning of the 20^{th} century, Hilbert proposed to the world his now famous twenty-three problems for the next hundred years to come. They were open problems of that time whose importance was deemed sufficient enough to be considered as landmarks for the dawning century. Most notably, the second problem was to prove that arithmetic was consistent. In Hilbert's mind, proving it would have somehow marked the end of the quest for the foundations, because arithmetic was then thought to be a *complete* basis for all mathematics.

As we now know, this hope was a chimera. In 1931 Gödel [6] showed that any reasonable consistent logical system featured a formula independent from this system, that is, which was not provable in this system, nor was its negation. As arithmetic fitted the requirements of reasonableness, this theorem washed ashore the idea that arithmetic would encode all mathematics. Worse, Gödel also proved that the consistency of a reasonable system was independent from this system. This was a hard stroke. To prove the consistency of arithmetic thus required a system strictly more powerful than arithmetic itself. But the consistency of that system would require in turn a more powerful one, and so on, effectively requiring consistent systems *all the way down*.

It seems that Gödel tried to work around this inherent limitation of logic by reconsidering the intrinsic value of mere consistency. This is particularly visible in his subsequent work, including the double-negation translation [4] and the Dialectica translation [5]. Both translations were trying to reduce consistency of arithmetic to a computational property, drawing from Brouwer's *intuitionism*. Rather than a purely formal game consisting in applying trusted rules, Brouwer opposed to Hilbert and defended the fact that logic needed to be rooted in a constructivist approach, effectively building up the objects it described. He rejected in particular the principles of classical logic that allowed to create objects that could not be made explicit. Instead of relying on abstract, static logical principles, the constructivist approach allowed to base the logical content of a proof on the dynamics of an effective procedure hidden inside the latter, i.e. in modern terms, a program. This is where the *realizability* techniques stem from.

This did not stop there though. Pushing forward the ideas sketched by intuitionism, Curry and Howard showed that not only programs could be extracted from proofs through realizability, but also that the proofs were *already* programs in their own right. This observation, known as the *Curry-Howard correspondence*, sparkled an important paradigmatic shift. This identity allowed for the design of objects that were at the same time programming languages and proof systems. Such languages allow to write programs and prove them in the same formalism, ensuring both correct-by-construction programs as well as computer-checked proofs.

Another consequence of this equivalence, maybe more surprising, manifested itself by a chance remark due to Griffin [7]. Trying to type a programming primitive called `call/cc` from the Scheme programming language, he realized that this primitive actually granted the expressiveness of classical logic. Thus the Curry-Howard correspondence was not restricted to intuitionistic logic, and could be extended by providing extraneous primitives that retained a computational content. Following this discovery, Krivine started to try to implement well-known axioms in a computational way, spawning a fruitful research program, the so-called classical realizability [14].

2 Overview

This thesis places itself in the wake of this bicephal tradition, that merges ideas coming from computer science with principles originating in logic. This paradigmatic standpoint allows for an interdisciplinary look at objects from both worlds, and can often give many enlightening hindsights from the other side of the bridge. The central contribution of this thesis is actually an instance of such a methodological manifesto. It consists in looking at a well-known object, Gödel's Dialectica, with a Curry-Howard era computer scientist's look, based on the pioneering work by De Paiva and Hyland [19, 20, 9].

Their work provided a firm categorical framework to synthetize Dialectica-like translations. Strangely enough, the Dialectica translation by itself did not benefit from this categorical apparatus. In particular, a clear understanding of the computational effects at work in the translation remained to be found. What does the program corresponding to the translation of a proof actually do? Our work brings an answer to this question which can be summarized by the following claim.

> "All the while, Gödel's Dialectica translation was a classical realizability program translation, manipulating stacks as first-class objects and treating substitution in a computationally relevant way, ultimately allowing to observe those stacks at variable access time."

Buried under a hefty crust of technicalities inherited from a time where the λ-calculus was but a toy and Howard was not even a teenager, the Dialectica translation actually turns out to be a jewel of the finest kind, exhibiting constructions that are elegantly explained in the classical realizability realm. It features in particular a first-class notion of stacks as concrete objects, while also dealing with such a subtle thing as delayed substitution, as found in the Krivine machine for example. These notions were virtually unknown at the time the Dialectica translation was published, let alone designed, effectively making the work of Gödel even more awing.

The main contributions of this thesis can be stated as follows.

1. A reformulation of the Dialectica translation as a pure, untyped, program translation that respects the underlying operational semantics of the λ-calculus. While the work of De Paiva and Hyland was an important step into this direction, they were not based on computational objects and relied both on an underlying categorical structure and an explicitly typed language. To the best of our knowledge, the issue of the lack of preservation of β-equivalence in the original Dialectica was never even clearly stated as such.

2. A computational description of the translation, in terms of new side-effects described in the Krivine machine. This presentation is heavily inspired by the work of Krivine in classical realizability, and draws many of the folklore ideas of this research topic. This is a novel approach, and it raises actually more questions than it solves, for we now have a new range of intuitions arising from this description.

3. An extension of the Dialectica translation to the dependently-typed case, and a study of its limitations. Contrarily to double-negation translations, the Dialectica translation can be adapted easily to cope with type-dependency, as long as the preservation of β-equivalence has been solved. This is not the case though for dependent elimination, and this hints at new variants of the Dialectica.

4. Various relatives of the Dialectica translation appearing when considering distinct linear decompositions. Many of them are folklore, but presenting them in the same place in the style of Oliva [17] allows to highlight interesting common patterns, as well as possible new variants.

5. A new formulation for on-demand computation (the *call-by-need* family of reductions) rooted in logical considerations rather than *ad-hoc* hypotheses. This problem is related to the representation of variable substitution in λ-calculus, and echoes with the clever encodings of substitutions in the Krivine machine by means of closures that play an important rôle in the Dialectica translation.

The results from 1, 2 and 3 were published at CSL-LICS 2014 [21]. The results from 5 where published at ESOP 2016 [22]. Although not exactly in the thesis, some of the results of point 4 were at the root of an article accepted at LICS 2016 [11]. We describe more lengthily each point in the following sections.

3 Gödel's Dialectica [Ch. 7]

The Dialectica transformation, also known as the functional interpretation, was introduced by Gödel in the eponymous journal in 1958 [5], 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.

The goal of Gödel's Dialectica translation was to enrich Heyting's intuitionistic arithmetic **HA**, but without relying on additional axioms. In practice, this is done by translating **HA** proofs into System T, a simply-typed λ-calculus with integers, s.t. the resulting terms verify some properties provable in (a higher-order version of) **HA**. Thanks to this translation Dialectica interprets indeed two semi-classical principles, respectively known as *Markov's principle* and *the independence of premise*. The exact statements of those formulae are given below.

$$\neg\forall x.\,\neg A \to \exists x.\,A \qquad (A \to \exists x.\,B) \to \exists x.\,(A \to B)$$

Such axioms are independent of pure intuitionistic logic, and of **HA** in particular. Their constructivist acceptance depends on the formula A considered. In general, one requires A to be decidable in the case of Markov's principle, while A should be somehow computationally irrelevant in the independence of premise. Let us justify them under these assumptions informally.

