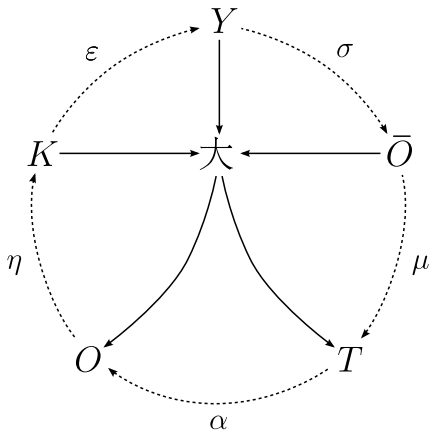
On the Semantics of the Effect Calculus

Pierre-Marie Pédrot

ENS of Lyon

M1 Research Internship: March–May 2010



Kyōto University, RIMS

Supervisor: Masahito Hasawa (RIMS)

Acknowledgements

Firstly, I would like to bow deeply to Hassei, for having accepted to supervise me during this internship, for having provided me with a bunch of interesting material about semantics and category theory and for his general and continuous support.

I definitely need to thank Étienne Duchesne for his kindness and for having been able to stand my whines alternatively about the deadly heat, the equatorial rain and the collapse of the yen-euro exchange rate.

I must also thank every member of the computer science team, in particular Hasuo-san for his helpful social skills and Asada-san for his vast knowledge about Grothendieck's crave for kimchi.

Last but not least, I would like to thank people who supported and amused me through electronic means, including but not limited to: the extravagant *Ours en Jetpack* clanic list, the perfidious Turkey Whisperess, the prolix M1IF list and the cheese-and-perfume supplying `tous.ens` list.

Introduction

The λ-calculus is amongst the most famous models of computation and is widely used throughout the study of programming languages. Yet it basically lacks of many features that can be found in practical languages, one of the most obvious being imperative constructs, that is, state-based programs.

Following the work of Moggi[10], it is now common practice to encode those non-intuitionistic (often called unpure) features using monads. Monads are a very generic pattern, and even though they stem from category theory, they do have practical interest: compare with their use in Haskell for example.

One may also have experimentally witnessed that some monads displayed remarkable properties as long as they were somehow *well-behaved*. Actually, a frequent case is the one of *linear* monads, that is, restricting the type system to some flavour of linear logic.

The most famous examples are the linearly-used continuation-passing style monad and the linearly-used state monad. Given a return type R and a state type S, these are:

	Intuitionistic monad	Linear monad
CPS	$(A \to R) \to R$	$(!A \multimap R) \multimap R$
State	$S \to (A \times S)$	$S \multimap (!A \otimes S)$

This linearity is a very deep property. Indeed, imperative languages do forbid both the duplication and the erasure of their current state. On the other hand, as long as a functional language does not have `call/cc`-like control primitives, its CPS translation is also linear.

Stepping further into linear logic, the nature of this tight relationship can be explicitely dug out. Those two constructions are dual, as the following isomorphism holds in classical linear logic:

$$(!A \multimap R) \multimap R \quad \cong \quad R^{\perp} \multimap (!A \otimes R^{\perp})$$

This duality between linear state and CPS had already been accidentally observed as the formal equivalence between SSA-form and CPS-form[7], though the requirement of linearity was awkwardly formulated and not made explicit.

Simpson & *al.* proposed a calculus[2], the extended effect calculus (EEC) designed to deal with linearly used effects, and particularly with those two monads[3]. The EEC was itself inspired by Levy's CBPV[8].

During this internship, we studied the categorical foundations of such calculus. We built a refinement of the EEC, the multiplicative effect calculus (MEC), both on syntactical and semantical grounds. As for syntax, it makes the CPS-state duality obvious; on the semantical side, it generalizes the models of EEC in order to get rid of technical impediments.

1 Effects in a Nutshell

Basically, effects are any non-intuitionistic features of programming languages. The main problem with effects is that they break the equational theory of λ-caculus. Surprisingly enough, many common constructions are actually non-intuitionistic. Very famous examples include imperative state, parallelism, non-determinism, control, etc.

One of the basic frameworks for expressing semantics of programs is category theory. Because of the lack of space, we won't recall every definition, and we will be using standard notations. For a general overview of category theory, the interested reader may refer to MacLane's book[9]. We only present here the semantic tools that will be needed later on.

1.1 Monads

In his pioneer work, Moggi[10] showed that many effects could be captured in λ-calculus through the use of monads.

Definition 1.1. A monad over a category $\mathbf{C}$ is a triple (T, η, μ) where $T : \mathbf{C} \to \mathbf{C}$ is a functor, and $\eta : \mathbb{1}_{\mathbf{C}} \to T$ and $\mu : T^2 \to T$ are two natural transformations such that the following diagrams commute for all A:

$$\begin{array}{ccc} T^3A & \xrightarrow{T\mu_A} & T^2A \\ {\scriptstyle \mu_{TA}}\downarrow & & \downarrow{\scriptstyle \mu_A} \\ T^2A & \xrightarrow{\mu_A} & TA \end{array} \qquad \begin{array}{ccccc} TA & \xrightarrow{\eta_{TA}} & T^2A & \xleftarrow{T\eta_A} & TA \\ & \searrow^{=} & \downarrow{\scriptstyle \mu_A} & \swarrow_{=} & \\ & & TA & & \end{array}$$

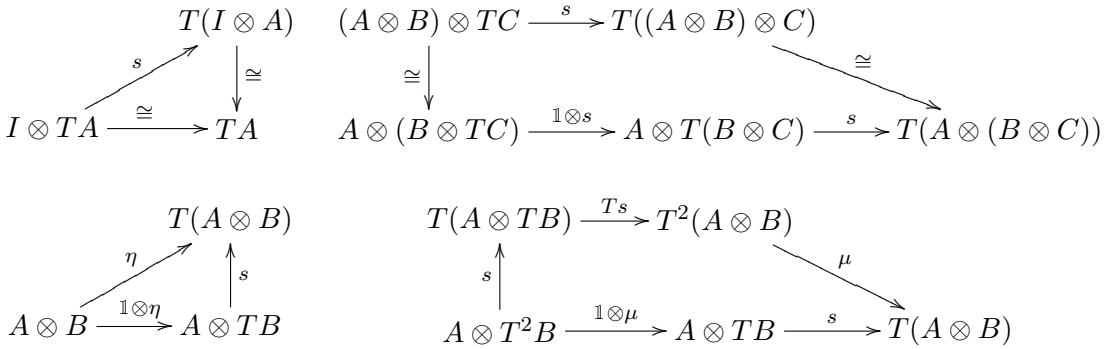
Monads are the categorical counterpart of closure operators from set theory. The first diagram states that closure is associative, and the second one is a generalization of idempotence of closure operators.

Example. We give here some examples of computationally interesting monads in **Set**:

Partiality	$TA = A + \{\bot\}$	$\eta_A(x) = \mathtt{inj}(x)$	$\mu_A(\mathtt{inj}(\mathtt{inj}(x)) = \mathtt{inj}(x)$ $\mu_A(\mathtt{inj}(\bot)) = \bot$
State	$TA = S \to A \times S$	$\eta_A(x) = \lambda s.(x, s)$	$\mu_A(m) = \lambda s.\mathtt{let}\ (m', s') = m(s)\ \mathtt{in}\ m'(s')$
Non-determinism	$TA = \mathcal{P}(A)$	$\eta_A(x) = \{x\}$	$\mu_A(m) = \bigcup_{x \in m} x$

General monads are usually not enough to model λ-calculus, for they lack a coherence morphism to model typing environments as finite products. One needs to add a tensorial *strength* to avoid this. Most often, the tensor is actually a cartesian product, but this has no effect whatsoever on the following definition.

