A linear-non-linear model for a computational call-by-value lambda calculus (extended abstract)

Peter Selinger[1] and Benoît Valiron[2]

[1] Dalhousie University, selinger@mathstat.dal.ca
[2] University of Ottawa, bvali087@uottawa.ca

Abstract. We give a categorical semantics for a call-by-value linear lambda calculus. Such a lambda calculus was used by Selinger and Valiron as the backbone of a functional programming language for quantum computation. One feature of this lambda calculus is its linear type system, which includes a duplicability operator "!" as in linear logic. Another main feature is its call-by-value reduction strategy, together with a side-effect to model probabilistic measurements. The "!" operator gives rise to a comonad, as in the linear logic models of Seely, Bierman, and Benton. The side-effects give rise to a monad, as in Moggi's computational lambda calculus. It is this combination of a monad and a comonad that makes the present paper interesting. We show that our categorical semantics is sound and complete.

1 Introduction

In the last few years, there has been some interest in the semantics of quantum programming languages. [18] gave a denotational semantics for a flow-chart language, but this language did not include higher-order types. Several authors defined quantum lambda calculi [21, 19] as well as quantum process algebras [11, 12], which had higher-order features and a well-defined operational semantics, but lacked denotational semantics. [20] gave a categorical model for a higher-order quantum lambda calculus, but omitted all the non-linear features (i.e., classical data). Meanwhile, Abramsky and Coecke [2, 9] developed categorical axiomatics for Hilbert spaces, but there is no particular language associated with these models.

In this paper, we give the first categorical semantics of an unabridged quantum lambda calculus, which is a version of the language studied in [19].

For the purposes of the present paper, an understanding of the precise mechanics of quantum computation is not required. We will focus primarily on the type system and language, and not on the structure of the actual "built-in" quantum operations (such as unitary operators and measurements). In this sense, this paper is about the semantics of a generic call-by-value linear lambda calculus, which is parametric on some primitive operations that are not further explained. It should be understood, however, that the need to support primitive quantum operations motivates particular features of the type system, which we briefly explain now.

The first important language feature is linearity. This arises from the well-known *no-cloning* property of quantum computation, which asserts that quantum data cannot be duplicated [23]. So if $x : qbit$ is a variable representing a quantum bit, and $y : bit$ is a variable representing a classical bit, then it is legal to write $f(y, y)$, but not $g(x, x)$. In order to keep track of duplicability at higher-order types we use a type system based on linear logic. We use the duplicability operator "!" to mark classical types. In the categorical semantics, this operator gives rise to a comonad as in the work of [16] and [6]. Another account of mixing copyable and non-copyable data is [10], where the copyability is internal to objects.

A second feature of quantum computation is its probabilistic nature. Quantum physics has an operation called measurement, which converts quantum data to classical data, and whose outcome is inherently probabilistic. Given a quantum state $\alpha|0\rangle + \beta|1\rangle$, a measurement will yield output 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$. To model this probabilistic effect in our call-by-value setting, our semantics requires a computational monad in the sense of [14]. The coexistence of the computational monad T and the duplicability comonad ! in the same category is what makes our semantics interesting and novel. It differs from the work of [7], who considered a monad and a comonad one two different categories, arising from a single adjunction.

The computational aspects of linear logic have been extensively explored by many authors, including [8, 6, 5, 1, 22]. However, these works contain explicit lambda terms to witness the structural rules of linear logic, for example, $x : !A \rhd \mathrm{derelict}(x) : A$. By contrast, in our language, structural rules are implicit at the term level, so that $!A$ is regarded as a subtype of A and one writes $x : !A \rhd x : A$. As we have shown in [19], linearity information can automatically be inferred by the type checker. This allows the programmer to program as in a non-linear language.

This use of subtyping is the main technical complication in our proof of well-definedness of the semantics. This is because one has to show that the denotation is independent of the choice of a potentially large number of possible derivations of a given typing judgment. We are forced to introduce a Church-style typing system, and to prove that the semantics finally does not depend on the additional type annotations.

Another technical choice we made in our language concerns the relation between the exponential ! and the pairing operation. Linear logic only requires $!A \otimes !B \rhd !(A \otimes B)$ and not the opposite implication. However, in our programming language setting, we find it natural to identify a classical pair of values with a pair of classical values, and therefore we will have an isomorphism $!A \otimes !B \cong !(A \otimes B)$.

The plan of the paper is the following. First, we describe the lambda calculus and equational axioms we wish to consider. Then, we develop a categorical model, called linear category for duplication, which is inspired by [8] and [14].

We then show that the language is an internal language for the category, thus obtaining soundness and completeness.

2 The language

We will describe a linear typed lambda calculus with higher-order functions and pairs. The language is designed to manipulate both classical data, which is duplicable, and quantum data, which is non-duplicable. For simplicity, we assume the language is strictly linear, and not affine linear as in [19]. This means duplicable values are both copyable and discardable, whereas non-duplicable values must be used once, and only once.

2.1 The type system

The set of types is given as follows: $\mathit{Type}\ A, B ::= \alpha \mid (A \multimap B) \mid (A \otimes B) \mid \top \mid\ !A$.

Here α ranges over type constants. While the remainder of this paper does not depend on the choice of type constants, in our main application [19] this is intended to include a type *qbit* of quantum bits, and a type *bit* of classical bits. $A \multimap B$ stands for functions from A to B, $A \otimes B$ for pairs, $\top$ for the unit type, and $!A$ for duplicable objects of types A. We denote $!! \cdots !A$ with n !'s by $!^n A$.

The intuitive definition of $!A$ is the key to the spirit in which we want the language to be understood: The ! on $!A$ is understood as specifying a property, rather than additional structure, on the elements of A. Therefore, we will have $!A \cong\ !!A$. Whether or not a given value of type A is also of type $!A$ should be something that is inferred, rather than specified in the code.

Since a term of type $!A$ can always be seen as a term of type A, we equip the type system with a subtyping relation as follows: Provided that $(m = 0) \vee (n \geqslant 1)$,

$$\frac{}{!^n\alpha <:\ !^m\alpha}\ (ax), \qquad \frac{}{!^n\top <:\ !^m\top}\ (\top),$$

$$\frac{A <: A' \quad B <: B'}{!^n(A' \multimap B) <:\ !^m(A \multimap B')}\ (\multimap), \qquad \frac{A <: A' \quad B <: B'}{!^n(A \otimes B) <:\ !^m(A' \otimes B').}\ (\otimes).$$

This relation encapsulates the main properties terms should satisfy with respect to duplicability.

2.2 Terms

The language consists of the following typed terms, divided into *values* on the one hand, and general terms, or *computations*, on the other. Both share a subset of the values, the *core values*.

$$\begin{aligned}
&\mathit{CoreValue}\ U ::= x^A \mid c^A \mid *^n \mid \lambda^n x^A.M,\\
&\mathit{Value}\ V, W\ ::= U \mid \langle V, W\rangle \mid\ \mathit{let}\ \ x^A = V\ \mathit{in}\ W \mid\ \mathit{let}\ \langle x^A, y^B\rangle^n = V\ \mathit{in}\ W \mid\\
&\qquad\qquad\qquad \mathit{let}\ * = V\ \mathit{in}\ W,\\
&\mathit{Term}\ M, N\ ::= U \mid \langle M, N\rangle \mid (MN) \mid\ \mathit{let}\ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ N \mid\ \mathit{let}\ * = M\ \mathit{in}\ N,
\end{aligned}$$

where n is an integer, c ranges over a set of constant terms, x over a set of term variables and α over a set of constant types. We abbreviate $(\lambda^0 x^A.M)N$ by $let\ x^A = N\ in\ M$, $\lambda^n x^{!^m \top}.\ let * = x^\top\ in\ M$ by $\lambda^n *^m.M$ and we omit numerical indexes when they are null.

