A Quick Overview on the Quantum Control Approach to the Lambda Calculus

Alejandro Díaz-Caro[*]

Departamento de Ciencia y Tecnología
Universidad Nacional de Quilmes
Bernal, Buenos Aires, Argentina

Instituto de Ciencias de la Computación
CONICET–Universidad de Buenos Aires
Buenos Aires, Argentina

adiazcaro@icc.fcen.uba.ar

In this short overview we start with the basics of quantum computing, explaining the difference between the quantum and the classical control paradigms. We give an overview of the quantum control line of research within the lambda calculus, ranging from untyped calculi up to categorical and realisability models. This is a summary of the last 10+ years of research in this area, starting from Arrighi and Dowek's seminal work until today.

1 Introduction

The study of quantum computing in the framework of the lambda calculus has more than one motivation.

On the one hand, it is a tool to develop programming languages with firm foundations. For example, from the study of Selinger and Valiron's "Quantum Lambda Calculus" (QLC) [47], Quipper [26] and QWIRE [37] arose. Both quantum languages are quite advanced and complex, and while they are not fully formalised, their cores are based on Knill's QRAM model [28], which proposes that a quantum computer is a device attached to a classical computer, and it is the classical computer which instructs the quantum computer on what operations to perform, over which qubits, etc. Selinger took this QRAM model and formalised it in what is called the "Quantum Data, Classical Control" approach [46]. This approach derived later into the QLC, and finally in Quipper and QWIRE. Most quantum programming languages follow the same model, including high level languages such as IBM's Qiskit [1] or Microsoft's Q$\sharp$ [50]. Indeed, the Quantum Data, Classical Control approach is the more practical approach.

On the other hand, typed lambda calculus provides a way to study logics, through the Curry-Howard isomorphism (see, for example, [48]), which connects logics with computation. The logic of quantum mechanics has been a challenging subject since the beginnings of quantum mechanics. The seminal work of Birkhoff and von Neumann [8], in which the authors tried to reconcile the apparent inconsistency of classical logic with the quantum measurement, was the birth of a long field of study among physicists. However, due to its origins, there is no formal connection of this line with computing, and so, an extension to the Curry-Howard isomorphism contemplating this logic is not easy to envisage. The apparition of quantum computing as a way of understanding quantum mechanics brought all the computer science machinery to the game. As a consequence, a quantum logic formally connected to a quantum lambda calculus seems reasonable. If Birkhoff and von Neumann's quantum logic takes into account the superposition of quantum states, and the uncertainty principle, a quantum lambda calculus aimed at being the basis for a quantum logic must also consider these non-classical aspects of the quantum theory. With this objective in mind is that the first developments in this direction dropped the idea of Quantum Data, Classical Control, embracing Quantum Control instead.

[*]Founded by PICT-2019-1272, STIC-AmSud 21STIC10 Qapla', ECOS-Sud A17C03 QuCa, and PIP 11220200100368CO.

The quantum control approach has several origins. Most notably, quantum automata exploit quantum control. The main difference with classical control is that in classical control the control flow of a program is classical in the sense that its description does not admit superpositions, measurement, or any other kind of quantum properties. For example, we can describe a quantum algorithm by describing its classical flow such as:

```
Prepare a quantum state $|\psi\rangle$
Apply the unitary transformation $U$ to $|\psi\rangle$
Apply the unitary transformation $V$ to the result of the previous step
Measure the obtained system
```

There is nothing quantum in the flow described here. This list of instructions can be done by a classical computer. The quantum computer is the device which will have to perform the operations prescribed by this classical machine. In quantum control instead the flow is not classical. For example, in the instruction

$$\texttt{If } c \texttt{ then } |\psi\rangle \texttt{ else } |\varphi\rangle$$

if we admit c to be a superposition such as $\alpha.|\mathsf{true}\rangle + \beta.|\mathsf{false}\rangle$, this instruction would become the superposition

$$\alpha.\Big(\texttt{If } |\mathsf{true}\rangle \texttt{ then } |\psi\rangle \texttt{ else } |\varphi\rangle\Big) + \beta.\Big(\texttt{If } |\mathsf{false}\rangle \texttt{ then } |\psi\rangle \texttt{ else } |\varphi\rangle\Big)$$

and so resulting in

$$\alpha.|\psi\rangle + \beta.|\varphi\rangle$$

The control flow is in superposition, and so it is not classic. Of course this instruction is only valid if the final state $\alpha.|\psi\rangle + \beta.|\varphi\rangle$ is a valid quantum state, which is not always the case. In particular, if the norm of the input $\alpha.|\mathsf{true}\rangle + \beta.|\mathsf{false}\rangle$ being 1 implies that the norm of the output is also 1, then this "quantum if" instruction can be implemented by a quantum operator. The first work in this line, defining the quantum-if in the lambda calculus, was that of Altenkirch and Grattage [2]. Since then, there has been a long line of independent research pursuing a "quantum computational logic", that is, some sort of quantum logic firmly founded on quantum computing.

This article intends to be an overview on this quest for a quantum computational logic.

Plan of the paper In Section 2 we give a brief introduction to the quantum computing formalism. In Section 3 we present Lineal, an untyped extension to the lambda calculus to deal with superpositions. It is the starting point on the quest for a quantum computational logic. In Section 4 we present Vectorial and some of its fragments, which is the first typed Lineal. In Section 5 we present Lambda-$\mathcal{S}$, which is another typed Lineal, extended with quantum measurements. We also provide a categorical interpretation of this calculus. In Section 6 we show the first restriction of Lineal into a quantum language, achieved by using realisability techniques. We then introduce the calculus Lambda-$\mathcal{S}_1$, which has been derived from this technique, and give some details on its categorical interpretation. Such an interpretation is related to that of Lambda-$\mathcal{S}$. In Section 7 we move from the opposite direction: we define a new connective $\odot$ (read "sup") in Natural Deduction, which induces the $\odot$-calculus. Then, we show how to transparently add complex scalars, defining the $\odot^{\mathbb{C}}$-calculus, and use it to encode a quantum language. In Section 8 we refer to a few recent works towards recursive types in the quantum control setting. Finally, we conclude in Section 9 with some final thoughts and open problems.

2 Quantum computing in terms of four postulates

This section does not pretend to be an extensive introduction to quantum computing but just for the basics, and we take the liberty to simplify many things for the sake of readability. For a quite complete introduction to quantum computing the reader is referred to the great book by Nielsen and Chuang [34].

Quantum mechanics can be described in four axioms or postulates.

The first postulate defines how the quantum states are represented.

Postulate 1 (State space)**.** The state of an isolated quantum system can be fully described by a *state vector*, which is a norm-1 vector in a Hilbert space, that is, Banach space with inner product.

In quantum computing we usually consider the Hilbert space $\mathbb{C}^{2^n}$, hence, from now on we only consider these spaces. For vectors in $\mathbb{C}^{2^n}$ we use the Dirac notation consisting of a binary encoding on vectors. For example,

$$\begin{pmatrix}\frac{\sqrt{3}}{2}\\ \frac{1}{2}\end{pmatrix} \in \mathbb{C}^2 \text{ is written } \tfrac{\sqrt{3}}{2}|0\rangle + \tfrac{1}{2}|1\rangle,$$

$$\text{and} \quad \begin{pmatrix}\frac{1}{\sqrt{3}}\\ 0\\ \frac{1}{\sqrt{3}}\\ \frac{1}{\sqrt{3}}\end{pmatrix} \in \mathbb{C}^4 \text{ is written } \tfrac{1}{\sqrt{3}}|00\rangle + \tfrac{1}{\sqrt{3}}|10\rangle + \tfrac{1}{\sqrt{3}}|11\rangle.$$

A generic state vector is written $|\psi\rangle$. We also write $\langle\psi|$ to the transpose conjugate of $|\psi\rangle$. This way, $|\psi\rangle\langle\varphi|$ is a matrix while $\langle\psi||\varphi\rangle$ (usually written $\langle\psi|\varphi\rangle$) is the inner product $(|\psi\rangle,|\varphi\rangle)$.

The second postulate defines how a quantum state evolves over time. We give its discrete time version, since in quantum computing we usually consider discrete time.

Postulate 2 (Evolution)**.** The *evolution* of an isolated quantum system can be described by a unitary matrix, that is, a matrix U such that $U^\dagger = U^{-1}$, where $U^\dagger$ is the conjugate transpose of U. If the state of a quantum system is described by $|\psi\rangle$, after the evolution U the new state is $|\varphi\rangle = U|\psi\rangle$.