Definition 1.2. A strong monad over a monoidal category $(\mathbf{C}, \otimes, I)$ is a monad (T, η, μ) together with a natural transformation $s_{A,B} : A \otimes TB \to T(A \otimes B)$ that verifies the following coherence requirements, where $\cong$ stands for the canonical isomorphisms:



1.2 Canonical adjunctions

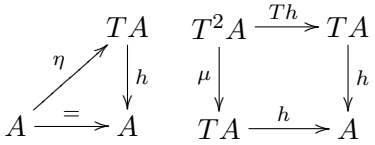
Given a λ-calculus extended with a strong monad T, the interpretation in usually done through the decomposition of T into two adjoint functors. Any monad gives rise to two canonical adjunctions[1]: through the Kleisli category $\mathbf{C}_T$ which corresponds to a call-by-value reduction, and through the Eilenberg-Moore category $\mathbf{C}^T$, which corresponds to a call-by-name reduction.

Definition 1.3. For any monad (T, η, μ) over $\mathbf{C}$, the Kleisli category $\mathbf{C}_T$ is defined as follow:

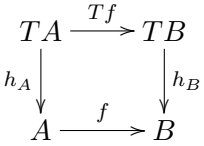
$$\begin{array}{ll}
\mathrm{Obj}(\mathbf{C}_T) \equiv \mathrm{Obj}(\mathbf{C}) & \\
\mathbf{C}_T(A, B) \equiv \mathbf{C}(A, TB) & \text{for } A, B \in \mathrm{Obj}(\mathbf{C}) \\
\mathbb{1}_A \equiv \eta_A : A \to TA & \text{for } A \in \mathrm{Obj}(\mathbf{C}) \\
f; g \equiv f; Tg; \mu_C : A \to TC & \text{for } f : A \to TB \text{ and } g : B \to TC
\end{array}$$

This also defines a straightforward adjunction $F_T \dashv G_T : \mathbf{C}_T \to \mathbf{C}$.

Definition 1.4. Given a monad (T, η, μ) over $\mathbf{C}$, a T-algebra is a pair (A, h) where $A \in \mathrm{Obj}(\mathbf{C})$ and $h : TA \to A$ such that the following diagrams commute:



A morphism of T-algebras $(A, h_A) \to (B, h_B)$ is a morphism $f : A \to B$ such that:



The Eilenberg-Moore category $\mathbf{C}^T$ of T is the category whose objects are T-algebras and morphisms are morphisms of T-algebras. There exists an adjunction $F^T \dashv G^T : \mathbf{C}^T \to \mathbf{C}$ where $G^T : \mathbf{C}^T \to \mathbf{C}$ is the forgetful functor and $F^T : \mathbf{C} \to \mathbf{C}^T$ is the free algebra functor $A \mapsto (TA, \mu_A)$.

[1] They are extremal in a bicategorical sense.

2 The Multiplicative Effect Calculus

2.1 Motivations

The Multiplicative Effect Calculus is a coherent fragment of the Extended Effect Calculus designed by Møgelberg and Simpson[2]. Initially, the EEC was designed as an extension of Levy's Call-by-Push-Value[8]. CBPV was a subsuming paradigm for CBV and CBN effects, yet it suffered from its categorical models which were defined in terms so-called *staggered* categories.

Staggered categories can be understood as an ugly semantical hack to overcome limitations of CBPV, namely, the lack of computation function space. EEC fixed this point by providing a nice categorical model which did have this function space. EEC models also feature interesting adjunctions which are neither Kleisli nor Eilenberg-Moore, hence underlying the indifference towards the traditional CBV–CBN schism.

EEC is also interesting because it expresses computations through linear logic formulas, and particularly the two aforementioned pervasive monads: the linear-state monad $\underline{S} \multimap !A \otimes \underline{S}$ and the linear-continuation monad $(A \to \underline{R}) \multimap \underline{R}$. To achieve this, EEC distinguishes values from computations and features three special type constructors:

1. The linear arrow $\multimap$ which corresponds to the missing computation function space of CBPV.
2. A tensor $\otimes$ which acts as an asymmetrical product.
3. A cotensor $\to$ which acts as an asymmetrical power.

Although EEC has rather nifty categorical semantics, it also suffers from several design flaws. First, its syntax is inspired by Moggi's computational call-by-value λ_c-calculus. This definitely hides away EEC's agnosticism on reduction strategy. Some unlucky presentations inherited from λ_c-calculus also cripple important duality results with syntactic burden. Finally, even its categorical models are too rich and hinder the comprehension of EEC.

In this section, we propose a refinement of EEC, the so-called Multiplicative Effect Calculus. Its (categorical) semantics is close, though its syntax is brand new.

2.2 Types

For the sake of clarity, we pursue and extend the use of linear logic notations. We will note value types $A, B \ldots$ as capital letters and computation types $\underline{A}, \underline{B} \ldots$ as underlined capital letters. We suppose given some base value types α and computation types $\underline{\alpha}$. Types are defined in figure 1.

$$A ::= \alpha \mid A \to B \mid \underline{A} \multimap \underline{B}$$
$$\underline{A} ::= \underline{\alpha} \mid ?A^{\perp} \parr \underline{B} \mid !A \otimes \underline{B}$$

Table 1: MEC types

Note that the tensor and the par are asymmetrical, as they take a value and a computation as arguments. As in the original EEC, computation abstractions must live as value types, thus we insist on the fact that there is no closed structure underneath. We also drop the $!A$ type from EEC in order to gain additional symmetry.

In MEC, computations are strictly distinct from value. We argue that the inclusion of computations into values in the original EEC was a legacy from CBPV, which lacked computation function space, hence bypassing this restriction through the use of an adjunction together with a null computation. More on this in section 4.4.

2.3 Terms

The MEC has three types of abstractions: the intuitionistic one, and two mixed abstractions which are dual to each other. Every abstraction comes with its own application. Additionally, we also have the null process ρ and the composition of processes $t; u$. The name choice for the two binders κ and ζ will be revealed later.

$$t ::= x \mid \rho \mid \lambda x.t \mid t(u) \mid \kappa x.t \mid t\langle u\rangle \mid \zeta x.t \mid t[u] \mid t; u$$

Table 2: MEC terms

2.4 Type system

The MEC has two kinds of typing judgments, one for values and one for computations. They are respectively of the form $\Gamma \vdash t : A$ and $\Gamma \mid \underline{A} \vdash t : \underline{B}$, where Γ is an intuitionistic environment, that is, an unordered list of variable bindings of the form $(x : A)$. The linear environment of the computation typing judgment is required to contain exactly one type (hence the linearity), and the bound computation variable is implicit. The typing derivations are given in table 3. It is worth noting that $\Gamma \mid \underline{A} \vdash t : \underline{B}$ can be just considered as a handy notation for $\Gamma \vdash t : \underline{A} \multimap \underline{B}$.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \mid \underline{A} \vdash \rho : \underline{A}}$$