- If A is decidable, Markov's principle can be algorithmically implemented as follows: start an unbounded loop from $n = 0$ and incrementally check whether $A[x := n]$ holds. This is possible because A is decidable. If ever some index satisfies the formula, stop and return it as the result for the existential. Thanks to the assumption $\neg\forall x.\,\neg A$, this loop must eventually terminate even though there is no definite way to know when. The termination argument is classical but it does not interfere with the operational behaviour of the algorithm.

- Assume we have a good notion of computationally irrelevant. One can imagine it as purely logical content, or in a weaker setting, a subset of types which are not observable, like purely negative types. Then if A is such a type, the content of the argument fed to the $A \to \exists x.\,B$ function does not matter to build the witness. One may pass it a default value. To ensure consistency, one should assume that the A type is inhabited. In our arithmetical system, we can safely produce a default integer when A is not inhabited, and ask the programmer for a subsequent proof of A, which would be impossible in this case. This actually justifies the above presentation of this axiom.

Some systems precisely implement the behaviours we just described. See for instance Herbelin's work [8], where the *return* part of Markov's principle is implemented thanks to an exception. It is known that the expressive power of the Dialectica translation is essentially equivalent to Heyting's arithmetic enriched with these two principles [1].

The global picture is the following: take a proof $\vdash A$ of **HA**, and translate it into a proof of a statement of the form:

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

where the witness (resp. counter) translation $\mathbb{W}(A)$ (resp. $\mathbb{C}(A)$) is a sequence of System T types, the variables $\vec{u}$ and $\vec{x}$ range over terms of System T of the corresponding type, and the *interpretation matrix* A^D is defined by induction over A. Furthermore, this translation preserves consistenty, as the translation of the absurdity is logically equivalent to the absurdity. These results are recalled in great detail in Chapter 7.

A careful scrutiny of this formulation reveals that, in our modern phraseology, we would call this a realizability interpretation: from a proof, extract a computational interpretation in a given programming language, here System T, together with a orthogonality relation A^D living in the meta-theory discriminating pseudo-proofs from actual proofs.

Alternatively, the translated formula can be seen as a sort of a game, as in game semantics: the existentially quantified terms $\vec{u}$ are going to be seen as the proponents (or witnesses) of A, the universally quantified terms $\vec{x}$ as opponents (or counters) of A, and A^D as the rules of the game on A. The goal of the witnesses is to defeat any possible opponent according to the rules of A^D.

4 Dialectica as a Program Translation (Part I) [Ch. 8]

In order to extract the computational content of this translation, one has to look closely into the translation of the proof terms. The original presentation of the Dialectica interpretation is a bit abstruse to our modern eyes. Indeed, System T is lacking some of the structure necessary to an elegant presentation, and Gödel had to resort to various tricks to make the translation work. Most notably, the translation emulates booleans and sum types with integer-based constructions, and replaces in-language pairs with meta-theoretical sequences.

Although it is common practice to present the Dialectica translation of a given set of minimal logical axioms, we exposed the historical translation in natural deduction style, which allows to reason almost immediatly in terms of λ-calculus. This simplifies a lot the mental burden associated to the translation, and shows that several points stand out as utterly necessary to it.

Fact 1. The following are crucial points of the Dialectica translation.

- Of all connectors, the implication is the only one that deserves a special treatment. While the witness definition of the other connectors is a matter of commutation, the arrow is expanded into a forward and a reverse component as follows:

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

- The translation requires for all formula A a dummy term $✠_A : \mathbb{C}(A)$ that exists thanks to the fact that all System T types are inhabited. It can be seen as a call-by-name exception, which is precisely the rôle devoted to the *daimon* from ludics, hence the notation.

- The translation requires for all formula A a term $\mathrm{merge}_A : \mathbb{C}(A) \to \mathbb{C}(A) \to \mathbb{W}(A) \to \mathbb{C}(A)$ s.t.

$$A_D[u, \mathrm{merge}_A\ x_1\ x_2\ u] \leftrightarrow A_D[u, x_1] \wedge A_D[u, x_2]$$

  for all $u : \mathbb{W}(A), x_1 : \mathbb{C}(A), x_2 : \mathbb{C}(A)$. This term is built out of the decidability of the A^D formula in System T.

By exploiting these observations, we build in Chapter 8 a slightly modernized presentation of Dialectica on the propositional fragment of **HA**, which has a well-understood computational representation, that is, the simply-typed λ-calculus. Thanks to the natural deduction approach taken in the historical translation, this is almost immediate. It is effectively presented as a translation from the simply-typed λ-calculus extended with algebraic datatypes into itself, while the A^D is expressed in a weak dependent type theory. We sketch it quickly below on the negative fragment.

Definition 1 (Translation)**.** The type translation of the arrow is given as

$$A \to B \quad \begin{cases} & \mathbb{W}(-) & \mathbb{C}(-) \\ \mathbb{W}(A) \to \mathbb{W}(B) & & \\ \times & & \mathbb{W}(A) \times \mathbb{C}(B) \\ \mathbb{W}(A) \to \mathbb{C}(B) \to \mathbb{C}(A) & & \end{cases}$$

while the orthogonality relation is given as

$$(A \to B)_D[(f, \varphi), (u, \pi)] := A_D[u, \varphi\ u\ \pi] \to B_D[f\ u, \pi]$$

and the term translation is divided into a forward translation $(- \vdash - : -)^\bullet$ together with a reverse translation $(- \vdash - : -)_x$ for every free variable x of the source term:

$$
\begin{array}{lcl}
(\Gamma \vdash x : A)^\bullet & := & x \\
(\Gamma \vdash \lambda x.\, t : A \to B)^\bullet & := & (\lambda x.\, (\Gamma, x : A \vdash t : B)^\bullet, \lambda x\, \pi.\, (\Gamma, x : A \vdash t : B)_x\ \pi) \\
(\Gamma \vdash t\ u : B)^\bullet & := & \mathrm{fst}\ (\Gamma \vdash t : A \to B)^\bullet\ (\Gamma \vdash u : A)^\bullet \\
(\Gamma \vdash x : A)_x & := & \lambda \pi.\, \pi \\
(\Gamma \vdash y : A)_x & := & \lambda \pi.\, \maltese_{\Gamma(x)} \\
(\Gamma \vdash \lambda y.\, t : A \to B)_x & := & \lambda \rho.\, \mathtt{match}\ \rho\ \mathtt{with}\ (y, \pi) \mapsto (\Gamma, y : A \vdash t : B)_x\ \pi \\
(\Gamma \vdash t\ u : B)_x & := & \lambda \pi.\, \mathrm{merge}^x_\Gamma\ \ ((\Gamma \vdash t : A \to B)_{\vec{x}}\ ((\Gamma \vdash u : A)^\bullet, \pi)) \\
 & & \qquad\qquad ((\Gamma \vdash u : A)_{\vec{x}}\ (\mathrm{snd}\ (\Gamma \vdash t : A \to B)^\bullet\ (\Gamma \vdash u : A)^\bullet\ \pi)) \\
 & & \qquad\qquad \vec{x}
\end{array}
$$

Theorem 1 (Soundness)**.** *For all closed λ-term $t : A$, the translated term $(\cdot \vdash t : A)^\bullet$ realizes A, i.e. $(\cdot \vdash t : A)^\bullet : \mathbb{W}(A)$ and furthermore for all $\pi : \mathbb{C}(A)$, $A_D[(\cdot \vdash t : A)^\bullet, \pi]$ holds.*

As described by De Paiva in her thesis [19], the previous construction can be recovered by carefully choosing a decomposition of intuitionistic types into linear types, and interpreting it into a syntactic category. Yet, it seems that the following constatation had never been formulated before our work.