Example 2.1. Let $H = \begin{pmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}}\end{pmatrix}$, then

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = |+\rangle \\ H|1\rangle &= \frac{1}{\sqrt{2}}|0\rangle + \frac{-1}{\sqrt{2}}|1\rangle = |-\rangle \end{aligned}$$

where $|+\rangle$ and $|-\rangle$ are just conventional notations for these vectors. In particular, notice that $\{|+\rangle,|-\rangle\}$ is an orthonormal basis of $\mathbb{C}^2$ as well as $\{|0\rangle,|1\rangle\}$. Hence, H is a basis change matrix.

H is known as the Hadamard gate.

The third postulate defines how a quantum system is measured. We give its general form as Postulate 3, and a way to simplify it in Theorem 2.3.

Postulate 3 (Quantum measurement)**.** The *quantum measurement* is described by a collection of square matrices $\{M_i\}_i$, where i is called the *output of the measurement*, such that

$$\sum_i M_i^\dagger M_i = I$$

If the state of a quantum system is described by $|\psi\rangle$, the probability of measuring i is given by $p_i = \langle\psi|M_i^\dagger M_i|\psi\rangle$ and the state after the measuring i is $|\varphi\rangle = \frac{1}{\sqrt{p_i}}M_i|\psi\rangle$.

Example 2.2. Consider the following measurement: $\{M_0, M_1\}$, with $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. That is, $M_0 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$ and $M_1 = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, and let $|\psi\rangle = \frac{\sqrt{3}}{2}|0\rangle + \frac{1}{2}|1\rangle$.

Then, $p_0 = \langle\psi|0\rangle\langle 0|0\rangle\langle 0|\psi\rangle = \frac{3}{4}$ and $p_1 = \langle\psi|1\rangle\langle 1|1\rangle\langle 1|\psi\rangle = \frac{1}{4}$. The state after measuring 0 is $|0\rangle$ and after measuring 1 is $|1\rangle$.

The previous example can be taken as a general rule. With these M_0 and M_1, in general, measuring $\alpha|0\rangle + \beta|1\rangle$ results in $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$.

Moreover, taking $\{P_b\}_{b\in\{0,1\}^n}$ with $P_b = |b\rangle\langle b|$ to measure the state $|\psi\rangle = \sum_b \alpha_b|b\rangle$ results in $|b\rangle$ with probability $|\alpha_b|^2$.

The following theorem states that such a measurement (usually called "measurement in the computational basis") is enough for quantum computing.

Theorem 2.3. *Any measurement $M = \{M_i\}_i$ can be simulated by a measurement in the computational basis given by $P = \{P_b\}_{b\in\{0,1\}^n}$ with $P_b = |b\rangle\langle b|$, and a unitary matrix U in the sense that measuring $|\psi\rangle$ with M is the same as measuring $U|\psi\rangle$ with P_b and applying U^{-1} afterwards.* □

The previous lemma justifies the fact that most quantum programming languages consider only measurements in the computational basis.

Finally, the fourth postulate defines how to compose quantum systems.

Postulate 4 (Composed system)**.** The state space of a *composed system* is the tensor product of the state of its components.

Given n systems in states $|\psi_1\rangle, \dots, |\psi_n\rangle$, the composed system is described by $|\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$.

Example 2.4. If $|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $|\varphi\rangle = \frac{1}{\sqrt{5}}|0\rangle + \frac{2}{\sqrt{5}}|1\rangle$, the composed system $|\psi\rangle \otimes |\varphi\rangle$ is

$$\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) \otimes \left(\frac{1}{\sqrt{5}}|0\rangle + \frac{2}{\sqrt{5}}|1\rangle\right) = \frac{1}{\sqrt{10}}|00\rangle + \frac{2}{\sqrt{10}}|01\rangle + \frac{1}{\sqrt{10}}|10\rangle + \frac{2}{\sqrt{10}}|11\rangle$$

Definition 2.5 (qubit)**.** We call qubit, or quantum bit, to the quantum states in $\mathbb{C}^2$ and n-qubit to quantum states in $\mathbb{C}^{2^n} = \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2$.

So, a qubit is written $\alpha|0\rangle + \beta|1\rangle$ and an n-qubit is $\sum_{b\in\{0,1\}^n} \alpha_b|b\rangle$.

An important surprisingly consequence of the four postulates is the no-cloning theorem [52], which states that an unknown quantum state cannot be duplicated.

Theorem 2.6. *There is no unitary matrix U such that for some fixed $|\varphi\rangle \in \mathbb{C}^{2^n}$ and for all $|\psi\rangle \in \mathbb{C}^{2^n}$ we have $U(|\varphi\rangle \otimes |\psi\rangle) = |\psi\rangle \otimes |\psi\rangle$.* □

3 Lineal: A linear algebraic lambda calculus

Lineal, a seminal work by Arrighi and Dowek, first published at [5] and then extended in [6], starts from the following simple idea: The Church encoding for booleans, where $\lambda x.\lambda y.x$ represents `true` and $\lambda x.\lambda y.y$ represents `false`, follows the premise that in the lambda calculus everything is a function, even the basic data. Therefore, to consider quantum bits we would need to extend the lambda calculus with complex linear combinations of lambda terms. This way, the qubit $\alpha|0\rangle + \beta|1\rangle$ would be represented by $\alpha.\lambda x.\lambda y.x + \beta.\lambda x.\lambda y.y$. More generally, we may consider the infinite-dimensional Hilbert space whose basis is given by the values of the lambda calculus.

Hence, Lineal proposes the following syntax of terms:

$$t := x \mid \lambda x.t \mid tt \mid \alpha.t \mid t+t \mid \vec{0}$$

where $\alpha \in \mathbb{C}$ and the symbol $+$ is considered modulo associativity and commutativity.

Its operational semantics has four groups of rules.

The beta group:

$$(\lambda x.t)b \longrightarrow (b/x)t$$

where b is a basis term, that is, a classical value (either a variable or an abstraction).

The elementary group:

$$\begin{array}{ccc} t+\vec{0} \longrightarrow t & 0.t \longrightarrow \vec{0} & 1.t \longrightarrow t \\ \alpha.\vec{0} \longrightarrow \vec{0} & \alpha.(\beta.t) \longrightarrow (\alpha\times\beta).t & \alpha.(t+r) \longrightarrow \alpha.t+\beta.t \end{array}$$

The factorization group:

$$\alpha.t+\beta.t \longrightarrow (\alpha+\beta).t \qquad \alpha.t+t \longrightarrow (\alpha+1).t \qquad t+t \longrightarrow 2.t$$

The application group:

$$\begin{array}{ccc} (t+r)s \longrightarrow (ts)+(rs) & (\alpha.t)r \longrightarrow \alpha.(tr) & \vec{0}t \longrightarrow \vec{0} \\ s(t+r) \longrightarrow (st)+(sr) & r(\alpha.t) \longrightarrow \alpha.(rt) & t\vec{0} \longrightarrow \vec{0} \end{array}$$

This way, for example, if $|0\rangle = \lambda x.\lambda y.x$ and $|1\rangle = \lambda x.\lambda y.y$, a term encoding a unitary matrix U will act as follows:

$$U(\alpha.|0\rangle+\beta.|1\rangle) \longrightarrow (U\alpha.|0\rangle)+(U\beta.|1\rangle) \longrightarrow^* \alpha.U|0\rangle+\beta.U|1\rangle$$

For instance, the Hadamard gate (see Example 2.1) can be encoded as

$$H := \lambda x.\left\{x\left[\frac{1}{\sqrt{2}}.|0\rangle+\frac{1}{\sqrt{2}}.|1\rangle\right]\left[\frac{1}{\sqrt{2}}.|0\rangle+\frac{-1}{\sqrt{2}}.|1\rangle\right]\right\} \tag{1}$$

where $[t] := \lambda x.t$ is a thunk and $\{t\} := t\lambda x.x$ releases the thunk (i.e. $\{[t]\} \longrightarrow^* t$).