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

$$\frac{\Gamma, x : A \mid \underline{C} \vdash t : B}{\Gamma \mid \underline{C} \vdash \zeta x.t : ?A^{\perp} \parr \underline{B}} \qquad \frac{\Gamma \mid \underline{C} \vdash t : ?A^{\perp} \parr \underline{B} \qquad \Gamma \vdash u : A}{\Gamma \mid \underline{C} \vdash t[u] : \underline{B}}$$

$$\frac{\Gamma, x : A \mid \underline{B} \vdash t : \underline{C}}{\Gamma \mid !A \otimes \underline{B} \vdash \kappa x.t : \underline{C}} \qquad \frac{\Gamma \mid !A \otimes \underline{B} \vdash t : \underline{C} \qquad \Gamma \vdash u : A}{\Gamma \mid \underline{B} \vdash t\langle u\rangle : \underline{C}}$$

$$\frac{\Gamma \mid \underline{A} \vdash t : \underline{B}}{\Gamma \vdash t : \underline{A} \multimap \underline{B}} \qquad \frac{\Gamma \vdash t : \underline{A} \multimap \underline{B}}{\Gamma \mid \underline{A} \vdash t : \underline{B}}$$

$$\frac{\Gamma \mid \underline{A} \vdash t : \underline{B} \qquad \Gamma \mid \underline{B} \vdash u : \underline{C}}{\Gamma \mid \underline{A} \vdash t; u : \underline{C}}$$

Table 3: Typing rules of MEC

One can see that the mixed abstractions share a dual typing rule: the $\kappa x.t/t\langle u\rangle$ pair acts on the argument type of the typing derivation while the $\zeta x.t/t[u]$ pair acts on the return type.

Intuitively, this typing system permits to have a sharp distinction between values and computations. Whereas values are first-class citizens, one can only manipulate computations through blackboxes given as computation transformers $\underline{A} \multimap \underline{B}$, hence ensuring some form of linearity.

Computations do not exist by themselves and are only dynamic entities. It follows that the null process ρ can be seen as the function that returns immediately, while the composition $t; u$, as its name suggests, is just sequential composition of two processes t and u.

Note also that there is no canonical way to retrieve values from computations, e.g. there is no closed term which has type $(\underline{S} \multimap !A \otimes \underline{S}) \to A$.

2.5 Operational semantics

The definition of MEC operational semantics requires a rather tricky equivalence relation[2] on terms which turns out to be obvious when one thinks in terms of an abstract machine. We define the relation $\equiv$ as the smallest subterm-closed congruence verifying the equations in table 4 (and compatible with α-conversion).

$t; (u; v) \equiv (t; u); v$		(Assoc)
$\kappa x.(t; u) \equiv \kappa x.t; u$	$x \notin u$	(AbsL-Ctx)
$\zeta x.(t; u) \equiv t; \zeta x.u$	$x \notin t$	(AbsR-Ctx)
$(t; u)\langle v\rangle \equiv t\langle v\rangle; u$		(AppL-Ctx)
$(t; u)[v] \equiv t; u[v]$		(AppR-Ctx)
$\zeta x.\kappa y.t \equiv \kappa y.\zeta x.t$	$x \neq y$	(Abs-Comm)
$(\zeta x.t)\langle u\rangle \equiv \zeta x.t\langle u\rangle$	$x \notin u$	(LR-Comm)
$(\kappa x.t)[u] \equiv \kappa x.t[u]$	$x \notin u$	(RL-Comm)
$t\langle v\rangle[u] \equiv t[u]\langle v\rangle$		(App-Comm)

Table 4: Congruence definition

The operational semantics is only given by three β-reductions together with their respective η-expansions, and two identity reductions in table 5. The reduction relation $\to$ is then the closure of these reductions up to congruence and subterm compatibility. We shall note $t = u$ if these two terms are $\beta\eta$-equivalent.

$$(\lambda x.t)(u) \to_\beta t\{x \leftarrow u\} \qquad \lambda x.t(x) =_\eta t \quad x \notin t$$
$$(\zeta x.t)[u] \to_\beta t\{x \leftarrow u\} \qquad \zeta x.t[x] =_\eta t \quad x \notin t$$
$$(\kappa x.t)\langle u\rangle \to_\beta t\{x \leftarrow u\} \qquad \kappa x.t\langle x\rangle =_\eta t \quad x \notin t$$

$$t; \rho \to t \qquad \rho; t \to t$$

Table 5: Core reductions of MEC

Lemma 2.1. *The MEC enjoys the subject reduction property.*

Lemma 2.2. *The reduction relation is locally confluent.*

[2]Which bears a striking similarity with the one of π-calculus.

This result should be easily turned into a confluence result through proof techniques *à la* Tait-Martin-Löf. Nonetheless, handling of the congruence relation is rather cumbersome, and as confluence issues were not essential, we did not check the in-depth details.

We also conjecture that the MEC is no stronger than the simply-typed λ-calculus, in the sense that one could find a way to encode MEC terms into simply-typed λ-calculus ones while preserving the reduction up to transitive closure. The main problem of this approach is that the congruence relation is very difficult to encode.

2.6 Abstract machine

We don't give here a detailed definition of an abstract machine for MEC but rather explain its principles with much handwaving.

Intuitively, MEC can be seen as a two-level language with a clear separation between high-level intuitionistic functions (thereafter noted Λ) and low-level unpure processes (thereafter noted Π). The meta-language Λ is hardly an extension of simply-typed lambda-calculus, except that is has distinguished process types $\underline{A} \multimap \underline{B}$, hence it can be put into your favorite abstract machine using whatever reduction pleases you, as long as you don't fall across a process.

During the Λ-reduction, should the machine encounter a process in Π, it must be reduced in its own abstract machine. One can see processes as a two-stack machine, a left one and a right one, containing closures from Λ. The pair $\kappa x.t/t\langle u\rangle$ deals with popping and pushing on the left stack, while the pair $\zeta x.t/t[u]$ deals with popping and pushing on the right one. The null process is the identity on stacks, whereas the composition $t; u$ amounts to combine stacks componentwise.

This provides a clear understanding of the commutativity requirements of the operational semantics: left and right stacks do not mess together. It also underlines the lack of *parti pris* in favour of either of the usual reduction strategies.

3 CPS-like translation

3.1 Implicit monads

The MEC is designed in such a way that it gives naturally rise to two families of monads: the linealy-used state and the linearly-used CPS. The language itself permits to decompose both into their respective adjunctions.

CPS	State
$\vdash \lambda x.\rho[x] : A \to ((?A^\perp \parr \underline{R}) \multimap \underline{R})$	$\vdash \lambda x.\rho\langle x\rangle : A \to (\underline{S} \multimap (!A \otimes \underline{S}))$
$\vdash \zeta x.x : ?A^\perp \parr \underline{R} \multimap ?(?A^\perp \parr \underline{R} \multimap \underline{R})^\perp \parr \underline{R}$	$\vdash \kappa x.x : !(\underline{S} \multimap !A \otimes \underline{S}) \otimes \underline{S} \multimap !A \otimes \underline{S}$

There is evident syntactic duality between terms and between types. The next section extends and formalizes this observation.

3.2 CPS duality

We define in table 6 the CPS translation $(-)^\bullet$ of MEC into itself. Its name is justified by the fact that this translation can be seen as a double-negation on types through the use of linear logic notations. The auxilliary transformation $(-)^\circ$ is then the simple linear negation.

Lemma 3.1. *The CPS translation is sound, that is, if* $\Gamma \vdash t = u$ *then* $\Gamma^\bullet \vdash t^\bullet = u^\bullet$.

Contrary to the usual λ-calculus, the CPS translation of MEC exhibits a very specific property, viz. it is involutive.

Lemma 3.2. *The CPS translation is involutive.*

The proof is syntactically evident. From this involutivity, one easily gets the fullness and faithfullness of the CPS translation.

Theorem 3.3. *The CPS translation is faithful, that is, if* $\Gamma^\bullet \vdash t^\bullet = u^\bullet : A^\bullet$ *then* $\Gamma \vdash t = u : A$.

