An Algebraic Extension of Intuitionistic Linear Logic: The $\mathcal{L}_!^{\mathcal{S}}$-Calculus and Its Categorical Model

Extended abstract of: `arXiv:2504.12128`

Alejandro Díaz-Caro[1,2] Malena Ivnisky[3,4,5] Octavio Malherbe[6]

[1]Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
[2]Universidad Nacional de Quilmes, DCyT, Bernal, PBA, Argentina
[3]Universidad de Buenos Aires, DC, FCEyN, Buenos Aires, Argentina
[4]Universidad de Buenos Aires-CONICET, ICC, Buenos Aires, Argentina
[5]Universidad de la República–MEC, PEDECIBA, Montevideo, Uruguay
[6]Universidad de la República, IMERL, FIng, Montevideo, Uruguay

Abstract

We introduce the $\mathcal{L}_!^{\mathcal{S}}$-calculus, a linear lambda-calculus extended with scalar multiplication and term addition, that acts as a proof language for intuitionistic linear logic (ILL). These algebraic operations enable the direct expression of linearity at the syntactic level, a property not typically available in standard proof-term calculi. Building upon previous work, we develop the $\mathcal{L}_!^{\mathcal{S}}$-calculus as an extension of the $\mathcal{L}^{\mathcal{S}}$-calculus with the ! modality. A denotational semantics is provided in the framework of linear categories with biproducts. This work in progress is part of a broader programme aiming to build a measurement-free quantum programming language grounded in linear logic.

1 Introduction

The design of proof languages plays a central role in both logic and programming languages, particularly when reasoning about resource usage and algebraic computation. Linear Logic [22] provides a framework for such reasoning, but its standard proof-term calculi do not make linear properties—such as distribution over addition or scalar multiplication—explicit at the syntactic level. This lack of syntactic expressiveness limits its applicability in settings like quantum computing, where linearity is not just a property but a fundamental constraint.

Linear Logic is named as such because it is modelled by vector spaces and linear maps, and more generally by monoidal categories [5,6,21,25]. These types of categories also include the so-called Cartesian categories, generating a formal place of interaction between purely algebraic structures and purely logical structures, i.e. between algebraic operations and the exponential connective "!". In the strictly linear fragment (without !), functions between two propositions are linear functions. However, expressing this linearity within the proof-term language itself is challenging. Properties such as $f(u+v) = f(u) + f(v)$ and $f(a \cdot u) = a \cdot f(u)$, for some scalar a, require operations like addition and scalar multiplication, which are typically absent in the proof language.

In [13] this challenge has been addressed by introducing the $\mathcal{L}^{\mathcal{S}}$-calculus, a proof language for the strictly linear fragment of intuitionistic linear logic (IMALL) extended with syntactic constructs for addition ($+$) and scalar multiplication ($\bullet$). While the extension does not change provability, it enables the expression of linear properties at the term level, such as distributivity of functions over sums and scalar multiplication. For example, any proof-term t of $A \multimap B$ satisfies that $t\ (u + v)$ is observationally equivalent to $t\ u + t\ v$. Similarly, $t\ (a \bullet u)$ is equivalent to $a \bullet t\ u$.

This extension involves changing the proof-term $\star$ of proposition $\mathtt{1}$ into a family of proof-terms $a.\star$, one for each scalar a in a given fixed semiring $\mathcal{S}$:

$$\frac{}{\vdash a.\star : \mathtt{1}}\ \mathtt{1}_i(a)$$

The following two deduction rules have also been added:

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : A}{\Gamma \vdash t + u : A}\ \text{sum} \qquad\qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash a \bullet t : A}\ \text{prod}(a)$$

Incorporating these rules requires adding commuting rules to preserve cut-elimination. Indeed, the new rules may appear between an introduction and an elimination of some connective. For example, consider the following derivation.

$$\frac{\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\ \&_i}{\Gamma \vdash A \,\&\, B}\ \text{prod}(a) \qquad \Gamma, A \vdash C}{\Gamma \vdash C}\ \&_{e1}$$

To achieve cut-elimination, the $\text{prod}(a)$ rule must be commuted either with the introduction or with the elimination, as shown below.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash A}{\Gamma \vdash A}\,\text{prod}(a) \quad \dfrac{\Gamma \vdash B}{\Gamma \vdash B}\,\text{prod}(a)}{\Gamma \vdash A \,\&\, B}\,\&_i \qquad \Gamma, A \vdash C}{\Gamma \vdash C}\,\&_{e1} \qquad\qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \,\&\, B}\,\&_i \qquad \Gamma, A \vdash C}{\Gamma \vdash C}\,\&_{e1}}{\Gamma \vdash C}\,\text{prod}(a)$$

