On a fully abstract model for a quantum linear functional language (extended abstract)

Peter Selinger[1,2]

Dalhousie University, Halifax, Nova Scotia, Canada

Benoît Valiron[3]

University of Ottawa, Ottawa, Ontario, Canada

Abstract

This paper studies the linear fragment of the programing language for quantum computation with classical control described in [4]. We sketch the language, and discuss equivalence of terms. We also describe a fully abstract denotational semantics based on completely positive maps.

Keywords: Quantum computing, functional programming, linear lambda calculus, higher order, semantics.

1 Introduction

This work studies a linear functional programming language for quantum computation with classical control, derived from the language in [4].

The first denotational semantics of a quantum programming language was given by the first author in [3], for the quantum flowchart language QFC. The semantics given there was compositional and took place in a category of superoperators, which are special completely positive maps. However, the language lacked a crucial feature found in functional programming languages, namely, the notion of higher-order functions.

In [4], we sought to address this omission by introducing a typed lambda calculus for quantum computation. This language resembles QFC in that it combines quantum and classical data types with classical control features, but it also includes lambda abstractions and therefore function closures. The problem of deciding the

[1] Research supported by NSERC.
[2] Email: `selinger@mathstat.dal.ca`
[3] Email: `bvali087@uottawa.ca`

duplicability of data was solved syntactically, by giving a type system that distinguishes duplicable and non-duplicable types. The quantum lambda calculus possesses a reduction semantics, but no denotational semantics has been given for it so far.

In this paper, we study the restriction of the quantum lambda calculus to the purely linear case. This means we study the fragment of the language where each value, classical and quantum, must be used exactly once. The linear quantum lambda calculus differs from its nonlinear cousin in that it is less sensitive to the evaluation order of terms. We give a denotational semantics for this language in a category of completely positive maps, and we show that it is fully abstract with respect to the operational semantics.

The question of finding a denotational semantics for the full quantum lambda calculus is still open, but we hope that this work is a step in that direction.

The plan of the paper is as follows. First we briefly describe the language and the type system. Then we develop an operational semantics for it, and we define a notion of equivalence of terms. Finally we build a denotational semantics for the language, and we show the full abstraction result.

2 A linear lambda-calculus for quantum computation

In [4], we have defined an operational semantics for a lambda calculus for quantum computation with classical control. Here, we study the purely linear fragment of this language. We begin by re-adapting the definitions and results from this earlier paper for the linear setting.

2.1 Terms and Programs

Definition 2.1 The linear quantum lambda calculus has the following terms:

$$\begin{aligned} M,N,P ::= {} & x \mid MN \mid \lambda x.M \mid \mathit{if}\ M\ \mathit{then}\ N\ \mathit{else}\ P \mid 0 \mid 1 \mid \\ & \mathit{meas} \mid \mathit{new} \mid U \mid * \mid \langle M,N\rangle \mid \mathit{let}\ \langle x,y\rangle = M\ \mathit{in}\ N \mid \Omega. \end{aligned}$$

We follow Barendregt's convention for identifying terms up to α-equivalence [1]. The set of free variables of a term M is written $FV(M)$. We also sometimes use the shorthand notation $\langle M_1,\dots,M_n\rangle = \langle M_1,\langle M_2,\dots\rangle\rangle$. In the following, we often write c for an arbitrary constant of the language, i.e., 0, 1, *meas*, *new*, U, or $*$.

The set of types is defined by

$$A, B \ ::= \ \mathit{bit} \mid \mathit{qbit} \mid A \multimap B \mid \top \mid A \otimes B.$$

Note that, unlike the language of [4], there is no type constructor $!A$. We use the same shortcut for the product type as we did for the product term, to define $A \otimes \dots \otimes A$. To each constant term c, we associate a fixed type A_c, namely $\mathit{meas} : \mathit{qbit} \multimap \mathit{bit}$, $0, 1 : \mathit{bit}$, $\mathit{new} : \mathit{bit} \multimap \mathit{qbit}$, $* : \top$ and $U : \mathit{qbit}^{\otimes n} \multimap \mathit{qbit}^{\otimes n}$. A *typing judgement* is a triple $\Delta \rhd M : A$, where Δ is a list of distinct typed variables called a *typing context*, M is a term, and A is a type. We say that a typing judgement is *valid* if it follows from the typing rules given in Table 1.

$$\frac{}{x : A \rhd x : A}\ (ax_1) \qquad \frac{}{\rhd c : A_c}\ (ax_2)$$

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

$$\frac{\Gamma_1 \rhd M : A \multimap B \quad \Gamma_2 \rhd N : A}{\Gamma_1, \Gamma_2 \rhd MN : B}\ (\mathit{app}) \qquad \frac{\Delta, x : A \rhd M : B}{\Delta \rhd \lambda x.M : A \multimap B}\ (\lambda_1)$$

$$\frac{\Gamma_1 \rhd M_1 : A_1 \quad \Gamma_2 \rhd M_2 : A_2}{\Gamma_1, \Gamma_2 \rhd \langle M_1, M_2 \rangle : A_1 \otimes A_2}\ (\otimes.I) \qquad \frac{}{\rhd * : \top}\ (\top) \qquad \frac{}{\Delta \rhd \Omega : A}\ (\Omega)$$

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

Table 1
Typing rules for the linear quantum lambda calculus

Although this language is intended to manipulate quantum information, no constants of type *qbit* are provided in the definition of lambda-terms. Indeed, while it would be possible to allow constant qubit expressions such as $\alpha|0\rangle + \beta|1\rangle$, such a notation would not lend itself to expressing entangled states. Instead, we introduce the concept of a quantum array and a linking function to express terms with embedded quantum data.