Theorem 3.4. *The CPS translation is full, that is, if* $\Gamma^\bullet \vdash t : A^\bullet$, *then there exists* $\Gamma \vdash u : A$ *such that* $\Gamma^\bullet \vdash t = u^\bullet : A^\bullet$. *More precisely,* $u = t^\bullet$ *by involutivity.*

$$\begin{array}{rclcrcl} \alpha^\bullet & = & \alpha & \qquad & \underline{\alpha}^{\circ\circ} & = & \underline{\alpha} \\ (A \to B)^\bullet & = & A^\bullet \to B^\bullet & & (?A^\perp \parr \underline{B})^\circ & = & !A^\bullet \otimes \underline{B}^\circ \\ (\underline{A} \multimap \underline{B})^\bullet & = & \underline{B}^\circ \multimap \underline{A}^\circ & & (!A \otimes \underline{B})^\circ & = & ?(A^\bullet)^\perp \parr \underline{B}^\circ \end{array}$$

$$x^\bullet = x \quad (\lambda x.t)^\bullet = \lambda x.t^\bullet \quad (t(u))^\bullet = t^\bullet(u^\bullet)$$

$$\rho^\bullet = \rho \qquad (t;u)^\bullet = u^\bullet; t^\bullet$$

$$\begin{array}{ll} (\kappa x.t)^\bullet = \zeta x.t^\bullet & (\zeta x.t)^\bullet = \kappa x.t^\bullet \\ (t\langle u\rangle)^\bullet = t^\bullet[u^\bullet] & (t[u])^\bullet = t^\bullet\langle u^\bullet\rangle \end{array}$$

Table 6: CPS translation

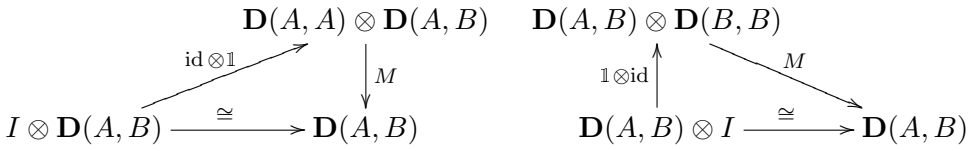
These results also held for the original EEC, yet the duality was not obvious because it was burried under a lot of technical issues.

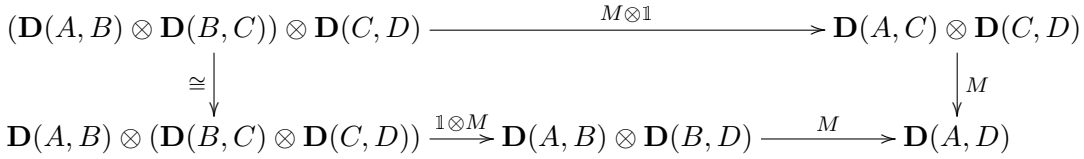
4 Models

4.1 Enriched categories

MEC models are defined in terms of enriched category, while most other comparable models use fibred categories. In our opinion, enrichement is a notion simpler than fibers, and as it befits perfectly our needs here, we should stick to it. For the unacquainted reader, we first recall the basics of enriched category theory using notations from MEC. More details can be found in Kelly's reference book[6].

Definition 4.1. Given a monoidal category $(\mathbf{C}, \otimes, I)$, a $\mathbf{C}$-enriched category $\mathbf{D}$ is defined by a class of $\mathbf{D}$-objects $\mathrm{Obj}(\mathbf{D})$, and for every $A, B \in \mathrm{Obj}(\mathbf{D})$ an object $\mathbf{D}(A,B) \in \mathrm{Obj}(\mathbf{C})$. We also require the existence of morphisms $\mathrm{id}_A : I \to \mathbf{D}(A,A)$ and $M_{A,B,C} : \mathbf{D}(A,B) \otimes \mathbf{D}(B,C) \to \mathbf{D}(A,C)$ for every $\mathbf{D}$-objects A, B, C such that the following diagrams commute:





Enriched categories are a natural way to express hom-sets of a category in terms of objects of another category without having to deal with size issues. For example, a **Set**-enriched category is simply a locally small category, a **Cat**-enriched category is a 2-category, etc. Monoidal closed categories can also be seen as self-enriched categories.

We won't discuss further the extension of usual categorical objects (functors, natural transformations, etc.) to enriched categories and we will focus immediately on the structure we need to interpret MEC. From now on, let us assume a cartesian closed category[3] **C** and a **C**-enriched category **D**.

Definition 4.2. We say that **D** has **C**-powers (or cotensors) if for every $A \in \mathrm{Obj}(\mathbf{C})$ and $\underline{B} \in \mathrm{Obj}(\mathbf{D})$ there exists a **D**-object $?A^{\perp} \parr \underline{B}$ and a **C**-isomorphism $\varphi : \mathbf{D}(\underline{C}, ?A^{\perp} \parr \underline{B}) \to \mathbf{D}(\underline{C}, \underline{B})^{A}$ **C**-natural in $\underline{C}$, i.e. the following diagram commutes:

$$\begin{array}{ccccc}
\mathbf{D}(\underline{C'},\underline{C}) \times (\mathbf{D}(\underline{C}, ?A^{\perp} \parr \underline{B}) \times A) & & \xrightarrow{\mathbb{1}\times(\varphi\times\mathbb{1};\mathrm{ev})} & & \mathbf{D}(\underline{C'},\underline{C}) \times \mathbf{D}(\underline{C},\underline{B}) \\
\cong\downarrow & & & & \downarrow M \\
\mathbf{D}(\underline{C'},\underline{C}) \times \mathbf{D}(\underline{C}, ?A^{\perp} \parr \underline{B}) \times A & \xrightarrow{M\times\mathbb{1}} & \mathbf{D}(\underline{C'}, ?A^{\perp} \parr \underline{B}) \times A & \xrightarrow{\varphi\times\mathbb{1};\mathrm{ev}} & \mathbf{D}(\underline{C'},\underline{B})
\end{array}$$

Definition 4.3. We say that **D** has **C**-tensors (or copowers) if for every $A \in \mathrm{Obj}(\mathbf{C})$ and $\underline{B} \in \mathrm{Obj}(\mathbf{D})$ there exists a **D**-object $!A \otimes \underline{B}$ and a **C**-isomorphism $\psi : \mathbf{D}(!A \otimes \underline{B}, \underline{C}) \to \mathbf{D}(\underline{B}, \underline{C})^{A}$ **C**-natural in $\underline{C}$, i.e. the following diagram commutes:

$$\begin{array}{ccccc}
\mathbf{D}(!A \otimes \underline{B},\underline{C}) \times A \times \mathbf{D}(\underline{C},\underline{C'}) & & \xrightarrow{(\psi\times\mathbb{1};\mathrm{ev})\times\mathbb{1}} & & \mathbf{D}(\underline{B},\underline{C}) \times \mathbf{D}(\underline{C},\underline{C'}) \\
\cong\downarrow & & & & \downarrow M \\
\mathbf{D}(!A \otimes \underline{B},\underline{C}) \times \mathbf{D}(\underline{C},\underline{C'}) \times A & \xrightarrow{M\times\mathbb{1}} & \mathbf{D}(!A \otimes \underline{B},\underline{C'}) \times A & \xrightarrow{\psi\times\mathbb{1};\mathrm{ev}} & \mathbf{D}(\underline{B},\underline{C'})
\end{array}$$

[3] A symmetric monoidal closed one is sufficient, but we will not use such models here.