Proposition 1. *There exist two λ-terms $\Gamma \vdash t : A$ and $\Gamma \vdash u : A$ such that $t \equiv_\beta u$ but $(\Gamma \vdash t : A)^\bullet \not\equiv_\beta (\Gamma \vdash u : A)^\bullet$.*

This incompatibility is not a mere technical issue related to some missing administrative reduction step. We can find indeed two terms whose translations are not equatable for any sensible equivalence over λ-terms. Note that this incompatibility is actually already present in the historical translation, as long as we think of **HA** and System T as two extensions of the simply-typed λ-calculus. Therefore, the current presentation cannot qualify as a program translation, because it does not respect the underlying semantics of the program.

5 Dialectica as a Program Translation (Part II) [Ch. 9]

The source of the above failure is the requirement of the merge operations and their evil counterparts, the dummy terms, for they are totally lacking any form of naturality in the categorical acceptance of the term. Indeed, the dummy term is not canonical in general. Worse, the very need to be able to define dummy terms forces use to have all interpreted types inhabited, and in particular requires that we interpret the empty type 0 by the singleton type 1.

Therefore, we are going to algebraize this behaviour by assuming an abstract datatype featuring the same properties as $\maltese$ and merge, but compatible with β-equivalence. This will be the rôle of *abstract multisets*.

Definition 2 (Abstract multisets)**.** An abstract multiset datastructure is given by:

- a parameterized type $\mathfrak{M}\,(-)$;
- two operations giving it a monad signature:

$$
\begin{array}{lcl}
\{\cdot\} & : & A \to \mathfrak{M}\,A \\
\gg\!= & : & \mathfrak{M}\,A \to (A \to \mathfrak{M}\,B) \to \mathfrak{M}\,B
\end{array}
$$

- two operations giving it a monoid signature:

$$
\begin{array}{lcl}
\varnothing & : & \mathfrak{M}\,A \\
\odot & : & \mathfrak{M}\,A \to \mathfrak{M}\,A \to \mathfrak{M}\,A
\end{array}
$$

We also expect some β-equivalences on abstract multisets which we will not detail here. One may truthfully think of the abstract multisets as a monad equipped with a compatible commutative monoidal structure, i.e. some quotient of the corresponding free structure, that is, finite multisets.

We sketch the revised Dialectica translation below. The main difference with the historical Dialectica lies in the addition of a monadic multiset layer, and the replacement of $\maltese$ by $\emptyset$ and merge with $\odot$.

Definition 3 (Type translation)**.** We just adapt the translation of types by adding a multiset return type to the reverse component of the arrow:

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

A term is then translated according to two schemes, the forward translation $(-)^\bullet$ and the reverse translation $(-)_x$, where x is a free variable.

Definition 4 (Term translation)**.** Given a λ-term t and a variable x, we mutually define the translations $t^\bullet$ and t_x by induction on t below.

$$\begin{aligned}
x^\bullet &:= x \\
(\lambda x.\, t)^\bullet &:= (\lambda x.\, t^\bullet, \lambda \pi\, x.\, t_x\ \pi) \\
(t\ u)^\bullet &:= \mathrm{fst}\ t^\bullet\ u^\bullet \\
x_x &:= \lambda \pi.\, \{\pi\} \\
x_y &:= \lambda \pi.\, \emptyset \\
(\lambda y.\, t)_x &:= \lambda (y, \pi).\, t_x\ \pi \\
(t\ u)_x &:= \lambda \pi.\, (\mathrm{snd}\ t^\bullet\ \pi\ u^\bullet \gg\!= \lambda \rho.\, u_x\ \rho) \odot (t_x\ (u^\bullet, \pi))
\end{aligned}$$

As expected, typing is preserved by the translation.

Theorem 2 (Typing soundness)**.** *If* $x_1 : \Gamma_1, \dots, x_n : \Gamma_n \vdash t : A$ *then*

$$\begin{aligned}
x_1 : \mathbb{W}(\llbracket \Gamma_1 \rrbracket_{\mathrm{n}}), \dots, x_n : \mathbb{W}(\llbracket \Gamma_n \rrbracket_{\mathrm{n}}) &\vdash t^\bullet : \mathbb{W}(\llbracket A \rrbracket_{\mathrm{n}}) \\
x_1 : \mathbb{W}(\llbracket \Gamma_1 \rrbracket_{\mathrm{n}}), \dots, x_n : \mathbb{W}(\llbracket \Gamma_n \rrbracket_{\mathrm{n}}) &\vdash t_{x_i} : \mathbb{C}(\llbracket A \rrbracket_{\mathrm{n}}) \to \mathfrak{M}\,\mathbb{C}(\llbracket \Gamma_i \rrbracket_{\mathrm{n}})
\end{aligned}$$

for all $1 \leq i \leq n$.

The main difference with the historical Dialectica is the following fundamental property.

Theorem 3 (Computational soundness)**.** *If* $t_1 \equiv_\beta t_2$ *then* ${t_1}^\bullet \equiv_\beta {t_2}^\bullet$ *and* ${t_1}_x \equiv_\beta {t_2}_x$ *for any variable* x.

The important point is that this equivalence also makes sense on untyped programs, that is, the revised Dialectica is not only a logical translation, but also a true program translation that happens in addition to preserve typing. This result puts it on par with much more standard program translations such as the various flavours of CPS.

Actually, our translation is not novel per se, at least typewise. It is indeed a generalization of the Diller-Nahm construction [3].

Proposition 2. *If we take* $\mathfrak{M} := \mathfrak{P}_f$ *to be the finite set structure with the expected operations we recover the so-called Diller-Nahm variant of the Dialectica translation.*

There is quite a methodologic difference though. The Diller-Nahm translation is motivated by the fact that the original translation requires atomic types to be inhabited and their orthogonality to be decidable, which may be much too strong from the logical point of view. That is not at all our motivation, as we aim to recover the preservation of β-equivalence, which is a much more proof-theoretical demeanour. Moreover, the target system of the Diller-Nahm translation is often thought of as a variant of some set theory, which is essentially non-computable. We, in turn, take it to be a true programming language equipped with types. This is another major difference.

We find it rather peculiar that the same modification brings us with two sharp improvements over the traditional Dialectica. Yet, the most intriguing fact to us is that up to the present work, it seems that nobody ever realized that the Diller-Nahm variant also solved the β-equivalence preservation issue. We conjecture that the Dialectica community was not interested in the proof-theoretical properties *per se* of their various translations, preferring the study of their logical expressiveness instead.