Both of these are reducible. We refer to the sum and $\text{prod}(a)$ rules as *interstitial rules*, as they can appear in the interstice between an introduction and an elimination. We choose to commute these rules with the introductions as much as possible. This means we introduce the following commutation rule $a \bullet \langle t, u\rangle \longrightarrow \langle a \bullet t, a \bullet u\rangle$ instead of the alternative rule $\delta^1_{\&}(a \bullet t, x^A.u) \longrightarrow a \bullet \delta^1_{\&}(t, x^A.u)$. This choice provides a better introduction property, for example, a closed irreducible proof-term of a proposition $A \,\&\, B$ is a pair.

In [14] we extended the proof system to second-order intuitionistic linear logic, adding the exponential connective and a universal quantifier. We proved that the linearity result still holds for second order.

In [17], which is the first journal version of [14], polymorphism has been removed to allow for a thorough study of the ILL fragment. We provide a denotational semantics for this fragment of the language, which we refer to as the $\mathcal{L}^{\mathcal{S}}_{!}$-calculus.

While our primary focus is on introducing a minimal extension to the proof language within the realm of intuitionistic linear logic, our work draws inspiration from various domains, particularly quantum programming languages. These languages were trailblazers in merging programming constructs with algebraic operations, such as addition and scalar multiplication.

Indeed, QML [1] introduced the concept of superposition of terms through the if° constructor, allowing the representation of linear combinations $a.u + b.v$ by the expression $\mathsf{if}^\circ\ a.|0\rangle + b.|1\rangle$ then u else v. The linearity (and even unitarity) properties of QML were established through a translation to quantum circuits.

The ZX calculus [10], a graphical language based on a categorical model, lacks direct syntax for addition or scalar multiplication but defines a framework where such constructs can be interpreted. This language is extended by the Many Worlds Calculus [9] which allows for linear combinations of diagrams.

The algebraic lambda-calculus [26] and Lineal [4] exhibit syntax similarities with $\mathcal{L}^{\mathcal{S}}$-calculus. However, the algebraic lambda-calculus lacks a proof of linearity in its simple intuitionistic type system. In contrast, Lineal is untyped and axiomatizes the linearity, relying on explicit definitions like $f(u+v) := f(u)+f(v)$ and $f(a.u) := a.f(u)$. Several type systems have been proposed for Lineal [2, 3, 15, 16, 18]. However, none of these systems are related to linear logic, and their purpose is not to prove linearity, as we do, but rather to axiomatize it.

One of the objectives of our line of research is to define a measurement-free quantum programming language rooted in Linear Logic. Or, another way to state it, is to extend a proof language of Intuitionistic Linear Logic with the necessary constructs to represent quantum algorithms. The $\mathcal{L}^{\mathcal{S}}$-calculus was the first step in this direction [13].

In our conference paper [14], we introduced the $\mathcal{L}^{\mathcal{S}}_{!\forall}$-calculus as the second step, extending the language to Second-Order Intuitionistic Linear Logic, where we can encode natural numbers, iterators, and more generally, classical computing, apart from vectors and matrices for quantum computing. The $\mathcal{L}^{\mathcal{S}}_{!}$-calculus is a minimal extension of the $\mathcal{L}^{\mathcal{S}}$-calculus, adding just the exponential connective. The next step is to add polymorphism to the semantics, appealing to linear hyperdoctrines [23]. The $\mathcal{L}^{\mathcal{S}}_{!}$-calculus is a step towards a quantum programming language based on Linear Logic. Further steps include adding measurement, and restricting the language to unitary maps following the techniques from [16].

2 The $\mathcal{L}^{\mathcal{S}}_{!}$-calculus

2.1 Syntax and Typing Rules

The $\mathcal{L}^{\mathcal{S}}_{!}$-logic is intuitionistic linear logic (we follow the presentation of F_{DILL} [23], extended with the additive linear connectives).

$$A = \mathtt{1} \mid A \multimap A \mid A \otimes A \mid \top \mid \mathtt{0} \mid A \,\&\, A \mid A \oplus A \mid {!A}$$

Let $\mathcal{S}$ be a semiring of *scalars*, for instance $\{\star\}$, $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, or $\mathbb{C}$. The proof-terms of the $\mathcal{L}^{\mathcal{S}}_{!}$-calculus are given in Figure 1, where a is a scalar in $\mathcal{S}$.