4.2 MEC models

The basic structures to interpret MEC are given by a pair $(\mathbf{V}, \mathbf{C})$ where:

- $\mathbf{V}$ is a cartesian closed category, called the *value category*;
- $\mathbf{C}$ is a $\mathbf{V}$-enriched category with $\mathbf{V}$-powers and copowers, called the *computation category*.

We suppose given an interpretation for base value types (as $\mathbf{V}$-objects) and base computation types (as $\mathbf{C}$-objects), and we define inductively the type interpretation in the evident way. The term interpretation is then given in table 7. Pure terms $\Gamma \vdash t : A$ are interpreted as morphisms $[\![\Gamma]\!] \to [\![A]\!]$ while processes $\Gamma \mid \underline{A} \vdash t : \underline{B}$ are interpreted as morphisms $[\![\Gamma]\!] \to ([\![\underline{A}]\!] \multimap [\![\underline{B}]\!])$ where $[\![\Gamma]\!]$ denotes the left-associative product of types in Γ and $\underline{A} \multimap \underline{B}$ denotes the enriched homset $\mathbf{C}(\underline{A}, \underline{B})$.

$$
\begin{aligned}
[\![x_1 : A_1, \dots, x_n : A_n \vdash x_i : A_i]\!] &= \pi_i^n \\
[\![\Gamma \vdash \lambda x.t : A \to B]\!] &= \Lambda([\![\Gamma, x : A \vdash t : B]\!]) \\
[\![\Gamma \vdash t(u) : B]\!] &= \langle [\![\Gamma \vdash t : A \to B]\!] \mid [\![\Gamma \vdash u : A]\!] \rangle ; \mathrm{ev} \\
[\![\Gamma \mid \underline{C} \vdash \zeta x.t : ?A^\perp \parr \underline{B}]\!] &= \Lambda([\![\Gamma, x : A \mid \underline{C} \vdash t : \underline{B}]\!]) ; \varphi^{-1} \\
[\![\Gamma \mid \underline{C} \vdash t[u] : B]\!] &= \langle [\![\Gamma \mid \underline{C} \vdash t : ?A^\perp \parr \underline{B}]\!] ; \varphi \mid [\![\Gamma \vdash u : A]\!] \rangle ; \mathrm{ev} \\
[\![\Gamma \mid !A \otimes \underline{B} \vdash \kappa x.t : \underline{C}]\!] &= \Lambda([\![\Gamma, x : A \mid \underline{B} \vdash t : \underline{C}]\!]) ; \psi^{-1} \\
[\![\Gamma \mid \underline{B} \vdash t\langle u \rangle : \underline{C}]\!] &= \langle [\![\Gamma \mid !A \otimes \underline{B} \vdash t : \underline{C}]\!] ; \psi \mid [\![\Gamma \vdash u : A]\!] \rangle ; \mathrm{ev} \\
[\![\Gamma \mid \underline{A} \vdash t : \underline{B}]\!] &= [\![\Gamma \vdash t : \underline{A} \multimap \underline{B}]\!] \\
[\![\Gamma \mid \underline{A} \vdash t; u : \underline{C}]\!] &= \langle [\![\Gamma \mid \underline{A} \vdash t : \underline{B}]\!] \mid [\![\Gamma \mid \underline{B} \vdash u : \underline{C}]\!] \rangle ; M \\
[\![\Gamma \mid \underline{A} \vdash \rho : \underline{A}]\!] &= 1_{[\![\Gamma]\!]} ; \mathrm{id}_{[\![\underline{A}]\!]}
\end{aligned}
$$

Table 7: Categorical interpretation of MEC

The usual results hold for MEC models, namely that they are sound and complete with respect to MEC equational theory.

Lemma 4.4. *The following diagram commutes, where θ is the canonical isomorphism $(Z^Y)^X \to (Z^X)^Y$:*

$$
\begin{array}{ccc}
!A \otimes \underline{C} \multimap ?B^\perp \parr \underline{D} & \xrightarrow{\varphi} & (!A \otimes \underline{C} \multimap \underline{D})^B \\
\downarrow{\scriptstyle \psi} & & \downarrow{\scriptstyle \psi^B ;\ \theta} \\
(\underline{C} \multimap ?B^\perp \parr \underline{D})^A & \xrightarrow{\varphi^A} & (\underline{C} \multimap \underline{D})^{B^A}
\end{array}
$$

Theorem 4.5. *The categorical semantics is sound with respect to the operational semantics, i.e. if $\vdash t = u$ then $[\![t]\!] = [\![u]\!]$.*

Proof. This is proven using a routine induction on the typing judgement. The substitution cases are handled with the same lemma as original λ-calculus. The non-trivial congruence cases are reduced thanks to lemma 4.4.

□

Completeness is proved by building up a syntactical model. For this to work, one needs to add (value) pairs $A \times B$ and (value) unit type 1 to MEC. As we wanted to have a core language, we did not provide MEC with such constructions, though their addition is plain as day, with the usual λ-calculus inherited typing and equational theory.

Theorem 4.6. *The categorical semantics is complete with respect to the operational semantics, i.e. if for every MEC model $\mathcal{M}$ we have $[\![t]\!]_{\mathcal{M}} = [\![u]\!]_{\mathcal{M}}$ then $\vdash t = u$.*

Proof. The syntactical model $\mathcal{M}_{syn}$ is built as follows:

1. **C**-objects are computation types
2. **V**-objects are value types
3. **V**-morphisms $A \to B$ are judgements $[x : A \vdash t : B]$ identified up to $\beta\eta$-equivalence and α-conversion

As MEC is an extension of λ-calculus, the same arguments apply and we can show that **V** is a cartesian closed category with the substitution-as-composition structure. The additional requirements are defined as follows:

$$\begin{array}{rcl}
\mathrm{id}_{\underline{A}} & \equiv & [x : 1 \vdash \rho : \underline{A} \multimap \underline{A}] \\
M_{\underline{A},\underline{B},\underline{C}} & \equiv & [p : \underline{A} \multimap \underline{B} \times \underline{B} \multimap \underline{C} \vdash \mathtt{fst}(p); \mathtt{snd}(p) : \underline{A} \multimap \underline{C}] \\
\varphi_{\underline{A},\underline{B},\underline{C}} & \equiv & [f : \underline{C} \multimap ?A^{\perp} \parr \underline{B} \vdash \lambda x.(f; \rho[x]) : A \to \underline{C} \multimap \underline{B}] \\
\varphi^{-1}_{\underline{A},\underline{B},\underline{C}} & \equiv & [f : A \to \underline{C} \multimap \underline{B} \vdash \zeta x.f(x) : \underline{C} \multimap ?A^{\perp} \parr \underline{B}] \\
\psi_{\underline{A},\underline{B},\underline{C}} & \equiv & [f : !A \otimes \underline{B} \multimap \underline{C} \vdash \lambda x.(\rho\langle x\rangle; f) : A \to \underline{B} \multimap \underline{C}] \\
\psi^{-1}_{\underline{A},\underline{B},\underline{C}} & \equiv & [f : A \to \underline{B} \multimap \underline{C} \vdash \kappa x.f(x) : !A \otimes \underline{B} \multimap \underline{C}]
\end{array}$$

The syntactical enrichment mimics the categorical structure, hence **C** is **V**-enriched by construction. Proof of isomorphisms is just a rewriting sequence. The **V**-naturality requirement is syntactically equivalent to the Ctx-rules for the congruence, while the Comm-rules are related to the commutating diagram of lemma 4.4.