Note that the above terms carry Church-style type annotations, as well as integer superscripts; we call these terms *indexed terms*. We also define a notion of *untyped terms* as terms with no index:

$$\begin{aligned} PureTerm\ M, N\ ::=\ & x \mid c \mid * \mid \lambda x.M \mid (MN) \mid \langle M, N \rangle \mid \\ & let\langle x, y\rangle = M\ in\ N \mid\ let * = M\ in\ N\,. \end{aligned}$$

The erasure operation $Erase : Term \rightarrow PureTerm$ is defined as the operation of removing the types and integers attached to a given indexed term. If $M = Erase(\bar{M})$, we say that $\bar{M}$ is an *indexation of M*

Finally, we define an α-equivalence on terms, denoted by $=_\alpha$, in the usual way (see for example [3]).

2.3 Duplicable pairs and pairs of duplicable elements

Before we formally present the type system, let us informally motivate our choice of typing rules. One non-obvious choice we had to make is for the interaction of pairs and duplicability. Unlike previous works with comonads [8, 5], we want to think of the type $!(A \otimes B)$ as a type of pairs of elements of type A and B: we want to use the same operation to access the components as one would use for a pair of type $A \otimes B$, without having to use a dereliction operation.

This immediately raises a concern: consider a pair of elements $\langle x, y\rangle$ of type $!(A \otimes B)$. Are x and y duplicable? In the usual linear logic interpretation, they are not. Having a infinite supply of pair of shoes does not mean one has an infinite supply of right shoes: we cannot discard the left shoes. On the other hand, in our interpretation of "classical" data as residing in "classical" memory and therefore being duplicable, if the string $\langle x, y\rangle$ is duplicable, then so should be the elements x and y. In other words, we want the duplication to "permeate" the pairing.

The choice of such a "permeable" pairing is more or less forced on us by our desire to have no explicit term syntax for structural rules. Consider the following untyped terms, which can be typed if t is of type $!(A \otimes !(B \otimes C))$:

$$let\langle x, u\rangle = t\ in\ let\langle y, z\rangle = u\ in\ \langle\langle z, y\rangle, x\rangle, \tag{1}$$

$$let\langle x, u\rangle = t\ in\ \langle let\langle y, z\rangle = u\ in\ \langle z, y\rangle, x\rangle. \tag{2}$$

First, we expect these two terms to be axiomatically equal. Term (2) should be of type $!(!(C \otimes B) \otimes A)$, regardless of the permeability of the pairing: if $\langle y, z\rangle$ is duplicable, so should be $\langle z, y\rangle$. Now, consider the term (1) with a non-permeable pairing. In the naive type system, u ends up being of type $B \otimes C$, and the variables y and z in the final recombination end up being respectively of type B and C. It is not possible to make $\langle z, y\rangle$ of the duplicable type $!(C \otimes B)$.

We therefore choose a permeable pairing, which will be reflected, albeit subtly, in the typing rule $(\otimes.I)$ and $(\otimes.E)$ in the following section.

$$\frac{A <: B}{!\Delta, x : A \rhd x^B : B}\ (ax_1) \quad \frac{!A_c <: B}{!\Delta \rhd c^B : B}\ (ax_2) \quad \frac{\Gamma_1, !\Delta \rhd M : A \multimap B \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd MN : B}\ (app)$$

$$\frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda^0 x^A.M : A \multimap B}\ (\lambda_1) \quad \frac{!\Delta, x : A \rhd M : B}{!\Delta \rhd \lambda^{n+1} x^A.M : !^{n+1}(A \multimap B)}\ (\lambda_2) \quad \frac{}{!\Delta \rhd *^n : !^n \top}\ (\top.I)$$

$$\frac{!\Delta, \Gamma_1 \rhd M_1 : !^n A_1 \quad !\Delta, \Gamma_2 \rhd M_2 : !^n A_2}{!\Delta, \Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle^n : !^n (A_1 \otimes A_2)}\ (\otimes.I) \quad \frac{!\Delta, \Gamma_1 \rhd M : \top \quad !\Delta, \Gamma_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd let * = M\ in\ N : A}\ (\top.E)$$

$$\frac{!\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2) \quad !\Delta, \Gamma_2, x_1 : !^n A_1, x_2 {:} !^n A_2 \rhd N : A}{!\Delta, \Gamma_1, \Gamma_2 \rhd let \langle x_1^{A_1}, x_2^{A_2} \rangle^n = M\ in\ N : A}\ (\otimes.E)$$

Table 1. Typing rules

2.4 Typing judgments

A typing judgment is a tuple $\Delta \rhd M : A$, where M is an indexed term, A is a type, and Δ is a typing context. To each constant term c we assign a type $!A_c$. A *valid typing judgment* is a typing judgment that can be derived from the typing rules in Table 1. We use the notation $!\Delta$ for a context where all variables have a type of the form $!A$. Finally, when we write a context Γ, Δ, we assume the contexts Γ and Δ to be disjoint.

The following lemmas are proved by structural induction on terms or type derivations as appropriate.

Lemma 1. *If V is a value such that $\Delta \rhd V : !A$ is a valid typing judgment, then $\Delta = !\Delta'$ for some context Δ'.* □

Lemma 2. *Consider the following valid typing judgment: $\Delta, x : A \rhd M : B$. Then for every free instance $x^{A'}$ in M, $A <: A'$.* □

Definition 1. In a typing judgment $\Delta \rhd M : A$, a term variable $x \in |\Delta|$ is called *dummy* if $x \notin FV(M)$.

Lemma 3. *Any dummy variable x in $\Delta \rhd M : B$ satisfies $\Delta(x) = !A$, for some A. Conversely, if $\Delta \rhd M : B$ is valid and if $x \notin FV(M)$, then for all types A the typing judgment $\Delta, x : !A \rhd M : B$ is valid.* □

Typing derivations are not unique *per se*. However for a given valid typing judgement $\Delta \rhd M : A$ two typing derivations will only differ with respect to the placement of *dummy variables*, namely the unused variables in context.

2.5 Type casting and substitution Lemma

Lemma 4. *Suppose $\Delta \rhd M : A$ is a valid typing judgment, and suppose $\Delta' <: \Delta$ and $A <: A'$. Then there exists a canonical valid typing judgment $\Delta' \rhd M' : A'$ such that $Erase(M) = Erase(M')$. Moreover, if M is a value, so is M'.*

Proof. By induction on M. □

We will denote this M' with $\{\Delta' <: \Delta \rhd M : A <: A'\}$. If $\Delta' = \Delta$ or $A' = A$, we omit them for clarity.