Definition 2.2 A *quantum closure* is a triple $[Q, L, M]$ where Q is a normalized vector in $\otimes_{i=1}^n \mathbb{C}^2$, for some $n \geqslant 0$, L is a bijective function from a set $|L|$ of term variables to $\{0, \dots, n-1\}$, and M is a term. Q is called a *quantum array*, and L is called a *linking function*. We write $|Q| = n$. If $L(x_i) = i$, we will sometimes write L as the ordered list $|x_1 \cdots x_n\rangle$. The idea is that the variable x_i is bound in the term M to qubit number $L(x_i)$ of the state Q. We also call the pair (Q, L) a *quantum context*.

We extend the notion of α-equivalence to quantum closures:

$$[Q, |x \cdots y \cdots z\rangle, M] =_\alpha [Q, |x \cdots y' \cdots z\rangle, M[y'/y]]$$

if $y \notin FV(M) \cup |L|$.

Definition 2.3 A quantum closure $[Q, L, M]$ is *well-typed* (or *valid*) of type A in the typing context Γ, written $\Gamma \vDash [Q, L, M] : A$, if $|L| \cap |\Gamma| = \emptyset$ and $\Gamma, x_1{:}qbit, \dots, x_k{:}qbit \rhd M : A$ is a valid typing judgement, where $\{x_1 \dots x_k\} = |L| \cap FV(M)$.

A well-typed quantum closure is *closed* if $|\Gamma| = \emptyset$, and a closed well-typed quantum closure is also called a *program*.

A well-typed quantum closure is called a *computation* if $|L| \subseteq FV(M)$, or equivalently, if $FV(M) = |L| \cup |\Gamma|$.

3 Operational semantics

3.1 Small step semantics

The language contains a probabilistic operation: the measurement. This probabilistic operation forces us to choose a reduction strategy.

$$[Q, L, \Omega] \to_1^\omega [Q, L, \Omega] \qquad [Q, L, (\lambda x.M)V] \to_1^\beta [Q, L, M[V/x]]$$

$$[Q, L, \mathit{let}\ \langle x_1, x_2 \rangle = \langle V_1, V_2 \rangle\ \mathit{in}\ N] \to_1^{\mathit{let}} [Q, L, N[V_1/x_1, V_2/x_2]]$$

$$[Q, L, \mathit{if}\ 0\ \mathit{then}\ M\ \mathit{else}\ N] \to_1^{\mathit{if}_0} [Q, L, N]$$

$$[Q, L, \mathit{if}\ 1\ \mathit{then}\ M\ \mathit{else}\ N] \to_1^{\mathit{if}_1} [Q, L, M]$$

$$\frac{[Q, L, N] \to_p^\kappa [Q', L', N']}{[Q, L, MN] \to_p^\kappa [Q', L', MN']} \qquad \frac{[Q, L, M] \to_p^\kappa [Q', L', M']}{[Q, L, MV] \to_p^\kappa [Q', L', M'V]}$$

$$\frac{[Q, L, M_1] \to_p^\kappa [Q', L', M_1']}{[Q, L, \langle M_1, M_2 \rangle] \to_p^\kappa [Q', L', \langle M_1', M_2 \rangle]} \qquad \frac{[Q, L, M_2] \to_p^\kappa [Q', L', M_2']}{[Q, L, \langle V_1, M_2 \rangle] \to_p^\kappa [Q', L', \langle V_1, M_2' \rangle]}$$

$$\frac{[Q, L, P] \to_p^\kappa [Q', L', P']}{[Q, L, \mathit{if}\ P\ \mathit{then}\ M\ \mathit{else}\ N] \to_p^\kappa [Q', L', \mathit{if}\ P'\ \mathit{then}\ M\ \mathit{else}\ N]}$$

$$\frac{[Q, L, M] \to_p^\kappa [Q', L', M']}{[Q, L, \mathit{let}\ \langle x_1, x_2 \rangle = M\ \mathit{in}\ N] \to_p^\kappa [Q', L', \mathit{let}\ \langle x_1, x_2 \rangle = M'\ \mathit{in}\ N]}$$

Table 2
Reduction rules for the quantum lambda calculus

Definition 3.1 We define the call-by-value reduction strategy for the linear quantum lambda calculus by structural induction. For this purpose we need the notion of a value. We define a *value term* to be of the form $V, W \ ::= \ x \mid c \mid \lambda x.M \mid \langle V, W \rangle$. A *value program* is a program of the form $[Q, L, V]$, where V is a value term. The rules for the reduction are an adaptation of the ones found in [4].

We set the rules to be the "classical" ones found in Table 2, plus the following "quantum" rules. In the first two rules, let $Q = \sum_j Q_j^0 \otimes \alpha_j|0\rangle \otimes \tilde{Q}_j^0 + \sum_j Q_j^1 \otimes \beta_j|1\rangle \otimes \tilde{Q}_j^1$, where $Q_j^b \in \mathbb{C}^{2^{i-1}}$ and $\tilde{Q}_j^b \in \mathbb{C}^{2^{n-i}}$.

$$[Q, |x_1 \cdots x_n\rangle, \mathit{meas}\ \ x_i] \to_{|\alpha|^2}^{m_0} [\textstyle\sum_j Q_j^0 \otimes \tilde{Q}_j^0, |x_1 \cdots x_{i-1}x_{i+1} \cdots x_n\rangle, 0],$$
$$[Q, |x_1 \cdots x_n\rangle, \mathit{meas}\ \ x_i] \to_{|\alpha|^2}^{m_1} [\textstyle\sum_j Q_j^1 \otimes \tilde{Q}_j^1, |x_1 \cdots x_{i-1}x_{i+1} \cdots x_n\rangle, 1].$$

If w is a fresh term variable not yet in use:

$$[Q, |x_1 \cdots x_n\rangle, \mathit{new}\ \ 0] \to_1^{n_0} [Q \otimes |0\rangle, |x_1 \cdots x_n w\rangle, w],$$
$$[Q, |x_1 \cdots x_n\rangle, \mathit{new}\ \ 1] \to_1^{n_1} [Q \otimes |1\rangle, |x_1 \cdots x_n w\rangle, w].$$

If Q' is the result of applying U to the quantum bits $L(x_1), \dots, L(x_n)$ in Q:

$$[Q, L, U\langle x_1, \dots, x_n \rangle] \to_1^U [Q', L, \langle x_1, \dots, x_n \rangle].$$

Note that since we want a linear language, we modified the measurement rule from [4] by deleting the quantum bit measured from the quantum array.

