A Rewriting Theory for Quantum λ-Calculus

Claudia Faggian
IRIF, CNRS, Université Paris Cité, France

Gaetan Lopez
IRIF, CNRS, Université Paris Cité, France

Benoît Valiron
Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, Inria,
Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Abstract

Quantum lambda calculus has been studied mainly as an idealized programming language – the evaluation essentially corresponds to a deterministic abstract machine. Very little work has been done to develop a rewriting theory for quantum lambda calculus. Recent advances in the theory of probabilistic rewriting give us a way to tackle this task with tools unavailable a decade ago. Our primary focus are standardization and normalization results.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Operational semantics; Theory of computation → Equational logic and rewriting; Theory of computation → Linear logic

Keywords and phrases quantum lambda-calculus, probabilistic rewriting, operational semantics, asymptotic normalization, principles of quantum programming languages

Digital Object Identifier 10.4230/LIPIcs.CSL.2025.47

Related Version *Extended Version*: `http://arxiv.org/abs/2411.14856` [25]

Funding This work has been partially funded by the French National Research Agency (ANR) by the projects PPS ANR-19-CE48-0014, TaQC ANR-22-CE47-0012 and within the framework of "Plan France 2030", under the research projects EPIQ ANR-22-PETQ-0007, OQULUS ANR-23-PETQ-0013, HQI-Acquisition ANR-22-PNCQ-0001 and HQI-R&D ANR-22-PNCQ-0002.

1 Introduction

Quantum computation is a model of computation in which one has access to data coded on the state of objects governed by the law of quantum physics. Due to the unique nature of quantum mechanics, quantum data exhibits several non-intuitive properties [37]: it is non-duplicable, it can exist in superposition, and reading the memory exhibits a probabilistic behavior. Nonetheless, the mathematical formalization is well-established: the state of a quantum memory and the available manipulations thereof can be expressed within the theory of Hilbert spaces.

Knill's QRAM model [34] describes a generic interface for interacting with such a quantum memory. The memory is modeled with uniquely identified quantum registers, each holding one quantum bit – also called a qubit. The interface should make it possible to create and discard registers and apply elementary operations on arbitrary registers. These operations consist of *unitary gates* and *measurements*. The former are internal, local modifications of the memory state, represented by a quantum circuit, while the latter are the operations for reading the memory. Measurements are *probabilistic operations* returning a classical bit of information.

Quantum algorithms are typically designed with a model akin to Knill's QRAM [37]. A quantum algorithm consists of the description of a sequence of quantum operations, measurements, and classical processing. The control flow is purely classical, and generally, the algorithm's behavior depends on the results of past measurements. An algorithm, therefore, mixes classical processing and interaction with quantum memory in a potentially arbitrary way: Quantum programming languages should be designed to handle it.

Quantum λ-Calculus and Linear Logics. In the last 20 years, many proposals for quantum programming languages have emerged [28, 30, 40, 39, 8, 11]. Similar to classical languages, the paradigm of higher-order, functional quantum programming languages have been shown to be a fertile playground for the development of well-founded, formal quantum languages aiming at the formal analysis of quantum programs [40, 11].

The *quantum λ-calculus* of Selinger&Valiron [42] lies arguably at the foundation of the development of higher-order, quantum functional programming languages [14, 13, 38, 12, 35]. Akin to other extensions of lambda-calculus with probabilistic [17, 15, 21] or non-deterministic behavior [16], the quantum lambda calculus extends the regular lambda calculus – core of functional programming languages – with a set interface to manipulate a quantum memory. Due to the properties of the quantum memory, quantum lambda-calculi should handle non-duplicable data and probabilistic behavior.

One of the critical points that a quantum programming language should address is the problem of the *non-duplicability* of quantum data. In the literature, non-duplicable data is usually captured with tools coming from linear logic. The first approach [42, 38, 12] consists in using types, imposing all functions to be linear by default, and with the use of a special type $!A$ to explicitly pinpoint duplicable objects of type A. An alternative – untyped – approach [13, 14] considers instead an untyped lambda calculus, augmented with a special term construct "!" and validity constraints to forbid the duplication of qubits.

Probabilistic and Infinitary Behavior. A quantum λ-calculus featuring all of quantum computation should not only permit the manipulation of quantum register with unitary operations but should also give the possibility to *measure* them, and retrieve a classical bit of information. As the latter is a probabilistic operation, an operational semantics for a quantum λ-calculus is inherently probabilistic. As in the non-quantum case, probabilistic choice and unbounded recursion yield subtle behaviors.

Fair Coin. Consider the following program L, written in a mock-up ML language with quantum features, similar to the language of [42]:

$$L \coloneqq \texttt{if}\ \ \texttt{meas}(H\texttt{new})\ \texttt{then}\ I\ \texttt{else}\ \Omega.$$

For this introduction, we only describe its behavior informally. The term L above produces a qubit[1] in state $\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$ by creating a fresh qubit in state $|0\rangle$ (this is the role of `new`), and applying the Hadamard gate H. Measuring this qubit amounts to flipping a fair coin with equal probability $\frac{1}{2}$. In one case, the program returns the identity function I; otherwise, it diverges – the term Ω stands for the usual, non-terminating looping term.

[1] The reader unfamiliar with the notation should not worry, as the formal details are not essential at this point: just retain that *the state of our qubit is a superposition of two (basis) states*, which play the role of head and tail. When needed, in Section 3 we provide a brief introduction to the mathematical formalism for quantum computation [37].

The program L, therefore, uses the quantum memory only once (at the beginning of the run of the program), and it terminates with probability $\frac{1}{2}$.

Unbounded Use of Fair Coin. In the context of probabilistic behavior, unbounded loops might terminate asymptotically: A program may terminate *with probability* 1, but only at the limit (*almost sure termination*). A simple example suffices to illustrate this point.

Consider a quantum process R that flips a coin by creating and measuring a fresh qubit. If the result is head, the process stops, outputting I. If the result is tail, it starts over. In our mock-up ML, the program R is

$$R \coloneqq \texttt{letrec}\ fx = \big(\texttt{if}\ (\texttt{meas}\ x)\texttt{then}\ I\ \texttt{else}\ f(H\texttt{new})\big)\ \texttt{in}\ f(H\texttt{new}). \tag{1}$$

After n iterations, the program R is in normal form with probability $\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^n}$. Even if the termination probability is 1, this probability of termination is not reached in a finite number of steps but *as a limit*. The program in Equation (1) is our leading example: we formalize it in Example 4.3.

Operational Semantics of Quantum Programs. As it is customary when dealing with *choice effects*, the probabilistic behavior is dealt with by fixing an *evaluation strategy*. Think of tossing a (quantum) coin and duplicating the result, versus tossing the coin twice, which is indeed one key issue at the core of confluence failure in such settings (as observed in [16, 13]). Following the standard approach adopted for functional languages with side effects, the evaluation strategy in quantum λ-calculi such as [42, 38, 12] is a deterministic call-by-value strategy: an argument is reduced to a value before being passed to a function.