$$
\begin{array}{rlll}
(\beta_\lambda) & \Delta \rhd \mathit{let}\ x = V\ \mathit{in}\ M & \approx_{ax} M[V/x] & : A \\
(\beta_\otimes) & \Delta \rhd \mathit{let} \langle x, y\rangle^n = \langle V, W\rangle^n\ \mathit{in}\ M & \approx_{ax} M[V/x, W/y] & : A \\
(\beta_*) & \Delta \rhd \mathit{let}\ * = *\ \mathit{in}\ M & \approx_{ax} M & : A \\
(\eta_\lambda) & \Delta \rhd \lambda^n x^A.\{V : !^n(A \multimap B) <: A \multimap B\}x^A & \approx_{ax} V & : !^n(A \multimap B). \\
(\beta^2_\lambda) & \Delta \rhd \mathit{let}\ x^A = N\ \mathit{in}\ x^A & \approx_{ax} N & : A. \\
(\eta_\otimes) & \Delta \rhd \mathit{let} \langle x^A, y^B\rangle^n = N\ \mathit{in}\ \langle x^{!^n A}, y^{!^n B}\rangle^n & \approx_{ax} N & : !^n(A \otimes B). \\
(\eta_*) & \Delta \rhd \mathit{let}\ * = N\ \mathit{in}\ *^n & \approx_{ax} N & : !^n \top.
\end{array}
$$

$$
\begin{array}{rlll}
(\mathit{let}_1) & \Delta \rhd \mathit{let}\ -_1 = (\mathit{let}\ -_2 = M\ \mathit{in}\ N)\ \ \mathit{in}\ P & \approx_{ax} \mathit{let}\ \ -_2 = M\ \mathit{in}\ \mathit{let}\ -_1 = N\ \mathit{in}\ P & : A \\
(\mathit{let}_2) & \Delta \rhd \mathit{let}\ -_1 = V\ \mathit{in}\ \mathit{let}\ -_2 = W\ \mathit{in}\ M & \approx_{ax} \mathit{let}\ \ -_2 = W\ \mathit{in}\ \mathit{let}\ -_1 = V\ \mathit{in}\ M & : A \\
(\mathit{let}^{app}) & \Delta \rhd \mathit{let}\ x^{A \multimap B} = M\ \mathit{in}\ \mathit{let}\ y^A = N\ \mathit{in}\ xy & \approx_{ax} MN & : B \\
(\mathit{let}^{\lambda}) & \Delta \rhd \mathit{let}\ x^D = V\ \mathit{in}\ \lambda^n y^A.M & \approx_{ax} \lambda^n y^A.\ \mathit{let}\ \ x^D = V\ \mathit{in}\ M & : !^n(A \multimap B) \\
(\mathit{let}^{\otimes}) & \Delta \rhd \mathit{let}\ x^A = M\ \mathit{in}\ \mathit{let}\ y^B = N\ \mathit{in}\ \langle x^A, y^B\rangle^n & \approx_{ax} \langle M, N\rangle^n & : !^n(A \otimes B)
\end{array}
$$

$$
\begin{array}{rl}
(\mathit{app}_{<:}) & \{M : !^n(A \multimap D) <: B \multimap D'\}\{N : C <: B\} \\
 & \qquad \approx_{ax} \{\{M : !^n(A \multimap D) <: A \multimap D\}\{N : C <: A\} : D <: D'\} \\
(\mathit{let}^{\otimes}_{<:}) & \mathit{let}\langle x^{A'}, y^{B'}\rangle^{n'} = \{M : !^n(A \otimes B) <: !^{n'}(A' \otimes B')\}\ \mathit{in}\ N \\
 & \qquad \approx_{ax} \mathit{let}\ \langle x^A, y^B\rangle^n = M\ \mathit{in}\ \{\Delta, x : !^n A, y : !^n B <: \Delta, x : !^{n'} A', y : !^{n'} B' \rhd N\} \\
(\mathit{let}^{x}_{<:}) & \mathit{let}\ x^{A'} = \{M : A <: A'\}\ \mathit{in}\ N \approx_{ax} \mathit{let}\ x^A = M\ \mathit{in}\ \{\Delta, x : A <: \Delta, x : A' \rhd N\} \\
(\mathit{let}^{*}_{<:}) & \mathit{let}\ * = \{M : !^m \top <: !^n \top\}\ \mathit{in}\ N \approx_{ax} \mathit{let}\ * = M\ \mathit{in}\ N
\end{array}
$$

Table 2. Axiomatic equivalence axioms

$$
\begin{array}{rlll}
(\alpha_{let}) & \Delta, x : A \rhd \mathit{let}\ y^A = x^A\ \mathit{in}\ M : B & \approx_{ax} \Delta, y : A \rhd M & : B \\
(\mathit{let}^{!\lambda}) & !\Delta \rhd \mathit{let}\ x^{!C} = V\ \mathit{in}\ \lambda y.M & \approx_{ax} \lambda^{n+1} y.\ \mathit{let}\ \ x^{!C} = V\ \mathit{in}\ M & : !^{n+1}(A \multimap B) \\
(\mathit{let}^{\otimes}_1) & \Delta \rhd \langle V, \mathit{let}\ - = M\ \mathit{in}\ N\rangle & \approx_{ax} \mathit{let}\ - = M\ \mathit{in}\ \langle V, N\rangle & : !^n(A \otimes B) \\
(\mathit{let}^{\otimes}_2) & \Delta \rhd \langle \mathit{let}\ - = M\ \mathit{in}\ N, V\rangle & \approx_{ax} \mathit{let}\ - = M\ \mathit{in}\ \langle N, V\rangle & : !^n(A \otimes B) \\
(\mathit{let}^{app}_1) & \Delta \rhd V(\mathit{let}\ - = M\ \mathit{in}\ N) & \approx_{ax} \mathit{let}\ - = M\ \mathit{in}\ VN & : B \\
(\mathit{let}^{app}_2) & \Delta \rhd (\mathit{let}\ - = M\ \mathit{in}\ N)V & \approx_{ax} \mathit{let}\ - = M\ \mathit{in}\ NV & : B
\end{array}
$$

Table 3. Axiomatic equivalence: derived rules

Definition 2. Given two valid typing judgments $!\Delta, \Gamma_1 \rhd V : A$ and $!\Delta, \Gamma_2, x : A \rhd M : B$ where V is a value, we define the *substitution* (with capture avoiding) $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ as follows: we replace each free instance $x^{A'}$ (where $A <: A'$ from Lemma 2) in M by $\{\Delta \rhd V : A <: A'\}$.

Lemma 5 (Substitution Lemma). *In Definition 2, $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ is well-typed. Also, if M is a value, so is $M[V/x]$.*

Proof. Proof by structural induction on M, using Lemmas 2 and 4. □

2.6 Axiomatic equivalence

We define an equivalence relation on (indexed) typing judgments. We write $\Delta \rhd M \approx_{ax} M' : A$, or simply $M \approx_{ax} M'$, to indicate that $\Delta \rhd M : A$ and $\Delta \rhd M' : A$ are equivalent. Axiomatic equivalent is defined as the reflexive, symmetric, transitive, and congruence closure of the rules from Tables 2, so long as both sides of the equivalences are well-typed. The symbol "$-$" is a place holder for x, $*$, or $\langle x, y\rangle$.

Lemma 6. *The equivalences of Table 3 are derivable.* □

The following result stipulates that all the indexations of a given erasure live in the same axiomatic class. In other words, the axiomatic equivalence class of a term is independent of its indexation.

Theorem 1. *If $Erase(M) = Erase(M')$ and if $\Delta \rhd M, M' : A$ are valid typing judgments, then $M \approx_{ax} M'$.*

Proof (Sketch). The actual proof is long and technical, and is omitted here for space reasons. We proceed by first defining a special subset of terms, called *neutral* terms, for which the Theorem is obvious. We then prove that every term is axiomatically equivalent to a neutral term via a series of rewrite systems. □

3 Linear category for duplication

As it was advertised, the structure of the categorical semantics will closely follow the one proposed by Bierman [8], but with the added twist of a computational monad à la Moggi [14]. Indeed, since one has tensor products and a tensor unit, one can expect the categorical model to be symmetric monoidal. Since one can construct candidate maps for building a comonad, a comonoid structure for each $!A$ and coherence maps for the comonad, we have a linear category. Finally, the computational aspect will be taken care by Moggi's computational monad.

3.1 Linear exponential comonads

In his Ph.D. thesis, Bierman [8] gives the definition of a *linear category*. We prefer here the terminology given in [15], and use the concept of *linear exponential comonad*.