A judgement has the form $\Upsilon; \Gamma \vdash t : A$, where Υ is the non-linear context and Γ is the linear one. The deduction rules are exactly the deduction rules of intuitionistic linear natural deduction, with proof-terms, with two differences: the interstitial rules and the scalars (see Section 1). These three rules are shown in Figure 2.

The reduction rules are those of Figure 3. The first group of rules correspond to the reduction of cuts on the connectives $\mathtt{1}$, $\multimap$, $\otimes$, $\&$, $\oplus$, and $!$. The next two groups enable us to commute the interstitial rules sum and $\text{prod}(a)$ with the introduction rules of the connectives $\mathtt{1}$, $\multimap$, $\top$, and $\&$, and with the elimination rule of the connectives $\otimes$, $\oplus$, and $!$. For instance, the rule $\langle t, u\rangle + \langle v, w\rangle \longrightarrow \langle t + v, u + w\rangle$ pushes the symbol $+$ inside the pair. The zero-ary commutation rules add and multiply the scalars: $a.\star + b.\star \longrightarrow (a + b).\star$, $a \bullet b.\star \longrightarrow (a \times b).\star$.

2.2 Linearity

The linearity results from [13] are trivially false when we consider the exponential connective. Hence, these results refer to the original $\mathcal{L}^{\mathcal{S}}$-calculus (Theorems 2.1 and 2.2).

$$
\begin{array}{llll}
t = x \mid t + u \mid a \bullet t & & & \\
& \mid a.\star & \mid \delta_{1}(t,u) & (1) \\
& \mid \lambda x^A.t & \mid t\ u & (\multimap) \\
& \mid t \otimes u & \mid \delta_{\otimes}(t, x^A y^B.u) & (\otimes) \\
& \mid \langle\rangle & & (\top) \\
& & \mid \delta_{0}(t) & (0) \\
& \mid \langle t,u \rangle & \mid \delta^1_{\&}(t, x^A.u) \mid \delta^2_{\&}(t, x^B.u) & (\&) \\
& \mid \mathit{inl}(t) \mid \mathit{inr}(t) & \mid \delta_{\oplus}(t, x^A.u, y^B.v) & (\oplus) \\
& \mid !t & \mid \delta_{!}(t, x^A.u) & (!) \\
& \text{Introductions} & \text{Eliminations} & \text{Connective}
\end{array}
$$

Figure 1: The proof-terms of the $\mathcal{L}^{\mathcal{S}}_{!}$-calculus.

$$
\frac{}{\Upsilon; \varnothing \vdash a.\star : 1}\ 1_i(a)
\qquad
\frac{\Upsilon;\Gamma \vdash t : A \quad \Upsilon;\Gamma \vdash u : A}{\Upsilon;\Gamma \vdash t + u : A}\ \text{sum}
\qquad
\frac{\Upsilon;\Gamma \vdash t : A}{\Upsilon;\Gamma \vdash a \bullet t : A}\ \text{prod}(a)
$$

Figure 2: Some of the deduction rules in the $\mathcal{L}^{\mathcal{S}}_{!}$-calculus.

The set of vector types $\mathcal{V}$ is inductively defined as follows: $1 \in \mathcal{V}$, and if A and B are in $\mathcal{V}$, then so is $A \,\&\, B$. To each proposition $A \in \mathcal{V}$, we associate a positive natural number $d(A)$, which is the number of occurrences of the symbol 1 in A.

For each vector $\mathbf{u}$ in $\mathcal{S}^m$, we can define a term $\overline{\mathbf{u}}^A$, of type $A \in \mathcal{V}$ with $d(A) = m$.