Lemma 3.2 (Substitution) *If $\Delta, x : A \rhd M : B$ and if $\Gamma \rhd N : A$ then $\Delta, \Gamma \rhd M[N/x] : B$ is a valid typing judgement.*

Note that, in the non-linear case, the Substitution Lemma only holds when $N = V$ is a value. However, in the linear calculus considered here, it holds for general N.

Theorem 3.3 (Safety properties) *If P is a closed valid computation of type A, either it is a value or $P \to_\rho P'$, with P' a closed valid computation of type A.*

Proof. The proof is an adaptation of the proof of Theorems 1 and 2 in [4]. □

The language being linear, the reduction has a strong relation on the length of the terms.

Definition 3.4 Let $l(M)$ be the length of a term M, defined recursively as $l(x) = l(c) = l(\Omega) = 1$, $l(\lambda x.M) = l(M) + 1$, $l(\mathit{let}\ \langle x, y\rangle = M\ \mathit{in}\ N) = l(M) + l(N) + 1$, $l(\mathit{if}\ P\ \mathit{then}\ M\ \mathit{else}\ N) = l(P) + \mathit{max}(l(M), l(N))$, $l(MN) = l(\langle M, N\rangle) = l(M) + l(N) + 1$.

Lemma 3.5 *If $x \in FV(M)$, then $l(M[N/x]) = l(M) + l(N) - 1$.*

Lemma 3.6 *If $[Q, L, M] \to^x_\rho [Q', L', M']$, then either $x \neq \omega$ and $l(M') < l(M)$ or $x = \omega$ and $l(M') = l(M)$. In the latter case, $[Q, L, M] = [Q', L', M']$.*

Definition 3.7 A program $[Q, L, M]$ that reduces with a ω-rule is called a *fixed point.*

Theorem 3.8 (Strong Normalization) *If $P = [Q, L, M]$ is a closed valid computation, P reduces to a value or to a fixed point in at most $l(M)$ steps.*

Proof. Follows from Lemma 3.6. □

3.2 Quantum context and reduction

When describing the reduction rules, we carefully separated the quantum context from the lambda-term. In this subsection we show that the precise order of quantum bits in the quantum array does not matter.

Definition 3.9 If σ is a permutation of $\{1, \dots, n\}$, we extend σ to $\mathbb{N}$ with $\sigma(j) = j$ for $j > n$, and we define $\bar\sigma$ to be the corresponding permutation of quantum bits $\bar\sigma|x_1 \cdots x_n \cdots x_{n+k}\rangle = |x_{\sigma(1)} \cdots x_{\sigma(n)} x_{n+1} \cdots x_{n+k}\rangle$. We say that (Q_1, L_1) is σ-equivalent to (Q_2, L_2) if Q_1 and Q_2 have the same size, $Q_2 = \bar\sigma(Q_1)$ and $L_2 = \sigma^{-1} \circ L_1$. We write $(Q_1, L_1) =_\alpha{}^\sigma (Q_2, L_2)$. We define an equivalence relation called *alpha-equivalence on quantum contexts* by $(Q_1, L_1) =_\alpha (Q_2, L_2)$ if there exists a σ such that $(Q_1, L_1) =_\alpha{}^\sigma (Q_2, L_2)$.

The alpha-equivalence is sound with respect to the semantics:

Lemma 3.10 *If $[Q, L, M] \to_p [Q', L', M']$ then $[\bar\sigma Q, \sigma L, M] \to_p [\bar\sigma Q', \sigma L', M']$.*

3.3 Reduction to values

In the reduction process, what we are really seeking is the final result of the computation. In this section we explicate the relation of programs to values.

Definition 3.11 We informally recall a notion defined in [4]: If X is the set of closed valid programs and U the set of values, let $\mathit{prob}_U : X \times U \to [0, 1]$ be the map $\mathit{prob}_U(P, V)$ that returns the total probability for a program P to end up on the value V in zero or more steps. This function is called the *big-step reduction.*

We also define the *small-step reduction* operation $\mathit{prob} : X \times X \to [0, 1]$: for closed programs P, P', we define $\mathit{prob}(P, P') = p$ if there is a single-step probabilistic reduction $P \to_p P'$, $\mathit{prob}(V, V) = 1$ if V is a value program, and $\mathit{prob}(P, P') = 0$ in all other cases. Note that for all well-typed P, $\sum_{P' \in X} \mathit{prob}(P, P') = 1$.

Definition 3.12 If P is a closed well-typed program of type bit, and $b \in \{0, 1\}$, we define $(P \Downarrow b) = \sum_{V \in U_b} \mathit{prob}'(P, V)$, where U_b is the set of valid programs with term the value b. We say that P *evaluates to b with probability $P \Downarrow b$.*

Definition 3.13 We define a *formal probability distribution* of quantum closures to be $\Gamma \vDash \sum_i \rho_i[Q_i, L_i, M_i] : A$, where each $\Gamma \vDash [Q_i, L_i, M_i] : A$ is valid and $\sum \rho_i \leqslant 1$. The distribution is said to be closed if $|\Gamma| = \emptyset$.