One may have noticed that we did not mention the orthogonality relation defined on the multiset-using types. This is perfectly doable, as long as we have a notion of being true at each elements for our abstract multisets. The corresponding rule for the arrow type would be the following.

$$(A \to B)_D[(f,\varphi),(u,\pi)] := (\forall \rho \in \varphi\ u\ \pi.\, A_D[u,\rho]) \to B_D[f\ u,\pi]$$

Nonetheless, the necessity of orthogonality in the present context is greatly reduced.

- First, as far as preservation of reduction is concerned, we actually do not need orthogonality anymore. If we bothered about orthogonality, our translated terms would actually verify the realizability relation. Another way to state it is that we now only care of the computational contents of the skeleton of our proofs, not on the internalized logical properties they convey.
- Second, orthogonality can chiefly be seen as a way to rule out unsoundness of the resulting model when allowing proof terms inhabiting any type. Once we get rid of the dummy terms we were using all along, soundness will be preserved by construction, i.e. by choosing to interpret the empty type as itself.

6 A Classical Realizability Intepretation of Dialectica [Ch. 9]

We present here our main result on the Dialectica translation. As we have shown, the Dialectica translation is indeed a full-fledged program translation. The natural question that follows is to understand what the translation is actually doing from the point of view of the dynamics of the program. In particular, we need to account for the reverse translation $(-)_x$, which is the main component that makes the translation non-trivial.

We want to exploit the Curry-Howard isomorphism in the proof-to-program direction, and describe the target of the Dialectica translation as a λ-calculus enriched with some side-effect. The essential tool on which this work is based is the Krivine abstract machine, We will use in quite an involved way the structures it uses.

To summarize it quickly, the Dialectica translation is a way to manipulate first-class observable stacks in a program, in a way similar to delimited continuations. Such stacks are constructed thanks to the reverse translations.

Before giving details, we recall a minute amount of background.

Definition 5 (Krivine Abstract Machine)**.** We recall quickly the various components that constitute the machine: processes p, environments σ, closures c, stacks π and usual λ-terms t.

$$\begin{array}{rcl} p & := & \langle c \mid \pi \rangle \\ c & := & (t,\sigma) \\ \sigma & := & \cdot \mid \sigma + (x := c) \\ \pi & := & \varepsilon \mid c \cdot \pi \end{array}$$

The reduction rules of the machine are the following.

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

We now want to point out an obvious fact that was hidden in plain sight.

Fact 2. The counter types $\mathbb{C}(-)$ from the Dialectica translation are actually the types of stacks from the KAM accepting the corresponding witnesses. Therefore, the Dialectica translation manipulates KAM stacks in a first-class fashion.

But there is even more relation between the KAM and the Dialectica translation.

Fact 3. The computational soundness of the translation relies crucially on the following substitution lemma:

$$(t[x := r])_y \equiv_\beta \lambda\pi.\, (t_y[x := r^\bullet]\ \pi) \odot (t_x[x := r^\bullet]\ \pi \gg\!= \lambda\rho.\, r_y\ \rho)$$

where x is not free in r. While the left-hand side of the union is expected, the right-hand side is a bit more mysterious. It shows that substitution is *not* transparent in the Dialectica translation and that it has a computational content. This is precisely mimicked by the KAM environment that delay the substitution and make it observable by the machine. Such a feature had already been used by Miquel [16] to give a computational interpretation to the forcing translation.

Starting from those two observations, one can prove the following theorem.

Theorem 4 (KAM simulation)**.** *Intuitively, given a λ-term t, the term t_x produces all the stacks that appear in front of the head variable x in the reduction of $\langle (t, \sigma) \mid \pi \rangle$ (i.e. when the* Grab *rule is fired).*

Formally, let t be a λ-term, σ an KAM environment and π a KAM stack, such that (t, σ) is closed. Assume some variable x not free in σ nor in any closure of π. If $\langle (t, \sigma) \mid \pi \rangle \longrightarrow^* \langle (x, \tau) \mid \rho \rangle$ for some τ and ρ, then $\rho^\bullet \in t_x\, \{\sigma^\bullet\}\ \pi^\bullet$.

We now have the global picture of the dynamic contents of the Dialectica translation. The $(-)_x$ translation allows to capture the current stack each time the variable x reaches head position, that is, when it is evaluated in call-by-name. As this may occur several times, we have to return a multiset instead of only one stack. In addition, we also need the current stack to somehow delimit our returned stacks. The access to the current stack may be essential, because it contains terms that may in turn reach head position, taking control over the whole machine. This gives a very higher-orderish flavour to the Dialectica translation.

Fact 4. The substitution lemma can be revisited in a meaningful way. It tells us no more than that the accesses to the variable y in the term $t[x := r]$ can be classified in two kinds of accesses.

- The left-hand side of the union corresponds to the accesses to y in the term $t[x := r]$ itself. This was expected.
- The substitution may nonetheless have created new accesses to y: these are precisely described by the accesses to y in r when r is to be substituted by x when it comes in head position. But this is exactly what is constructed by the right-hand side term

  $$t_x[x := r^\bullet]\ \pi \gg\!= \lambda\rho.\, r_y\ \rho$$

  Indeed, the left argument of the bind constructs all accesses to x in t, that are then passed to the accessors of y created by r, which is the right argument of the bind.

Fact 5. We can revisit the historical presentation of the Dialectica translation, and compare it with our hybrid of Diller-Nahm and Curry-Howardesque translation. The main differences lies in the way the historical translation gets rid of the abstract multiset datastructure.

- Instead of returning an empty multiset, the historical translation choose a canonical placeholder materialized by the dummy term.
- When having to make the union of two multisets, the historical translation dynamically determines one the two terms to be picked, according to the decidable orthogonality.

The main issue with this process is that the selection occurring when picking one of the two terms is incorrect with respect to the operational semantics: it may choose a dummy term just because in the current context, it wins against the considered term, even though it is not a proper stack.

Fact 6. While the simulation theorem is particularly informative, one may already have realized that there is a huge discrepancy between the Dialectica translation and its KAM interpretation. The simulation theorem relates indeed reduction sequences of the KAM with *multisets* of stacks. This is were there is a catch.

- The KAM is fully sequential. With the reduction rules we provided, there is even only one possible reduction path. This means that the stacks we observe in the accesses to variables ought to be ordered with respect to the sequential order by which they appear along the reduction.
- The Dialectica is not sequential. The use of multisets, as free commutative monoids by excellence, forgoes any hope to recover some order relating it to the KAM.

One could naively use the natural candidate for free non-commutative monoids, namely, lists. Unluckily, this is not possible.

Proposition 3. *If we implement abstract multisets by standard lists, disregarding the required equivalence axioms, then the Dialectica translation does not preserve β-equivalence anymore.*

One can immediately remark that there is something really wrong here with respect to the translation, when using lists instead of multisets. The accesses of x in $t\ u$ can be indeed interleaved between t and u in the KAM. This has nothing to do with the Dialectica interpretation which sharply separates the two sources of accesses: the left side of the union corresponds to terms from t and the right side to those from u. No such interleaving is present in the translation.

