Basis-Sensitive Quantum Typing via Realizability*

Alejandro Díaz-Caro
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
Universidad Nacional de Quilmes, Departamento de Ciencia y Tecnología, Argentina
Universidad de la República, CENUR Suroeste, InCo, Uruguay

Octavio Malherbe
Universidad de la República, Facultad de Ingeniería, IMERL, Uruguay

Rafael Romero
Universidad Nacional de Quilmes, Departamento de Ciencia y Tecnología, Argentina
ICC, CONICET-Universidad de Buenos Aires, Argentina

1 Introduction

The no-cloning theorem [7] and the no-deleting theorem [6] are two well-known results that state it is impossible to copy or delete an arbitrary qubit. There is, however, a subtlety: although arbitrary qubits cannot be copied or deleted, this is possible for known—and, in the case of deletion, separable—qubits. This implies that qubits with known values behave as classical data and can be treated accordingly. Moreover, it suffices to know the basis to which a qubit belongs in order to copy and delete it.

In most quantum programming languages, qubits are defined with respect to a canonical basis—often referred to as the computational basis. In this setting, classical bits correspond to the basis vectors, whereas qubits are unit-norm linear combinations of them. Classical bits can be copied and deleted freely, while such operations on arbitrary qubits are restricted. In this work, we introduce a quantum λ-calculus in the quantum-control paradigm—by contrast with the classical-control one, where the control flow is classical and cannot be superposed.

To do so, we work with intuitionistic realisability, which provides a constructive framework that connects operational semantics with type systems. In our context, it allows the extraction of a sound type system directly from the reduction semantics of the calculus, ensuring that safety properties hold by construction. Each typing rule corresponds to a provable theorem in this setting. Rather than building ad-hoc typing rules, the system is derived from the computational content of the calculus.

2 The λ_B calculus

For the calculus, we adopt the grammar described in Table 1. It includes pair constructors and destructors, two boolean values, and a quantum conditional case construct. These terms are closed under linear combination, forming a vector space over $\mathbb{C}$. We define a language in the "quantum data and control" paradigm, which allows the superposition of programs. Such superpositions are represented as norm-1 linear combinations of terms. We will equip the vector space with an inner product, which itself enables the notion of orthogonal terms.

*Full paper can be found in: arXiv:2510.18542v1.

Submitted to:

$$v ::= x \mid \lambda x^B.\vec{t} \mid (v,v) \mid |0\rangle \mid |1\rangle \qquad (\mathrm{V})$$
$$t ::= v \mid tt \mid (t,t) \mid \mathsf{let}\,(x^{B_1},y^{B_2}) = \vec{t}\ \mathsf{in}\ \vec{t} \mid \mathsf{case}\ \vec{t}\ \mathsf{of}\ \{\vec{v} \mapsto \vec{t} \mid \cdots \mid \vec{v} \mapsto \vec{t}\} \qquad (\Lambda)$$
$$\vec{v} ::= v \mid \vec{v} + \vec{v} \mid \alpha\vec{v} \mid \vec{0} \qquad (\alpha \in \mathbb{C}) \qquad (\vec{\mathrm{V}})$$
$$\vec{t} ::= t \mid \vec{t} + \vec{t} \mid \alpha\vec{t} \mid \vec{0} \qquad (\alpha \in \mathbb{C}) \qquad (\vec{\Lambda})$$

Table 1: Syntax of the calculus, where $B, B_1, B_2 \subseteq \vec{\mathrm{V}}$.

$$(\lambda x^X.\vec{t})\vec{v} \rightsquigarrow \vec{t}\langle\vec{v}/x\rangle_X$$
$$\mathsf{let}\,(x^X,y^Y) = \sum_{i=1}^n \alpha_i(\vec{v_i},\vec{w_i})\ \mathsf{in}\ \vec{t} \rightsquigarrow \sum_{i=1}^n \alpha_i\,\vec{t}\langle\vec{v_i}/x\rangle_X\langle\vec{w_i}/y\rangle_Y$$
$$\mathsf{case}\ \sum_{i=1}^n \alpha_i\vec{v_i}\ \mathsf{of}\ \{\vec{v_1} \mapsto \vec{t_1} \mid \cdots \mid \vec{v_n} \mapsto \vec{t_n}\} \rightsquigarrow \sum_{i=1}^n \alpha_i\vec{t_i}$$

Table 2: Reduction system. We omit contextual rules

The novel feature of the calculus lies in its abstraction construct, which is decorated with an orthogonal basis B. This will inform the reduction system on how to treat the values passed as arguments. Intuitively, values expressed in the chosen basis will represent classical data, while linear combinations of these values correspond to quantum data and reduce linearly within the term. This refinement enables duplication and erasure for qubits in known bases while maintaining linear handling for unknown qubits. We then have to define a substitution operation that incorporates this behaviour:

Definition 2.1 (Basis-dependent substitution)**.** *Let $B = \{\vec{b_i}\}_{i\in I}$ be an orthonormal basis, $\vec{t}$ a term distribution, x a variable and, $\vec{v}$ a value distribution that can be decomposed as $\vec{v} = \sum_{i\in I}\alpha_i\vec{b_i}$. We define the linear substitution as: $\vec{t}\langle\vec{v}/x\rangle_B = \sum_{i\in I}\alpha_i\,\vec{t}\,[\vec{b_i}/x]$.*

As usual, the expression $\vec{t}\,[\vec{v}/x]$ denotes the capture-avoiding substitution of $\vec{v}$ for x in $\vec{t}$. Intuitively, this operation substitutes variables with vectors expressed in the chosen basis; the accompanying coefficients are those of the value distribution being substituted.