Definition 3. Let $(\mathcal{C}, \otimes, \top)$ be a symmetric monoidal category [13], where $\alpha_{A,B,C} : A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, $\lambda_A : \top \otimes A \to A$, $\rho_A : A \otimes \top \to A$ and $\sigma_{A,B} : A \otimes B \to B \otimes A$ are the usual associativity, left unit, right unit and symmetry morphisms. Let $(L, \delta, \epsilon, d^L, d^L)$ be a monoidal comonad [8], where $\epsilon_A : LA \to A$, $\delta_A : LA \to LLA$, $d^L_{A,B} : LA \otimes LB \to L(A \otimes B)$ and $d^L_\top : \top \to L\top$. We say that L is a *linear exponential comonad* [15] provided that

1. each object in $\mathcal{C}$ of the form LA is equipped with a commutative comonoid $(LA, \triangle_A, \diamondsuit_A)$, where $\triangle_A : LA \to LA \otimes LA$ and $\diamondsuit_A : LA \to \top$;
2. $\triangle_A$ and $\diamondsuit_A$ are monoidal natural transformations;
3. $\triangle_A : (LA, \delta_A) \to (LA \otimes LA, (\delta_A \otimes \delta_A); d_A)$ and $\diamondsuit_A : (LA, \delta_A) \to (\top, d^L_\top)$ are L-coalgebra morphisms;
4. Every map δ_A is a comonoid morphism $(LA, \diamondsuit_A, \triangle_A) \to (L^2A, \diamondsuit_{LA}, \triangle_{LA})$.

The equations for 2–4 are to be found in Table 4.

3.2 Strong monad and *T*-exponentials.

To capture the computational effect of the probabilistic measurement, we use the notion of *strong monad*, as in [14]. Recall that a *monad* over a category $\mathcal{C}$ is a triple (T, η, μ) where $T : \mathcal{C} \to \mathcal{C}$ is a functor, $\eta : id \dot{\to} T$ and $\mu : T^2 \dot{\to} T$ are natural transformations and such that $T\mu_A; \mu_A = \mu_{TA}; \mu_A$ and $\eta_{TA}; \mu_A = id_{TA} = T\eta_A; \mu_A$. Given a map $f : A \to TB$, we define the map $f^* : TA \to TB$ by $Tf; \mu_B$.

$$
\begin{tikzcd}
LA \otimes LB \arrow[r, "\triangle_A \otimes \triangle_B"] \arrow[dd, "d^L_{A,B}"'] & (LA \otimes LA) \otimes (LB \otimes LB) \arrow[d, "sw"] \\
& (LA \otimes LB) \otimes (LA \otimes LB) \arrow[d, "d^L_{A,B} \otimes d^L_{A,B}"] \\
L(A \otimes B) \arrow[r, "\triangle_{(A \otimes B)}"'] & L(A \otimes B) \otimes L(A \otimes B)
\end{tikzcd}
\quad
\begin{tikzcd}
\top \arrow[r, "\lambda_\top^{-1}"] \arrow[d, "d^L_\top"'] & \top \otimes \top \arrow[d, "d^L_\top \otimes d^L_\top"] \\
L\top \arrow[r, "\triangle_\top"'] & L\top \otimes L\top
\end{tikzcd}
\quad
\begin{tikzcd}
LA \otimes LB \arrow[r, "\Diamond_A \otimes \Diamond_B"] \arrow[d, "d^L_{A,B}"'] & \top \otimes \top \arrow[d, "\lambda_\top"] \\
L(A \otimes B) \arrow[r, "\Diamond_{A \otimes B}"'] & \top
\end{tikzcd}
\quad
\begin{tikzcd}
\top \arrow[rr, "id"] \arrow[dr, "d^L_\top"'] & & \top; \\
& L\top \arrow[ur, "\Diamond_\top"'] &
\end{tikzcd}
$$

$\triangle_A$ and $\Diamond_A$ are monoidal natural transformations.

$$
\begin{tikzcd}
LA \arrow[r, "\triangle_A"] \arrow[dd, "\delta_A"'] & LA \otimes LA \arrow[d, "\delta_A \otimes \delta_A"] \\
& L^2A \otimes L^2A \arrow[d, "d^L_{LA,LA}"] \\
L^2A \arrow[r, "L\triangle_A"'] & L(LA \otimes LA),
\end{tikzcd}
\quad
\begin{tikzcd}
LA \arrow[r, "\Diamond_A"] \arrow[d, "\delta_A"'] & \top \arrow[d, "d^L_\top"] \\
L^2A \arrow[r, "L\Diamond_A"'] & L\top;
\end{tikzcd}
\quad
\begin{tikzcd}
LA \arrow[r, "\delta_A"] \arrow[d, "\triangle_A"'] & L^2A, \arrow[d, "\triangle_{LA}"] \\
LA \otimes LA \arrow[r, "\delta_A \otimes \delta_A"'] & L^2A \otimes L^2A
\end{tikzcd}
\quad
\begin{tikzcd}
LA \arrow[rr, "\delta_A"] \arrow[dr, "\Diamond_A"'] & & L^2A. \arrow[dl, "\Diamond_{LA}"] \\
& \top &
\end{tikzcd}
$$

$\triangle_A$ and $\Diamond_A$ are L-coalgebra maps. δ_A is a comonoid morphism.

Table 4. Equations for a linear exponential comonad

Definition 4. A *strong monad* over a monoidal category $\mathcal{C}$ is a monad (T, η, μ) together with a natural transformation $t_{A,B} : A \otimes TB \to T(A \otimes B)$, called the *tensorial strength*, subject to a number of coherence conditions.

Remark 1. If the category $\mathcal{C}$ is symmetric, the tensorial strength t induces two natural transformations $TA \otimes TB \to T(A \otimes B)$, namely

$$
\begin{aligned}
\Psi_1 : TA \otimes TB &\xrightarrow{\sigma_{TA,TB}} TB \otimes TA \xrightarrow{t_{TB,A}} T(TB \otimes A) \xrightarrow{(\sigma_{TB,A};t_{A,B})^*} T(A \otimes B), \\
\Psi_2 : TA \otimes TB &\xrightarrow{t_{TA,B}} T(TA \otimes B) \xrightarrow{(\sigma_{TA,B};t_{B,A})^*} T(B \otimes A) \xrightarrow{T\sigma_{B,A}} T(A \otimes B).
\end{aligned}
$$

Note that Ψ_1 and Ψ_2 might not be equal: the map Ψ_1 "evaluates" the first variable and then the second one. The map Ψ_2 does the opposite. The strength is called *commutative* if $\Psi_1 = \Psi_2$.

Lemma 7. *If (T, η, μ, t) is a commutative strong monad on a symmetric monoidal category $\mathcal{C}$, then (T, η, μ, Ψ_1) and (T, η, μ, Ψ_2) are monoidal monads.* □

Definition 5. A symmetric monoidal category $(\mathcal{C}, \otimes, \top)$ together with a strong monad (T, η, μ) is said to have *T-exponentials* [14], or *Kleisli exponentials*, if it is equipped with a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, and a natural isomorphism

$$\Phi : \mathcal{C}(A, B \multimap C) \xrightarrow{\cong} \mathcal{C}(A \otimes B, TC).$$

Lemma 8. *The map Φ induces a natural transformation $\varepsilon_{A,B} : (A \multimap B) \otimes A \to TB$ defined by $\Phi(id_{A \multimap B})$.* □

3.3 Idempotent, strong monoidal comonad

A comonad (L, ϵ, δ) on some category is said to be *idempotent* if $\delta : L \dot{\to} LL$ is an isomorphism. A monoidal comonad $(L, \delta, \epsilon, d^L, d^L)$ is *strong monoidal* if $d^L_\top$ and $d^L_{A,B}$ are isomorphisms.