Theorem 2.1 ([17, Theorem 2.20])**.** *Let $A, B \in \mathcal{V}$ with $d(A) = m$ and $d(B) = n$ and let M be a matrix with m columns and n lines, then there exists a closed proof-term t of $A \multimap B$ such that, for all vectors $\mathbf{u} \in \mathcal{S}^m$, $t\ \overline{\mathbf{u}}^A \longrightarrow^* \overline{M\mathbf{u}}^B$.* □

Theorem 2.2 ([17, Corollary 2.24])**.** *For any A and B, if $\vdash \lambda x^A.t : A \multimap B$, $\vdash u_1 : A$, and $\vdash u_2 : B$, then*

$$
(\lambda x^A.t)(u_1 + u_2) \sim (\lambda x^A.t)u_1 + (\lambda x^A.t)u_2
$$
$$
(\lambda x^A.t)(a \bullet u_1) \sim a \bullet (\lambda x^A.t)u_1
$$

where $\sim$ is the computational equivalence. □

3 Denotational semantics

3.1 The categorical construction

In this section, we introduce the fundamental categorical constructions that serve as the basis for defining the denotational semantics of the $\mathcal{L}^{\mathcal{S}}_{!}$-calculus. These constructions are inspired by the work of Bierman [7] on intuitionistic linear logic. Many of the additive properties required for our setting have been established in a recent draft [20], which develops a categorical semantics for the $\mathcal{L}\odot^{\mathcal{S}}$-calculus [13], an extension of the $\mathcal{L}^{\mathcal{S}}$-calculus incorporating the non-deterministic *sup* connective $\odot$. We adapt these constructions to incorporate the exponential connective !.

The first key definition is that of a linear category (Definition 3.1).

Definition 3.1 (Linear category [7, Definition 35])**.** A linear category consists of a symmetric monoidal closed category $(\mathcal{C}, \otimes, I)$ with finite products and coproducts, equipped with a symmetric monoidal comonad $(!, \varepsilon, \delta, m_{A,B}, m_I)$, such that:

1. For every free !-coalgebra $(!A, \delta_A)$ there are two distinguished monoidal natural transformations with components $e_A : !A \to I$ and $d_A : !A \to !A \otimes !A$ which form a commutative comonoid and are coalgebra morphisms.
2. Whenever $f : (!A, \delta_A) \to (!B, \delta_B)$ is a coalgebra morphism between free coalgebras, then it is also a comonoid morphism.

The next key ingredient is the notion of semiadditive category (Definition 3.2) and semiadditive functor (Definition 3.3).

Definition 3.2 (Semiadditive category [24, Chapter I, Section 18])**.** A semiadditive category is a category $\mathcal{C}$ together with an abelian monoid structure on each of its morphism sets, subject to the following conditions:

1. The composition functions $[B \to C] \times [A \to B] \to [A \to C]$ are bilinear.
2. The zero elements of the monoids behave as zero morphisms.

Definition 3.3 (Semiadditive functor)**.** Let $\mathcal{C}, \mathcal{D}$ be semiadditive categories. A functor $F : \mathcal{C} \to \mathcal{D}$ is semiadditive when the mapping of morphisms preserves the monoid structure.

$$\begin{aligned}
\delta_{\mathtt{1}}(a.\star, t) &\longrightarrow a \bullet t \\
(\lambda x^A.t)\ u &\longrightarrow (u/x)t \\
\delta_\otimes(u \otimes v, x^A y^B.w) &\longrightarrow (u/x, v/y)w \\
\delta^1_\&(\langle t_1, t_2\rangle, x^A.v) &\longrightarrow (t_1/x)v \\
\delta^2_\&(\langle t_1, t_2\rangle, x^A.v) &\longrightarrow (t_2/x)v \\
\delta_\oplus(\mathit{inl}(t), x^A.v, y^B.w) &\longrightarrow (t/x)v \\
\delta_\oplus(\mathit{inr}(u), x^A.v, y^B.w) &\longrightarrow (u/y)w \\
\delta_!(!t, x^A.u) &\longrightarrow (t/x)u
\end{aligned}$$