With the substitution defined, we can design the reduction system in Table 2. We take the liberty to omit the contextual rules, since they are fairly standard. The reduction is weak, meaning there is no action under lambda abstractions nor in the branches of the case construct. The evaluation of terms only occurs in cases where the basis-dependent substitution is well-defined. In case an argument cannot be decomposed onto a basis, the term gets stuck.

Example 2.2. *Using this system, we can consider the duplicator function in the Hadamard basis $\mathcal{X} = \{|+\rangle,|-\rangle\}$ applied to either $|+\rangle$ or $|0\rangle$ as such:*

$$(\lambda x^{\mathcal{X}}.(x,x))|+\rangle \rightsquigarrow (|+\rangle,|+\rangle)$$
$$(\lambda x^{\mathcal{X}}.(x,x))|0\rangle = (\lambda x^{\mathcal{X}}.(x,x))(\frac{1}{\sqrt{2}}|+\rangle + \frac{1}{\sqrt{2}}|-\rangle) \rightsquigarrow \frac{1}{\sqrt{2}}(|+\rangle,|+\rangle) + \frac{1}{\sqrt{2}}(|-\rangle,|-\rangle)$$

This is not a new operation, any language that implements arbitrary unitary gates is able to encode this program. The main point here is that *the operation is native to the language*, without even introducing the

$$[\![\flat_X]\!] = X \qquad [\![A \times B]\!] := \left\{(\vec{v},\vec{w}) : \vec{v} \in [\![A]\!],\ \vec{w} \in [\![B]\!]\right\}$$
$$[\![\sharp A]\!] := ([\![A]\!]^{\perp})^{\perp} \qquad [\![A \to B]\!] := \left\{\vec{w}\ :\ \forall \vec{v} \in [\![A]\!],\ \vec{w}\vec{v} \rightsquigarrow^* \vec{u} \in [\![B]\!]\right\}$$

With X an orthogonal basis and, $A^{\perp}$ the set of norm-1 vectors orthogonal to every member of A.

Table 3: Unitary semantics of types

concept of unitary gate. Adding more information into the term allow us more flexibility when writing functions. In doing so, we hope to add a layer of abstraction between language and quantum circuits, more in line with modern classical programming languages.

Typing system. Working via intuitionistic realisability, the typing system is first built from sets of value distributions. We will choose sets of orthogonal qubits to act as bases and a simple algebra to form more complex types. The final piece is an operation which allows us to span a base of values, we call this operation $\sharp$ (*sharp*). Members of this type represent linear combinations of values, and will be our *unknown qubits*. The $\sharp$ operator can be thought as the opposite of the exponential bang !. While the ! marks resources as non-linear, the $\sharp$ instead marks them as linear (i.e. non-duplicable). The type semantics is described in Table 3.

Once we have a type algebra defined, we can start proving the validity of some rules. This will ensure that the extracted system will be correct by construction. Intuitively, we will state that a judgement $\Gamma \vdash \vec{t} : A$ is valid when: "Every substitution σ such that $\sigma(x) \in [\![\Gamma(x)]\!]$ validates that $\sigma(\vec{t})$, reduces to $\vec{w} \in [\![A]\!]$". In the same way, a rule is valid if starting from valid hypotheses we arrive to a valid conclusion. There are infinite valid rules, since typing rules are just theorems that we can prove, but we are interested in those which will help us to build a sufficiently expressive language.

The main result establishes a connection between λ-abstractions and operators in $\mathbb{C}^{2^n}$. We will say a lambda term represents a function $F : \mathbb{C}^n \to \mathbb{C}^n$ if it encodes the action of F on vectors. With the typing judgements we have, we can characterize unitary operators as functions in the unitary semantic of arrow types.

Theorem 2.3 (Characterisation of Unitary Operators)**.** *Let X and Y be orthonormal bases of size 2^n. A closed λ-abstraction $\lambda x^X.\vec{t} \in [\![\sharp\flat_X \to \sharp\flat_Y]\!]$ if and only if it represents a unitary operator $F : \mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$.*

3 Conclusions

In this work we have explored a quantum-control λ-calculus equipped with the feature of allowing abstractions to be expressed relative to arbitrary bases, beyond the canonical one. The benefit of this design becomes clear in the realisability model. The inclusion of atomic types $\flat_X$ enables a direct characterisation of abstractions representing unitary operators, generalising the result from [3]. The use of basis types yields a simpler and more transparent proof.

In the paper linked to this abstract we also prove a set of valid rules which could give form to a typing system for a programming language. Additionally, we provide two use-case examples for the language, Deutsch's algorithm and quantum teleportation.

References

[1] Pablo Arrighi and Gilles Dowek. Lineal: A linear-algebraic λ-calculus. *Logical Methods in Computer Science*, 13(1:8):1–33, 2017.

[2] Alejandro Díaz-Caro, Gilles Dowek, and Juan Pablo Rinaldi. Two linearities for quantum computing in the lambda calculus. *BioSystems*, 186:104012, 2019. Postproceedings of TPNC 2017.

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

[4] Alejandro Díaz-Caro and Nicolas A. Monzon. A quantum-control lambda-calculus with multiple measurement bases. In *APLAS 2025: 23rd Asian Symposium on Programming Languages and Systems*, 2025. To appear at LNCS. Full version at arXiv:2506.16244.

[5] Alejandro Díaz-Caro and Octavio Malherbe. Quantum control in the unitary sphere: Lambda-s1 and its categorical model. *Logical Methods in Computer Science*, Volume 18, Issue 3, September 2022.

[6] Arun K. Pati and Samuel L. Braunstein. Impossibility of deleting an unknown quantum state. *Nature*, 404(6774):164–165, 2000.

[7] William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982.