The thunk is used to stop the linearity. Otherwise, $H|0\rangle \longrightarrow |0\rangle\left(\frac{1}{\sqrt{2}}.|0\rangle+\frac{1}{\sqrt{2}}.|1\rangle\right)\left(\frac{1}{\sqrt{2}}.|0\rangle+\frac{-1}{\sqrt{2}}.|1\rangle\right)$, which is just $(\lambda x.\lambda y.x)\left(\frac{1}{\sqrt{2}}.|0\rangle+\frac{1}{\sqrt{2}}.|1\rangle\right)\left(\frac{1}{\sqrt{2}}.|0\rangle+\frac{-1}{\sqrt{2}}.|1\rangle\right)$ would reduce, using the rules at the application group, to

$$\frac{1}{2}.(\lambda x.\lambda y.x)|0\rangle|0\rangle+\frac{-1}{2}.(\lambda x.\lambda y.x)|0\rangle|1\rangle+\frac{1}{2}.(\lambda x.\lambda y.x)|1\rangle|0\rangle+\frac{-1}{2}.(\lambda x.\lambda y.x)|1\rangle|1\rangle$$

and then to

$$\frac{1}{2}.|0\rangle+\frac{-1}{2}.|0\rangle+\frac{1}{2}.|1\rangle+\frac{-1}{2}.|1\rangle \longrightarrow^* \vec{0}$$

instead of the expected $\frac{1}{\sqrt{2}}.|0\rangle+\frac{1}{\sqrt{2}}.|1\rangle$.

So, the linearity achieved by the rules at the application group can be frozen with thunks when needed, making Lineal a very expressive calculus of computation combining matrices and vectors.

There is no measurement in Lineal. In addition, the rules of the operational semantics have some constraints such as

$$\text{“rule } \alpha.t+\beta.t \longrightarrow (\alpha+\beta).t \text{ applies only if } t \text{ is closed normal”} \tag{2}$$

These constraints are needed in this untyped case where non-terminating terms can break confluence. For example, let $\Delta_b = \lambda x.(xx+b)$ and $Y_b = \Delta_b\Delta_b$. Then $Y_b \longrightarrow Y_b + b$. If b represents 1, this Y_b can add up to infinity. Then, since we have infinity and algebraic operations we may run into undefined forms such as $\infty - \infty$, which here is represented by $Y_b + (-1).Y_b$. Indeed, without any restrictions, $Y_b + (-1).Y_b \longrightarrow 0.Y_b \longrightarrow \vec{0}$, but also $Y_b + (-1).Y_b \longrightarrow Y_b + b + (-1).Y_b \longrightarrow^* b$. Hence, restriction (2) removes this form of indeterminacy. There are other restrictions, but since we will consider types in the next section, which ensures strong normalisation, we can just ignore them.

Remark 3.1. The algebraic linearity of Lineal implies that the no-cloning theorem (Theorem 2.6) is valid using this encoding of unitary matrices. Intuitively, the fact that $U(\alpha.|0\rangle + \beta.|1\rangle) \longrightarrow^* \alpha.U|0\rangle + \beta.U|1\rangle$ means that α and β are never duplicated by U, and so it is not possible to construct a term U such that $U(\alpha.|0\rangle + \beta.|1\rangle)$ reduce to $(\alpha.|0\rangle + \beta.|1\rangle) \otimes (\alpha.|0\rangle + \beta.|1\rangle)$, for any encoding of $\otimes$.

4 Vectorial: The first typed Lineal

Typing Lineal with simply types, or second order polymorphic types, is possible by adding the following straightforward typing rules to simply typed lambda calculus or System F, in order to type the extra constructions:

$$\frac{}{\Gamma \vdash \vec{0} : A} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash r : A}{\Gamma \vdash t + r : A} \qquad \frac{\Gamma \vdash t : A}{\Gamma \vdash \alpha.t : A}$$

This way, the term H from Equation (1) would be typed with $(\tau \Rightarrow \tau \Rightarrow \tau) \Rightarrow (\tau \Rightarrow \tau \Rightarrow \tau)$, for some basic type τ.

A straightforward extension like this excludes from the calculus several valid interesting terms. For example, let t be a term typed with $\forall X.A \Rightarrow B$ for some A. It could be applied not only to terms typed with $[C/X]A$, but also to a linear combination $\sum_i \alpha_i.r_i$ as soon as each r_i is typed with some $[C_i/X]A$. Indeed, $t\sum_i \alpha_i.r_i$ reduces to $\sum_i \alpha_i.tr_i$. However, $\sum_i \alpha_i.r_i$ may have type $\forall X.A$, but from there, only one C_i can be chosen to type the whole term. A solution to this lack of expressivity is provided by the Vectorial calculus [4], which types this term with $\sum_i \alpha_i.[C_i/X]A$.

The Vectorial calculus introduces linear combinations of types, in the same way as Lineal considers linear combinations of terms. However, due to the application group of rewrite rules, term variables must be typed only with types that are not linear combinations of types (here called "unit types"). Indeed, suppose we admit variables of type $\alpha.U$, so $\lambda x.x+y$ is typed with $(\alpha.U) \Rightarrow (\alpha.U)+V$, where V is the type of y. Then, if u is typed with U, we may expect $\alpha.u$ to be typed with $\alpha.U$, and so $(\lambda x.x+y)(\alpha.u)$ has type $(\alpha.U)+V$. However,

$$(\lambda x.x+y)(\alpha.u) \longrightarrow \alpha.(\lambda x.x+y)u \longrightarrow \alpha.(u+y) \longrightarrow \alpha.u + \alpha.y$$

so its type should be $\alpha.U + \alpha.V$ instead.

Indeed, the abstracted variable x has been substituted by u and not by $\alpha.u$ during reduction, and so, the type of the abstraction should reflect this being just $U \Rightarrow U+V$. Therefore, term variables must be typed with unit types.

On the other hand, type variables do not always need to be unit types. For example, we may consider variables $\mathcal{X}$, $\mathcal{Y}$, which can be substituted only by unit types, and variables $\mathbb{X}$, $\mathbb{Y}$, which can be substituted by any type.

For instance, the Hadamard term H from Equation (1) may be typed as follows. Let $\mathcal{T} = \forall\mathcal{X}.\forall\mathcal{Y}.\mathcal{X} \Rightarrow \mathcal{Y} \Rightarrow \mathcal{X}$ and $\mathcal{F} = \forall\mathcal{X}.\forall\mathcal{Y}.\mathcal{X} \Rightarrow \mathcal{Y} \Rightarrow \mathcal{Y}$. Also, if t has type A, we let thunks $[t]$ be typed with $[A]$, which is just a notation for $(\forall\mathcal{X}.\mathcal{X}\Rightarrow\mathcal{X})\Rightarrow A$. Then,

$$\vdash H:\forall\mathbb{X}.\left(\left[\frac{1}{\sqrt{2}}.\mathcal{T}+\frac{1}{\sqrt{2}}.\mathcal{F}\right]\Rightarrow\left[\frac{1}{\sqrt{2}}.\mathcal{T}+\frac{-1}{\sqrt{2}}.\mathcal{F}\right]\Rightarrow[\mathbb{X}]\right)\Rightarrow\mathbb{X}$$

The full grammar of types is the following.

$$\begin{array}{ll} A := U \mid \alpha.A \mid A+A \mid \mathbb{X} & \text{Types} \\ U := \mathcal{X} \mid U\Rightarrow A \mid \forall\mathcal{X}.U \mid \forall\mathbb{X}.U & \text{Unit types}\end{array}$$

where we use A,B,C for types, U,V,W for unit types, $\mathbb{X},\mathbb{Y},\mathbb{Z}$ for variables, and $\mathcal{X},\mathcal{Y},\mathcal{Z}$ for unit variables. We use X,Y,Z to refer to both kinds of variables.

There is also an equivalence between types given by the axioms of vector spaces as follows.

$$\begin{array}{ccc} 1.A\equiv A & \alpha.A+\beta.B\equiv\alpha.(A+B) & A+B\equiv B+A \\ \alpha.(\beta.A)\equiv(\alpha\beta).A & \alpha.A+\beta.A\equiv(\alpha+\beta).A & A+(B+S)\equiv(A+B)+S\end{array}$$

There is no general null type 0, but one type 0 for each type A. So, the term $\vec{0}$ is typed with the rule

$$\frac{\Gamma\vdash t:A}{\Gamma\vdash\vec{0}:0.A}$$

Taking again the example that started this section, to type $t\sum_i\alpha_i.r_i$ where $\vdash t:\forall X.A\Rightarrow B$, and for each i, $\vdash r_i:[C_i/X]A$ we consider an $\Rightarrow$-elimination typing rule such as

$$\frac{\Gamma\vdash t:\forall X.A\Rightarrow B\quad\Gamma\vdash r:\sum_i\alpha_i.[C_i/X]A}{\Gamma\vdash tr:\sum_i\alpha_i.[C_i/X]B}$$

which is not only an $\Rightarrow$-elimination but also a $\forall$-elimination at the same time.