$$\begin{aligned}
a.\star + b.\star &\longrightarrow (a+b).\star \\
(\lambda x^A.t) + (\lambda x^A.u) &\longrightarrow \lambda x^A.(t+u) \\
\delta_\otimes(t+u, x^A y^B.v) &\longrightarrow \delta_\otimes(t, x^A y^B.v) + \delta_\otimes(u, x^A y^B.v) \\
\langle\rangle + \langle\rangle &\longrightarrow \langle\rangle \\
\langle t, u\rangle + \langle v, w\rangle &\longrightarrow \langle t+v, u+w\rangle \\
\delta_\oplus(t+u, x^A.v, y^B.w) &\longrightarrow \delta_\oplus(t, x^A.v, y^B.w) + \delta_\oplus(u, x^A.v, y^B.w) \\
\delta_!(a \cdot t, x^A.s) &\longrightarrow a \cdot \delta_!(t, x^A.s)
\end{aligned}
\qquad
\begin{aligned}
a \bullet b.\star &\longrightarrow (a \times b).\star \\
a \bullet \lambda x^A.t &\longrightarrow \lambda x^A.a \bullet t \\
\delta_\otimes(a \bullet t, x^A y^B.v) &\longrightarrow a \bullet \delta_\otimes(t, x^A y^B.v) \\
a \bullet \langle\rangle &\longrightarrow \langle\rangle \\
a \bullet \langle t, u\rangle &\longrightarrow \langle a \bullet t, a \bullet u\rangle \\
\delta_\oplus(a \bullet t, x^A.v, y^B.w) &\longrightarrow a \bullet \delta_\oplus(t, x^A.v, y^B.w) \\
\delta_!(t+u, x^A.s) &\longrightarrow \delta_!(t, x^A.s) + \delta_!(u, x^A.s)
\end{aligned}$$

Figure 3: The reduction rules of the $\mathcal{L}^{\mathcal{S}}_!$-calculus.

3.1.1 The category $\mathbf{C}^{\mathcal{S}}_!$

We define the category $\mathbf{C}^{\mathcal{S}}_!$ (Definition 3.4), which will serve as the basis for the denotational semantics of the $\mathcal{L}^{\mathcal{S}}_!$-calculus. We make use of the following two facts: any category with biproducts admits a semiadditive structure [24, Proposition 18.4] and its hom-sets carry a natural semiring structure [20, Corollary 3.4].

Definition 3.4 (The category $\mathbf{C}^{\mathcal{S}}_!$)**.** Let $\mathcal{S}$ be a fixed semiring. We define $\mathbf{C}^{\mathcal{S}}_!$ to be a linear category with biproduct $\oplus$ where there is a monomorphism from $\mathcal{S}$ to $\mathit{Hom}(I, I)$.

The following is an example of a concrete category with the properties stated in Definition 3.4.

Example 3.5. The category $(\mathsf{SM}_{\mathcal{S}}, \otimes, \mathcal{S}, \oplus)$, where objects are semimodules over the semiring $\mathcal{S}$, arrows are semimodule homomorphisms, the tensor product is the tensor product over semimodules and the biproduct is the Cartesian product. This category is symmetric monoidal closed, and the semiring monomorphism is defined as $(\!|a|\!) = b \mapsto a \cdot_{\mathcal{S}} b$ for every $a \in \mathcal{S}$. The first model for the $\mathcal{L}^{\mathcal{S}}$-calculus has been defined in this category [19]. We can define a monoidal adjunction between $\mathsf{SM}_{\mathcal{S}}$ and Set, which induces a monoidal comonad, and it satisfies the properties of a linear category.

3.2 Interpretation of the calculus

In this section, we define the interpretation of the $\mathcal{L}^{\mathcal{S}}_!$-calculus in $\mathbf{C}^{\mathcal{S}}_!$. We begin by introducing the interpretation of types (Definition 3.6) and typing contexts (Definition 3.7). We then define the interpretation of the typing rules (partially shown in Figure 4), where we use the following standard notation.

Notation. $!\varnothing = \varnothing \qquad !(x^A, \Gamma) = x^{!A}, !\Gamma \qquad \varnothing, \Gamma = \Gamma, \varnothing = \Gamma$

Definition 3.6 (Interpretation of types)**.**

$$[\![\mathtt{1}]\!] = I \qquad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!] \qquad [\![\mathtt{0}]\!] = \mathbf{0} \qquad [\![A \oplus B]\!] = [\![A]\!] \oplus [\![B]\!]$$
$$[\![A \multimap B]\!] = [[\![A]\!] \to [\![B]\!]] \qquad [\![\top]\!] = \mathbf{0} \qquad [\![A \,\&\, B]\!] = [\![A]\!] \oplus [\![B]\!] \qquad [\![!A]\!] = !\,[\![A]\!]$$

Definition 3.7 (Interpretation of typing contexts)**.**

$$[\![\varnothing]\!] = I \qquad\qquad \left[\!\!\left[x^A, \Gamma\right]\!\!\right] = [\![A]\!] \otimes [\![\Gamma]\!]$$

3.3 Soundness and adequacy

The soundness theorem (Theorem 3.8) states that the interpretation is stable by reduction.