Lemma 3.14 *Given a set Z, let $\mathcal{C}Z$ be the set of probability distributions over it. The small-step reduction $\mathit{prob} : X \times X \to [0, 1]$ can be curried to a map $\mathit{prob}' : \mathcal{C}X \to \mathcal{C}X$: $\mathit{prob}'(\sum_i \alpha_i P_i) = \sum_i \alpha_i \sum_{P' \in X} \mathit{prob}(P_i, P')P'$. Similarly, $\mathit{prob}_U : X \times U \to [0, 1]$ can be curried to a map $\mathit{prob}'_U : \mathcal{C}X \to \mathcal{C}U$: $\mathit{prob}'_U(\sum_i \alpha_i P_i) = \sum_i \alpha_i \mathit{prob}_U(P_i, V)V$. The definition of $P \Downarrow b$ can be extended in the same way to probability distributions of programs.*

Lemma 3.15 *If $\vDash P : A$ is valid, so is $\vDash \mathit{prob}'P : A$ and $\vDash \mathit{prob}'_U P : A$.*

Due to the strong normalization theorem, applying the map prob'_U is applying the map prob' finitely many times.

Proposition 3.16 *If $[Q, L, M]$ is a closed valid computation, then the reduction verifies*

$$\mathit{prob}'^{l(M)}[Q, L, M] = \sum_{i=1}^{n} \rho_i[Q_i, L_i, V_i] + \sum_{j=1}^{m} \rho'_j P_j,$$

with $\mathit{prob}'_U[Q, L, M] = \sum_{i=1}^n \rho_i[Q_i, L_i, V_i]$, the P_j being fixed points, $\sum_i \rho_i + \sum_j \rho'_j = 1$, and $n + m < 2^{l(M)}$.

4 Denotational semantics

In [3] the notion of completely positive map is used to model the notion of quantum computation. We aim to show that the linear subset of the quantum lambda calculus has the category **CPM** as a fully abstract model. Note that the interpretation will not be "onto" all completely positive maps, but will only be "onto up to scalar multiples". In this section we set the definition of the denotational semantics.

4.1 The category **CPM**

We recall the definition of the category **V** [3].

- The objects are signatures $\sigma = n_1, \dots, n_k$, i.e., finite tuples of positive integers,

$$\llbracket x : A \rhd x : A \rrbracket(v) = v \qquad \llbracket \rhd 0 : bit \rrbracket(x) = (x, 0)$$

$$\llbracket \rhd * : \top \rrbracket(x) = x \qquad \llbracket \rhd 1 : bit \rrbracket(x) = (0, x)$$

$$\llbracket \Delta \rhd \Omega : A \rrbracket = 0 : \llbracket \Delta \rrbracket \to \llbracket A \rrbracket$$

$$\llbracket \rhd new : bit \multimap qbit \rrbracket = \Phi(\iota) : 1 \to \llbracket bit \rrbracket \otimes \llbracket qbit \rrbracket$$

$$\llbracket \rhd meas : qbit \multimap bit \rrbracket = \Phi(p) : 1 \to \llbracket qbit \rrbracket \otimes \llbracket bit \rrbracket$$

$$\llbracket \rhd U : qbit^{\otimes n} \multimap qbit^{\otimes n} \rrbracket = \Phi(U) : 1 \to 2^{2n}$$

$$\frac{\llbracket \Delta, x : A \rhd M : B \rrbracket = f : \llbracket \Delta \rrbracket \otimes \llbracket A \rrbracket \to \llbracket B \rrbracket}{\llbracket \Delta \rhd \lambda x.M : A \multimap B \rrbracket = \Phi(f) : \llbracket \Delta \rrbracket \to \llbracket A \rrbracket \otimes \llbracket B \rrbracket}$$

$$\frac{\begin{array}{c}\llbracket \Delta \rhd M : A \multimap B \rrbracket = \Phi(g) : \llbracket \Delta \rrbracket \to \llbracket A \rrbracket \otimes \llbracket B \rrbracket \\ \llbracket \Gamma \rhd N : A \rrbracket = f : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket\end{array}}{\llbracket \Delta, \Gamma \rhd MN : B \rrbracket \ : \ x \otimes y \mapsto g(x \otimes (fy)) : \llbracket \Delta \rrbracket \otimes \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket}$$

$$\frac{\begin{array}{c}\llbracket \Delta \rhd P : bit \rrbracket \ : \ x \mapsto (px, qx) : \llbracket \Delta \rrbracket \to \llbracket bit \rrbracket \\ \llbracket \Gamma \rhd M : A \rrbracket = f : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket \\ \llbracket \Gamma \rhd N : A \rrbracket = g : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket\end{array}}{\llbracket \Delta, \Gamma \rhd if\ P\ then\ M\ else\ N : A \rrbracket \ : \ x \otimes y \mapsto (px)(fy) + (qx)(gy)}$$

$$\frac{\begin{array}{c}\llbracket \Delta \rhd M : A \rrbracket = f : \llbracket \Delta \rrbracket \to \llbracket A \rrbracket \\ \llbracket \Gamma \rhd N : B \rrbracket = g : \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket\end{array}}{\llbracket \Delta, \Gamma \rhd \langle M, N \rangle : A \otimes B \rrbracket \ : \ x \otimes y \mapsto fx \otimes gy : \llbracket \Delta \rrbracket \otimes \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket \otimes \llbracket B \rrbracket}$$

Table 3
Denotational semantics for typing judgments.

- the arrows $\sigma \to \sigma'$ are linear maps $V_\sigma \to V_{\sigma'}$, where $V_{n_1,\dots,n_k} = \mathbb{C}^{n_1 \times n_1} \times \dots \times \mathbb{C}^{n_k \times n_k}$.