> *One aspect that has been seldom examined is however the* properties *of the* general reduction *associated to the quantum lambda-calculus: this is the purpose of this paper.*

A Rewriting Theory for the Quantum λ-Calculus. Lambda calculus has a rich, powerful notion of reduction, whose properties are studied by a vast amount of literature. Such a general *rewriting theory* provides a *sound framework* for reasoning about programs transformations, such as compiler optimizations or parallel implementations, and a base on which to reason about program equivalence. The two fundamental operational properties of lambda calculus are *confluence* and *standardization*. Confluence guarantees that normal forms are unique, standardization that if a normal form exists, there is a strategy that is guaranteed to terminate in such a form.

As pioneered by Plotkin [41], standardization allows to bridge between the general reduction (where programs transformation can be studied), and a specific evaluation strategy, which implements the execution of an idealized programming language. Summarizing the situation, for programming languages, there are two kinds of term rewriting: run-time rewriting ("evaluation") and compile-time rewriting (program transformations).

In the context of quantum lambda-calculi, the only line of research discussing *rewriting* (rather than fixing a deterministic strategy) has been pursued by Dal Lago, Masini, and Zorzi [14, 13]: working with an untyped quantum lambda-calculus, they establish confluence results (and also a form of standardization, but only for the sub-language without measurement [13] – therefore, without the probabilistic behavior).

In this paper, we study not only confluence but also *standardization and normalization results* for a quantum λ-calculus featuring *measurement*, and where β reduction (the engine of λ-calculus) is fully unrestricted. Recent advances in probabilistic and monadic rewriting theory [9, 12, 19, 33, 6, 23, 27, 32] allow us to tackle this task with a formalism and powerful

techniques unavailable a decade ago. Still, quantum rewriting is *more challenging* than probabilistic rewriting because we need to manage the states of the quantum memory. The *design of the language* is, therefore, also delicate: we need to control the duplication of qubits while allowing the full power of β-reduction.

Contributions. We can summarize the contributions of the paper as follows. These are described in more details in Section 5, once all the necessary materials have been set up.

- An untyped quantum lambda-calculus, closely inspired by [14] but re-designed to allow for a more general reduction, now encompassing the full strength of β-reduction; validity constraints make it quantum-compatible.
- The calculus is equipped with a rich operational semantics, which is sound with respect to quantum computation. The *general reduction* enables arbitrary β-reduction; surface reduction (in the spirit of [43] and other calculi based on Linear Logic) plays the role of an *evaluation strategy*.
- We study the rewriting theory for the system, proving *confluence* of the reduction, and *standardization*.
- We obtain a normalization result that scales to the *asymptotic* case, defining a *normalizing strategy* w.r.t. termination at the limit.

Missing proofs and some more technical details are given in the extended version [25].

2 Setting the Scene: the Rewriting Ingredients

This section is devoted to a more detailed (but still informal) discussion of two key elements: the style of λ-calculus we adopt, and what standardization results are about. The calculus is then defined in Section 3, its operational semantics in Section 4; standardization and normalization in the following sections.

Untyped Quantum λ-Calculus. Our quantum calculus is built on top of Simpson's calculus $\Lambda^!$ [43], a variant of untyped λ-calculus inspired by Girard's Linear Logic [29]. In this choice, we follow [14, 13, 12]. Indeed, the fine control of duplication which $\Lambda^!$ inherits from linear logic makes it an ideal base for quantum computation.

The Bang operator ! plays the role of a marker for non-linear management: duplicability and discardability of resources. Abstraction is refined into linear abstraction $\lambda x.M$ and non-linear abstraction $\lambda^! x.M$. The latter allows duplication of the argument, which is required to be suspended as thunk $!N$, behaving as the !-box of linear logic.

▶ **Example 2.1** (duplication, or not). $(\lambda x.Hx)(\mathtt{new})$ is a valid term, but $(\lambda x.\langle x, x\rangle)(\mathtt{new})$ which would duplicate the qubit created by `new` is not. Instead, we can validly write $(\lambda^! x.\mathsf{CNOT}\langle Hx, x\rangle)(!\mathtt{new})$ which thunks `new` and then duplicate it, yielding $\mathsf{CNOT}\langle H\mathtt{new}, \mathtt{new}\rangle$. Notice that this term produces *an entangled pair* of qubits.

In our paper, as well as in [14, 13, 12], surface reduction (*i.e.*, no reduction is allowed in the scope of the ! operator) is the key ingredient to allow for the coexistence of quantum bits with duplication and erasing. Unlike previous work however, in our setting β-reduction – the engine of λ-calculus – is unconstrained. We prove that only quantum operations needs to be surface, making ours a conservative extension of the usual λ-calculus, with its full power.

Table 1 Summarizing Standard Factorization and Normalization Results.

Call-by-name λ-calculus	**Call-by-value λ_v-calculus**	**Linear $\lambda_!$-calculus**
General reduction: $(\to_\beta)$	General reduction: $(\to_{\beta_v})$	General reduction: $(\to_{\beta_!})$
Evaluation: head $(\to_h)$	Evaluation: weak-left $(\to_l)$	Evaluation: surface $(\to_s)$
1. Head factorization:	*1. Weak-left factorization*:	*1. Surface factorization*:
$M \to_\beta^* N$ iff $M \to_h^* \cdot \to_{\neg h}^* N$	$M \to_{\beta_v}^* N$ iff $M \to_l^* \cdot \to_{\neg l}^* N$	$M \to_{\beta_!}^* N$ iff $M \to_s^* \cdot \to_{\neg s}^* N$
2. Head normalization:	*2. Convergence to a value*:	*2. Surface normalization*:
$M \to_\beta^* H$ iff $M \to_h^* H'$	$M \to_{\beta_v}^* V$ iff $M \to_l^* V'$	$M \to_{\beta_!}^* S$ iff $M \to_s^* S'$

Call-by-Value... or rather, Call-by-Push-Value. The reader may have recognized that reduction in our calculus follows the Call-by-Push-Value paradigm, with the Bang operator *thunking* a computation. In fact, Simpson's calculus [43], more precisely the fragment without linear abstraction, is essentially an untyped version of Call-by-Push-Value, and it has been extensively studied in the literature of Linear Logic also with the name of *Bang calculus* [20, 31, 10], as a unifying framework which subsumes both Call-by-Name (CbN) and Call-by-Value(CbV) calculi.

A Taste of Standardization and Normalization: Pure λ-Calculi. Termination and confluence concern the existence and the uniqueness of normal forms, which are the results of a computation. Standardization and normalization are concerned with *how to compute* a given outcome. For example, is there a strategy which guarantees termination, if possible? The desired outcome is generally a specific kind of terms. In the classical theory of λ-calculus (à la Barendregt), the terms of interest are *head normal forms*. In the Call-by-Value approach, the terms of computational interest are values.

Classical λ-calculus. The simplest form of standardization is *factorization*: any reduction sequence can be re-organized so as to first performing specific steps and then everything else. A paradigmatic example is the head factorization theorem of classical λ-calculus (theorem 11.4.6 in [7]): every β-reduction sequence $M \to_\beta^* N$ can be re-organized/factorized so as to first reducing head redexes and then everything else – in symbols $M \to_h^* \cdot \to_{\neg h}^* N$.