Definition 6. Given a monoidal category $(\mathcal{C}, \otimes, \top)$ with an idempotent, strong monoidal comonad (L, ϵ, δ), a bifunctor $\multimap : \mathcal{C}^{op} \times \mathcal{C} \to \mathcal{C}$, we define a *canonical arrow for* $\mathcal{C}$ *with respect to duplication* by induction: For all objects A, the arrows id_A, ϵ_A, δ_A, $d^L_\top$ and $d^L_{A,B}$ are canonical. All *expansions of canonical arrows with respect to duplication* are also canonical. An expansion of an arrow $f : A \to B$ is defined to be either f or any of Lg,$X \otimes g$, $g \otimes X$, $X \multimap g$, $g \multimap X$, where g is an expansion of f and X ranges over the objects of the category. Finally, compositions of canonical arrows are also canonical.

Theorem 2 (Coherence for idempotent comonads). *Given a category* $\mathcal{C}$ *with the structure in Definition 6, if* $f, g : A \to B$ *are two canonical arrows with respect to duplication, then they are equal.* □

3.4 Linear category for duplication

We now have enough background to define a candidate for the categorical model of the language we describe in Section 2.

Definition 7. A *linear category for duplication* is a category $\mathcal{C}$ with the following structure:

- a symmetric monoidal structure $(\otimes, \top, \alpha, \lambda, \rho, \sigma)$;
- an idempotent, strongly monoidal, linear exponential comonad $(L, \delta, \epsilon, d^L, d^L, \diamondsuit, \triangle)$;
- a strong monad (T, μ, η, t);
- a Kleisli exponential $\multimap$.

The *computational linear category* is defined to be the Kleisli category $\mathcal{C}_T$, as defined in [14].

Remark 2. A linear category for duplication gives rise to a double adjunction

$$\begin{tikzcd} \mathcal{C}_L \arrow[r, bend left, "U^L"] \arrow[r, phantom, "\perp"] & \mathcal{C} \arrow[l, bend left, "F^L"] \arrow[r, bend left, "U^T"] \arrow[r, phantom, "\perp"] & \mathcal{C}_T \arrow[l, bend left, "F^T"] \end{tikzcd},.$$

Here the left adjunction arises from the co-Kleisli category $\mathcal{C}_L$ of the comonad L. It is as in the linear-non-linear models of [4], and $\mathcal{C}_L$ is a category of classical (non-quantum) values. The right adjunction arises from the Kleisli category $\mathcal{C}_T$ of the computational monad T, as in [14]. Here $\mathcal{C}_T$ is a category of (effectful) quantum computations.

3.5 The category $\mathcal{C}_\lambda$

Definition 8. We can define a category $\mathcal{C}_\lambda$ as follows: Objects are types, and arrows $A \to B$ are axiomatic classes of valid typing judgments of the form $x : A \rhd V : B$, where V is a value. We define the composition of arrows $x : A \rhd V : B$ and $y : B \rhd W : C$ to be $x : A \rhd \mathit{let}\ y = V\ \mathit{in}\ W : C$. The identity on A is set to be the arrow $x : A \rhd x : A$.

Lemma 9. *The category* $\mathcal{C}_\lambda$ *is well-defined.*

$\alpha_{A,B,C} =$	$x : A\otimes(B\otimes C) \rhd let\langle y,z\rangle = x\ in\ let\langle t,u\rangle = z\ in\ \langle\langle y,t\rangle,u\rangle$	$: (A\otimes B)\otimes C$
$\lambda_A =$	$x : \top\otimes A \rhd let\langle y,z\rangle = x\ in\ let * = y\ in\ z$	$: A$
$\rho_A =$	$x : A\otimes\top \rhd let\langle y,z\rangle = x\ in\ let * = z\ in\ y$	$: A$
$\sigma_{A,B} =$	$x : A\otimes B \rhd let\langle y,z\rangle = x\ in\ \langle z,y\rangle$	$: B\otimes A$
$\eta_A =$	$x : A \rhd \lambda *.x$	$: \top\multimap A$
$\mu_A =$	$x : \top\multimap(\top\multimap A) \rhd \lambda *.(x*)*$	$: \top\multimap A$
$t_{A,B} =$	$z : A\otimes(\top\multimap B) \rhd let\langle x,y\rangle = z\ in\ \lambda *.\langle x,y*\rangle$	$: \top\multimap(A\otimes B)$
$\epsilon_A =$	$x : !A \rhd x^A$	$: A$
$\delta_A =$	$x : !A \rhd x^{!^2 A}$	$: !^2 A$
$d^!_{A,B} =$	$z : !A\otimes !B \rhd let\langle x,y\rangle = z\ in\ \langle x,y\rangle$	$: !(A\otimes B)$
$d^!_\top =$	$z : \top \rhd let * = z\ in\ *$	$: !\top$
$\triangle_A =$	$x : !A \rhd \langle x,x\rangle$	$: !A\otimes !A$
$\diamondsuit_A =$	$x : !A \rhd *$	$: \top$

$(x : A \rhd V : B)\otimes(y : C \rhd W : D) =$	$z : A\otimes B \rhd let\langle x,y\rangle = z\ in\ \langle V,W\rangle$	$: C\otimes D$
$(x : A \rhd V : B)\multimap(y : C \rhd W : D) =$	$z : B\multimap C \rhd \lambda x.(let\ y = zV\ in\ W)$	$: A\multimap D$
$(x : A \rhd V : \top\multimap B)^* =$	$y : \top\multimap A \rhd \lambda *.\ let\ \ x = (y*)\ in\ (V*)$	$: \top\multimap B$
$\Phi_{A,B,C}\,(x : A \rhd V : B\multimap C) =$	$t : A\otimes B \rhd \lambda *.\ let\ \langle x,y\rangle = t\ in\ Vy$	$: \top\multimap C$

Table 5. Definitions of maps and operations on maps in $\mathcal{C}_\lambda$

Proof. The composition of two arrows yields an arrow axiomatically equivalent to a value due to Axiom (β_λ) and Lemma 5. Composition is associative due to Axiom (let_1). The arrow $x : A \rhd x : A$ is indeed the identity on A due to axioms (α_{let}) and (β^2_λ). $\square$

Lemma 10. *Given a valid typing judgment $\Delta \rhd V : A$ where V is a value, there exists a canonical value V' such that $Erase(V') = Erase(V)$ and such that $!\Delta \rhd V' : !A$. We denote this V' by $\{!\Delta \lessdot \Delta \rhd V : A \gtrdot A'\}$.*

Proof. By induction on V. $\square$

Lemma 11. *If $\Delta \rhd V \approx_{ax} W : A$, and if $V' = \{!\Delta \lessdot \Delta \rhd V : A \gtrdot A'\}$ and $W' = \{!\Delta \lessdot \Delta \rhd W : A \gtrdot A'\}$, then $V' \approx_{ax} W'$.*

Proof. By induction on $V \approx_{ax} W$. $\square$

Theorem 3. *If we define $T(A) := \top \multimap A$ and $L(A) =!A$, together with the maps and the operations on maps defined in Table 5, $\mathcal{C}_\lambda$ is a linear category for duplication.*

Proof. The proof is mainly a long list of verifications. It uses Theorem 1, Lemmas 9, 10 and 11. $\square$

4 Denotational semantics

4.1 Interpretation of the language

The lambda-calculus defined in Section 2 is thought as a *computational* lambda-calculus. Using Moggi's technique, we split the interpretation of the language into the interpretation of the *values* in a linear category for duplication $\mathcal{C}$ and the interpretation of the *computations*, i.e. general terms, in its Kleisli category $\mathcal{C}_T$. Without loss of generality, for notation purposes, we assume the category to be strictly monoidal.