There is a tensor product $(n_1, \dots, n_k) \otimes (m_1, \dots, m_l) = n_1 m_1, \dots, n_1 m_l, n_2 m_1, \dots, n_k m_l$ and a canonical isomorphism $V_{\sigma \otimes \tau} \simeq V_\sigma \otimes V_\tau$. Thus $\mathbf{V}$ has a structure of symmetric monoidal closed category. The unit element is the signature 1. From the vector spaces property, $\mathbf{V}(\sigma \otimes \tau, \sigma') =_\Phi \mathbf{V}(\sigma, \tau \otimes \sigma')$: If $B(\tau)$ is a basis for V_τ, then from $f \in \mathbf{V}(\sigma \otimes \tau, \sigma')$ we construct $g = \Phi(f) \in \mathbf{V}(\sigma, \tau \otimes \sigma')$ by $g(s) = \sum_{b \in B(\tau)} b \otimes f(s \otimes b)$. Conversely, given such a g, if $g(s) = \sum_{b \in B(\tau), u \in B(\sigma')} \alpha_{b,u} b \otimes u$, one constructs $f = \Phi^{-1}(g)$ by $f(s \otimes t) = \sum_{u \in B(\sigma')} \alpha_{t,u} u$. This makes $\mathbf{V}$ monoidal closed.

The category **CPM** has the same objects as $\mathbf{V}$ and as arrows completely positive maps (see [3]).

4.2 Modeling the quantum lambda-calculus

We set the denotation of types to be $\llbracket bit \rrbracket = (1,1)$, $\llbracket A \otimes B \rrbracket = \llbracket A \rrbracket \otimes \llbracket B \rrbracket$, $\llbracket qbit \rrbracket = 2$, and $\llbracket A \multimap B \rrbracket = \llbracket A \rrbracket \otimes \llbracket B \rrbracket$, and the denotation of contexts to be

$$\llbracket x_1 : A_1, \dots, x_n : A_n \rrbracket = \llbracket A_1 \rrbracket \otimes \dots \otimes \llbracket A_n \rrbracket, \quad \llbracket \emptyset \rrbracket = 1.$$

The denotation for typing judgments of the form $\Delta \rhd M : A$ is a linear map $\llbracket \Delta \rrbracket \to \llbracket A \rrbracket$, defined inductively as in Table 3. Here, $\Phi : \hom(A \otimes B, C) \to \hom(A, B \otimes C)$ is the bijection from the compact closed structure and ι and p are respectively the quantum bits creation and the measurement operation:

$$\begin{array}{rrcl} \iota : & 1,1 & \to & 2 \\ & (a,b) & \mapsto & \left(\begin{smallmatrix} a & 0 \\ 0 & b \end{smallmatrix}\right) \end{array} \qquad \begin{array}{rrcl} p : & 2 & \mapsto & 1,1 \\ & \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) & \mapsto & (a,d) \end{array}$$

We also define the denotation for a computation. Consider the valid computation $\Delta \vDash [Q, L, M] : A$ where $L = |x_1 \cdots x_n\rangle$ and $|Q| = n$. The quantum context (Q, L) can be seen as a map $g : 1 \to 2^{\otimes n}$ such that $g(1) = Q$. Then if

$$\llbracket \Delta, x_1 : qbit, \dots, x_n : qbit \rhd M : A \rrbracket = f : \llbracket \Delta \rrbracket \otimes 2^{\otimes n} \to \llbracket A \rrbracket,$$

one defines $\llbracket \Delta \vDash [Q, L, M] : A \rrbracket$ as the composition

$$\begin{tikzcd} \llbracket \Delta \rrbracket \arrow[r, "id_{\llbracket \Delta \rrbracket} \otimes g"] & \llbracket \Delta \rrbracket \otimes 2^{\otimes n} \arrow[r, "f"] & \llbracket A \rrbracket. \end{tikzcd}$$

One extends this definition to probabilistic distributions of computations using linearity:

$$\llbracket \Delta \vDash \sum_i \rho_i P_i \rrbracket = \sum_i \rho_i \llbracket \Delta \vDash P_i \rrbracket.$$

Lemma 4.1 *Denotations of terms and computations are completely positive maps.*

Lemma 4.2 (Substitution) *If $|\Gamma| \cap |\Delta| = \emptyset$ and $\llbracket \Delta, x : A \rhd M : B \rrbracket = G : \llbracket \Delta \rrbracket \otimes \llbracket A \rrbracket \to \llbracket B \rrbracket$, $\llbracket \Gamma \rhd N : A \rrbracket = F : \llbracket \Gamma \rrbracket \to \llbracket A \rrbracket$, then $\llbracket \Delta, \Gamma \rhd M[N/x] : B \rrbracket = H : \llbracket \Delta \rrbracket \otimes \llbracket \Gamma \rrbracket \to \llbracket B \rrbracket$, with $H(d \otimes g) = G(d \otimes (Fg))$*

Lemma 4.3 *Given a valid closed computation $P : A$ with $prob' P = \sum_i \rho_i P_i$ then $\llbracket P : A \rrbracket = \llbracket prob' P : A \rrbracket = \llbracket prob'_U P : A \rrbracket$.*

4.3 Fullness of the semantics for the first order fragment

Proposition 4.4 *For any types $A = U_1 \otimes \dots \otimes U_n$ and $B = V_1 \otimes \dots \otimes V_m$, where each U_i and V_j is either bit or qbit, if $F : \llbracket A \rrbracket \to \llbracket B \rrbracket$ is any superoperator, then there exists a valid typing judgement $x : A \rhd M : B$ of denotation F.*

Corollary 4.5 *For every type $A = U_1 \otimes \dots \otimes U_n$, where each U_i is either bit or qbit, and for every hermitian positive element $v \in V_{\llbracket A \rrbracket}$ of trace at most 1, $v = \llbracket M : A \rrbracket(1)$ for some closed valid term M.*

5 Equivalence classes of terms

Being able to build terms, we need some tools to compare them. One can compare them through syntactic manipulations, or one can have a finer approach using the two semantics we have built: in the case of the operational semantics, the *behavior* of the terms is what defines the equivalence, and in the case of the denotational semantics, the equivalence is expressed by the *denotation* of the terms.