A major consequence is *head normalization*, relating arbitrary β reduction with *head* reduction, w.r.t. *head normal forms*, the terms of computational interest in classical λ-calculus. A term M has head normal form if and only if head reduction terminates:

$$M \to_\beta^* H(\text{hnf}) \Leftrightarrow M \to_h^* H'(\text{hnf})$$

Plotkin's Call-by-Value. This kind of results takes its full computational meaning in Plotkin's [41] Call-by-Value λ-calculus The terms of interest are here *values*. Plotkin relates arbitary β reduction $(\to_{\beta_v})$ and the evaluation strategy $\to_l$ which only performs *weak-left* steps (no reduction in the scope of abstractions), by establishing

$$M \to_{\beta_v}^* V(\text{value}) \Leftrightarrow M \to_l^* V'(\text{value})$$

In words: the unconstrained reduction $(\to_{\beta_v})$ returns a value if and only if the evaluation strategy $(\to_l)$ returns a value. The proof relies on a factorization: $M \to_{\beta_v}^* N$ iff $M \to_l^* \cdot \to_{\neg l}^* N$.

Simpson's pure calculus. Standardization and Normalization results have been established by Simpson also for its calculus $\Lambda^!$ [43]. Here, the evaluation strategy is *surface reduction*, *i.e.* no reduction is allowed in the scope of a ! operator.

Summary. The table in Table 1 summarize the factorization and normalization result for the three calculi (respectively based on $\beta, \beta_v, \beta_!$) which we have discussed.

3 Untyped Quantum λ-Calculus

Quantum lambda-calculus is an idealization of functional quantum programming language: following Selinger and Valiron [42], it consists of a regular λ-calculus together with specific constructs for manipulating quantum data and quantum operations. One of the problems consists in accomodating the non-duplicability of quantum information: in a typed setting [42] one can rely on a linear type system. In our untyped setting, we instead base our language on Simpson's λ-calculus [43], extended with constructs corresponding to quantum data and quantum operations.

Due of entanglement, the state of an array of quantum bits cannot be separated into states of individual qubits: the information is *non-local*. A corollary is that quantum data cannot easily be written inside lambda-terms: unlike Boolean values or natural numbers, one cannot put in the grammar of terms a family of constants standing for all of the possible values a quantum bit could take. A standard procedure [42] relies on an external memory with register identifiers used as placeholders for qubits inside the lambda-term. As they stands for qubits, these registers are taken as non-duplicable.

In the original quantum lambda-calculus [42], regular free variables of type qubit were used to represent registers. In this work, being untyped we prefer to consider two kinds of variables: regular term variables, and special variables, called *registers* and denoted by r_i with $i \in \mathbb{N}$, corresponding to the qubit number i in the quantum memory. The language is also equipped with three term constructs to manipulate quantum information. The first term construct is `new`, producing the allocation of a fresh qubit[2]. The second term construct is $\texttt{meas}(r_i, M_0, M_1)$, corresponding to a destructive measurement of the qubit r_i. The evaluation then *probabilistically* continues as M_0 or M_1, depending on the measure being $|0\rangle$ or $|1\rangle$. Finally, assuming that the memory comes with a set of built-in unitary gates ranged over by letters A, B, C, the term U_A corresponds to a function applying the gate A to the indicated qubits.

Raw Terms. Formally, *raw terms* $M, N, P, \ldots$ are built according to the following grammar.

$$M, N, P ::= x \mid !M \mid \lambda x.M \mid \lambda^! x.M \mid MN \mid r_i \mid U_A \mid \texttt{new} \mid \texttt{meas}(P, M, N) \qquad (\textbf{terms } \Lambda_q)$$

where x ranges over a countable set of *variables*, r_i over a disjoint set of *registers* where $i \in \mathbb{N}$ is called the *identifier* of the register, and U_A over a set of build-in n-ary *gates*. In this paper, we limit the arity n to be 1 or 2. Pairs do not appear as a primitive construct, but we adopt the standard encoding, writing $\langle M, N\rangle$ as sugar for $\lambda f.\,(f\ M)\,N$. We also use the shorthand $\langle M_1, \ldots, M_n\rangle$ for $\langle M_1 \langle M_2, \ldots\rangle\rangle$. The variable x is bound in both $\lambda x.P$ and $\lambda^! x.P$. As usual, we silently work modulo α-equivalence. Given a term of shape $\texttt{meas}(P, M, N)$, we call M and N its *branches*. As usual, the set of free variables of a term M are denoted with $\texttt{FV}(M)$. The set of registers identifiers for a term M is denoted with $\texttt{Reg}(M)$.

▶ Remark 3.1. Without any constraints, one could write terms such as $\langle r_0, r_0\rangle$ or $\lambda x.\langle x, x\rangle$. Both are invalid: the former since a qubit cannot be duplicated, the latter since λ-abstractions are meant to be linear in Simpson's calculus.

[2] Unlike the original quantum λ-calculus [42], the term `new` literally evaluates to a qubit.

Terms Validity. To deal with the problem in Remark 3.1, we need to introduce the notions of context and surface context, to speak of occurrences and surface occurrences of subterms.

A context is a term with a hole. We define general *contexts* where the hole can appear anywhere, and *surface contexts* for contexts where holes do not occur in the scope of a ! operator, nor in the branches of a $\texttt{meas}(-,-,-)$. They are generated by the grammars

$$\begin{aligned}\mathbf{C} ::= {} & (\!|\,|\!) \mid M\mathbf{C} \mid \mathbf{C}M \mid \lambda x.\mathbf{C} \mid \lambda^! x.\mathbf{C} \mid \texttt{meas}(\mathbf{C},M,N) \mid && \textbf{(contexts)}\\ & \texttt{meas}(M,\mathbf{C},N) \mid \texttt{meas}(M,N,\mathbf{C}) \mid !\mathbf{C},\\ \mathbf{S} ::= {} & (\!|\,|\!) \mid M\mathbf{S} \mid \mathbf{S}M \mid \lambda x.\mathbf{S} \mid \lambda^! x.\mathbf{S} \mid \texttt{meas}(\mathbf{S},M,N), && \textbf{(surface contexts)}\end{aligned}$$

where $(\!|\,|\!)$ denotes the *hole* of the corresponding context. The notation $\mathbf{C}(\!|R|\!)$ (or $\mathbf{S}(\!|R|\!)$) stands for the term where the only occurrence of a hole $(\!|\,|\!)$ in $\mathbf{C}$ (or $\mathbf{S}$) is replaced with the term R, potentially capturing free variables of R.

Contexts and surface contexts allow us to formalize two notions of occurence of a subterm T. The pair $(\mathbf{C},T)$ (resp. $(\mathbf{S},T)$) is an *occurence* (resp. *surface occurence*) of T in M whever $M=\mathbf{C}(\!|T|\!)$ (resp. $M=\mathbf{S}(\!|T|\!)$). By abuse of notation, we will simply speak of occurrence of a subterm in M, the context being implicit.

We can now define a notion of valid terms, agreeing with the quantum principle of no-cloning.

▶ **Definition 3.2** (Valid Terms, and Linearity)**.** A term M is **valid** whenever

- no register occurs in M more than once, and every occurrences of registers are surface;
- for every subterm $\lambda x.P$ of M, x is *linear* in P, *i.e.* x occurs free exactly once in P and, moreover, this occurrence of x is surface.