This is actually even worse. No such interleaving is *possible* in the Dialectica translation. To cope with this sequentialization one would need to know the order of relative accesses to the free variables in the term. The Dialectica is totally oblivious of this issue, because all terms t_x are constructed in parallel for each variable x, regardless of the other variables. There is no further dependency on the free variables of the source term.

We conjecture that this phenomenon is actually a consequence of the linear factorization of the Dialectica translation, rather than a true defect of the translation itself. Linear logic is indeed essentially a model of commutative monads. Requiring the translation to respect the sequentiality of the KAM would break this commutativity.

There are another puzzling constatation to make from the simulation theorem. A remark we found enlightening is that we can see the Dialectica translation as a way to *count* things, i.e. to measure the complexity of a program. Indeed, by just considering the length of the multisets returned by the reverse translations, we can actually compute the number of dynamic accesses to x by a given term. This only would be already interesting, but there is more. It turns out that it is well-known that the Dialectica can count, but through two very different paths.

- First, quite an important part of Kohlenbach's book on proof mining [12] is dedicated to the study of majorizability and its implementation in the Dialectica translation.

  The purpose of this translation is to provide bounds on the growth of arithmetical functions, thus the inherent underlying counting approach.

- The second instance of a *déjà-vu* counting Dialectica can be found in the trend of recent work on the so-called *quantitative semantics*. This technique aims at extracting quantitative properties from denotational semantics, thanks to a clever use of linear logic. The paper [15] in particular designs such a semantics atop of a variation of coherence spaces.

  It is folklore that coherence spaces are a degenerate instance of the double-glueing construction, which is actually itself the generalization of De Paiva categories, from which our Dialectica proceeds.

We conjecture those two families of objects to be essentially the same, albeit seen through the prism of two distinct communities' practices and history.

7 Variants and Uses of Dialectica [Ch. 10]

We studied some variants of the Dialectica translation in various calling conventions.

Up to now, we only translated the purely negative fragment of the call-by-name λ-calculus, that is, our one and only type was the call-by-name arrow. Positive datatypes are often overlooked for various reasons. We believe that one of the main reasons of their mistreatment is that they often raise issues in call-by-name. Luckily, in our case, we already introduced positive datatypes in the target calculus, to be able to construct stacks. It is indeed a standard fact that stacks corresponding to some terms have the opposite polarity, so that our all-negative terms must have all-positive stacks. We give such a translation in Chapter 10.

The reverse translation of the positive fragment of call-by-name shows a repeating pattern: while introduction rules are essentially constructing a CPS translation of the future pattern-matching they will be applied to, elimination rules construct a two-part object. This CPS-like translation can be intuitively explained as follows. Let us consider for instance the term (t, u) and let us track when z is used in this pair. Unluckily, even in call-by-name, a pair is somehow a value, i.e. an object that would be inert if we put it the KAM together with an empty stack. Therefore, we need its surrounding context, which is the one deciding how to use the various components. And this context is necessarily a functional object, because it eventually boils down to some pattern-matching context of the form $\texttt{match} \cdot \texttt{with}\ (x, y) \mapsto r$. But the only information we need about this context is the way it uses x and y in r, that is, r_x and r_y, which is exactly what the translation observes. The KAM simulation theorem extends to this interpretation seamlessly.

The translation is also adapted into a classical-by-name and a call-by-value variants, which reinforce the CPS-look-alikeness of the translation. The classical variant is performed in the $\lambda\mu$-calculus, and shows that dually stack variables α have a rôle which is very similar to term variables, featuring a translation $(-)_\alpha$ that tracks the substitutions of those variables in the KAM by returning the corresponding term.

Thanks to our multiset-using reformulation of the Dialectica translation, as well as the computational intuition provided by the simulation theorem, we revisit the interpretation of the two additional semi-classical axioms featured by the historical Dialectica translation.

Apparently, the fact that the original Dialectica interprets the axiom of independence of premise is more due to chance rather than to a deliberate design choice. Indeed, it essentially comes from the fact that a lot of type interpretations are collapsed into the empty sequence, and that the realizability emerging from them comes uniquely from the meta-level logic and not from their trivial realizers, just as what happens in Kreisel's modified realizability. Our proof-theoretical presentation is much more computationally aware and cannot realize independence of premises.

The interpretation of Markov's principle is more interesting, as it critically relies on the reverse component of arrows. Furthermore, as seen through the multiset prism, it also requires the orthogonality to hold, allowing to guarantee that in precise cases, the multiset produced by the reverse component is not empty. In a nutshell, this is what allows to extract a witness from the reverse component of a proof of $\neg(\forall x.\, P\ x)$, by ensuring that the function must be fed with at least an argument which we observe as a stack, i.e. an inhabitant of $\exists x.\, \neg(P\ x)$. Recovering this argument is done dynamically, and in particular requires that the

multisets are finite. In our setting, we do not recover the full Markov's principle though, only a weak version of it. Indeed, we can only interpret a variant of this principle where negation $\neg A := A \to 0$ has been turned into $\sim A := A \to \bot$ where $\bot$ stands for an intuitionistic type similar to the corresponding type in **LL**, that is, a type which cannot be inhabited but which does not feature the *ex falso quod libet* principle either. The realizer of this principle shows a strong similarity with the one obtained using delimited continuations.

8 A Dependently-typed Dialectica [Ch. 11]

We showed that our version of the Dialectica translation accommodates quite well within a dependent framework, at least for the purely negative fragment. We can give indeed a translation of $\mathbf{CC}_\omega$ into itself extended with dependent pairs and abstract multisets. The $\mathbf{CC}_\omega$ system is quite expressive, and although it is predicative, the translation can accomodate easily impredicativity as well. For simplicity, we stick to a Curry-style presentation, otherwise we would require rather heavily annotated terms.

Definition 6 ($\mathbf{CC}_\omega$)**.** Terms of $\mathbf{CC}_\omega$ are inductively generated by the following grammar.

$$M, N, A, B := \Box_{i \in \mathbb{N}} \mid x \mid M\ N \mid \lambda x.\, M \mid \Pi x : A.\, B$$

The β-equivalence $\equiv_\beta$ is the contextual closure of the rule

$$(\lambda x.\, M)\ N \equiv_\beta M[x := N]$$

Environments are defined as ordered lists associating variables to terms, as usual, i.e.

$$\Gamma, \Delta := \cdot \mid \Gamma, x : A$$

The typing rules are mutually defined below.

$$\frac{}{\vdash_{\mathrm{wf}} \cdot} \qquad \frac{\Gamma \vdash A : \Box_i}{\vdash_{\mathrm{wf}} \Gamma, x : A} \qquad \frac{\Gamma \vdash M : B \qquad \Gamma \vdash A : \Box_i}{\Gamma, x : A \vdash M : B}$$

$$\frac{\Gamma \vdash A : \Box_i}{\Gamma, x : A \vdash x : A} \qquad \frac{\vdash_{\mathrm{wf}} \Gamma \qquad i < j}{\Gamma \vdash \Box_i : \Box_j} \qquad \frac{\Gamma \vdash A : \Box_i \qquad \Gamma, x : A \vdash B : \Box_j}{\Gamma \vdash \Pi x : A.\, B : \Box_{\max(i,j)}}$$

$$\frac{\Gamma, x : A \vdash M : B \qquad \Gamma \vdash \Pi x : A.\, B : \Box_i}{\Gamma \vdash \lambda x.\, M : \Pi x : A.\, B} \qquad \frac{\Gamma \vdash M : \Pi x : A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B[x := N]}$$

$$\frac{\Gamma \vdash M : B \qquad \Gamma \vdash A : \Box_i \qquad A \equiv_\beta B}{\Gamma \vdash M : A}$$

The target system is a slight extension to the previous one.