(β)	$\Gamma \rhd (\lambda x.M)N$	$\approx_{ax}$	$M[N/x] : A$
(η)	$\Gamma \rhd \lambda x.Mx$	$\approx_{ax}$	$M : A \multimap B$
$(\beta_\otimes)$	$\Gamma \rhd$ *let* $\langle x, y\rangle = \langle N, P\rangle$ *in* M	$\approx_{ax}$	$M[N/x, P/y] : A$
$(\eta_\otimes)$	$\Gamma \rhd$ *let* $\langle x, y\rangle = M$ *in* $\langle x, y\rangle$	$\approx_{ax}$	$M : A \otimes B$
(β^1_{if})	$\Gamma \rhd$ *if* 1 *then* M *else* N	$\approx_{ax}$	$M : A$
(β^0_{if})	$\Gamma \rhd$ *if* 0 *then* M *else* N	$\approx_{ax}$	$N : A$
(Ω)	$\Gamma \rhd M[\Omega/x]$	$\approx_{ax}$	$\Omega : A$
(η_{if})	$\Gamma \rhd$ *if* B *then* $M[1/x]$ *else* $M[0/x]$	$\approx_{ax}$	$M[B/x] : A$
(id)	$\Gamma \rhd$ *meas*(*new* M)	$\approx_{ax}$	$M : qbit$

Table 4
Axiomatic equivalence

5.1 Axiomatic equivalence

A first notion of equality of terms can be defined by a set of syntactic rules. This is known as the axiomatic equivalence.

Definition 5.1 We define an equivalence relation $\approx_{ax}$ on typing judgement. We write the relation as $\Gamma \rhd M \approx_{ax} N : A$, and we define it to be the smallest relation satisfying the rules in Table 4, the alpha-equivalence and one congruence rule (ξ) per term constructor. This means for example:

$$\frac{\Gamma \rhd M \approx_{ax} M' : A \multimap B \quad \Delta \rhd N \approx_{ax} N' : A}{\Gamma, \Delta \rhd MN \approx_{ax} M'N' : B}\ (\xi_{app})$$

$$\frac{\Gamma, x : A \rhd M \approx_{ax} M' : B}{\Gamma \rhd \lambda x.M \approx_{ax} \lambda x.M' : A \multimap B}\ (\xi_\lambda)$$

We call it *the axiomatic equivalence relation.*

Lemma 5.2 *The order of the arguments in an application does not matter. Similarly, one can apply the arguments as a pairing or sequentially. More precisely:*

$$[Q, L, ((\lambda xy.M)N)P] \approx_{ax} [Q, L, ((\lambda yx.M)P)N]$$
$$[Q, L, \textit{let}\ \langle x, y\rangle = \langle N, P\rangle\ \textit{in}\ M] \approx_{ax} [Q, L, ((\lambda xy.M)N)P]$$

5.2 Operational context

To say that two arbitrary terms have the same behavior, we need a way to *observe* them. The only *observable* types at our disposal are the types *bit* and $\top$. So the fact that two terms M and M' have the same behavior can be understood as the fact that in whichever context $C[-]$ we "use" them, if $C[-] : bit$, then $C[M]$ reduces to 0, respectively, 1, with the same probability as $C[M']$. Such a term $C[-]$ is called an *operational context.*

Definition 5.3 We define a *formal operational context* to be a formula defined by the following BNF:

$$\begin{aligned} C[-] ::= {} & [-] \mid (C[-]M) \mid (MC[-]) \mid \lambda x.C[-] \mid \langle C[-], M\rangle \mid \langle M, C[-]\rangle \\ & \mid \textit{let}\ \langle x, y\rangle = C[-]\ \textit{in}\ M \mid \textit{let}\ \langle x, y\rangle = M\ \textit{in}\ C[-] \\ & \mid \textit{if}\ C[-]\ \textit{then}\ M\ \textit{else}\ N \mid \textit{if}\ M\ \textit{then}\ C[-]\ \textit{else}\ C'[-]. \end{aligned}$$

We call $[-]$ the *hole* of the context.

The notions of well-typed contexts and free variables in contexts are defined the same ways as for terms. Note that there exists a new notion: the notion of *captured variables*, which are the variables whose scope includes the hole. We can make this more precise by speaking of typed contexts:

Definition 5.4 A *typed operational context* is a typing tree with root $\Gamma' \rhd C[-] : B$, considering the additional axiom $\Gamma \rhd [-] : A$, i.e., a typing tree of the form

$$\cfrac{\cfrac{\Gamma \rhd [-] : A}{\vdots}}{\Gamma' \rhd C[-] : B.}$$

We say that this context is of type B, with free variables Γ', a hole of type A, and captured variables Γ. We also use the notation $\Gamma' \rhd C[\Gamma \rhd - : A] : B$ for a typed operational context.

Lemma 5.5 *If*

$$\cfrac{\cfrac{\Gamma \rhd [-] : A}{\vdots}}{\Gamma' \rhd C[-] : B.}$$

is a valid typing derivation, then so is

$$\cfrac{\cfrac{\Delta, \Gamma \rhd [-] : A}{\vdots}}{\Delta, \Gamma' \rhd C[-] : B,}$$

provided the variables that occur in Δ are fresh.

5.3 Operational equivalence

We define a notion of operational equivalence, based on the reduction rules and observations of type *bit*, as in [2]. (Equivalently, it would suffice to consider observations of type $\top$).

Definition 5.6 Let $\rhd C[\Gamma \rhd - : A] : bit$ be a closed typed operational context of type *bit*, and let $P = [Q, L, M]$ be a well-typed computation with typing judgement $\Gamma \vDash [Q, L, M] : A$. In this case we define the substitution $C[P]$ by $[Q, L, C[M]]$, where M is syntactically replacing $[-]$ in $C[-]$. We linearly extend this definition to probabilistic distributions of programs of the form $\Gamma \vDash \sum_i \rho_i P_i : A$ by setting $C[\sum_i \rho_i P_i] = \sum_i \rho_i C[P_i]$.