We define an *interpretation of the type system* to be a map Θ that assigns to each constant type α an object $\Theta(\alpha)$. Each type A is interpreted as an object of $\mathcal{C}$: $[\![\alpha]\!]_\Theta = \Theta(\alpha)$, $[\![\top]\!]_\Theta = \top$, $[\![!A]\!]_\Theta = L[\![A]\!]_\Theta$, $[\![A \otimes B]\!]_\Theta = [\![A]\!]_\Theta \otimes [\![B]\!]_\Theta$ and $[\![A \multimap B]\!]_\Theta = [\![A]\!]_\Theta \multimap [\![B]\!]_\Theta$.

Given a valid subtyping $A <: B$, there exists a canonical arrow $[\![A]\!]_\Theta \to [\![B]\!]_\Theta$ in $\mathcal{C}$ with respect to duplication, as defined in Definition 6. Moreover, this arrow is unique by Theorem 2. We extend the map Θ to interpret $A <: B$ as this unique arrow and we denote it by $I_{A,B}$.

We use the following straightforward shortcut definitions, where A, A', B, B' are types and Δ, Γ and Γ' are typing contexts:

- $\mathit{Split}_{!\Delta,\Gamma,\Gamma'} : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![\Gamma']\!] \to [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![!\Delta]\!] \otimes [\![\Gamma']\!]$.
- Given $f : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \to [\![A]\!]$ and $g : [\![!\Delta]\!] \otimes [\![\Gamma']\!] \to [\![B]\!]$, we define the map $f \otimes_{!\Delta} g : [\![!\Delta]\!] \otimes [\![\Gamma]\!] \otimes [\![\Gamma']\!] \to A \otimes B$.
- Given a natural transformation $n_A : FA \to GA$, if $\Delta = \{x_1 : A_1 \dots x_n : A_n\}$ we define $n_\Delta = n_{[\![A_1]\!]} \otimes \dots \otimes n_{[\![A_n]\!]}$.

Definition 9. The map Θ is said to be an *interpretation of the language* if moreover it assigns to each constant term $c : A_c$ an arrow $\Theta(c) : \top \to [\![A_c]\!]$ in $\mathcal{C}$.

Given a linear category for duplication $\mathcal{C}$, it is possible to interpret the typing derivation of a well-typed value as a map in $\mathcal{C}$ and the typing derivation of a valid computation as a map in the Kleisli category $\mathcal{C}_T$. We define them inductively.

- If $x_1 : A_1, \dots x_n : A_n \rhd V : B$ is a value with typing derivation π, its *value interpretation* $[\![\pi]\!]^v_\Theta$ is an arrow $[\![A_1]\!] \otimes \dots \otimes [\![A_n]\!] \to_{\mathcal{C}} [\![B]\!]$;
- if $x_1 : A_1, \dots x_n : A_n \rhd M : A$ is a term with typing derivation π, its *computational interpretation* $[\![\pi]\!]^c_\Theta$ is an arrow $[\![A_1]\!] \otimes \dots \otimes [\![A_n]\!] \to_{\mathcal{C}} T([\![B]\!])$.

Table 6 formulates the definition in the simple case where the contexts Δ, Γ_1 and Γ_2 contain only one variable. One can easily extend this to the general setting.

As we already noted in Section 2.4, a valid typing judgment does not have a unique typing tree *per se*. However the following result holds:

Theorem 4. *Given a valid typing judgment with two typing derivations π and π', for any interpretation Θ we have $[\![\pi]\!]^c_\Theta = [\![\pi']\!]^c_\Theta$ (and $[\![\pi]\!]^v_\Theta = [\![\pi']\!]^v_\Theta$ if the typing judgment is a value).*

Proof. The proof is done by showing that given any typing judgment $\Delta \rhd M : A$ with denotation f one can factor f as $\diamondsuit_{!\Gamma} \otimes \bar{f}$, where $\bar{f}$ is the denotation of $\Delta' \rhd M : A$, where $\Delta', !\Gamma = \Delta$ and $|\Gamma|$ is the set of dummy variables. $\square$

Definition 10. Given a interpretation Θ of the language in a category $\mathcal{C}$, we define the denotation of a valid typing judgment $\Delta \rhd M : A$ with typing derivation π to be $[\![\Delta \rhd M : A]\!]^c_\Theta = [\![\pi]\!]^c_\Theta$ and $[\![\Delta \rhd M : A]\!]^v_\Theta = [\![\pi]\!]^v_\Theta$ if M is a value.

Interpretation of core values:

$$\llbracket !\Delta, x : A \rhd x : B \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{\Diamond_\Delta \otimes I_{A,B}} \llbracket B \rrbracket$$

$$\llbracket !\Delta \rhd c : B \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \xrightarrow{\Diamond_\Delta} \top \xrightarrow{\Theta(c)} \llbracket A_c \rrbracket \xrightarrow{I_{A_c,B}} \llbracket B \rrbracket$$

$$\llbracket !\Delta \rhd * : !^n\top \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \xrightarrow{\Diamond_\Delta} \top \xrightarrow{d^L_\top} L\top \xrightarrow{I_{!\top,!^n\top}} L^n\top$$

$$\frac{\llbracket \Delta, x : A \rhd M : B \rrbracket^c_\Theta = \llbracket \Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{f} T(\llbracket B \rrbracket)}{\llbracket \Delta \rhd \lambda x.M : A \multimap B \rrbracket^v_\Theta = \llbracket \Delta \rrbracket \xrightarrow{\Phi^{-1}(f)} \llbracket A \rrbracket \multimap \llbracket B \rrbracket}$$

$$\frac{\llbracket !\Delta, x : A \rhd M : B \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{f} T(\llbracket B \rrbracket)}{\llbracket !\Delta \rhd \lambda x.M : !^{n+1}(A \multimap B) \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \xrightarrow{L(\Phi^{-1}f);I_{!(A\multimap B),!^{n+1}(A\multimap B)}} L^{n+1}(\llbracket A \rrbracket \multimap \llbracket B \rrbracket)}$$

Interpretation of extended values:

$$\frac{\llbracket !\Delta, \Gamma_1 \rhd V : A \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} \llbracket A \rrbracket \qquad \llbracket !\Delta, \Gamma_2, x : A \rhd W : B \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{g} \llbracket B \rrbracket}{\llbracket !\Delta, \Gamma_2, \Gamma_1 \rhd let\ x = V\ in\ W : B \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{id \otimes_{!\Delta} f} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket A \rrbracket \xrightarrow{g} \llbracket B \rrbracket}$$

$$\frac{\begin{array}{l}\llbracket !\Delta, \Gamma_1, \rhd V : !^n(A_1 \otimes A_2) \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} L^n(\llbracket A_1 \rrbracket \otimes \llbracket A_2 \rrbracket) \\ \llbracket !\Delta, \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd W : C \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes L^n\llbracket A_1 \rrbracket \otimes L^n\llbracket A_2 \rrbracket \xrightarrow{g} \llbracket C \rrbracket\end{array}}{\begin{array}{l}\llbracket !\Delta, \Gamma_2, \Gamma_1 \rhd let\langle x, y\rangle^n = V\ in\ W : C \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{id \otimes_{!\Delta} f} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes L^n(\llbracket A_1 \rrbracket \otimes \llbracket A_2 \rrbracket) \\ \qquad \xrightarrow{id \otimes \left(d^{L^n}_{\llbracket A_1 \rrbracket, \llbracket A_2 \rrbracket}\right)^{-1}} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes L^n\llbracket A_1 \rrbracket \otimes L^n\llbracket A_2 \rrbracket \xrightarrow{g} \llbracket C \rrbracket\end{array}}$$

$$\frac{\llbracket !\Delta, \Gamma_2 \rhd V : \top \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{f} \top \qquad \llbracket !\Delta, \Gamma_1 \rhd W : C \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{g} \llbracket C \rrbracket}{\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd let\ * = V\ in\ W : C \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{id \otimes_{!\Delta} f} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{g} \llbracket C \rrbracket}$$

$$\frac{\llbracket !\Delta, \Gamma_1 \rhd V : !^n A \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} L^n\llbracket A \rrbracket \qquad \llbracket !\Delta, \Gamma_2 \rhd W : !^n B \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{g} L^n\llbracket B \rrbracket}{\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd \langle V, W\rangle^n : !^n(A \otimes B) \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{f \otimes_{!\Delta} g} L^n\llbracket A \rrbracket \otimes L^n\llbracket B \rrbracket \xrightarrow{d^{L^n}_{A,B}} L^n(\llbracket A \rrbracket \otimes \llbracket B \rrbracket)}$$