▶ Remark 3.3. The validity conditions for registers and linear variables do not allow us to put registers inside branches. So for instance a term such as

$$\lambda z.\,\texttt{meas}(r_0, z\,(U_A\,r_1), z\,(U_B\,r_1)).$$

is invalid in our syntax. This function would measure r_0 and performs an action on r_1 based on the result. If one cannot write such a term directly with the constraints we have set on the language, one can however encode the corresponding behavior as follows:

$$(\lambda^! f.fzr_1)\,\texttt{meas}(r_0, !(\lambda ux.u(U_A x)), !(\lambda ux.u(U_B x)).$$

The action on the register r_1 is the function f whose definition is based on the result of the measurement of r_0.

Quantum Operations. Before diving into the definition of the notion of program, we briefly recall here the mathematical formalism for quantum computation [37].

The basic unit of information in quantum computation is a quantum bit or *qubit*. The state of a single qubit is a normalized vector of the 2-dimensional Hilbert space $\mathbb{C}^2$. We denote the standard basis of $\mathbb{C}^2$ as $\{|0\rangle,|1\rangle\}$, so that the general state of a single qubit can be written as $\alpha|0\rangle+\beta|1\rangle$, where $|\alpha|^2+|\beta|^2=1$.

The basic operations on quantum states are unitary operations and measurements. A *unitary operation* maps an n-qubit state to an n-qubit state, and is described by a *unitary* $2^n\times 2^n$-matrix. We assume that the language provides a set of built-in unitary operations, including for example the *Hadamard gate* H and the *controlled not gate* CNOT:

$$H := \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad \mathsf{CNOT} := \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix} \tag{2}$$

Measurement acts as a projection. When a qubit $\alpha|0\rangle + \beta|1\rangle$ is measured, the observed outcome is a classical bit: either 0 or 1, with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Moreover, the state of the qubit collapses to $|0\rangle$ (resp. $|1\rangle$) if 0 (resp. 1) was observed.

Programs. In order to associate quantum states to registers in lambda-terms, we introduce the notion of program, consisting of a *quantum memory* and a lambda-term of Λ_q. A program is closed under permutation of register identifiers.

The *state* of one qubit is a normalized vector in $\mathcal{E} = \mathbb{C}^2$. The *state* of a quantum memory (also called *qubits state* in the remainder of the document) consisting of n qubits is a normalized vector $\mathtt{Q} \in \mathcal{E}^n = (\mathbb{C}^2)^{\otimes n}$, the n-fold Kronecker product of $\mathcal{E}$. The size of $\mathtt{Q}$ is written $|\mathtt{Q}| := n$. We identify a canonical basis element of $\mathcal{E}^n$ with a string of bits of size n, denoted with $|b_0 \ldots b_{n-1}\rangle$. A state $\mathtt{Q} \in \mathcal{E}^n$ is therefore in general of the form $\mathtt{Q} = \sum_{b_0,\ldots b_{n-1} \in \{0,1\}} \alpha_{b_0 \ldots b_{n-1}} |b_0 \ldots b_{n-1}\rangle$. If σ is a permutation of $\{0, \ldots n-1\}$, we define $\sigma(\mathtt{Q})$ as $\sigma(\mathtt{Q}) = \sum_{b_0,\ldots b_{n-1} \in \{0,1\}} \alpha_{b_0 \ldots b_{n-1}} |b_{\sigma(0)} \ldots b_{\sigma(n-1)}\rangle$.

A *raw program* $\mathbf{p}$ is then a pair $[\mathtt{Q}; M]$, where $\mathtt{Q} \in \mathcal{E}^n$ and where M is a valid term such that $\mathtt{Reg}(M) = \{0, \ldots, n-1\}$. We call n the size of $\mathtt{Q}$ and we denote it with $|\mathtt{Q}|$. If σ is a permutation over the set $\{0..n-1\}$, the *re-indexing* $\sigma(\mathbf{p})$ of $\mathbf{p}$ is the pair $[\sigma(\mathtt{Q}); \sigma(M)]$ where $\sigma(M)$ is M with each occurence of r_i replaced by $r_{\sigma(i)}$.

▶ **Definition 3.4** (Program). A *program* is an equivalence class of raw programs under re-indexing. We identify programs with their representative elements. The set of all programs is denoted with $\mathcal{P}$.

▶ **Example 3.5.** The following two raw programs are equal modulo re-indexing: $[|\psi\rangle \otimes |\phi\rangle; \langle r_0, r_1\rangle] = [|\phi\rangle \otimes |\psi\rangle; \langle r_1, r_0\rangle]$. In both cases, $|\psi\rangle$ is the first qubit in the pair and $|\phi\rangle$ the second one. Re-indexing is agnostic with respect to entanglement, and we also have the raw program $[\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle; \langle r_0, r_1\rangle]$ being a re-indexation of the raw program $[\alpha|00\rangle + \gamma|01\rangle + \beta|10\rangle + \delta|11\rangle; \langle r_1, r_0\rangle]$: they are two representative elements of the same program.

4 Operational Semantics

The operational semantics of λ-calculus is usually formalized by means of a *rewriting system*. In the setting of λ-calculus, the rewriting rules are also known as *reductions*.

Rewriting System. We recall that a *rewriting system* is a pair $(\mathcal{A}, \to)$ consisting of a set $\mathcal{A}$ and a binary relation $\to$ on $\mathcal{A}$ whose pairs are written $t \to s$ and called *reduction steps*. We denote $\to^*$ (resp. $\to^=$) the transitive-reflexive (resp. reflexive) closure of $\to$. We write $t \leftarrow u$ if $u \to t$. If $\to_1, \to_2$ are binary relations on $\mathcal{A}$ then $\to_1 \cdot \to_2$ denotes their composition.

Probabilistic Rewriting. In order to define the operational semantics of the quantum lambda calculus, we need to formalize probabilistic reduction. We do so by means of a rewrite system over *multidistributions*, adopting the *monadic* approach recently developed in the literature of probabilistic rewriting (see *e.g.* [6, 12, 19, 23]). Reduction is here defined not simply on programs, but on (monadic) structures representing probability distributions over programs, called *multidistributions*.

The operational semantics of the language is defined by specifying the probabilistic behavior of programs, then lifting reduction to multidistributions of programs. Let us recall the notions.

Probability Distributions. Given a countable set Ω, a function $\mu\colon \Omega \to [0,1]$ is a probability *subdistribution* if $\|\mu\| := \sum_{\omega\in\Omega} \mu(\omega) \leq 1$ (a *distribution* if $\|\mu\| = 1$). Subdistributions allow us to deal with partial results. We write $\mathtt{D}(\Omega)$ for the set of subdistributions on Ω, equipped with the pointwise order on functions: $\mu \leq \rho$ if $\mu(\omega) \leq \rho(\omega)$ for all $\omega \in \Omega$. $\mathtt{D}(\Omega)$ has a bottom element (the subdistribution $\mathbf{0}$) and maximal elements (all distributions).