Next, we also consider that t can be a linear combination of terms, and, since it is also a $\forall$-elimination, we may have more variables to replace, which gives us the more general rule

$$\frac{\Gamma\vdash t:\sum_i\alpha_i.\forall\vec{X}.A\Rightarrow B_i\quad\Gamma\vdash r:\sum_j\alpha_j.[\vec{C}_j/\vec{X}]A}{\Gamma\vdash tr:\sum_i\sum_j\alpha_i\beta_j.[\vec{C}_j/\vec{X}]B_i}$$

where, as usual, notation $\forall\vec{X}.A$ stands for $\forall X_1.\dots.\forall X_n.A$ and $[\vec{C}/\vec{X}]$ for $[C_1/X_1]\cdots[C_n/X_n]$.

The main theorem in [4] is a strong normalisation result for this calculus. However, subject reduction is proved only to some extent. Indeed, if t has both types A and B, $\alpha.t+\beta.t$ may be typed with $\alpha.A+\beta.B$, while its reduct, $(\alpha+\beta).t$ cannot. So, a weaker subject reduction is proved, which states that if $\Gamma\vdash t:A$ and $t\longrightarrow r$, then $\Gamma\vdash r:B$ with B related to A by an ad-hoc relation. Later, [35] slightly modifies Vectorial obtaining a proper subject reduction result, without losing its main properties.

Some fragments of Vectorial are the Scalar type system [3], which only includes scalars on types but not sums. The Scalar type system can track the “weight” of a type, defined as the ℓ_1-norm of a term, which is useful, for example, to define a probabilistic type system out of Lineal, by enforcing ℓ_1-norm to be equal to 1 on each term. Another fragment is the Additive type system [21], with sums but not scalars, which is shown to be equivalent to System F with pairs.

5 Lambda-$\mathcal{S}$: Lineal plus measurement

Lineal takes care of three of the four postulates stated in Section 2.

- Postulate 1, referring to state vectors, can be encoded in several ways in Lineal. For example, we can take $|0\rangle := \lambda x.\lambda y.x$ and $|1\rangle := \lambda x.\lambda y.y$, and then make its linear combination to express any vector in $\mathbb{C}^2$. Also, for $\vec{v} \in \mathbb{C}^{2^n}$ it suffices to consider the basis $\{\lambda\vec{x}.x_1,\dots,\lambda\vec{x}.x_{2^n}\}$ where $\lambda\vec{x}.x_i$ stands for $\lambda x_1.\dots.\lambda x_{2^n}.x_i$.
- Postulate 2, referring to unitary matrices, can be encoded easily also. For example, any $U = (\alpha_{ij})_{ij} \in \mathbb{C}^2 \times \mathbb{C}^2$, can be written as

$$U := \lambda x.\,\{x\,[\alpha_{00}.|0\rangle + \alpha_{01}.|1\rangle]\,[\alpha_{10}.|0\rangle + \alpha_{11}.|1\rangle]\}$$

- Postulate 4, referring to composing systems, can be also represented by taking the usual Church encoding for pairs, since the linearity given by the application group of rewrite rules will make these pairs bi-linear (i.e. left and right linear), as a tensor product.

The only missing postulate is Postulate 3, referring to measurements. Suppose we want to add a term π representing a measurement operator in the computational basis such that $\pi(\alpha.|0\rangle+\beta.|1\rangle)$ reduces to $|0\rangle$ with probability $|\alpha|^2$ and to $|1\rangle$ with probability $|\beta|^2$ (if $|\alpha|^2+|\beta|^2=1$, otherwise it suffices to divide this term by $|\alpha|^2+|\beta|^2$ before reducing). The problem is that π should not be linear, but then, $\lambda x.\pi x$ would not behave as π since

$$(\lambda x.\pi x)(\alpha.|0\rangle+\beta.|1\rangle) \longrightarrow^* \alpha.(\lambda x.\pi x)|0\rangle+\beta.(\lambda x.\pi x)|1\rangle \longrightarrow^* \alpha.\pi|0\rangle+\beta.\pi|1\rangle \longrightarrow^* \alpha.|0\rangle+\beta.|1\rangle$$

which is definitely not what we meant to do. Indeed, $(\lambda x.\pi x)r$ should reduce to πr whatever r is, even if it is a superposition. This jeopardises the entire encoding. As we pointed out in Remark 3.1, the algebraic linearity (i.e. the application group of rewrite rules) is needed to forbid cloning. An alternative solution, first described in Lambda-$\mathcal{S}$ [10, 14], is to allow for certain functions to be call-by-name, that is, $(\lambda x.t)r$ reduces to $(r/x)t$ wherever r is, at the condition that x appears at most once in t, which also forbids duplication. With this goal in mind, Lambda-$\mathcal{S}$ is typed with simple types, but adding a new symbol S which marks the "superpositions", as those types that forbid duplication. The grammar of types is given by

$$\begin{aligned} \Psi &:= \mathbb{B} \mid \Psi\times\Psi \mid S\Psi && \text{Qubit types}\\ A &:= \Psi \mid \Psi\Rightarrow A \mid A\times A \mid SA && \text{Types}\end{aligned}$$

Since superposition of superpositions is still a superposition, $SSA = SA$, and since a basis term such as $|0\rangle$ can be seen as the superposition of $1.|0\rangle+0.|1\rangle$, $A \le SA$.

Lambda-$\mathcal{S}$ includes constants $|0\rangle$ and $|1\rangle$, instead of using Church encodings, as well as an if-then-else construction. However, the most interesting characteristic, other than the application group from Lineal, is the fact that there are two beta-reductions, depending on types. If b is a basis term, then $(\lambda x:\mathbb{B}^n.t)b$ reduces to $(b/x)t$, which is exactly Lineal's beta-reduction. Instead, $(\lambda x:S\Psi.t)r$ reduces directly to $(r/x)t$, for any r. However, the typing system ensures that x does not appear more than once in t in this second case. This way,

$$(\lambda x:\mathbb{B}.t)(\alpha.|0\rangle+\beta.|1\rangle) \longrightarrow \alpha.(\lambda x:\mathbb{B}.t)|0\rangle+\beta.(\lambda x:\mathbb{B}.t)|1\rangle \longrightarrow^* \alpha.(|0\rangle/x)t+\beta.(|1\rangle/x)t$$

while

$$(\lambda x:S\mathbb{B}.\pi x)(\alpha.|0\rangle+\beta.|1\rangle) \longrightarrow \pi(\alpha.|0\rangle+\beta.|1\rangle)$$

as expected.

The term H from Example 2.1 is still valid, typed with $\mathbb{B} \Rightarrow S\mathbb{B}$ on this system.

A first denotational semantics (in environment style) is given where the type $\mathbb{B}$ is interpreted as $[\![\mathbb{B}]\!] = \{|0\rangle, |1\rangle\}$ while SA is interpreted as $\mathsf{Span}[\![A]\!]$, the vector space generated by $[\![A]\!]$. For example, $[\![S\mathbb{B}]\!] = \mathbb{C}^2$. In [16, 18] a concrete categorical interpretation is given, where S is considered as a function of an adjunction between the category **Set** and the category **Vec**. Explicitly, when we evaluate S we obtain formal linear combinations of elements of a set with complex numbers as coefficients. The other functor in the adjunction, U is just the forgetful functor allowing us to forget its vectorial structure.

Later, in [17], an abstract categorical semantics of Lambda-$\mathcal{S}$ has been defined. The main structural feature of such a model is that it is expressive enough to describe the bridge between the property-less elements such as $\alpha.v + \beta.v$, without any equational theory, and the result of its algebraic manipulation into $(\alpha + \beta).v$, explicitly controlling its interaction.