Interpretation of computations: First, if U is a core value, $\llbracket \Delta \rhd U : A \rrbracket^c_\Theta = \llbracket \Delta \rhd U : A \rrbracket^v_\Theta; \eta_A$.

$$\frac{\llbracket !\Delta, \Gamma_1 \rhd M : A \multimap B \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} T(\llbracket A \rrbracket \multimap \llbracket B \rrbracket) \qquad \llbracket !\Delta, \Gamma_2 \rhd N : A \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{g} T(\llbracket A \rrbracket)}{\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd MN : B \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{f \otimes_{!\Delta} g} T(\llbracket A \rrbracket \multimap \llbracket B \rrbracket) \otimes T(\llbracket A \rrbracket) \xrightarrow{\Psi_1} T((\llbracket A \rrbracket \multimap \llbracket B \rrbracket) \otimes \llbracket A \rrbracket) \xrightarrow{\varepsilon^*_{A,B}} T(\llbracket B \rrbracket)}$$

$$\frac{\begin{array}{l}\llbracket !\Delta, \Gamma_1 \rhd M : !^n(A_1 \otimes A_2) \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} TL^n(\llbracket A_1 \rrbracket \otimes \llbracket A_2 \rrbracket) \\ \llbracket !\Delta, \Gamma_2, x : !^n A_1, y : !^n A_2 \rhd N : C \rrbracket^v_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes L^n\llbracket A_1 \rrbracket \otimes L^n\llbracket A_2 \rrbracket \xrightarrow{g} T(\llbracket C \rrbracket)\end{array}}{\begin{array}{l}\llbracket !\Delta, \Gamma_2, \Gamma_1 \rhd let\langle x, y\rangle^n = M\ in\ N : !^n C \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{id \otimes_{!\Delta} f} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes TL^n(\llbracket A_1 \rrbracket \otimes \llbracket A_2 \rrbracket) \\ \qquad \xrightarrow{t;T\left(id \otimes \left(d^{L^n}\right)^{-1}\right)} T(\llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes L^n\llbracket A_1 \rrbracket \otimes L^n\llbracket A_2 \rrbracket) \xrightarrow{g^*} T\llbracket C \rrbracket\end{array}}$$

$$\frac{\llbracket !\Delta, \Gamma_2 \rhd M : \top \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{f} T(\top) \qquad \llbracket !\Delta, \Gamma_1 \rhd N : C \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{g} T(\llbracket C \rrbracket)}{\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd let\ * = M\ in\ N : C \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{id \otimes_{!\Delta} f} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes T(\top) \xrightarrow{t;g^*} T(\llbracket C \rrbracket)}$$

$$\frac{\llbracket !\Delta, \Gamma_1 \rhd M : !^n A \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \xrightarrow{f} TL^n\llbracket A \rrbracket \qquad \llbracket !\Delta, \Gamma_2 \rhd N : !^n B \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{g} TL^n\llbracket B \rrbracket}{\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd \langle M, N\rangle^n : !^n(A \otimes B) \rrbracket^c_\Theta = \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \xrightarrow{f \otimes_{!\Delta} g} TL^n\llbracket A \rrbracket \otimes TL^n\llbracket B \rrbracket \xrightarrow{\Psi_1;Td^{L^n}_{A,B}} TL^n(\llbracket A \rrbracket \otimes \llbracket B \rrbracket)}$$

Table 6. Interpretation of values and computations.

Lemma 12. *Suppose that $\Delta \rhd V : A$ is a valid typing judgment where V is a value. Then $\llbracket \Delta \rhd V : A \rrbracket^c = \llbracket \Delta \rrbracket \xrightarrow{\llbracket \Delta \rhd V : A \rrbracket^v} \llbracket A \rrbracket \xrightarrow{\eta_{\llbracket A \rrbracket}} T(\llbracket A \rrbracket)$.*

Proof. Proof by induction on V, using the bifunctoriality of $\otimes_{LA}$ and the equations for strong monadicity in Definition 4. $\square$

4.2 Soundness of the denotation

The axiomatic equivalence and the categorical semantics are two faces of the same coin. Indeed, as we will prove in this section, two terms in the same axiomatic equivalence class have the same denotation. A corollary is that the indexation of terms does not influence the denotation. This proves semantically the fact that it is safe to work with untyped terms. An alternate justification of this fact is of course the operational semantics, which was given in [19].

Lemma 13. *Suppose* $M' = \{\Delta' <: \Delta \rhd M : A <: A'\}$. *Then* $\llbracket \Delta' \rhd M' : A' \rrbracket^c = I_{\Delta',\Delta}; \llbracket \Delta \rhd M : A \rrbracket^c; T(I_{A,A'})$. *If* $M = V$ *is a value, from Lemma 4,* $M' = V'$ *is a value. Then* $\llbracket \Delta' \rhd V' : A' \rrbracket^v = I_{\Delta',\Delta}; \llbracket \Delta \rhd V : A \rrbracket^v; I_{A,A'}$.

Proof. Proof by structural induction on M. □

Lemma 14 (Substitution). *Given two valid typing judgments* $!\Delta, \Gamma_1, x : A \rhd M : B$ *and* $!\Delta, \Gamma_2 \rhd V : A$, *the typing judgment* $!\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B$ *is valid. Let* h *be* $\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd M[V/x] : B \rrbracket^c$ *and* h' *be* $\llbracket !\Delta, \Gamma_1, \Gamma_2 \rhd W[V/x] : B \rrbracket^v$, *in the case where* $M = W$ *is a value. Then they are defined by*

$$\begin{tikzcd} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \arrow[r, dotted, "h"] \arrow[d, "Split_{!\Delta,\Gamma_1,\Gamma_2}"] & T(\llbracket B \rrbracket) \\ \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \arrow[r, "id \otimes \llbracket !\Delta, \Gamma_2 \rhd V : A \rrbracket^v"] & \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket A \rrbracket, \arrow[u, "\llbracket !\Delta, \Gamma_1, x:A \rhd M:B \rrbracket^c"'] \end{tikzcd} \quad \begin{tikzcd} \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \arrow[r, dotted, "h'"] \arrow[d, "Split_{!\Delta,\Gamma_1,\Gamma_2}"] & \llbracket B \rrbracket \\ \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_2 \rrbracket \arrow[r, "id \otimes \llbracket !\Delta, \Gamma_2 \rhd V : A \rrbracket^v"] & \llbracket !\Delta \rrbracket \otimes \llbracket \Gamma_1 \rrbracket \otimes \llbracket A \rrbracket. \arrow[u, "\llbracket !\Delta, \Gamma_1, x:A \rhd W:B \rrbracket^v"'] \end{tikzcd}$$

Proof. Proof by induction on M, using Lemma 1, Lemma 12 and the naturality of Φ. □

Theorem 5. *If* $\Delta \rhd M \approx_{ax} M' : A$ *then* $\llbracket \Delta \rhd M : A \rrbracket^c_\Theta = \llbracket \Delta \rhd M' : A \rrbracket^c_\Theta$ *(and* $\llbracket \Delta \rhd M : A \rrbracket^v_\Theta = \llbracket \Delta \rhd M' : A \rrbracket^v_\Theta$ *if* M *is a value) for every interpretation* Θ.