□

4.3 Remarks

It is obvious that the syntactical CPS translation, when transposed into the categorical world, simply corresponds to the the dual interpretation $(\mathbf{V}, \mathbf{C}^{op})$. Indeed, $\mathbf{C}^{op}$ is also **V**-enriched and its powers (resp. tensors) are the tensors (resp. powers) of **C**: the isomorphisms of $\mathcal{M}_{syn}$ themselves are syntactically dual through the CPS transformation.

This also implies that if one wants to extend the MEC with additional structure while preserving the CPS involutivity, any construction should be added with its dual: products and coproducts, like in the original EEC, self-powers and self-copowers, etc.

The model study also provides us with cases where the built-in monads do not satisfy the mono requirement, i.e. their lift η is not monomorphic. An extreme example is the case where we only consider a cartesian closed category **V** and identify any process $\underline{A} \multimap \underline{B}$ with the cartesian unit 1. This is a perfectly valid model in which the processes are collapsed and do not have any informative content, though it is rather useless from a computational point of view.

4.4 Comparison with EEC models

MEC models are basically a restriction of EEC models. Indeed, adding a **V**-enriched adjunction together with **C**-products and coproducts gives you back an EEC model. We argue here that EEC models are not basic and that MEC models are the true underlying foundations for the computational properties of EEC.

First because **C**-products and coproducts are an additional property of the object language of computations that can be traced back to additive linear logic connectives. Dealing with them is usually not much of a fuss, and furthermore, they do not really relate to the computational capacities of the language, and are a rather straightforward extension of MEC. Thanks to the enriched structure, products and coproducts can be described by the existence of **V**-natural isomorphisms:

$$\begin{aligned} \underline{C} \multimap \underline{A} \,\&\, \underline{B} &\cong (\underline{C} \multimap \underline{A}) \times (\underline{C} \multimap \underline{B}) \\ \underline{A} \oplus \underline{B} \multimap \underline{C} &\cong (\underline{A} \multimap \underline{C}) \times (\underline{B} \multimap \underline{C}) \end{aligned}$$

Second because **V**-adjunctions are mostly irrelevant from a semantical standpoint. We recall that a **V**-adjunction $F \dashv U : \mathbf{C} \to \mathbf{V}$ is described through a family of natural **V**-isomorphisms:

$$\chi_{A,\underline{B}} : (FA \multimap \underline{B}) \to (U\underline{B})^A$$

The following lemma permits to get rid of adjunctions from EEC models.

Lemma 4.7. *Computations types and* **V***-adjunctions are in a one-one correspondance (up to isomorphism).*

Proof. Let us assume a **V**-adjunction $F \dashv U$. Then the following natural **V**-isomorphisms hold:

$$U(\underline{A}) \cong U(\underline{A})^1 \cong F(1) \multimap \underline{A}$$
$$F(A) \multimap \underline{B} \cong U(\underline{B})^A \cong (F(1) \multimap \underline{B})^A \cong !A \otimes F(1) \multimap \underline{B}$$

By naturality in $\underline{B}$ in the second equation, we obtain a **C**-isomorphism $F(A) \cong !A \otimes F(1)$. Hence $F \dashv U$ is uniquely defined by $F(1)$ up to isomorphism.

Conversely, given any computation type $\underline{R}$, let us define $F(A) = !A \otimes \underline{R}$ and $U(\underline{A}) = \underline{R} \multimap \underline{A}$. Then $F \dashv U$ is a **V**-adjunction. □

This lemma sheds a new light on the translations of λ-calculus into EEC as defined by Møgelberg[3]. Having an **V**-adjunction is just fixing a particular return type $\underline{R} \cong F(1)$, which corresponds to an empty stack for the κ binder. In other words, such an adjunction is exactly a canonical monad $\underline{R} \multimap !A \otimes \underline{R}$ for a chosen $\underline{R}$. This also implies a lack of symmetry, because in EEC we do not have the dual $?A^\perp \parr \underline{R}^\perp \multimap \underline{R}^\perp$ canonical monad, which leads to a lot of bureaucracy in the CPS translation. This concludes the justification for dropping $!A$ types and inclusion of computations into values from the original EEC.

5 Phylogeny of MEC

We describe in this section some phylogenic relationships that bounds MEC to other commonplace structures.

5.1 LNL models

Linear-nonlinear models are the usual interpretative framework of Dual Intuitionistic Linear Logic (DILL), which is a refinement of ILL[1]. We briefly recall the specificities of that particular proof system.

1. DILL types are those of ILL, namely:
$$A ::= \alpha \mid I \mid A \otimes B \mid A \multimap B \mid !A$$
2. DILL judgements are of the form $\Gamma \mid \Delta \vdash A$, where Γ is called the *intuitionnistic* environment and Δ the *linear* environment. Such a sequent can be thought as the following ILL sequent: $!\Gamma, \Delta \vdash A$. The deduction rules are then a simple adaptation of those from ILL.

LNL models reflect this separation between two different environments by requiring *two* categories and a well-behaved adjunction to glue them together.

Definition 5.1. LNL models are defined by a triple $(\mathbf{V}, \mathbf{C}, F \dashv G)$ where:

1. $\mathbf{V}$ is a cartesian closed category[4]
2. $\mathbf{C}$ is a symmetrical monoidal closed category
3. $F \dashv G : \mathbf{C} \to \mathbf{V}$ is a symmetric monoidal adjunction, i.e. its unit and counit preserve the monoidal tensor and unit together with the symmetrical structure

The following result already held in EEC, and the proof is a direct adaptation. As we dropped the additive connectives, we do not have to impose product-coproduct requirements on $\mathbf{C}$, hence we stick more closely to plain DILL.

Theorem 5.2. *Every LNL model gives rise to a MEC model.*

Proof. We take the two categories according to the obvious choice: $\mathbf{V}$ is already a CCC and shall be the value category. The $\mathbf{V}$-enrichment[5] is provided by the right adjoint combined with the closed structure of $\mathbf{C}$, and the tensor-cotensors are built out of the SMCC structure of $\mathbf{C}$:

$$\underline{A} \multimap \underline{B} \equiv G(\underline{A} \multimap \underline{B})$$
$$!A \otimes \underline{B} \equiv F(A) \otimes \underline{B}$$
$$?A^{\perp} \parr \underline{B} \equiv F(A) \multimap \underline{B}$$

The required morphisms are the obvious ones, and the rest of the proof is diagram chasing. □

Remark. This categorical result correspond syntactically to the sound embedding of MEC into DILL that naturally results from the following transparent translation on types:

$$[\![A \to B]\!] = ![\![A]\!] \multimap [\![B]\!]$$
$$[\![\underline{A} \multimap \underline{B}]\!] = [\![\underline{A}]\!] \multimap [\![\underline{B}]\!]$$
$$[\![!A \otimes \underline{B}]\!] = ![\![A]\!] \otimes [\![\underline{B}]\!]$$
$$[\![?A^{\perp} \parr \underline{B}]\!] = ![\![A]\!] \multimap [\![\underline{B}]\!]$$

[4]The closedness is not necessary, and one can show that it follows from the other requirements. Yet it eases the expression of the forthcoming result.

[5]We abuse the definition by identifying the enriched category with the underlying plain category.

Remark. LNL models are a very special case of MEC models where $\mathbf{C}$ is itself a SMCC and the enriched tensor can actually be decomposed into a comonad and a symmetric monoidal tensor, and likewise for the enriched cotensor.

The converse does not hold, and there does not seem to be any canonical way to retrieve a LNL model out of an MEC model, because of the many ways of lifting the enriched tensor into a monoidal one. What would be, for example, the monoidal unit? As a matter of fact, the LNL models do not feature the CPS duality, hence they are too rich to capture the essence of MEC.