Multidistributions. We use multidistributions [6] to *syntactically* represent distributions. A *multidistribution* $\mathtt{m} = \{\!\{ q_i\mathbf{p}_i \}\!\}_{i\in I}$ on the set of programs $\mathcal{P}$ is a finite multiset of pairs of the form $q_i\mathbf{p}_i$, with $q_i \in]0,1]$, $\mathbf{p}_i \in \mathcal{P}$, and $\sum_i q_i \leq 1$. The set of all multidistributions on $\mathcal{P}$ is $\mathtt{MD}(\mathcal{P})$. The sum of two multidistributions is noted $+$, and is simply the union of the multisets. The product $q \cdot \mathtt{m}$ of a scalar q and a multidistribution $\mathtt{m}$ is defined pointwise $q \cdot \{\!\{ p_i\mathbf{p}_i \}\!\}_{i\in I} := \{\!\{ (q\cdot p_i)\mathbf{p}_i \}\!\}_{i\in I}$. We write $\{\!\{ \mathbf{p} \}\!\}$ for $\{\!\{ 1\mathbf{p} \}\!\}$.

4.1 The Rewrite System $\mathcal{Q}$

$\mathcal{Q}$ is the rewrite system $(\mathtt{MD}(\mathcal{P}), \Rightarrow)$ where the relation $\Rightarrow \subseteq \mathtt{MD}(\mathcal{P}) \times \mathtt{MD}(\mathcal{P})$ is monadically defined in two phases. First, we define *one-step reductions* from a program to a multidistribution. For example, if $\mathbf{p}$ is the program $[\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle); \mathtt{meas}(r_0, M, N)]$, the program $\mathbf{p}$ reduces in one step to $\{\!\{ \frac{1}{2}[|\rangle; M], \frac{1}{2}[|\rangle; N] \}\!\}$. Then, we *lift* the definition of reduction to a binary relation on $\mathtt{MD}(\mathcal{P})$, in the natural way. So for instance, reusing $\mathbf{p}$ above, $\{\!\{ \frac{1}{2}\mathbf{p}, \frac{1}{2}[\mathtt{Q}; (\lambda x.xx)F] \}\!\}$ reduces in one step to $\{\!\{ \frac{1}{4}[|\rangle; M], \frac{1}{4}[|\rangle; N], \frac{1}{2}[\mathtt{Q}; FF] \}\!\}$. Let us see the details.

I. Programs Reduction. We first define the reduction of a program $\mathbf{p}$ to a multidistribution. The operational behavior of $\mathbf{p}$ is given by beta reduction, denoted with $\to_\beta$, and specific rules for handling quantum operations – the corresponding reduction is denoted with $\to_q$. Formally, the relations $\to_\beta$ and $\to_q$ (also called reduction steps) are subsets of $\mathcal{P} \times \mathtt{MD}(\mathcal{P})$ and are defined in Figure 2 by *contextual closure* of the *root rules* $\mapsto_\beta$ and $\mapsto_q$, given in Figure 1. The relation $\to$ is then the union $\to_\beta \cup \to_q$.

The root rules. They are given in Figure 1. We call *redex* the term on the left-hand side of a rule. Beta rules come in two flavors, the linear one (b), which does not allow for duplication, and the non-linear one (b!), which possibly duplicate boxes (*i.e.* terms of shape $!M$). Quantum rules act on the qubits state, exactly implement the operation which we had informally described in Section 3. Notice that the rule (m) has a probabilistic behaviour. The qubit which has been measured can be discharged (as we actually do).

Contextual Closures. They are defined in Figure 2. Observe that while the β rules are closed under arbitrary contexts $\mathbf{C}$, while the quantum rules are restricted to surface contexts $\mathbf{S}$ (no reduction in the scope of a ! operator, nor in the branches of $\mathtt{meas}(-,-,-)$). This constraints guarantee that qubits are neither duplicated nor delated.

▶ Remark 4.1 (Reindexing). As in [42], reduction is defined on programs, which are equivalence classes. We define the rules on a convenient representative. For example, in Figure 1 rule (u_1) reduces the redex $U_A r_0$. Modulo reindexing, the same rules can be applied to any other register.

II. Lifting. The lifting of a relation $\to_r \subseteq \mathcal{P} \times \mathtt{MD}(\mathcal{P})$ to a relation on multidistributions is defined in Figure 3. In particular, $\to, \to_\beta, \to_q$, lift to $\Rightarrow, \Rightarrow_\beta, \Rightarrow_q$, respectively.

Reduction Sequences. A $\Rightarrow$-sequence (or *reduction sequence*) from $\mathtt{m}$ is a sequence $\mathtt{m} = \mathtt{m}_0, \mathtt{m}_1, \mathtt{m}_2, \ldots$ such that $\mathtt{m}_i \Rightarrow \mathtt{m}_{i+1}$ for every i. We write $\mathtt{m}_0 \Rightarrow^* \mathtt{m}$ to indicate the existence of a finite reduction sequence from $\mathtt{m}_0$, and $\mathtt{m}_0 \Rightarrow^k \mathtt{m}$ to specify the number k of $\Rightarrow$-steps. Given a program $\mathbf{p}$ and $\mathtt{m}_0 = \{\!\{ \mathbf{p} \}\!\}$, the sequence $\mathtt{m}_0 \Rightarrow \mathtt{m}_1 \Rightarrow \cdots$ naturally models the evaluation of $\mathbf{p}$; each $\mathtt{m}_k$ expresses the "expected" state of the system after k steps.

β rules

(b) $[\mathtt{Q}; (\lambda x.M)N] \mapsto_\beta \{\!\{ [\mathtt{Q}; M\{N/x\}] \}\!\}$
$(!b)$ $[\mathtt{Q}; (\lambda^! x.M)!N] \mapsto_\beta \{\!\{ [\mathtt{Q}; M\{N/x\}] \}\!\}$

Quantum rules

(n) $[\mathtt{Q}; \mathtt{new}] \mapsto_q \{\!\{ [\mathtt{Q} \otimes |0\rangle; r_n] \}\!\}$ where $|\mathtt{Q}| = n$
(m) $[\mathtt{Q}; \mathtt{meas}(r_n, M, N)] \mapsto_q \{\!\{ |\alpha_0|^2[\mathtt{Q}_0; M], |\alpha_1|^2[\mathtt{Q}_1; N] \}\!\}$ where $\mathtt{Q} = \alpha_0\mathtt{Q}_0 \otimes |0\rangle + \alpha_1\mathtt{Q}_1 \otimes |1\rangle$ and Q has $n+1$ qubits
(u_1) for A unary operator: $[\mathtt{Q}; U_A\, r_0] \mapsto_q \{\!\{ [\mathtt{Q}'; r_0] \}\!\}$ where $\mathtt{Q}'$ is $(A \otimes \mathsf{Id})\mathtt{Q}$
(u_2) for A binary operator: $[\mathtt{Q}; (U_A\, \langle r_0, r_1\rangle)] \mapsto_q \{\!\{ [\mathtt{Q}'; \langle r_0, r_1\rangle] \}\!\}$ where $\mathtt{Q}'$ is $(A \otimes \mathsf{Id})\mathtt{Q}$.

Figure 1 Root rules ($\mapsto$).