Lemma 5.7 *In Definition 5.6, the substitution is well typed and a valid typing judgement for it is* $\vDash [Q, L, C[M]] : bit$.

Proof. A direct consequence of Lemma 5.5. □

Definition 5.8 Given two well-typed computations $\Gamma \rhd P, P' : A$, we say that P is operationally equivalent to P' with respect to Γ if for all closed typed operational contexts $\rhd C[\Gamma \rhd - : A] : bit$, $C[P]\Downarrow 0 = C[P']\Downarrow 0$ and $C[P]\Downarrow 1 = C[P']\Downarrow 1$. In this case, we write $\Gamma \rhd P \approx_{op} P' : A$. If M, M' are terms, we say that $\Gamma \rhd M \approx_{op} M' : A$ if $\Gamma \vDash [|\rangle, |\rangle, M] \approx_{op} [|\rangle, |\rangle, M'] : A$.

5.4 Denotational equivalence

The last equivalence we can define is the *denotational equivalence.* This equivalence relation is simply stated:

Definition 5.9 We say that two typing judgments $\Gamma \rhd M, M' : A$ are *denotationally equivalent* if $[\![\Gamma \rhd M : A]\!]$ and $[\![\Gamma \rhd M' : A]\!]$ are the same map in **CPM**. In that case we write $\Gamma \rhd M \approx_{den} M' : A$.

We extend this definition to computations: $\Gamma \vDash P \approx_{den} P' : A$ is true if $[\![\Gamma \vDash P : A]\!] = [\![\Gamma \vDash P' : A]\!]$.

6 Soundness of the axiomatic equivalence and full abstraction of the denotational semantics

The three defined equivalence relations we have built have the expected behavior: The axiomatic equivalence is sound with respect to the operational equivalence and the denotational semantic is fully abstract with respect to the operational semantics:

Theorem 6.1 (Soundness) *If $\Gamma \vDash M \approx_{ax} M' : A$ then $\Gamma \vDash M \approx_{op} M' : A$.*

An immediate corollary is that the quantum context is not a side-effect, i.e., the order of evaluation does not affect the outcome.

Corollary 6.2 $\Gamma \rhd ((\lambda xy.P)M)N \approx_{op} ((\lambda yx.P)N)M$.

Proof. Using Lemma 5.2 and the soundness theorem. □

Theorem 6.3 (Full abstraction) *The denotational semantics is fully abstract with respect to the operational equivalence of typing judgments, i.e.*

$$[\![\Gamma \rhd M : A]\!] = [\![\Gamma \rhd M' : A]\!] \text{ if and only if } \Gamma \rhd M \approx_{op} M' : A.$$

Remark 6.4 The presence of the non-terminating term Ω is necessary for full abstraction to hold. Without it, every program terminates with probability 1, and there is only one definable map $bit \to \top$. Thus, although $\lambda f.(f0)$ and $\lambda f.(f1)$ of type $(bit \multimap \top) \multimap \top$ have different denotations, no context will distinguish them.

6.1 Proof of the soundness theorem

Assume Theorem 6.3. It suffices to show that if $\Gamma \rhd M \approx_{ax} M' : A$ then $\Gamma \rhd M \approx_{den} M' : A$. We show this by structural induction on the proof of $\Gamma \rhd M \approx_{ax} M' : A$.

We detail the cases (β^1_{if}) and (η).

For the (β^1_{if}) case, let $[\![\Gamma \rhd M, N : A]\!] = f, g : [\![\Delta]\!] \to [\![A]\!]$ and $[\![\rhd 1 : bit]\!] = (p, q) : 1 \to 1, 1$. Then $[\![\Gamma \rhd \textit{if } 1 \textit{ then } M \textit{ else } N : A]\!] : 1 \otimes y \mapsto (p1)(fy) + (q1)(gy)$ which is $y \mapsto fy$, namely $[\![\Gamma \rhd M : A]\!]$.

The case (η) is done as follows. Consider the proof tree

$$\dfrac{\dfrac{\Gamma \rhd M : A \multimap B \quad x : A \rhd x : A}{\Gamma, x : A \rhd Mx : B}}{\Gamma \rhd \lambda x.(Mx) : A \multimap B}$$

and set

$$\begin{aligned} [\![x : A \rhd x : A]\!] &= Id_A : [\![a]\!] \to [\![A]\!] \\ [\![\Gamma \rhd M : A \multimap B]\!] &= \Phi(g) : [\![\Delta]\!] \to [\![A]\!] \otimes [\![B]\!]. \quad \text{Then} \\ [\![\Gamma, x : A \rhd Mx : B]\!] &= f : [\![\Gamma]\!] \otimes [\![A]\!] \to [\![B]\!] \end{aligned}$$

with $f(x \otimes y) = g(x \otimes Id_A(y)) = g(x \otimes y)$, and we are done.