5.2 The $\kappa\zeta$-calculus

Few people are aware of the existence of the $\kappa\zeta$-calculus, designed and studied by Hasegawa in his Master's thesis[4] as a decomposition of λ-calculus into two categorical languages, a context-oriented one (the κ-calculus) and a continuation-oriented one (the ζ-calculus). Quite surprisingly, after having spent those eons in a timeless sleep, we accidentally discovered that $\kappa\zeta$-calculus was eventually the closest relative of MEC in the jungle of λ-calculus variants.

As the complete description of $\kappa\zeta$-calculus would not fit here, we advise the reader to refer to Hasegawa's paper.

For our description, it is sufficient to consider that $\kappa\zeta$-calculus is a flavour of MEC where we forget about its λ-calculus fragment, we add a terminal computation $\underline{I}$ and we restrict value types appearing in enriched structures. More precisely, types are described by the following grammar:

$$A ::= \underline{A} \multimap \underline{B}$$
$$\underline{A} ::= \underline{\alpha} \mid \underline{I} \mid\ !(\underline{I} \multimap \underline{A}) \otimes \underline{B} \mid\ ?(\underline{I} \multimap \underline{A})^{\perp} \parr \underline{B}$$

For the sake of clarity, we simply forget about values and we abuse the tensor and cotensor notations. The name of the binders for our language were given after those of $\kappa\zeta$-calculus, which have the same (restricted) typing rules and reductions in this paradigm. For comparison, we give its typing rules in table 8 using an MEC-like syntax.

$$\frac{}{\Gamma, x : A \multimap B \vdash x : A \multimap B} \qquad \frac{}{\Gamma \vdash \rho : A \multimap A} \qquad \frac{}{\Gamma \vdash * : A \multimap I}$$

$$\frac{\Gamma, x : I \multimap A \vdash t : C \multimap B}{\Gamma \vdash \zeta x.t : C \multimap ?A^{\perp} \parr B} \qquad \frac{\Gamma \vdash t : I \multimap A}{\Gamma \vdash \rho[t] : ?A^{\perp} \parr C \multimap C}$$

$$\frac{\Gamma, x : I \multimap A \vdash t : B \multimap C}{\Gamma \vdash \kappa x.t : !A \otimes B \multimap C} \qquad \frac{\Gamma \vdash t : I \multimap A}{\Gamma \vdash \rho\langle t \rangle : C \multimap !A \otimes C}$$

$$\frac{\Gamma \vdash t : A \multimap B \qquad \Gamma \vdash u : B \multimap C}{\Gamma \vdash t; u : A \multimap C}$$

Table 8: Core rules of $\kappa\zeta$-calculus

Remark. The relationship between MEC models and $\kappa\zeta$-models are rather unclear, even though the syntactical similarities are more than obvious. This mismatch can be explained by the lack of structure of both models:

1. The computation category of an MEC model has no proper structure, and in particular no terminal object

2. $\kappa\zeta$-models lack everything related to the cartesian structure of $\mathbf{V}$

Nonetheless, we conjecture that, if we forget about the closedness requirement on $\mathbf{V}$, those two types of models are essentially the same. The trick lies in the good choice of the enriching category.

1. Given an MEC model $(\mathbf{V}, \mathbf{C})$, by adding a terminal object in $\mathbf{C}$, we do obtain a $\kappa\zeta$-model.

2. Given a $\kappa\zeta$-model $\mathbf{M}$ and putting aside the size issues, $(\widehat{\mathbf{M}}, \mathbf{M})$ should make a decent MEC model, where $\widehat{\mathbf{M}} = \mathbf{Set}^{\mathbf{M}^{op}}$ is the presheaf category of $\mathbf{M}$.

The interesting part about this constatation lies on the fact that $\kappa\zeta$-calculus was a subtle refinement of λ-calculus conceived to have greater control on the reduction process, not unlike explicit substitution. As such, this is a strong piece of evidence that the two linear-state and linear-continuation monads are tightly bound to the very core of λ-calculus semantics, regardless of the effects we may add to it.

5.3 Monads

As monads where the starting point of the construction of MEC, it was natural to check whenever a genuine monad would permit to construct an MEC model. Alas, most of the results are negative: there is no natural way to get a non-trivial MEC model from the two canonical adjunctions derived from T. For the Kleisli category, it is seemingly hopeless (the main problem is defining the cotensor). For the Eilenberg-Moore category, we have a purely negative result.

Lemma 5.3. *Given a strong monad T on a cartesian closed category $\mathbf{V}$, one can consider the pair $(\mathbf{V}, \mathbf{V}^T)$. Then $\mathbf{V}^T$ is trivially $\mathbf{V}$-enriched. The functors $T(-) \times (-) : \mathbf{V} \times \mathbf{V}^T \to \mathbf{V}^T$ and $(-)^{T(-)} : \mathbf{V}^{op} \times \mathbf{V}^T \to \mathbf{V}^T$ fail to define a tensor-cotensor structure. In the general case, instead of an isomorphism, they only define a retraction. Actually, $(\mathbf{V}, \mathbf{V}^T)$ is an MEC model iff it is a (degenerated) LNL model.*

5.4 Classical structures

The CPS duality result can be seen as a weak classical property of MEC. Although MEC lack any other classical property, the duality results are rather well-known in classical-capable λ-calculi, i.e. $\lambda\mu$-calculus and its variants. For example, CBN $\lambda\mu$-calculus with classical sums is dual to CBV $\lambda\mu$-calculus with classical products. Selinger designed control and co-control categories to model those languages. Once again, we won't recall the very definition of such categories, but only give the underlying ideas. The bravehearted reader may refer to Selinger's article[12].

Definition 5.4. A control category $\mathbf{P}$ is defined by the following requirements:

- $(\mathbf{P}, \times, 1)$ is a cartesian closed category
- $(\mathbf{P}, \parr, \bot)$ is a symmetric premonoidal category
- An isomorphism $(A \parr B)^C \cong A^C \parr B$ ($\parr$ is to be thought as a classical $\vee$)

- A herd of coherence diagrams!

A co-control category is the categorical dual of a control category.

A particularly remarkable subcategory of $\mathbf{P}$ is $\mathbf{P}^\bullet$, the category of central morphisms of $\mathbf{P}$, i.e. the largest subcategory in which $⅋$ is a monoidal tensor. In a control category, this is also the largest subcategory in which $⅋$ is a cartesian sum.

Theorem 5.5. *If* $\mathbf{P}$ *is a control category, then* $\mathbf{P}$ *is* $\mathbf{P}^{\bullet op}$*-enriched with* $\underline{A} \multimap \underline{B} = \bot\underline{B^A}$ *and features a cotensor* $?A^\bot ⅋ \underline{B}$ *where* $⅋$ *is the premonoidal tensor and* $?(-)^\bot : \mathbf{P}^{\bullet op} \to \mathbf{P}$ *is the reverse inclusion functor. Note that in general,* $\mathbf{P}^{\bullet op}$ *is* not *cartesian closed!*

Dually, any co-control category gives rise to a center-enriched tensor.

As a valuable effect, this result further justifies the linear logic notation *a posteriori*. Selinger also made this remark; the very terminology of control category is related to linearity, e.g. central morphisms are said to be copyable and discardable.

Unluckily, we only get part of the picture with each structure: control gives power, and co-control gives co-power. Even worse, in both cases the enriching category is not closed. The half-empty glass issue can be easily resolved by requiring a control-co-control-category. The CPS duality would then become a particular case of the classical duality. The absence of closed structure on the enriching category is more tricky. There is no way to impose it without implying some degeneracy to our model. The next section is a more precise remark about this.