Theorem 3.8 (Soundness)**.** *Let $\Upsilon; \Gamma \vdash t : A$. If $t \to r$, then $[\![t]\!] = [\![r]\!]$.* ☐

The adequacy theorem (Theorem 3.11) states that equality of interpretation implies observational equivalence. Observational equivalence (Definition 3.10) means that the proofs *produce* identical results in any elimination context (Definition 3.9).

Unlike the standard notion of context, an elimination context is specifically designed to eliminate connectives: a proof of an implication is applied, a proof of a pair is projected, and so on.

$$\left\llbracket\frac{\Upsilon;\Gamma\vdash t:A \quad \Upsilon;\Gamma\vdash u:A}{\Upsilon;\Gamma\vdash t+u:A}\ \mathrm{sum}\right\rrbracket = \llbracket !\Upsilon\rrbracket\otimes\llbracket\Gamma\rrbracket \xrightarrow{\Delta} (\llbracket !\Upsilon\rrbracket\otimes\llbracket\Gamma\rrbracket)\oplus(\llbracket !\Upsilon\rrbracket\otimes\llbracket\Gamma\rrbracket) \xrightarrow{\llbracket t\rrbracket\oplus\llbracket u\rrbracket} \llbracket A\rrbracket\oplus\llbracket A\rrbracket \xrightarrow{\nabla} \llbracket A\rrbracket$$

$$\left\llbracket\frac{\Upsilon;\Gamma\vdash t:A}{\Upsilon;\Gamma\vdash a\bullet t:A}\ \mathrm{prod}(a)\right\rrbracket = \llbracket !\Upsilon\rrbracket\otimes\llbracket\Gamma\rrbracket \xrightarrow{\llbracket t\rrbracket} \llbracket A\rrbracket \xrightarrow{\widehat{(\!|a|\!)}} \llbracket A\rrbracket$$

$$\left\llbracket\frac{}{\Upsilon;\varnothing\vdash a.\star:1}\ 1_i(a)\right\rrbracket = \llbracket !\Upsilon\rrbracket\otimes I \xrightarrow{\rho} \llbracket !\Upsilon\rrbracket \xrightarrow{e_\Upsilon} I \xrightarrow{(\!|a|\!)} I$$

$$\left\llbracket\frac{\Upsilon;\varnothing\vdash t:A}{\Upsilon;\varnothing\vdash !t:!A}\ !_i\right\rrbracket = \llbracket !\Upsilon\rrbracket\otimes I \xrightarrow{\rho} \llbracket !\Upsilon\rrbracket \xrightarrow{\delta_\Upsilon} !\,\llbracket !\Upsilon\rrbracket \xrightarrow{!(\rho^{-1})} !(\llbracket !\Upsilon\rrbracket\otimes I) \xrightarrow{!\llbracket t\rrbracket} !\,\llbracket A\rrbracket$$

$$\left\llbracket\frac{\Upsilon;\varnothing\vdash t:!A \quad \Upsilon;\Delta,x^A\vdash u:B}{\Upsilon;\Delta\vdash \delta_!(t,x^A.u):B}\ !_e\right\rrbracket = \llbracket !\Upsilon\rrbracket\otimes\llbracket\Delta\rrbracket \xrightarrow{\bar{d}_{\Upsilon,\varnothing,\Delta}} \llbracket !\Upsilon\rrbracket\otimes\llbracket !\Upsilon\rrbracket\otimes\llbracket\Delta\rrbracket \xrightarrow{\llbracket t\rrbracket\otimes id} !\,\llbracket A\rrbracket\otimes\llbracket !\Upsilon\rrbracket\otimes\llbracket\Delta\rrbracket$$
$$\xrightarrow{\varepsilon_A\otimes id} \llbracket A\rrbracket\otimes\llbracket !\Upsilon\rrbracket\otimes\llbracket\Delta\rrbracket \xrightarrow{\sigma} \llbracket !\Upsilon\rrbracket\otimes\llbracket\Delta\rrbracket\otimes\llbracket A\rrbracket \xrightarrow{\llbracket u\rrbracket} \llbracket B\rrbracket$$

Figure 4: Interpretation of some of the typing rules of the $\mathcal{L}^{\mathcal{S}}_!$-calculus. The maps Δ and ∇ are the diagonal and codiagonal morphisms. The maps ρ and σ are the right unit and symmetry of the monoidal structure. $\hat{a} = \rho\circ(id\otimes a)\circ\rho^{-1} : A\to A$ is the scalar map between arbitrary objects. The maps $e\_$ and $\delta\_$ are generalisations of the counit of the comonoid and the comultiplication of the comonad, respectively. The map $\bar{d}$ is a composition of the generalised comultiplication map $d\_$ and the coherence maps of the monoidal structure, that duplicates the banged contexts.