Note that cases (β) and (let) are done using Lemma 4.2. □

6.2 Full abstraction: preliminary lemmas

Lemma 6.5 *For any two valid closed computations P, P' of type bit, they have the same denotation if and only if for all $b \in \{0, 1\}$, $P \Downarrow b = P' \Downarrow b$.*

Proof. Consider two valid computations $P, P' : bit$. Suppose they have the same denotation f. Then from Lemma 4.3, f is also the denotation of $prob'_U P$ and of $prob'_U P'$. But by definition, $[\![prob'_U P]\!] = (p, q)$, where $p = (prob'_U P)\Downarrow 0 = P\Downarrow 0$ and $q = (prob'_U P)\Downarrow 1 = P \Downarrow 1$, and similarly for P'. Thus $P \Downarrow b = P' \Downarrow b$. The argument being reversible, we get the other implication. □

Lemma 6.6 *If $[\![\Gamma \rhd M{:}A]\!] = [\![\Gamma \rhd M'{:}A]\!]$ and $C[\Gamma \rhd - : A] : bit$ is a valid context, then $[\![C[M] : bit]\!] = [\![C[M'] : bit]\!]$.*

Proof. The proof uses Lemma 4.2. □

Definition 6.7 Given a type A, we define the *canonical first-order representation $\overline{A}$ of A* by the following: $\overline{bit} = bit$, $\overline{qbit} = qbit$, $\overline{\top} = \top$, $\overline{A \otimes B} = \overline{A} \otimes \overline{B}$, $\overline{A \multimap B} = \overline{A} \otimes \overline{B}$.

Lemma 6.8 *For all types A, $[\![A]\!] = [\![\overline{A}]\!]$.*

Lemma 6.9 *For all types A, there exist two terms $x : A \rhd \Upsilon_A : \overline{A}$ and $x : \overline{A} \rhd \overline{\Upsilon}_A : A$ and constants $\lambda, \lambda' > 0$, such that $[\![\Upsilon_A]\!] = \lambda id_{[\![A]\!]}$ and $[\![\overline{\Upsilon}_A]\!] = \lambda' id_{[\![A']\!]}$.*

Proof. The terms Υ_A and $\overline{\Upsilon}_A$ are simultaneously constructed by structural induction on A. □

Lemma 6.10 *For all types A, let $\sigma = [\![A]\!]$ and let m, m' be hermitian positive elements in V_σ, $m \neq m'$. Then there exists a well-typed term $x : A \rhd M : bit$ such that $[\![M]\!](m) \neq [\![M]\!](m')$.*

Proof. By Lemma 6.9, it suffices, without loss of generality, to consider the case where $A = bit \otimes \ldots \otimes bit \otimes qbit \otimes \ldots \otimes qbit$. However, in this case, the claim follows easily from Proposition 4.4 on the fullness of the semantics in first order. □

Lemma 6.11 *Given any type A and any hermitian positive $v \in V_{\llbracket A \rrbracket}$, there exists a closed term $M : A$ and $\lambda > 0$ such that $\llbracket M \rrbracket(1) = \lambda v$.*

Proof. By Corollary 4.5 and Lemma 6.9. □

6.3 Proof of the full abstraction theorem

If $\llbracket \Gamma \rhd M : A \rrbracket = \llbracket \Gamma \rhd M' : A \rrbracket$, take any valid context $C[\Gamma \rhd - : A] : bit$ for those two terms. Then from Lemma 6.6, $\llbracket \rhd C[M] : bit \rrbracket = \llbracket \rhd C[M'] : bit \rrbracket$. From Lemma 6.5, $C[M] \Downarrow b = C[M'] \Downarrow b$, for $b \in \{0, 1\}$. Since this holds for arbitrary contexts, M and M' are operationally equivalent.

The opposite implication follows from Lemma 6.10 and Lemma 6.11. Consider two typing judgments $\Gamma \rhd M, M'{:}A$ with denotations $F{=}\llbracket \Gamma \rhd M : A \rrbracket$ and $G = \llbracket \Gamma \rhd M' : A \rrbracket$, such that $F \neq G$.. Since the vector space $V_{\llbracket \Gamma \rrbracket}$ is spanned by hermitian positive elements, there exists a hermitian positive $v \in V_{\llbracket \Gamma \rrbracket}$ such that $F(v) \neq G(v)$.

If $\Gamma = x_1{:}A_1, \ldots, x_n{:}A_n$, let $B = A_1 \otimes \ldots \otimes A_n$. By Lemma 6.11, there exists a closed term $R : B$ such that $\llbracket R : B \rrbracket(1) = \lambda v$, for some $\lambda > 0$. By Lemma 6.10, there exists a term $x : A \rhd S : bit$ such that $\llbracket x : A \rhd S : bit \rrbracket(Fv) \neq \llbracket x : A \rhd S : bit \rrbracket(Gv)$. Now consider $C[\Gamma \rhd - : A] : bit$ defined by

$$let\ \langle x_1, \ldots x_n \rangle = R\ in\ let\ x = [-]\ in\ S.$$

Then $\llbracket C[M] \rrbracket(1) \neq \llbracket C[M'] \rrbracket(1)$, hence by Lemma 6.5, $C[M] \Downarrow b \neq C[M'] \Downarrow b$, for some $b \in \{0, 1\}$. It follows that $M \not\approx_{op} M'$, which completes the proof of full abstraction. □

7 Conclusion

In this paper we have restricted our study to the linear fragment of the programming language of [4]. We gave a syntactic notion of equivalence of terms and an operational one, together with a fully abstract model for the latter.

Several questions remain open. First, the exact image of the denotational semantics is still to be characterized as a subset of the completely positive maps. Then it would be interesting to explore the categorical semantics of the linear language. Finally, we may want to add weakening and duplication and find a denotational semantics for the full quantum lambda calculus.

References

[1] Barendregt, H. P., "The Lambda-Calculus, its Syntax and Semantics," Studies in Logic and the Foundation of Mathematics **103**, North Holland, 1984, second edition.

[2] Danos, V. and R. S. Harmer, *Probabilistic game semantics*, ACM Transactional on Computational Logic **3** (2002), pp. 359–382.

[3] Selinger, P., *Towards a quantum programming language*, Mathematical Structures in Computer Science **14** (2004), pp. 527–586.

[4] Selinger, P. and B. Valiron, *A lambda calculus for quantum computation with classical control*, in: P. Urzyczyn, editor, *Proceedings of the Seventh International Conference on Typed Lambda Calculi and Applications (TLCA 2005)*, Lecture Notes in Computer Science **3461** (2005), pp. 354–368.