Definition 7 ($\mathbf{CC}_\omega^\times$)**.** We extend $\mathbf{CC}_\omega$ terms, reductions and typing rules as follows.

$$M, N, A, B := \ldots \mid \Sigma x : A.\, B \mid (M, N) \mid \texttt{let}\ (x, y) := M\ \texttt{in}\ N$$

$$(\texttt{let}\ (x, y) := (M, N)\ \texttt{in}\ P) \equiv_\beta P[x := M, y := N]$$

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

$$\frac{\Gamma \vdash \Sigma x : A.\, B : \Box_i \qquad \Gamma \vdash M : A \qquad \Gamma \vdash N : B[x := M]}{\Gamma \vdash (M, N) : \Sigma x : A.\, B}$$

$$\frac{\Gamma \vdash M : \Sigma x : A.\, B \qquad \Gamma, x : A, y : B \vdash N : C[z := (x, y)] \qquad z \text{ fresh}}{\Gamma \vdash \texttt{let}\ (x, y) := M\ \texttt{in}\ N : C[z := M]}$$

As for the non-dependent translation, we also need to add multisets. While pairs needed to be made dependent, this is not the case for multisets. We simply adapt our definitions.

Definition 8 (Abstract multisets)**.** We assume that $\mathbf{CC}_\omega^\times$ features the following structure:

$$M, N, A, B := \ldots \mid \mathfrak{M}\, A \mid \{M\} \mid M \gg\!= N \mid M \odot N \mid \emptyset$$

together with the following typing rules:

$$\frac{\Gamma \vdash A : \square_i}{\Gamma \vdash \mathfrak{M}\, A : \square_i} \qquad \frac{\Gamma \vdash A : \square_i}{\Gamma \vdash \emptyset : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash M : A}{\Gamma \vdash \{M\} : \mathfrak{M}\, A} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash N : A}{\Gamma \vdash M \odot N : \mathfrak{M}\, A}$$

$$\frac{\Gamma \vdash M : \mathfrak{M}\, A \qquad \Gamma \vdash N : A \to \mathfrak{M}\, B \qquad \Gamma \vdash A \to \mathfrak{M}\, B : \square_i}{\Gamma \vdash M \gg\!= N : \mathfrak{M}\, B}$$

as well as some reduction rules adapted from the non-dependent case not described here.

We give here the principles underlying what a dependent Dialectica translation is. This can be summarized into the two main additions from the simply-typed λ-calculus into $\mathbf{CC}_\omega$:

- The shift from a simple arrow $A \to B$ to a dependent product $\Pi x : A.\, B$;
- The need to handle properly types as terms.

The handling of the dependent arrow turns out to be almost immediate, actually. Indeed, by reordering the two arguments in the reverse proof of the arrow, and making the arrows and the product over $\mathbb{W}(A)$ dependent, we can pose

$$\begin{aligned} \mathbb{W}(\Pi x : A.\, B) &:= \begin{cases} \Pi x : \mathbb{W}(A).\, \mathbb{W}(B) \\ \qquad \times \\ \Pi x : \mathbb{W}(A).\, \mathbb{C}(B) \to \mathfrak{M}\, \mathbb{C}(A) \end{cases} \\ \mathbb{C}(\Pi x : A.\, B) &:= \Sigma x : \mathbb{W}(A).\, \mathbb{C}(B) \end{aligned}$$

such that all translations of B have an $x : \mathbb{W}(A)$ in their scope.

The translation of types as term may seem more difficult at first sight, but it is easy as well. We can simply identify the translation of types seen as *terms* with the pair of their $\mathbb{W}(-)$ and $\mathbb{C}(-)$ type translations. Hence, the type translations can be reduced to macros, typically by enforcing

$$A^\bullet := (\mathbb{W}(A), \mathbb{C}(A))$$

for any type A. This allows to *define* $\mathbb{W}(-)$ and $\mathbb{C}(-)$ from $(-)^\bullet$ rather than as an external translation. Therefore, we simply pose

$$\begin{aligned} \mathbb{W}(A) &:= \mathrm{fst}\ A^\bullet \\ \mathbb{C}(A) &:= \mathrm{snd}\ A^\bullet \end{aligned}$$

and we are done.

While we actually already defined $(\Pi x : A.\, B)^\bullet$ in the previous paragraph, one can wonder what the translation $\square_i{}^\bullet$ could be. From what we just said, looking at the typing rule $\square_i : \square_j$ for $i < j$, we have the constraints

$$\begin{aligned} \square_i{}^\bullet &: \mathrm{fst}\ \square_j{}^\bullet \\ \mathrm{snd}\ \square_i{}^\bullet &: \square_k \end{aligned}$$

for some k. As we will see, the practical choice for $\mathrm{snd}\ \square_i^\bullet$ is essentially irrelevant, as long as it is a type. This is why we simply pose

$$\square_i^\bullet \quad := \quad (\square_i \times \square_i, \square_i)$$

which satisfies the constraints.

Finally, we did not talk at all about the reverse translation for types. There does not seem to be a lot of design space for it. The simplest choice to do is to deny any computational content to types, that is, we morally enforce

$$A_x := \lambda\pi.\, \emptyset$$

when A is a closed type.

We summarize all of our above rationale in one definition.

Definition 9 (Dependent Dialectica)**.** We mutually define the translations $(-)^\bullet$ and $(-)_x$ by induction below, where $\mathbb{W}(M) := \mathrm{fst}\ M^\bullet$ and $\mathbb{C}(M) := \mathrm{snd}\ M^\bullet$.

$$\begin{array}{lcl}
x^\bullet & := & x \\
(\lambda x.\, M)^\bullet & := & (\lambda x.\, M^\bullet, \lambda x\, \pi.\, M_x\ \pi) \\
(M\ N)^\bullet & := & \mathrm{fst}\ M^\bullet\ N^\bullet \\
(\Pi x : A.\, B)^\bullet & := & \left(\begin{pmatrix} \Pi x : \mathbb{W}(A).\, \mathbb{W}(B) \\ \times \\ \Pi x : \mathbb{W}(A).\, \mathbb{C}(B) \to \mathfrak{M}\, \mathbb{C}(A) \end{pmatrix}, \begin{pmatrix} \\ \Sigma x : \mathbb{W}(A).\, \mathbb{C}(B) \\ \\ \end{pmatrix} \right) \\
(\square_i)^\bullet & := & (\square_i \times \square_i, \square_i) \\
x_x & := & \lambda\pi.\, \{\pi\} \\
x_y & := & \lambda\pi.\, \emptyset \\
(\lambda y.\, M)_x & := & \lambda(y, \pi).\, t_x\ \pi \\
(M\ N)_x & := & \lambda\pi.\, (\mathrm{snd}\ M^\bullet\ N^\bullet\ \pi \gg\!= \lambda\rho.\, N_x\ \rho) \odot (M_x\ (N^\bullet, \pi)) \\
(\Pi x : A.\, B)_x & := & \lambda\pi.\, \emptyset \\
(\square_i)_x & := & \lambda\pi.\, \emptyset
\end{array}$$

This is almost the same as the simply-typed translation, except for the commutations explained before, and the special handling of types. The following results are similar to their simply-typed counterparts and show that the Dialectica translation naturally interprets dependent type theory, assuming the abstract multisets are computational enough.