Definition 3.9 (Elimination context)**.** An elimination context is a proof produced by the following grammar, where _ denotes a distinguished variable.

$$K = \_ \mid K\ u \mid \delta_\otimes(K, x^Ay^B.v) \mid \delta^1_\&(K, x^A.r) \mid \delta^2_\&(K, x^B.r) \mid \delta_\oplus(K, x^A.r, y^B.s) \mid \delta_!(K, x^A.r)$$

Notation. We write $K[t]$ for $(t/\_)K$, and $\_^A\vdash K:1$ means either $\_^A;\varnothing\vdash K:1$ or $\varnothing;\_^A\vdash K:1$.

Definition 3.10 (Observational equivalence)**.** A proof $\vdash t:A$ is observationally equivalent to a proof $\vdash u:A$ (written $t\equiv u$) if, for every elimination context $\_^A\vdash K:1$, we have

$$K[t]\longrightarrow^* a.\star \quad \text{if and only if} \quad K[u]\longrightarrow^* a.\star\,.$$

Theorem 3.11 (Adequacy)**.** *Let $\vdash t:A$ and $\vdash r:A$. If $\llbracket t\rrbracket = \llbracket r\rrbracket$, then $t\equiv r$.* ☐

4 Conclusion and future work

We have introduced the $\mathcal{L}^{\mathcal{S}}_!$-calculus, an extension of the $\mathcal{L}^{\mathcal{S}}$-calculus with the exponential connective, allowing non-linear functions and making it a more expressive language.

The $\mathcal{L}^{\mathcal{S}}$-calculus was originally introduced as a core language for quantum computing. Its ability to represent matrices and vectors (Section 2.2) makes it suitable for expressing quantum programs when taking $\mathcal{S}=\mathbb{C}$. Moreover, by taking $\mathcal{S}=\mathbb{R}^+$, one can consider a probabilistic language, and by taking $\mathcal{S}=\{\star\}$, a linear extension of the parallel lambda calculus [8].

To consider this calculus as a proper quantum language, we would need not only to ensure algebraic linearity but also to ensure unitarity, using techniques such as those in [16]. Also, the language $\mathcal{L}^{\mathcal{S}}$ can be extended with a non-deterministic connective $\odot$ [12], from which a quantum measurement operator can be encoded. We did not add such a connective to our presentation, to stay in a pure linear logic setting, however, the extension is straightforward.

In [14] the $\mathcal{L}^{\mathcal{S}}_!$-calculus has been extended to include the second-order $\forall$ connective, preserving the linearity properties. This extension allows us to use polymorphism to encode recursive structures in a usual way [22, Chapter 5], as shown in the following example. Given that we are in a linear setting, the exponential connective is essential to achieve this.

Example 4.1. In the polymorphic extension of the $\mathcal{L}^{\mathcal{S}}_!$-calculus, we can define natural numbers.

$$\mathsf{Nat} = \forall X.X\multimap !(X\multimap X)\multimap X \qquad \mathsf{zero} = \Lambda X.\lambda x^X.\lambda f^{!(X\multimap X)}.x$$
$$\mathsf{succ} = \lambda n^{\mathsf{Nat}}.\Lambda X.\lambda x^X.\lambda f^{!(X\multimap X)}.\delta_!(f, g^{X\multimap X}.g)\ (n\ X\ x\ f)$$

We can express the application n times of a square matrix over a vector as follows, where $A\in\mathcal{V}$ with $d(A)=m$.

$$\mathsf{Miter} = \lambda n^{\mathsf{Nat}}.\lambda m^{!(A\multimap A)}.\lambda v^A.n\ A\ v\ m$$

Let M be a square matrix with m columns and lines, and t be the closed proof-term of $A\multimap A$ representing such a matrix. For any vector $\mathbf{u}\in\mathcal{S}^m$ we have $\mathsf{Miter}\ \hat{n}\ !t\ \overline{\mathbf{u}}^A \longrightarrow^* \overline{M^n\mathbf{u}}^A$, where $\hat{n}$ is the encoding of $n\in\mathbb{N}$.