5.5 The closedness issue

The intuitionnistic fragment of MEC seems to be very difficult to deal with. For instance, the CPS translation ignores it and leaves it unchanged. Indeed, this fragment is essentially the simply-typed λ-calculus. Thus getting rid of the intuitionistic function space sounds natural. This matter of fact seems to be the norm throughout the whole array of related structures we studied:

1. The CC of an LNL model is closed thanks to the closedness of its SMCC adjoint, hence the closedness of the former can be seen as a lucky accident.

2. The $\kappa\zeta$-models only induce an underlying enriching category with finite products (that is, context manipulation), and as such the closedness is irrelevant here.

3. As stated in the previous section, control and co-control categories fail to comply with the closedness requirement.

We argue that the pervasive λ-calculus structures that are deep-rooted in MEC have nothing to do with the closedness of the value category. Indeed, if $\kappa\zeta$-calculus is a kind of resource-sensitive λ-calculus, then this tensor-cotensor construction is the categorical counterpart of environment-management in plain λ-calculus.

6 Further work

In this part we have a glimpse into topics we did not intensively studied but which are nevertheless cite-worthy.

6.1 Practical use

The type system of MEC enforces a linear discipline on computation types. In particular, MEC can be used as a language able to manipulate state and continuations linearly. This can save a lot of hassle in compiling of functional languages: usually, programs are translated to CPS in order to be optimized. Using an adhoc MEC as an intermediate language, the typing ensures that the modifications are harmless.

Likewise, state manipulation in pure languages such as Haskell hides a lot of complexity in the compiler. If one can force the linearity of the state at the user-level, it may allow better optimizations and permit to get rid of under-the-carpet purple magic.

One of the others advantages of MEC, which is inherited from $\kappa\zeta$-calculus, lies in the fact that closures are decomposed into a context-accessing part (the κ-fragment) and a continuation-passing part (the ζ-fragment). It is quite common, when working on an intermediate compiling language, that we only need one of the two operations. Creating a plain closure would be costly, and using instead only one of those two operators tends to be rather cheap.

Actually, the very structure of MEC looks like a compiler. In the value realm dwells a meta-language, namely the simply-typed λ-calculus[6], while in the neighbouring land of computations live process transformers, forming the object language. The meta-language is about manipulating object programs, whereas these object programs themselves are the actual implementation for our machine.

6.2 ILL models

Historically, there are two successive families of models for intuitionnistic fragments of linear logic.

1. The ILL models, defined by a SMCC plus a so-called linear comonad.
2. The DILL models, defined by a SMCC, a CC and a symmetric monoidal adjunction in between.

The study of MEC provided one possible definition of *linearity*. In a linear setting, functions are *point-free*, i.e. one should not be able to manipulate formulas directly because there shall not be any pending hypothesis. It seems that this notion of point-freedom corresponds to the enrichment in category theory. To our knowledge, there is no model of linear logic using explicitely this arrow structure[7]. We think that using an enrichment instead of an adjunction may solve some internal language-related issues of DILL. Basically, we want to have this sort of morphisms:

$$1 \to (\underline{A} \multimap \underline{A})$$
$$(\underline{A} \multimap \underline{C}) \times (\underline{B} \multimap \underline{D}) \to (\underline{A} \otimes \underline{B} \multimap \underline{C} \otimes \underline{D})$$
$$A \cong (\underline{I} \multimap !A)$$

It is not difficult to enrich the simply-typed λ-calculus to get a similar structure. The main problem would be that this categorical structure leads to an ugly syntax for its internal language. Nevertheless, this novel model may lead to interesting results.

[6] But it can be enriched with whatever you want.
[7] Like Hughes' arrows.

Conclusion

We gave in this document a new presentation of the quite young extended effect calculus, stripped down to its bare skeleton. The original effect calculus suffered a lot of atavistic design flaws inherited from both call-by-push-value and the computational λ-calculus. The essential result of this refinement focused only on the multiplicative fragment, which is, from a computational standpoint, the most problematic one, and it seems to handle it quite well. The syntax is natural and the canonical models are very general. Alongside this study, we showed that MEC and its models featured a lot of lovely good-mannered results that one can expect for a theoretical language, such as various soundness and completness. As a side-effect, MEC is also a syntactical language to reason about (co)-tensored enriched categories.

Interestingly enough, the original effect calculus was built with computational effects in mind, but it turned out that this decomposition of two dual monads into a dual tensor-cotensor structure is much more general than one could expect. The linear logic notation is motivated by the fact that DILL models are MEC models, but actually, it seems that this tensor-cotensor structure is essentially related to the very nature of substitution of λ-calculus, without any reference to non-intuitionistic patterns. MEC provides a finer grain on control of resource-like substitution. As such, the two linear state and continuation monads may capture the whole array of behaviours stuck between call-by-value and call-by-name, just as call-by-push-value did, but with a genuine categorical model.

The tight relationship that entangles MEC with $\kappa\zeta$-calculus is also something quite new. With this particular insight, the CPS duality turns out to be a duality between context and continuations. MEC could be the categorical justification of the CPS–SSA duality.

The simplified MEC without intuitionistic functions also seems to be a powerful model. The results about $\lambda\mu$-calculus indicates that MEC should be augmented with classical skills. We conjecture that it may better explain the CBV–CBN duality of $\lambda\mu$-calculus, as it should be a unifying framework for both structures.

As we stated in the last section, MEC could be a promising language for two very different research directions. On the practical side, MEC looks like a perfect language for an intermediate representation of calculi and flows of computations. On the realm of logics, we did not find anyone who was interested in representing linear logic through enriched categories, and we think it may be an interesting path to explore.

References

[1] Andrew Barber. *Dual Intuitionistic Linear Logic*. Tech. rep. University of Edinburgh, 1996.

[2] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. "Enriching an Effect Calculus with Linear Types". In: *CSL*. 2009, pp. 240–254.

[3] Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. "Linearly-Used Continuations in the Enriched Effect Calculus". In: *FOSSACS*. 2010, pp. 18–32.

[4] Masahito Hasegawa. "Decomposing Typed Lambda Calculus Into a Couple of Categorical Programming Languages". In: *Proc. CTCS , Lect. Notes in Computer Science 953*. Springer, 1995, pp. 200–219.

[5] Masahito Hasegawa. "Linearly Used Effects: Monadic and CPS Transformations into the Linear Lambda Calculus". In: *FLOPS*. 2002, pp. 167–182.

[6] Max Kelly. *Basic Concepts of Enriched Category Theory*. London Mathematical Society Lecture Notes 64. Cambridge University Press, 1982. ISBN: 0521287022.

[7] Richard Kelsey. "A Correspondence between Continuation Passing Style and Static Single Assignment Form". In: *ACM SIGPLAN Notices*. ACM Press, 1995, pp. 13–22.

[8] Paul Blain Levy. "Call-by-push-value". University of London, 2001.

[9] Saunders MacLane. *Categories for the working mathematician*. Springer-Verlag, 1971.

[10] Eugenio Moggi. "Notions of Computation and Monads". In: *Inf. Comput.* 93.1 (1991), pp. 55–92.

[11] Valeria De Paiva. *Categorical Semantics of Linear Logic For All*. 2006.

[12] Peter Selinger. "Control categories and duality: on the categorical semantics of the lambda-mu calculus". In: *Mathematical Structures in Computer Science* 11.2 (2001), pp. 207–260.