A distinctive design choice of Lambda-$\mathcal{S}$ is that we mark the superpositions, which are the non-duplicable elements. This is somehow the opposite of what is done in Linear Logic, where a bang ! marks the duplicable elements. In fact, it is common that intuitionistic linear models (linear as in linear-logic) are obtained by a monoidal comonad determined by a monoidal adjunction $(S,m) \dashv (U,n)$, that is, the bang ! is interpreted by the comonad SU (see for example [7]). Instead, a crucial point of our model of Lambda-$\mathcal{S}$ is to consider the monad US for the interpretation of S, determined by a similar monoidal adjunction. This implies that on the one hand we have tight control of the Cartesian structure of the model (i.e. duplication, etc.) and on the other hand superpositions live in some sense inside the classical world determined externally by classical rules until we decide to explore it. This is given by the following composition of maps:

$$US\mathbb{B} \times US\mathbb{B} \xrightarrow{n} U(S\mathbb{B} \otimes S\mathbb{B}) \xrightarrow{Um} US(\mathbb{B} \times \mathbb{B})$$

Remark 5.1. Lambda-$\mathcal{S}$ includes the four postulates, plus classical computation, since simply typed lambda calculus is a subset of it. However, the first postulate is included in a too general way, as with Lineal: any superposition $\alpha.|0\rangle + \beta.|1\rangle$ is valid, for any α and β, so, instead of taking norm-1 vectors (cf. Postulate 1), we are taking any vectors. Moreover, an abstraction $\vdash \lambda x : \mathbb{B}.t : \mathbb{B} \Rightarrow S\mathbb{B}$ may not represent a unitary matrix. For example, $\lambda x : \mathbb{B}.\mathsf{if}\ x\ \mathsf{then}\ |\psi\rangle\ \mathsf{else}\ |\varphi\rangle$ is typable, but does not represent a unitary map unless $|\psi\rangle \perp |\varphi\rangle$. To ensure unitarity we should add some ad-hoc restrictions to ensure that the branches of an if-then-else are orthogonal. However, the orthogonality of two values is not so hard to define, but the orthogonality between arbitrary programs does not seem to be an easy task.

QML [2] introduced some syntactic judgements of the form $t \perp r$ for a limited subset of terms. In the next section we see how to produce a calculus from a model where only norm-1 vectors are allowed.

6 Realisability to the rescue

From Remark 5.1 it is clear that we need a quite non-standard restriction to the linear combinations. The fact that not every vector in $\mathbb{C}^2$ is a qubit, but only those in the unitary sphere, is a hard condition to ask in a type system. For example, in Vectorial (see Section 4) we may restrict types $\alpha.A + \beta.B$ to the case $|\alpha|^2 + |\beta|^2 = 1$, but if $A = B = \forall \mathcal{X}.\mathcal{X} \Rightarrow \mathcal{X} \Rightarrow \mathcal{X}$, this condition is not enough, for example, this is a valid type for $\alpha.|0\rangle + \beta.|0\rangle \longrightarrow (\alpha + \beta).|0\rangle$, which does not have norm 1 if $|\alpha|^2 + |\beta|^2 = 1$.

The restriction of Lambda-$\mathcal{S}$ to norm-1 vectors has been obtained in [15] by means of realisability techniques [27, 29, 32, 51], which proved to be a great method to add any kind of ad-hoc restrictions in a clean and easy way. The technique can be summarised as follows:

1. Take an untyped machine (i.e. a calculus with a fixed strategy). It is *confluent by definition*.
2. Then, define a grammar of types and give an interpretation $[\![A]\!]$ for each type A as a set of values, for example, allow only for values v such that $\|v\| = 1$. So, it has *norm*-1 *by definition*.
3. Then, define that a term t is a realizer of A (notation $t \vDash A$) whenever $t \longrightarrow^* v \in [\![A]\!]$. A sequent $\Gamma \vdash t : A$ is defined to be valid if for any valid substitution σ in Γ, $\sigma t \vDash A$. So, typed terms are *strongly normalising by definition*, and *subject reduction is also ensured by definition*.
4. Finally, a typing rule of the kind
$$\frac{\Delta \vdash r : B}{\Gamma \vdash t : A}$$
is valid as soon as $\Delta \vdash r : B$ implies $\Gamma \vdash t : A$. This way, typing rules become *theorems* (potentially infinite many of them).

So, instead of defining typing rules and proving all its desirable properties, we give the desirable properties by definition and prove the typing rules. Of course this recipe does not tell us how to statically determine whether $t \perp r$ for some arbitrary terms t and r, but it gives us a way to check whether any given typing rule is valid, and forces the system to only let pass whatever terms we want (for example, only allow for terms of type $S\mathbb{B} \Rightarrow S\mathbb{B}$ if those terms represent unitary maps).

The notation in this system is a bit modified from that of Lambda-$\mathcal{S}$, so instead of SA we write $\sharp A$ for the set $\mathsf{Span}[\![A]\!] \cap \{v \mid \|v\| = 1\}$ and we even have a type $\flat A$ which is interpreted as the smallest set V of values such that $\sharp V$ contains $[\![A]\!]$. We do not give more details in this quick overview, since there are many, but it is worth mentioning that a quantum lambda calculus in the spirit of the QLC [47] is translated into this calculus in [15].

As mentioned in the step 4 of the above enumeration, the typing rules are potentially infinite. So, in [19] we extracted a finite fixed type system, defining Lambda-$\mathcal{S}_1$, and gave a categorical interpretation for it. The model developed has some common grounds with the concrete model of Lambda-$\mathcal{S}$ [16, 18], however, the chosen categories this time are not **Set** and **Vec**, but categories that use the fact that values in this calculus form a "distributive-action space" (an algebraic structure similar to a vector space, where its additive structure is a semigroup). The two categories in the constructed adjunction are defined in terms of $\vec{\mathrm{V}}$, the set of values in the calculus, and their linear combinations (such a set forms a distributive-action space). Then, the categories for the adjunction are defined by:

- $\mathbf{Set}_{\vec{\mathrm{V}}}$: a category whose objects are the non-empty parts of $\vec{\mathrm{V}}$, and whose arrows are the arrows in **Set** that can be defined in Lambda-$\mathcal{S}_1$. This category also includes a product $\boxtimes$, which is the set of separable tensor products.
- $\mathbf{SVec}_{\vec{\mathrm{V}}}$: a category whose objects are the sub-distributive action spaces of $\vec{\mathrm{V}}$, and whose arrows are the linear maps which can be defined in Lambda-$\mathcal{S}_1$. It also includes a tensor product $\otimes$, which can also be defined as the span of the product $\boxtimes$.

The main novelty and contribution of [19] is presenting a model for quantum computing in the quantum control paradigm, which is show to be complete on qubits in the sense that if two closed terms with qubit types are interpreted by the same arrows in the model, then those terms are computationally equivalent.

7 Sup: A new connective in Natural Deduction

As mentioned in the introduction, one of the main goals of the quantum control approach to the lambda calculus is to envisage a quantum computational logic (i.e. a quantum logic founded by quantum computing and an extension to the Curry-Howard correspondence). In Section 4 we showed Vectorial, where if A and B are two propositions, so is $\alpha.A+\beta.B$. So, we could restrict the valid propositions A to those for whom there are valid proof terms t such that $\vdash t : A$ and t represents a quantum state. Vectorial can be seen as the propositional logic of vector spaces. In Section 5 we gave another form to the superposition of propositions: if A is a proposition, then SA is a superposition of propositions A. In this case the superposition symbol is unary, and from several proofs t_i of A we can construct a proof $\sum_i \alpha_i.t_i$ of SA. Similarly, in Section 6 we not only gave the superpositions SA (now noted $\sharp A$), but also another unary symbol $\flat$, from where if A is a proposition, $\flat A$ is the proposition whose set of proofs is the minimum set such that the unary span of such a set is the set of proofs of $\sharp A$.

In all these previous works we started from the lambda calculus Lineal, and worked out a logic (a type system) from it. In [13] we started from the other end. Starting from Natural Deduction we worked out a new connective for superpositions, and from there we gave a proof system which can be used to encode quantum computing on it.

The rest of this section paraphrases the extended abstract of [13] we have presented at QPL 2021. Two nice video presentations given by Gilles Dowek can be found at [22, 23].

Insufficient, harmonious, and excessive connectives In natural deduction, to prove a proposition C, the elimination rule of a connective $\triangle$ requires a proof of $A \triangle B$ and a proof of C using, as extra hypotheses, exactly the premises needed to prove the proposition $A \triangle B$, with the introduction rules of the connective $\triangle$. This principle of inversion, or of harmony, has been introduced by Gentzen [25] and developed, among others, by Prawitz [38] and Dummett [24] in natural deduction, by Miller and Pimentel [31] in sequent calculus, and by Read [40–42] for the rules of equality.