We are currently working on extending the categorical semantics to this polymorphic extension, using hyperdoctrines [11] and following the approach of [23].

Acknowledgments We thank the anonymous reviewer who found a flaw in one of the reduction rules. This version of the abstract is corrected, and the draft [17] will soon be updated.

References

[1] T. Altenkirch and J. Grattage. A functional quantum programming language. In *Proceedings of LICS 2005*, pages 249–258, 2005.

[2] P. Arrighi and A. Díaz-Caro. A System F accounting for scalars. *Logical Methods in Computer Science*, 8(1:11), 2012.

[3] P. Arrighi, A. Díaz-Caro, and B. Valiron. The vectorial lambda-calculus. *Information and Computation*, 254(1):105–139, 2017.

[4] P. Arrighi and G. Dowek. Lineal: A linear-algebraic lambda-calculus. *Logical Methods in Computer Science*, 13(1), 2017.

[5] M. Barr. $*$-Autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1(2), 1991.

[6] J. Bénabou. Catégories avec multiplication. *Comptes Rendus des Séances de l'Académie des Sciences*, 256:1887–1890, 1963.

[7] G. Bierman. On intuitionistic linear logic. Technical Report UCAM-CL-TR-346, University of Cambridge, Computer Laboratory, Aug. 1994.

[8] G. Boudol. Lambda-calculi for (strict) parallel functions. *Information and Computation*, 108(1):51–127, 1994.

[9] K. Chardonnet, M. de Visme, B. Valiron, and R. Vilamart. The many-worlds calculus. Logical Methods in Computer Science, to appear, 2025.

[10] B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, 2017.

[11] R. L. Crole. *Categories for types.* Cambridge University Press, 1993.

[12] A. Díaz-Caro and G. Dowek. A new connective in natural deduction, and its application to quantum computing. *Theoretical Computer Science*, 957:113840, 2023.

[13] A. Díaz-Caro and G. Dowek. A linear linear lambda-calculus. *Mathematical Structures in Computer Science*, 34:1103–1137, 2024.

[14] A. Díaz-Caro, G. Dowek, M. Ivnisky, and O. Malherbe. A linear proof language for second-order intuitionistic linear logic. In G. Metcalfe, T. Studer, and R. de Queiroz, editors, *Logic, Language, Information and Computation*, volume 14672 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2024.

[15] A. Díaz-Caro, G. Dowek, and J. Rinaldi. Two linearities for quantum computing in the lambda calculus. *BioSystems*, 186:104012, 2019.

[16] A. Díaz-Caro, M. Guillermo, A. Miquel, and B. Valiron. Realizability in the unitary sphere. In *Proceedings of LICS 2019*, pages 1–13, 2019.

[17] A. Díaz-Caro, M. Ivnisky, and O. Malherbe. An algebraic extension of intuitionistic linear logic: The $L_!^S$-calculus and its categorical model. `arXiv:2504.12128`, 2025.

[18] A. Díaz-Caro and O. Malherbe. Quantum control in the unitary sphere: Lambda-S_1 and its categorical model. *Logical Methods in Computer Science*, 18(3:32), 2022.

[19] A. Díaz-Caro and O. Malherbe. Semimodules and the (syntactically-)linear lambda calculus. `arXiv:2205.02142v2`, 2022.

[20] A. Díaz-Caro and O. Malherbe. The sup connective in IMALL: A categorical semantics. `arXiv:2205.02142`, 2024.

[21] S. Eilenberg and G. M. Kelly. Closed categories. In S. Eilenberg, D. K. Harrison, S. MacLane, and H. Röhrl, editors, *Proceedings of the Conference on Categorical Algebra*, pages 421–562, Berlin, Heidelberg, 1966. Springer Berlin Heidelberg.

[22] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[23] P. Maneggia. *Models of linear polymorphism.* PhD thesis, The University of Birmingham, UK, 2004.

[24] B. Mitchell. *Theory of Categories.* Pure and applied mathematics : a series of monographs and textbooks. Academic Press, 1965.

[25] R. Seely. Linear logic, $*$-autonomous categories and cofree coalgebras. In J. W. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, Contemporary Mathematics. American Mathematical Society, 1989.

[26] L. Vaux. The algebraic lambda calculus. *Mathematical Structures in Computer Science*, 19(5):1029–1059, 2009.