Theorem 5 (Computational soundness)**.** *If $M \equiv_\beta N$ then $M^\bullet \equiv_\beta N^\bullet$ and $M_x \equiv_\beta N_x$ for any variable x.*

Theorem 6 (Typing soundness)**.** *Assume $\Gamma \vdash M : A$, then*

$$\mathbb{W}(\Gamma) \vdash M^\bullet : \mathbb{W}(A)$$

$$\mathbb{W}(\Gamma) \vdash M_x : \mathbb{C}(A) \to \mathfrak{M}\, \mathbb{C}(U)$$

when $(x : U) \in \Gamma$ and where $\mathbb{W}(\Gamma)$ stands for the pointwise application of $\mathbb{W}(-)$ to Γ.

All the magic comes from the fact that our translation preserves β-equivalence, allowing us to make the conversion rule work unmodified. This can be an issue.

First, being a quotient, it is difficult to encode it without losing canonicity of the representation. We could think of it as an opaque datatype with built-in equivalence, but that would require a dedicated system.

Furthermore, we actually do not even know if the equivalence of two well-typed terms is decidable. If this were not the case, it would put a threat upon the feasibility of our translation implementationwise.

Alternatively, our finite multisets are naturally encoded using HITs. This would not give us the definitional equations we hope for though, as they only allow for additional propositional equalities. To work around this, we can probably use extensionality encodings, as described in [18], by making conversion rules in the term relevant, replacing them with a rewriting over a propositional equality constructed from the fact that the source terms are β-equivalent.

The interpretation of dependent elimination has been shown to be troublesome in the thesis, but it finally turned out that it was achievable by making the counter type dependent in a witness argument, a solution hinted at in the thesis as well.

9 Beyond Dialectica [Ch. 12]

The study of the Dialectica translation in this thesis has been particularly influenced by the work of Krivine [14] and Miquel [16] on forcing in the realm of classical realizability, as well as its application to dependent type theory due to Jaber et al. [10]. Seen as a program translation, it is essentially the same as the Kripke model construction [13]. The underlying translation at the level of terms is rather dull, as it is a monotonous variant of the usual reader monad described.

This chapter explores the strong kinship linking forcing, the Lafont-Streicher-Reus CPS and the Dialectica translation. It does so by providing a series of pairwise related translations factorizing through a linear decomposition.

1. The usual reader monad.
2. The forcing translation.
3. A stack reading translation, that changes dynamically the type of the readable cell to match the type of stacks corresponding to the current term.
4. An intuitionistic CPS close to the Lafont-Streicher-Reus one. The main difference comes from the fact that all boxed terms are delimited w.r.t. their return type.

Each of these translation comes with a study of their typing and computation preservation, as well as some other unrelated results. The main results are the following.

- The forcing translation as found in the literature is call-by-value, and as such does not preserve all β-reductions, only those whose substituend is a value. This is the reason why the interpretation of the conversion rule in the adaptation of forcing to type theory is so troublesome [10]. This result actually led to a better behaved forcing translation, published at LICS 2016.
- The intuitionistic CPS is complete for intuitionistic provability, and the proof of this fact actually amounts to an internal normalization-by-evaluation process. This is close to what the Kripke model does, except that the proof is entirely intuitionistic. This result shows that there is a strong link between the two structures, and also hints at stronger variants of classical realizability.
- Although not a proper result per se, the path going from the reader monad to the intuitionistic CPS gives important hindsight into the Dialectica translation, and in particular how to reimplement it in a different way.

10 Logical-by-Need [Ch. 6]

This chapter is somehow distinct from the remaining of the thesis and features a presentation of the so-called *call-by-need* calling convention based on considerations stemming from linear logic. Contrarily to historical presentations of the family of call-by-need calculi, the one we describe there is more uniform and shares strong bounds with other computation systems, most notably the KAM. The results from this chapter have been published at ESOP 2016.

The starting point is the observation that the linear head reduction, designed by Danos and Regnier at the beginning of the 90's gives an elegant way to obtain the demand-driven mechanism at the root of the various call-by-need reductions.

We contribute to the theory of linear head reduction and show that it can be made into a calculus. Doing so allows us to formally connect linear hear reduction to call-by-need, showing how to systematically derive well-known call-by-need calculi from linear head reduction. More precisely, we will justify the following motto.

Call-by-need	$\equiv$	Demand-driven comp. *(weak linear head reduction)*	$+$	Memoization *(by value)*	$+$	Sharing *(closure sharing)*

Our results are validated in two ways: first, the resulting calculi correspond to well-known call-by-need calculi, providing a validation to Chang-Felleisen recent single-axiom call-by-need calculus [2] with one axiom. Second, we extend our results to the classical case, defining linear head reduction for $\lambda\mu$-calculus and deriving from it a call-by-need $\lambda\mu$-calculus.

The central tool to turn LHR into a calculus are the closure contexts, that internalize Danos-Regnier σ-equivalence.

Definition 10 (Closure contexts)**.** Closure contexts are inductively defined as:

$$\mathcal{C} := [\cdot] \mid \mathcal{C}_1[\lambda x.\, \mathcal{C}_2]\ t$$

Although not totally new, as they were present in Chang-Felleisen revisitation of call-by-need [2], our work highlights their utmost importance. In particular, we show that they are precisely the λ-calculus embodiement of the delayed substitutions from the KAM.

Proposition 4. *Let t be a term, σ an environment, π a stack and $\mathcal{C}$ a closure context. We have the following reduction*

$$\langle (\mathcal{C}[t], \sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push}, \textsc{Pop}} \langle (t, \sigma + [\mathcal{C}]_\sigma) \mid \pi \rangle$$

where $[\mathcal{C}]_\sigma$ is defined by induction over $\mathcal{C}$ as follows:

$$[[\cdot]]_\sigma \equiv \emptyset \qquad [\mathcal{C}_1[\lambda x.\, \mathcal{C}_2]\ t]_\sigma \equiv [\mathcal{C}_1]_\sigma + (x := (t, \sigma)) + [\mathcal{C}_2]_{\sigma + [\mathcal{C}_1]_\sigma + (x := (t,\sigma))}$$

Conversely, for all t_0 and σ_0 such that

$$\langle (t, \sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push}, \textsc{Pop}} \langle (t_0, \sigma_0) \mid \pi \rangle$$

there exists $\mathcal{C}_0$ such that $t = \mathcal{C}_0[t_0]$, where $\mathcal{C}_0$ is inductively defined over σ_0.