For example, to prove the proposition $A\wedge B$, the introduction rule of the conjunction requires a proof of A and a proof of B, hence, to prove a proposition C, the generalised elimination rule of the conjunction [33, 36, 45] requires, a proof of $A\wedge B$ and a proof of C, using, as extra hypotheses, the propositions A and B

$$\frac{\Gamma\vdash A\wedge B \quad \Gamma,A,B\vdash C}{\Gamma\vdash C}\ \wedge\text{-e}$$

This principle of inversion permits to define a cut elimination process where the proof

$$\cfrac{\cfrac{\cfrac{\pi_1}{\Gamma\vdash A} \quad \cfrac{\pi_2}{\Gamma\vdash B}}{\Gamma\vdash A\wedge B}\ \wedge\text{-i} \quad \cfrac{\pi_3}{\Gamma,A,B\vdash C}}{\Gamma\vdash C}\ \wedge\text{-e}$$

reduces to $(\pi_1/A,\pi_2/B)\pi_3$.

In the same way, to prove the proposition $A\vee B$, the introduction rules of the disjunction require a proof of A or a proof of B, hence, to prove a proposition C, the elimination rule of the disjunction requires a proof of $A\vee B$ and two proofs of C, one using, as extra hypothesis, the proposition A and the other the proposition B

$$\frac{\Gamma\vdash A}{\Gamma\vdash A\vee B}\ \vee\text{-i1} \qquad \frac{\Gamma\vdash B}{\Gamma\vdash A\vee B}\ \vee\text{-i2} \qquad \frac{\Gamma\vdash A\vee B \quad \Gamma,A\vdash C \quad \Gamma,B\vdash C}{\Gamma\vdash C}\ \vee\text{-e}$$

and a cut elimination process can be defined similarly.

We also can imagine connectives that do not verify this inversion principle, because the introduction rules require an insufficient amount of information with respect to what the elimination rule provides, as extra hypotheses, in the required proof of C. An example of such an *insufficient* connective is Prior's *tonk* [39], with the introduction and elimination rules as follows

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \textit{ tonk } B}\ \textit{tonk}\text{-i} \qquad\qquad \frac{\Gamma \vdash A \textit{ tonk } B \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \textit{tonk}\text{-e}$$

where the elimination rule requires a proof of A *tonk* B and a proof of C, using the extra hypothesis B, that is not required in the proof of A *tonk* B, with the introduction rule. For such connectives, cuts *tonk*-i / *tonk*-e cannot be reduced.

But, it is also possible that a connective does not verify the inversion principle because the introduction rules require an excessive amount of information. An example of such an *excessive* connective is the connective $\odot$ that has the introduction rule of the conjunction and the elimination rule of the disjunction

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \odot B}\ \odot\text{-i} \qquad\qquad \frac{\Gamma \vdash A \odot B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \odot\text{-e}$$

In this case, cuts can be eliminated. Moreover, several cut elimination processes can be defined, exploiting, in different ways, the excess of the connective. For example, the $\odot$-cut

$$\frac{\dfrac{\dfrac{\pi_1}{\Gamma \vdash A} \quad \dfrac{\pi_2}{\Gamma \vdash B}}{\Gamma \vdash A \odot B}\ \odot\text{-i} \quad \dfrac{\pi_3}{\Gamma, A \vdash C} \quad \dfrac{\pi_4}{\Gamma, B \vdash C}}{\Gamma \vdash C}\ \odot\text{-e}$$

can be reduced to $(\pi_1/A)\pi_3$, it can be reduced to $(\pi_2/A)\pi_4$, it also can be reduced, non-deterministically, either to $(\pi_1/A)\pi_3$ or to $(\pi_2/A)\pi_4$. Finally, to keep both proofs, we can add a structural rule

$$\frac{\Gamma \vdash A \quad \Gamma \vdash A}{\Gamma \vdash A}\ \text{parallel} \qquad \text{and reduce it to} \qquad \frac{\dfrac{(\pi_1/A)\pi_3}{\Gamma \vdash C} \quad \dfrac{(\pi_2/B)\pi_4}{\Gamma \vdash C}}{\Gamma \vdash C}\ \text{parallel}$$

Information loss With harmonious connectives, when a proof is built with an introduction rule, the information contained in the proofs of the premises of this rule is preserved. For example, the information contained in the proof π_1 is *present* in the proof π

$$\frac{\dfrac{\pi_1}{\Gamma \vdash A} \quad \dfrac{\pi_2}{\Gamma \vdash B}}{\Gamma \vdash A \wedge B}\ \wedge\text{-i}$$

in the sense that π_1 is a subtree of π. But it is moreover *accessible*, in the sense that, for all π_1, putting the proof π in the right context yields a proof that reduces to π_1. And the same holds for the proof π_2.

The situation is different with an excessive connective: the excess of information, required by the introduction rule, and not returned by the elimination rule in the form of an extra hypothesis, in the required proof of C, is lost. For example, the information contained in the proofs π_1 and π_2 is present in the proof

$$\frac{\dfrac{\pi_1}{\Gamma \vdash A} \quad \dfrac{\pi_2}{\Gamma \vdash B}}{\Gamma \vdash A \odot B}\ \odot\text{-i}$$

but its accessibility depends on the way we decide to reduce the cut

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma\vdash A}\quad\dfrac{\pi_2}{\Gamma\vdash B}}{\Gamma\vdash A\odot B}\,\odot\text{-i}\qquad \dfrac{\pi_3}{\Gamma,A\vdash C}\qquad \dfrac{\pi_4}{\Gamma,B\vdash C}}{\Gamma\vdash C}\,\odot\text{-e}$$

If we reduce it systematically to $(\pi_1/A)\pi_3$, then the information contained in π_1 is accessible, but that contained in π_2 is not. If we reduce it systematically to $(\pi_2/A)\pi_4$, then the information contained in π_2 is accessible, but not that contained in π_1. If we reduce it not deterministically to $(\pi_1/A)\pi_3$ or to $(\pi_2/A)\pi_4$, then the information contained in both π_1 and π_2 is accessible but non-deterministically. If we reduce it with parallel, then the information contained in both π_1 and π_2 is inaccessible.

So, while harmonious connectives, that verify the inversion principle, model information preservation, reversibility, and determinism, these excessive connectives, that do not verify the inversion principle, model information erasure, non-reversibility, and non-determinism. Such information erasure, non-reversibility, and non-determinism, occur, for example, in quantum physics, where the measurement of the superposition of two states does not yield both states back.

A quantum language with $\odot$ As we have seen in the previous sections, several programming languages have been designed to express quantum algorithms with quantum control. In particular, in Lambda $\mathcal{S}$ (see Section 5), the measurement operator π comes together with the rule reducing $\pi(t+r)$ non-deterministically to t or to r.

The superposition $t+r$ can be considered as the pair $\langle t,r\rangle$, as stated by [21]. Hence, it should have the type $A\wedge A$. In other words, it is a proof-term of the proposition $A\wedge A$. In System I (first introduced in [11] and later extended in [12,20,49]), various type-isomorphisms have been introduced, in particular the commutativity isomorphism $A\wedge B\equiv B\wedge A$, hence $t+r\equiv r+t$. In such a system, where $A\wedge B$ and $B\wedge A$ are identical, it is not possible to define the two elimination rules, as the two usual projections rules π_1 and π_2 of the λ-calculus. They were replaced with a single projection parametrised with a proposition A: π_A, such that if t is typed by A and r by B then $\pi_A(t+r)$ reduces to t and $\pi_B(t+r)$ to r. When $A=B$, so t and r both have type A, the proof-term $\pi_A(t+r)$ reduces, non-deterministically, to t or to r. Thus, this modified elimination rule behaves like a measurement operator.

These works on Lambda-$\mathcal{S}$ and System I brought to light the fact that the pair superposition / measurement, in a quantum programming language, behaves like a pair introduction / elimination, for some connective, in a proof language, as the succession of a superposition and a measurement yields a term that can be reduced. In System I, the assumption was made that this connective was a commutative conjunction, with a modified elimination rule, leading to a non-deterministic reduction.