β steps

$$\frac{[\mathtt{Q}; M] \mapsto_\beta \{\!\{ [\mathtt{Q}; M'] \}\!\}}{[\mathtt{Q}; \mathbf{C}(\!|M|\!)] \to_\beta \{\!\{ [\mathtt{Q}; \mathbf{C}(\!|M'|\!)] \}\!\}}$$

Quantum steps

$$\frac{[\mathtt{Q}; M] \mapsto_q \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}}{[\mathtt{Q}; \mathbf{S}(\!|M|\!)] \to_q \{\!\{ p_i[\mathtt{Q}_i; \mathbf{S}(\!|M_i|\!)] \}\!\}}$$

$$\to := \to_\beta \cup \to_q$$

Figure 2 Contextual closure of root rules: ($\to$).

Validity. Validity of terms is preserved:

▶ **Proposition 4.2.** *If M is a valid term, and $[\mathtt{Q}; M] \to \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}$, then M_i is a valid term.*

Notice that the restriction of $\to_q$ to surface contexts is necessary to respect the linearity of quantum computation, avoiding duplication or deletion of qubits.

Examples. Let us see the definitions at work in a few examples. We first formalize the recursive program from Equation (1). Recall that H is the Hadamar gate, and $I := \lambda x.x$.

▶ **Example 4.3** (Flipping the quantum coin)**.** The program in Equation (1) can be written as $\mathbf{p} := [\,|\rangle; \Delta!\Delta]$, where $|\rangle$ is just the empty memory and $\Delta := \lambda^! x.\, \mathtt{meas}(H\mathtt{new}, I, x!x)$. A reduction from $\mathbf{p}$ behaves as follows. At every reduction step, we underline the redex.

$$\begin{aligned}\{\!\{ [\,|\rangle; \underline{\Delta!\Delta}] \}\!\} &\Rightarrow \{\!\{ [\,|\rangle; \mathtt{meas}(U_H\underline{\mathtt{new}}, I, \Delta!\Delta)] \}\!\} \Rightarrow \{\!\{ [\,|0\rangle; \mathtt{meas}(\underline{U_H r_0}, I, \Delta!\Delta)] \}\!\} \\ &\Rightarrow \{\!\{ [\,\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle); \underline{\mathtt{meas}(r_0, I, \Delta!\Delta)}] \}\!\} \Rightarrow \{\!\{ \tfrac{1}{2}[|\rangle; I], \tfrac{1}{2}[|\rangle; \Delta!\Delta] \}\!\} \\ &\Rightarrow \ldots \Rightarrow \{\!\{ \tfrac{1}{2}[|\rangle; I], \tfrac{1}{4}[|\rangle; I], \tfrac{1}{4}[|\rangle; \Delta!\Delta] \}\!\} \Rightarrow \ldots\end{aligned}$$

Notice that the first step is a non-linear β reduction. The reduction of `new` allocates a fresh qubit in the memory, corresponding to the register r_0. The redex $U_H r_0$ applies the Hadamar gate H to that qubit. The last reduction performs *measurement*, yielding a probabilistic outcome.

▶ **Example 4.4** (Entangled pair)**.** Let $\mathbf{p} := [|\rangle; \mathtt{let}\ \langle x, y\rangle = U_{\mathsf{CNOT}}\langle U_H\mathtt{new}, \mathtt{new}\rangle\ \mathtt{in}\ \mathtt{meas}(y, I, I)x]$ (where $\mathtt{let}\ \langle x, y\rangle \ldots$ is sugar for an opportune encoding). This program produces an entangled pair of qubits (notice how CNOT is applied to a pair of registers) and then measures one of the qubits. Let us formalize its behaviour:

$$\begin{aligned}\{\!\{ \mathbf{p} \}\!\} &\Rrightarrow_{\mathsf{s}}^* \{\!\{ [\frac{\sqrt{2}}{2}(|0\rangle + |1\rangle) \otimes |0\rangle; \mathtt{let}\ \langle x, y\rangle = \underline{U_{\mathsf{CNOT}}\langle r_0, r_1\rangle}\ \mathtt{in}\ \mathtt{meas}(y, I, I)x] \}\!\} \\ &\Rrightarrow_{\mathsf{s}} \{\!\{ [\frac{\sqrt{2}}{2}|00\rangle + \frac{\sqrt{2}}{2}|11\rangle; \mathtt{let}\ \langle x, y\rangle = \langle r_0, r_1\rangle\ \mathtt{in}\ \mathtt{meas}(y, I, I)x] \}\!\} \\ &\Rrightarrow_{\mathsf{s}}^* \{\!\{ [\frac{\sqrt{2}}{2}|00\rangle + \frac{\sqrt{2}}{2}|11\rangle; \mathtt{meas}(r_1, I, I)r_0] \}\!\} \Rrightarrow_{\mathsf{s}} \{\!\{ \tfrac{1}{2}[|0\rangle; Ir_0], \tfrac{1}{2}[|1\rangle; Ir_0] \}\!\}\end{aligned}$$

$$\frac{}{\{\!\{ \mathbf{p} \}\!\} \Rightarrow \{\!\{ \mathbf{p} \}\!\}} \quad \frac{\mathbf{p} \to \mathtt{m}}{\{\!\{ \mathbf{p} \}\!\} \Rightarrow \mathtt{m}} \quad \frac{(\{\!\{ \mathbf{p}_i \}\!\} \Rightarrow \mathtt{m}_i)_{i\in I}}{\{\!\{ p_i\mathbf{p}_i \mid i \in I \}\!\} \Rightarrow \sum_{i\in I} p_i \cdot \mathtt{m}_i}$$

Figure 3 Lifting of $\to$.

4.2 Surface Reduction and Surface Normal Forms

So far, we have defined a very liberal notion of reduction, in which β is unrestricted – it can validly be performed even inside a !-box. What shall we adopt as evaluation strategy?

In the setting of calculi based on linear logic, as Simpson's calculus [43] and the Bang calculus [20], the natural candidate is **surface reduction**: the restriction of *beta* to surface contexts ($\underset{\mathsf{s}}{\to}_\beta$) plays a role akin to that of head reduction in classical λ-calculus, yielding to similar factorization and normalization results which relate $\to_\beta$ and $\underset{\mathsf{s}}{\to}_\beta$ (as recalled in Table 1). The *terms of interest* are here **surface normal forms** (snf), such as x or $!M$. They are the analog of values in Plotkin's Call-by-Value λ-calculus and of head normal forms in classical λ-calculus – such an analogy can indeed be made precise [20, 31, 10][3].

In our setting, *surface reduction* and *surface normal forms*(snf) also play a privileged role.

Surface Reduction. Surface steps $\underset{\mathsf{s}}{\to} \subseteq \mathcal{P} \times \mathtt{MD}(\mathcal{P})$ (Figure 5) are the union $\to_q \cup \underset{\mathsf{s}}{\to}_\beta$ of quantum steps together with $\underset{\mathsf{s}}{\to}_\beta$, *i.e.* the closure *under surface contexts* $\mathbf{S}$ of the β rules. A program $\mathbf{p}$ is a **surface normal form** (snf) if $\mathbf{p} \not\underset{\mathsf{s}}{\to}$, *i.e.* no surface reduction is possible from it.