Proof. Proof by induction on $M \approx_{ax} M'$, using Lemmas 13 and 14. □

Corollary 1. *If* $Erase(M) = Erase(M')$ *and if* $\Delta \rhd M, M' : A$ *are valid typing judgments, then* $\llbracket M \rrbracket^c = \llbracket M' \rrbracket^c$ *(and* $\llbracket M \rrbracket^v = \llbracket M' \rrbracket^v$ *if* M *is a value).*

Proof. Corollary of Theorems 1 and 5. □

4.3 Completeness

The category $\mathcal{C}_\lambda$ being a linear category for duplication, one can interpret the language in it. This section states that the defined lambda-calculus is an *internal language* of linear categories for duplication.

Since the category $\mathcal{C}_\lambda$ is a monoidal category, one can w.l.o.g. generalize the notion of pairing to finite tensor products of terms. Then the following results are true:

Lemma 15. *In* $\mathcal{C}_\lambda$, *a valid typing judgment* $x_1 : A_1, \ldots x_n : A_n \rhd M : B$ *has for computational denotation* $(t : A_1 \otimes \cdots \otimes A_n \rhd let \langle x_1, \ldots x_n \rangle = t \; in \; \lambda *.M : \top \multimap B)$. *If* $M = V$ *is a value, the value interpretation is* $(t : A_1 \otimes \cdots \otimes A_n \rhd let \langle x_1, \ldots x_n \rangle = t \; in \; V : B)$.

Proof. Proof by structural induction on M and V. □

Theorem 6. *In* $\mathcal{C}_\lambda$, Θ *being the identity, one has* $\llbracket x : A \rhd M : B \rrbracket^c_\Theta \approx_{ax} (x : A \rhd \lambda *.M : \top \multimap B)$ *and* $\llbracket x : A \rhd V : B \rrbracket^v_\Theta \approx_{ax} (x : A \rhd V : B)$.

Proof. Corollary of Lemma 15 □

5 Towards a denotational model of quantum lambda calculus

As noted in the introduction, this paper is mostly concerned with the categorical requirements for modeling a generic call-by-value linear lambda calculus, i.e., its type system (which includes subtyping) and equational laws. We have not yet specialized the language to a particular set of built-in operators, for example, those that are required for quantum computation.

However, since the quantum lambda calculus [19] is the main motivation behind our work, we will comment very briefly on what additional properties would be required to interpret its primitives. The quantum lambda calculus is obtained by instantiating and extending the call-by-value language of this paper with the following primitive types, constants, and operations:

Types: bit, $qbit$

Constants: $0 : !bit$, $1 : !bit$

$new : !(bit \multimap qbit)$, $U : !(qbit^n \multimap qbit^n)$, $meas : !(qbit \multimap !bit)$

Operations: $$\frac{\Gamma_1, !\Delta \rhd P : bit \quad \Gamma_2, !\Delta \rhd M : A \quad \Gamma_2, !\Delta \rhd N : A}{\Gamma_1, \Gamma_2, !\Delta \rhd \mathit{if}\ P\ \mathit{then}\ M\ \mathit{else}\ N : A}\ (\mathit{if})$$

Here, U ranges over a set of built-in unitary operations. In the intended semantics, $!bit \cong bit$, while $!qbit$ is empty. *new* creates a new qubit, and *meas* measures a qubit.

The denotational semantics of these operations is already well-understood in the absence of higher-order types. They can all be interpreted in the category **Q** of superoperators from [18]. The part that is not yet well-understood is how these features interact with higher-order types.

In light of our present work, we can conclude that a model of the quantum lambda calculus consists of a linear category for duplication $(\mathcal{C}, L, T, \multimap)$, such that the associated category of computations $\mathcal{C}_T$ contains the category **Q** of [18] as a full monoidal subcategory. To construct an actual instance of such a model is still an open problem.

6 Conclusion

We have developed a call-by-value, computational lambda-calculus for manipulating duplicable and non-duplicable data, together with an axiomatic equivalence relation on typed terms. We use a subtyping relation in order to have implicit promotion, dereliction, copying and discarding. Then we developed categorical model for the language, inspired by the work of [8] and [14]. We finally showed that the model is sound and complete with respect to the axiomatic equivalence.

References

1. Abramsky, S.: Computational interpretations of linear logic. Theoretical Computer Science **111** (1993) 3–57

2. Abramsky, S., Coecke, B.: A categorical semantics of quantum protocols. In: Proceedings of LICS'04. (2004) 415–425
3. Barendregt, H.P.: The Lambda-Calculus, its Syntax and Semantics. North Holland (1984)
4. Benton, N.: A mixed linear and non-linear logic: Proofs, terms and models (extended abstract). In: Proceedings of CSL'94, Selected Papers. Volume 933 of Lecture Notes in Computer Science. (1994) 121–135
5. Benton, N., Bierman, G., de Paiva, V.C.V., Hyland, M.: A term calculus for intuitionistic linear logic. In: Proceedings of TLCA'93. Volume 664 of Lecture Notes in Computer Science. (1993) 75–90
6. Benton, N., Bierman, G., Hyland, M., de Paiva, V.C.V.: Linear lambda-calculus and categorical models revisited. In: Proceedings of CSL'92, Selected Papers. Volume 702 of Lecture Notes in Computer Science. (1992)
7. Benton, N., Wadler, P.: Linear logic, monads and the lambda calculus. In: Proceedings of LICS'96. (1996) 420–431
8. Bierman, G.: On Intuitionistic Linear Logic. PhD thesis, Computer Science Department, Cambridge University (1993)
9. Coecke, B.: Quantum information-flow, concretely, abstractly. [17] 57–73
10. Coecke, B., Pavlovic, D.: Quantum measurements without sums. In Chen, G., Kauffman, L., Lomonaco, S.J., eds.: Mathematics of Quantum Computation and Technology. Chapman & Hall (2007) 559–596
11. Gay, S.J., Nagarajan, R.: Communicating quantum processes. In: Proceedings of POPL'05, ACM Press (2005)
12. Lalire, M., Jorrand, P.: A process algebraic approach to concurrent and distributed computation: operational semantics. [17] 109–126
13. Mac Lane, S.: Categories for the Working Mathematician. Springer Verlag (1998)
14. Moggi, E.: Notions of computation and monads. Information and Computation **93** (1991) 55–92
15. Schalk, A.: What is a model for linear logic. Manuscript (2004)
16. Seely, R.A.G.: *-autonomous categories and cofree coalgebras. Contemporary Mathematics **92** (1989)
17. Selinger, P., ed.: Proceedings of QPL'04. TUCS General Publication No 33, Turku Centre for Computer Science (2004)
18. Selinger, P.: Towards a quantum programming language. Mathematical Structures in Computer Science **14** (2004) 527–586
19. Selinger, P., Valiron, B.: A lambda calculus for quantum computation with classical control. Mathematical Structures in Computer Science **16** (2006) 527–552
20. Selinger, P., Valiron, B.: On a fully abstract model for a quantum linear functional language. In: Preliminary proceedings of QPL'06. (2006) 103–115
21. van Tonder, A.: A lambda calculus for quantum computation. SIAM Journal of Computing **33** (2004) 1109–1135
22. Wadler, P.: There's no substitute for linear logic. Manuscript, presented at MFPS'92 (1992)
23. Wootters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature **299** (1982) 802–803