However, as the measurement of the superposition of two states does not yield both states back, this connective should probably be excessive. Moreover, as, to build the superposition $a.|0\rangle+b.|1\rangle$, we need both $|0\rangle$ and $|1\rangle$ and the measurement, in the basis $|0\rangle,|1\rangle$, yields either $|0\rangle$ or $|1\rangle$, this connective should have the introduction rule of the conjunction, and the elimination rule of the disjunction, that is that it should be the connective $\odot$.

In [13], we present a propositional logic with the connective $\odot$, a language of proof-terms, the $\odot$-calculus (read: "the sup-calculus"), for this logic, and we prove a cut elimination theorem. We then extend this calculus into the $\odot^{\mathbb{C}}$-calculus (sup-$\mathbb{C}$-calculus), introducing scalars to quantify the propensity of a proof to reduce to another and show that its proof language forms the core of a quantum programming language.

In particular, $\vdash \alpha.* : \top$ making of $\top$ a representation of $\mathbb{C}$. In addition, while $\top \vee \top$ represent the booleans in the lambda calculus, $\top \odot \top$ represents the qubits, $\mathbb{C}^2$. We use $+$ to the proof term of the propositions $\odot$, and so $\vdash: \alpha.*+\beta.* : \top \odot \top$ represents the vector $\alpha.|0\rangle + \beta.|1\rangle$.

The elimination of $\odot$ has two proof terms:

$$\frac{\Gamma \vdash t : A \odot B \quad \Gamma, x : A \vdash r : C \quad \Gamma, y : A \vdash r : C}{\Gamma \vdash \delta_{\odot}^{\parallel}(t, [x]r, [y]s) : C} \qquad \frac{\Gamma \vdash t : A \odot B \quad \Gamma, x : A \vdash r : C \quad \Gamma, y : A \vdash r : C}{\Gamma \vdash \delta_{\odot}(t, [x]r, [y]s) : C}$$

The first one reduces with the following rule

$$\delta_{\odot}^{\parallel}(\alpha.t + \beta.r, [x]s_1, [y]s_2) \longrightarrow \alpha.(t/x)s_1 \parallel \beta.(r/x)s_2$$

where the $\parallel$ is such that it behaves as a vectorial sum.

The second one reduces with the following rules instead

$$\delta_{\odot}(\alpha.t + \beta.r, [x]s_1, [y]s_2) \longrightarrow (t/x)s_1$$
$$\delta_{\odot}(\alpha.t + \beta.r, [x]s_1, [y]s_2) \longrightarrow (r/x)s_2$$

where the first reduction happens with probability $\frac{|\alpha|^2}{|\alpha|^2+|\beta|^2}$ and the second with probability $\frac{|\beta|^2}{|\alpha|^2+|\beta|^2}$.

The term $\delta_{\odot}^{\parallel}$ is then used to encode quantum gates and $\delta_{\odot}$ to encode the measurement. See [13] for more details.

8 Towards quantum recursive types

A while language called qGCL with quantum control has been introduced in [44], and an entire book on the subject has been written by Ying [53]. The idea is to consider an infinite-dimensional Hilbert space and the while is guarded by a binary measurement on one qubit, which stops when the outcome is 0.

In [43], a similar idea of quantum control loop has been taken to the lambda calculus, by endowing a typed, reversible, algebraic lambda calculus of structural recursive fixpoints linked to the convergence of sequences in infinite-dimensional Hilbert spaces. A categorical semantics of this language is given in [30]. In [9], a language, based on this reversible language, is typed in μMALL (linear logic extended with least and greatest fixed points) allowing inductive and coinductive statements. While the paper [9] is subtitled “work-in-progress”, it is a firm first step towards quantum recursive types.

9 Some final thoughts

This quick overview aims to give a bird’s eye view on the quest for a quantum computational logic. There are several clues on how this logic should be. Either by representing superposition of propositions as $\alpha.A + \beta.B$ (see Section 4), as SA or $\sharp A$ (see Sections 5 and 6) or by $A \odot B$ (see Section 7), the many options present different approaches, but all of them are founded by the Curry-Howard isomorphism: a proposition is a type and a proof is a term.

There is still work to do. One may wonder if from $\odot$ we may define a measurement in Vectorial, for example, where the $+$ symbol at $\alpha.A + \beta.B$ can be eliminated by a disjunction elimination. Or what is the meaning of $\odot$ in sequent calculus, what categorical construction may model it, etc.

All of those are open problems we are willing to investigate.

Acknowledgements

The author wishes to thank Eduardo Bonelli and Mauricio Ayala-Rincón for the inclusion of this overview in LSFA 2021 as an invited talk, and for their useful comments.

References

[1] Gadi Aleksandrowicz et. al. (2019): *Qiskit: An open-source framework for quantum computing*, doi:10.5281/zenodo.2562111.

[2] Thorsten Altenkirch & Jonathan J. Grattage (2005): *A functional quantum programming language*. In: *Proceedings of LICS-2005*, IEEE Computer Society, pp. 249–258, doi:10.1109/LICS.2005.1.

[3] Pablo Arrighi & Alejandro Díaz-Caro (2012): *A System F accounting for scalars*. *Logical Methods in Computer Science* 8(1:11), doi:10.2168/LMCS-8(1:11)2012.

[4] Pablo Arrighi, Alejandro Díaz-Caro & Benoît Valiron (2017): *The vectorial λ-calculus*. *Information and Computation* 254(1), pp. 105–139, doi:10.1016/j.ic.2017.04.001.

[5] Pablo Arrighi & Gilles Dowek (2008): *Linear-algebraic λ-calculus: higher-order, encodings, and confluence*. In Andrei Voronkov, editor: *Rewritting Techniques and Applications (RTA 2008)*, *Lecture Notes in Computer Science* 5117, Springer, pp. 17–31, doi:10.1007/978-3-540-70590-1_2.

[6] Pablo Arrighi & Gilles Dowek (2017): *Lineal: A linear-algebraic lambda-calculus*. *Logical Methods in Computer Science* 13(1:8), doi:10.23638/LMCS-13(1:8)2017.

[7] Nick Benton (1994): *A mixed linear and non-linear logic: Proofs, terms and models*. In Leszek Pacholski & Jerzy Tiuryn, editors: *Computer Science Logic (CSL 1994)*, *Lecture Notes in Computer Science* 933, Springer, pp. 121–135, doi:10.1007/BFb0022251.

[8] Garrett Birkhoff & John von Neumann (1936): *The logic of quantum mechanics*. *Annals of Mathematics* 37(4), pp. 823–843, doi:10.2307/1968621.

[9] Kostia Chardonnet, Alexis Saurin & Benoît Valiron (2020): *Towards a Curry-Howard equivalence for linear, reversible computation*. In Ivan Lanese & Mariusz Rawski, editors: *Reversible Computation (RC 2020)*, *Lecture Notes in Computer Science* 12227, Springer, pp. 348–364, doi:10.1007/978-3-030-52482-1_8.

[10] Alejandro Díaz-Caro & Gilles Dowek (2017): *Typing quantum superpositions and measurement*. In Carlos Martín-Vide, Roman Neruda & Miguel A. Vega-Rodríguez, editors: *Theory and Practice of Natural Computing (TPNC 2017)*, *Lecture Notes in Computer Science* 10687, Springer, pp. 281–293, doi:10.1007/978-3-319-71069-3_22.

[11] Alejandro Díaz-Caro & Gilles Dowek (2019): *Proof normalisation in a logic identifying isomorphic propositions*. In Herman Geuvers, editor: *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 131, pp. 14:1–14:23, doi:10.4230/LIPIcs.FSCD.2019.14.

[12] Alejandro Díaz-Caro & Gilles Dowek (2020): *Extensional proofs in a propositional logic modulo isomorphisms*. arXiv:2002.03762.

[13] Alejandro Díaz-Caro & Gilles Dowek (2021): *A new connective in natural deduction, and its application to quantum computing*. In Antonio Cerone & Peter Csaba Ölveczky, editors: *Theoretical Aspects of Computing (ICTAC 2021)*, *Lecture Notes in Computer Science* 12819, Springer, pp. 175–193, doi:10.1007/978-3-030-85315-0_11.

[14] Alejandro Díaz-Caro, Gilles Dowek & Juan Pablo Rinaldi (2019): *Two linearities for quantum computing in the lambda calculus*. *BioSystems* 186, p. 104012, doi:10.1016/j.biosystems.2019.104012.