We provide an alternative and more conventional definition for the linear head reduction.

Definition 11 (λ_{lh}-calculus)**.** The λ_{lh}-calculus is defined by the reduction rule:

$$L_1[\mathcal{C}[\lambda x.\, L_2[x]]\ u] \to_{\lambda_{lh}} L_1[\mathcal{C}[\lambda x.\, L_2[u]]\ u]$$

where L_1, L_2 are left contexts, $\mathcal{C}$ is a closure context, t and u are λ-terms, with the usual freshness conditions to prevent variable capture in u.

Theorem 7. *The λ_{lh}-calculus captures the historical presentation of LHR by Danos and Regnier.*

Next, we provide a call-by-need calculus whose elaboration is based on three systematic steps from the linear head reduction calculus.

1. The restriction to a weak reduction.
2. The restriction of substitution to values up-to-closure-contexts.
3. The introduction of sharing of closure contexts.

Theorem 8. *The resulting calculus is essentially Chang-Felleisen by-need calculus.*

We show that our modernized LHR can be easily adapted to the classical case by giving a variant in the $\lambda\mu$-calculus. The soundness of this approach is witnessed by the following theorem.

Theorem 9. *Up to a restriction to weak reduction, the classical LHR has the same intentional content as the KAM, as it shares the same substitution sequence.*

Finally, by applying the same three steps as we did in the intuitionistic case, we design a call-by-need $\lambda\mu$-calculus. We show that the resulting calculus is close to another classical call-by-need calculus, Ariola-Herbelin-Saurin presentation of call-by-need in the setting of the $\bar{\lambda}\mu\tilde{\mu}$-calculus. We base upon their adaptation of this calculus in the usual $\lambda\mu$-calculus. For technical reasons, we need to modify a bit the latter by replacing its destructive substitution rule by a linear one, leading to the definition of AHS' calculus.

Theorem 10. *A $\lambda\mu$-term in our classical-by-need calculus normalizes iff it normalizes in AHS'-calculus.*

References

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

[2] Stephen Chang and Matthias Felleisen. "The Call-by-need Lambda Calculus, Revisited". In: *European Symposium on Programming, ESOP 2012*. Lecture Notes in Computer Science. Springer, 2012.

[3] Justus Diller. "Eine Variante zur Dialectica-Interpretation der Heyting-Arithmetik endlicher Typen". German. In: *Archiv für mathematische Logik und Grundlagenforschung* 16.1-2 (1974), pp. 49–66. ISSN: 0003-9268. DOI: 10.1007/BF02025118. URL: http://dx.doi.org/10.1007/BF02025118.

[4] K. Gödel. "Zur intuitionistischen Arithmetik und Zahlentheorie". In: *Ergebnisse eines mathematisches Kolloquiums* 4 (1932), pp. 34–38.

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

[6] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme". In: *Monatshefte für Mathematik und Physik* 38.1 (1931), pp. 173–198.

[7] Timothy G. Griffin. "A Formulae-as-type Notion of Control". In: POPL '90. San Francisco, California, USA: ACM, 1990, pp. 47–58. ISBN: 0-89791-343-4.

[8] Hugo Herbelin. "An Intuitionistic Logic that Proves Markov's Principle". In: *Logic in Computer Science, Symposium on* (2010), pp. 50–56. ISSN: 1043-6871. DOI: http://doi.ieeecomputersociety.org/10.1109/LICS.2010.49.

[9] J. M. E. Hyland. "Proof theory in the abstract". In: *Ann. Pure Appl. Logic* 114.1-3 (2002), pp. 43–78.

[10] Guilhem Jaber, Nicolas Tabareau, and Matthieu Sozeau. "Extending Type Theory with Forcing". In: *LICS 2012 : Logic In Computer Science*. Dubrovnik, Croatia, June 2012, pp. –.

[11] Guilhem Jaber et al. “The Definitional Side of the Forcing”. In: *LICS 2016*. 2016.

[12] Ulrich Kohlenbach. *Applied Proof Theory: Proof Interpretations and their Use in Mathematics*. Springer Monographs in Mathematics. Springer Verlag, 2008.

[13] Saul Kripke. “A Completeness Theorem in Modal Logic”. In: *J. Symb. Log.* 24.1 (1959), pp. 1–14. DOI: 10.2307/2964568. URL: http://dx.doi.org/10.2307/2964568.

[14] Jean-Louis Krivine. “Realizability algebras: a program to well order R”. In: *Logical Methods in Computer Science* 7.3 (2011). DOI: 10.2168/LMCS-7(3:2)2011. URL: http://dx.doi.org/10.2168/LMCS-7(3:2)2011.

[15] J. Laird et al. “Weighted relational models of typed lambda-calculi”. In: *28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2013), 25-28 June 2013, New Orleans, USA, Proceedings*. 2013, pp. 301–310.

[16] Alexandre Miquel. “Forcing as a Program Transformation”. In: *LICS*. IEEE Computer Society, 2011, pp. 197–206. ISBN: 978-0-7695-4412-0.

[17] Paulo Oliva. “Unifying Functional Interpretations”. In: *Notre Dame Journal of Formal Logic* 47.2 (Apr. 2006), pp. 263–290. DOI: 10.1305/ndjfl/1153858651. URL: http://dx.doi.org/10.1305/ndjfl/1153858651.

[18] Nicolas Oury. “Extensionality in the Calculus of Constructions”. English. In: *Theorem Proving in Higher Order Logics*. Ed. by Joe Hurd and Tom Melham. Vol. 3603. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 278–293. ISBN: 978-3-540-28372-0. DOI: 10.1007/11541868_18. URL: http://dx.doi.org/10.1007/11541868_18.

[19] Valeria de Paiva. “A Dialectica-like Model of Linear Logic”. In: *Category Theory and Computer Science*. Ed. by David H. Pitt et al. Vol. 389. Lecture Notes in Computer Science. Springer, 1989, pp. 341–356.

[20] Valeria de Paiva. “The Dialectica Categories”. In: *Categories in Computer Science and Logic: Proc. of the Joint Summer Research Conference*. Ed. by J. W. Gray and A. Scedrov. Providence, RI: American Mathematical Society, 1989, pp. 47–62.

[21] Pierre-Marie Pédrot. “A Functional Functional Interpretation”. In: *Proceedings of the 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: ACM, 2014, 77:1–77:10. ISBN: 978-1-4503-2886-9. DOI: 10.1145/2603088.2603094. URL: http://doi.acm.org/10.1145/2603088.2603094.

[22] Pierre-Marie Pédrot and Alexis Saurin. “Classical By-Need”. In: *ESOP 2016*. 2016, pp. 616–643. DOI: 10.1007/978-3-662-49498-1_24.