A $\to$-step which is not surface is noted $\underset{\neg\mathsf{s}}{\to}$. The lifting of $\underset{\mathsf{s}}{\to}, \underset{\neg\mathsf{s}}{\to}$ to relations on multidistributions is denoted $\underset{\mathsf{s}}{\Rightarrow}, \underset{\neg\mathsf{s}}{\Rightarrow}$ respectively.

▶ Remark 4.5. Notice that $\underset{\neg\mathsf{s}}{\to}$ steps do not act on the qubits state, since they are β steps.

Strict Lifting. To guarantee normalization results (Section 7), we will need a stricter form of lifting, noted $\underset{\mathsf{s}}{\rightrightarrows}$ (Figure 4), forcing a reduction step to be performed in each program of the multidistribution $\mathtt{r}$, if a redex exists. Clearly $\underset{\mathsf{s}}{\rightrightarrows} \subseteq \underset{\mathsf{s}}{\Rightarrow}$.

$$\frac{\mathbf{p} \not\underset{\mathsf{s}}{\to}}{\{\!\{ \mathbf{p} \}\!\} \underset{\mathsf{s}}{\rightrightarrows} \{\!\{ \mathbf{p} \}\!\}} \quad \frac{\mathbf{p} \underset{\mathsf{s}}{\to} \mathtt{m}}{\{\!\{ \mathbf{p} \}\!\} \underset{\mathsf{s}}{\rightrightarrows} \mathtt{m}} \quad \frac{(\{\!\{ \mathbf{p}_i \}\!\} \underset{\mathsf{s}}{\rightrightarrows} \mathtt{m}_i)_{i\in I}}{\{\!\{ p_i\mathbf{p}_i \mid i \in I \}\!\} \underset{\mathsf{s}}{\rightrightarrows} \sum_{i\in I} p_i \cdot \mathtt{m}_i}$$

Figure 4 Strict lifting of $\underset{\mathsf{s}}{\to}$.

▶ **Example 4.6.** We will prove that the strict lifting $\underset{\mathsf{s}}{\rightrightarrows}$ guarantees to reach snf, if any exist. This is obviously not the case for $\underset{\mathsf{s}}{\Rightarrow}$-sequences:

$$\{\!\{ \tfrac{1}{2}I\mathtt{new}, \tfrac{1}{2}\underline{(\lambda^! x.x!x)!(\lambda^! x.x!x)} \}\!\} \underset{\mathsf{s}}{\Rightarrow} \{\!\{ \tfrac{1}{2}I\mathtt{new}, \tfrac{1}{2}\underline{(\lambda^! x.x!x)!(\lambda^! x.x!x)} \}\!\} \underset{\mathsf{s}}{\Rightarrow} \{\!\{ \tfrac{1}{2}I\mathtt{new}, \tfrac{1}{2}\underline{(\lambda^! x.x!x)!(\lambda^! x.x!x)} \}\!\} \underset{\mathsf{s}}{\Rightarrow} \dots$$

[3] A consequence of Girard's translation of Call-by-Name and Call-by-Value λ-calculi into Linear Logic.

On the Interest of Surface Normal Forms. What is the result of running a quantum program? In general, since computation is probabilistic, the result of executing a program will be a distribution over some outcomes of interest. A natural choice are programs of shape $\mathbf{p} \coloneqq [\mathtt{Q}; S]$, with S in surface normal form, ensuring that at this point, the qubits state $\mathtt{Q}$ is a *stable piece of information* (it will not further evolve in the computation). Indeed:

$$\text{a program } \mathbf{p} \not\xrightarrow[\mathsf{s}]{} \text{ (i.e. in } \mathsf{snf}\text{) will no longer modify the qubits state.}$$

▶ Remark 4.7. Notice instead that a program $\mathbf{p} \not\to_q$ (no quantum step is possible) is not necessarily done in manipulating the quantum memory. Further β reductions may unblock further quantum steps. Think of $(\lambda^! x.\mathsf{CNOT}\langle Hx, x\rangle)(!\mathtt{new})$ from Example 2.1.

4.3 Sum-up Tables

Let us conclude the section summarizing the reduction relations at play.

Relations.

$\mathcal{P} \times \mathtt{MD}(\mathcal{P})$	Definition	Lifted to $\mathtt{MD}(\mathcal{P}) \times \mathtt{MD}(\mathcal{P})$	Strict lifting
$\to_\beta$	contextual closure of β-rules	$\Rightarrow_\beta$	
$\xrightarrow[\mathsf{s}]{}_\beta$	closure by **surface context** of β-rules	$\underset{\mathsf{s}}{\Rightarrow}_\beta$	$\underset{\mathsf{s}}{\rightrightarrows}_\beta$
$\to_q$	closure by **surface context** of q-rules	$\Rightarrow_q$	$\rightrightarrows_q$
$\to$	$\to_\beta \cup \to_q$	$\Rightarrow$	
$\xrightarrow[\mathsf{s}]{}$	$\xrightarrow[\mathsf{s}]{}_\beta \cup \to_q$	$\underset{\mathsf{s}}{\Rightarrow}$	$\underset{\mathsf{s}}{\rightrightarrows}$
$\xrightarrow[\neg\mathsf{s}]{}$	$\to - \xrightarrow[\mathsf{s}]{}$	$\underset{\neg\mathsf{s}}{\Rightarrow}$	

Reduction Sequences.

Finite reduction sequence	
$\mathtt{m} \Rightarrow^* \mathtt{n}$	there is a $\Rightarrow$-sequence from $\mathtt{m}$ to $\mathtt{n}$
$\mathtt{m} \underset{\mathsf{s}}{\Rightarrow}^* \mathtt{n}$	there is a $\underset{\mathsf{s}}{\Rightarrow}$-sequence from $\mathtt{m}$ to $\mathtt{n}$
$\mathtt{m} \underset{\mathsf{s}}{\rightrightarrows}^* \mathtt{n}$	there is a $\underset{\mathsf{s}}{\rightrightarrows}$-sequence from $\mathtt{m}$ to $\mathtt{n}$

5 Rewriting Theory for $\mathcal{Q}$: Overview of the Results

We are now going to study reduction on multidistributions of programs, namely the *general reduction* $\Rightarrow$ (corresponding to the lifting of $\to$) and *surface reductions* (corresponding to the lifting of $\xrightarrow[\mathsf{s}]{}$), and the relation between the two. Let us discuss each point.

1. The reduction $\Rightarrow$ allows for *unrestricted* β reduction. For example, we can rewrite in the scope of a Bang operator ! (perhaps to optimize the thunked code before copying it several times). We prove that $\Rightarrow$ is confluent, providing a general framework for rewriting theory. This (very liberal) reduction has a foundational role, in which to study the equational theory of the calculus and to analyze programs transformations.
2. Surface reduction $\underset{\mathsf{s}}{\Rightarrow} \subseteq \Rightarrow$ plays the role of an evaluation strategy, in which however the scheduling (how redexes should be fired) is not fully specified[4]. For example $\mathbf{p} = [|\rangle; \langle \mathtt{new}, H\mathtt{new}\rangle]$ has two surface redexes, enabling two different steps. We will prove (by

[4] This is not only convenient, as it allows for parallel implementation, but it is necessary for standardization [26]

proving a diamond property) that surface reduction ($\rightrightarrows_s$) is "essentially deterministic" in the sense that while the choice of the redex to fire is non-deterministic, the order in which such choices are performed are irrelevant to the final result.