[15] Alejandro Díaz-Caro, Mauricio Guillermo, Alexandre Miquel & Benoît Valiron (2019): *Realizability in the unitary sphere*. In: *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2019)*, pp. 1–13, doi:10.1109/LICS.2019.8785834.

[16] Alejandro Díaz-Caro & Octavio Malherbe (2019): *A concrete categorical semantics for Lambda-S*. In Beniamino Accattoli & Carlos Olarte, editors: *Logical and Semantic Frameworks with Applications (LSFA'18)*, *Electronic Notes in Theoretical Computer Science* 344, Elsevier, pp. 83–100, doi:10.1016/j.entcs.2019.07.006.

[17] Alejandro Díaz-Caro & Octavio Malherbe (2020): *A categorical construction for the computational definition of vector spaces*. *Applied Categorical Structures* 28(5), pp. 807–844, doi:10.1007/s10485-020-09598-7.

[18] Alejandro Díaz-Caro & Octavio Malherbe (2021): *A concrete model for a linear algebraic lambda calculus*. arXiv:1806.09236.

[19] Alejandro Díaz-Caro & Octavio Malherbe (2021): *Quantum control in the unitary sphere: Lambda-$\mathcal{S}_1$ and its categorical model*. arXiv:2012.05887.

[20] Alejandro Díaz-Caro & Pablo E. Martínez López (2015): *Isomorphisms considered as equalities: Projecting functions and enhancing partial application through an implementation of λ^+*. In: *Proceedings of the 27th Symposium on the Implementation and Application of Functional Programming Languages (IFL 2015)*, ICPS Proceedings, ACM, pp. 9:1–9:11, doi:10.1145/2897336.2897346.

[21] Alejandro Díaz-Caro & Barbara Petit (2012): *Linearity in the non-deterministic call-by-value setting*. In Luke Ong & Ruy de Queiroz, editors: *Logic, Language, Information and Computation (WoLLIC 2012)*, *Lecture Notes in Computer Science* 7456, pp. 216–231, doi:10.1007/978-3-642-32621-9_16.

[22] Gilles Dowek (2021): *Presentation of [13] at ICTAC 2021*. Available at `https://drive.google.com/file/d/1E1DLQfTUg48jC325kOCNnftVgBe_k01A/view`.

[23] Gilles Dowek (2021): *Presentation of [13] at QPL 2021*. Available at `https://www.youtube.com/watch?v=au0TDDp5qSw`.

[24] Michael Dummett (1991): *The logical basis of metaphysics*. Duckworth.

[25] Gerhard Gentzen (1935): *Untersuchungen über das logische Schließen*. *Mathematische Zeitschrift* 39, pp. 176–210, doi:10.1007/BF01201353.

[26] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger & Benoît Valiron (2013): *Quipper: a scalable quantum programming language*. *ACM SIGPLAN Notices (PLDI'13)* 48(6), pp. 333–342, doi:10.1145/2491956.2462177.

[27] Stephen C. Kleene (1945): *On the interpretation of intuitionistic number theory*. *The Journal of Symbolic Logic* 10(4), pp. 109–124, doi:10.2307/2269016.

[28] Emanuel H. Knill (1996): *Conventions for quantum pseudocode*. Technical Report LA-UR-96-2724, Los Alamos National Laboratory, doi:10.2172/366453.

[29] Jean-Louis Krivine (2009): *Realizability in classical logic*. *Panoramas et synthèses: Interactive models of computation and program behaviour* 27, pp. 197–229. Available at `hal-00154500`.

[30] Louis Lemonnier, Kostia Chardonnet & Benoît Valiron (2021): *Categorical semantics of reversible pattern-matching*. arXiv:2109.05837.

[31] Dave Miller & Elaine Pimentel (2013): *A formal framework for specifying sequent calculus proof systems*. *Theoretical Computer Science* 474, pp. 98–116, doi:10.1016/j.tcs.2012.12.008.

[32] Alexandre Miquel (2011): *A survey of classical realizability*. In Luke Ong, editor: *Typed Lambda Calculi and Applications (TLCA 2011)*, *Lecture Notes in Computer Science* 6690, pp. 1–2, doi:10.1007/978-3-642-21691-6_1.

[33] Sara Negri & Jan von Plato (2008): *Structural Proof Theory*. Cambridge University Press.

[34] Michael Nielsen & Isaac Chuang (2000): *Quantum Computation and Quantum Information*. Cambridge University Press., doi:10.1017/CBO9780511976667.

[35] Francisco Noriega & Alejandro Díaz-Caro (2020): *The Vectorial Lambda Calculus Revisited*. arXiv:2007.03648.

[36] Michel Parigot (1991): *Free deduction: An analysis of "Computations" in classical logic*. In A. Voronkov, editor: *Logic Programming*, *Lecture Notes in Computer Science* 592, Springer, pp. 361–380, doi:10.1007/3-540-55460-2_27.

[37] Jennifer Paykin, Robert Rand & Steve Zdancewic (2017): *QWIRE: A Core Language for Quantum Circuits*. In: *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, ACM, pp. 846–858, doi:10.1145/3009837.3009894.

[38] Dag Prawitz (1965): *Natural deduction. A proof-theoretical study*. Almqvist & Wiksell.

[39] Arthur N. Prior (1960): *The runabout inference-ticket*. *Analysis* 21(2), pp. 38–39, doi:10.1093/analys/21.2.38.

[40] Stephen Read (2004): *Identity and harmony*. *Analysis* 64(2), pp. 113–119, doi:10.1093/analys/64.2.113.

[41] Stephen Read (2010): *General-elimination harmony and the meaning of the logical constants*. *Journal of Philosophical Logic* 39, pp. 557–576, doi:10.1007/s10992-010-9133-7.

[42] Stephen Read (2014): *Identity and harmony revisited*. Available at `https://www.st-andrews.ac.uk/~slr/identity_revisited.pdf`. Informal publication.

[43] Amr Sabry, Benoît Valiron & Juliana Kaizer Vizzotto (2018): *From Symmetric Pattern-Matching to Quantum Control*. In Christel Baier & Ugo Dal Lago, editors: *Foundations of Software Science and Computation Structures (FoSSaCS 2018)*, *Lecture Notes in Computer Science* 10803, Springer, pp. 348–364, doi:10.1007/978-3-319-89366-2_19.

[44] Jeff W. Sanders & Paolo Zuliani (2000): *Quantum programming*. In Roland Backhouse & José Nuno Oliveira, editors: *Mathematics of Program Construction (MPC 2000)*, *Lecture Notes in Computer Science* 1837, Springer, pp. 80–99, doi:10.1007/10722010_6.

[45] Peter Schroeder-Heister (1984): *A natural extension of Natural deduction*. *The Journal of Symbolic Logic* 49(4), pp. 1284–1300, doi:10.2307/2274279.

[46] Peter Selinger (2004): *Towards a quantum programming language*. *Mathematical Structures in Computer Science* 14(4), pp. 527–586, doi:10.1017/S0960129504004256.

[47] Peter Selinger & Benoît Valiron (2006): *A lambda calculus for quantum computation with classical control*. *Mathematical Structures in Computer Science* 16(3), pp. 527–552, doi:10.1017/S0960129506005238.

[48] Morten Heine Sørensen & Paweł Urzyczyn (1998): *Lectures on the Curry-Howard isomorphism*. *Studies in Logic and the Foundations of Mathematics* 149, Elsevier.

[49] Cristian Sottile, Alejandro Díaz-Caro & Pablo E. Martínez López (2020): *Polymorphic System I*. In: *Proceedings of the 32nd Symposium on the Implementation and Application of Functional Programming Languages (IFL 2020)*, ICPS Proceedings, ACM, pp. 127–137, doi:10.1145/3462172.3462198.

[50] Microsoft Quantum Team (2017): *The Q$\sharp$ programming language*. Available at `https://docs.microsoft.com/en-us/quantum/quantum-qr-intro?view=qsharp-preview`.

[51] Jaap van Oosten (2008): *Realizability. An introduction to its categorical side*. *Studies in Logic and the Foundations of Mathematics* 152, Elsevier.

[52] William K. Wootters & Wojciech H. Zurek (1982): *A single quantum cannot be cloned*. *Nature* 299, pp. 802–803, doi:10.1038/299802a0.

[53] Mingsheng Ying (2016): *Foundations of Quantum Programming*. Elsevier.