3. The two reductions are related by a standardization result (Theorem 6.7) stating that if $\mathtt{m} \Rightarrow^* \mathtt{n}$ then $\mathtt{m} \Rightarrow_s^* \cdot \rightrightarrows_{\neg s}^* \mathtt{n}$. Standardization is the base of normalization results, concerning properties such as "program $\mathbf{p}$ terminates with probability p."
4. We prove that $\rightrightarrows_s$ is a *normalization strategy* for $\Rightarrow$, namely if $\mathbf{p}$ *may* converge to surface normal form with probability p using the general reduction $\Rightarrow$, then $\rightrightarrows_s$ reduction *must* converge to surface normal form with probability p. Informally, we can write that $\mathtt{m} \Downarrow p$ implies $\mathtt{m} \Downarrow_s p$ (corresponding to the last line in Table 1). To formalize and prove such a claim we will need more tools, because probabilistic termination is *asymptotic*, *i.e.* it appears as a limit of a possibly infinite reduction. We treat this in Section 7, where we rely on techniques from [2, 23, 24].

6 Confluence and Finitary Standardization

We first recall standard notions which we are going to use.

Confluence, Commutation, and all That (a quick recap). The relation $\rightarrow$ is *confluent* if $\leftarrow^* \cdot \rightarrow^* \subseteq \rightarrow^* \cdot \leftarrow^*$. A stricter form is the diamond $\leftarrow \cdot \rightarrow$ implies $\rightarrow \cdot \leftarrow$, which is well known to imply confluence. Two relations $\rightarrow_\circ$ and $\rightarrow_\bullet$ on A *commute* if : $\leftarrow_\circ^* \cdot \rightarrow_\bullet^*$ implies $\rightarrow_\bullet^* \cdot \leftarrow_\circ^*$. Confluence and factorization are both commutation properties: a relation is confluent if it commutes with itself.

An element $u \in \mathcal{A}$ is a $\rightarrow$-*normal form* if there is no t such that $u \rightarrow t$ (written $u \not\rightarrow$).

On normalization. In general, a term may or may not reduce to a normal form. And if it does, not all reduction sequences necessarily lead to normal form. How do we compute a normal form? This is the problem tackled by *normalization*: by repeatedly performing *only specific steps*, a normal form will be computed, provided that t can reduce to any. Intuitively, a *normalizing strategy* for $\rightarrow$ is a reduction strategy which, given a term t, is guaranteed to reach normal form, if any exists.

6.1 Surface Reduction has the Diamond Property

In this section, we first prove that surface reduction ($\Rightarrow_s$ and $\rightrightarrows_s$) has the *diamond property*:

$$\mathtt{r} \Leftarrow_s \mathtt{m} \Rightarrow_s \mathtt{s} \text{ implies } \mathtt{r} \Rightarrow_s \mathtt{n} \Leftarrow_s \mathtt{s} \text{ (for some } \mathtt{n}) \qquad \text{(Diamond)}$$

then we show that $\Rightarrow$ is confluent.

Here we adapt techniques used in probabilistic rewriting [6, 22, 26]. Proving the diamond property is however significantly *harder* than in the case of probabilistic λ-calculi, because we need to take into account also the *qubits state*, and the corresponding registers. If a program $\mathbf{p} = [\mathtt{Q}; M]$ has two different reductions, we need to join in one step not only the terms, but also their qubits states, working up to re-indexing of the registers (recall that programs are equivalence classes modulo re-indexing, see also Example 3.5). The following is an example, just using the simple construct `new`. Measurement makes the situation even more delicate.

▶ **Example 6.1.** Let $\mathbf{p} = [|\rangle; \langle \mathtt{new}, (H\mathtt{new})\rangle]$. The following are two different reduction sequences form $\mathbf{p}$. The two normal forms are the same program (Definition 3.4). Here, $|+\rangle := \frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$.

$$[|\rangle;\langle \underline{\texttt{new}},(H\texttt{new})\rangle] \to_{\mathsf{s}} \{\!\{ [|0\rangle;\langle r_0,(H\underline{\texttt{new}})\rangle] \}\!\} \Rrightarrow_{\mathsf{s}} \{\!\{ [|0\rangle;\langle r_0,(\underline{Hr_1})\rangle] \}\!\} \Rrightarrow_{\mathsf{s}} \{\!\{ [|0\rangle\otimes|+\rangle;\langle r_0,r_1\rangle] \}\!\}$$

$$[|\rangle;\langle \texttt{new},(\underline{H\texttt{new}})\rangle] \to_{\mathsf{s}} \{\!\{ [|0\rangle;\langle \texttt{new},(\underline{Hr_0})\rangle] \}\!\} \Rrightarrow_{\mathsf{s}} \{\!\{ [|+\rangle;\langle \underline{\texttt{new}},(r_0)\rangle] \}\!\} \Rrightarrow_{\mathsf{s}} \{\!\{ [|+\rangle\otimes|0\rangle;\langle r_1,r_0\rangle] \}\!\}$$

The key result is the following version of diamond (commutation). The proof – quite technical – is given in the extended version [25]. Recall that $\rightrightarrows_{\mathsf{s}} \subseteq \Rrightarrow_{\mathsf{s}}$.

▶ **Lemma 6.2** (Pointed Diamond). *Assume* $\mathbf{p} = [\mathtt{Q}; M]$ *and that* M *has two distinct redexes, such that* $\mathbf{p} \to_{\mathsf{s}} b\ \mathtt{m}_1$ *and* $\mathbf{p} \to_{\mathsf{s}} c\ \mathtt{m}_2$*. Then there exists* $\mathtt{n}$ *such that* $\mathtt{m}_1 \rightrightarrows_{\mathsf{s}} c\ \mathtt{n}$ *and* $\mathtt{m}_2 \rightrightarrows_{\mathsf{s}} b\ \mathtt{n}$*. Moreover, no term* M_i *in* $\mathtt{m}_1 = \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}_{i\in I}$ *and no term* M_j *in* $\mathtt{m}_2 = \{\!\{ p_j[\mathtt{Q}_j; M_j] \}\!\}_{j\in J}$ *is in* snf*.*

From the above result we obtain the diamond property.

▶ **Proposition 6.3** (Diamond). *Surface reductions* $\Rrightarrow_{\mathsf{s}}$ *and* $\rightrightarrows_{\mathsf{s}}$ *have the diamond property.*

In its stricter form, the diamond property guarantees that the non determinism in the choice of the redex is *irrelevant* – hence the reduction $\rightrightarrows_{\mathsf{s}}$ is essentially deterministic. The technical name for this property is Newman's *random descent* [36]: no matter the choice of the redex, all reduction sequences behave the same way, *i.e.* have the same length, and if terminating, they do so in the same normal form. Formalized by Theorem 7.3, we use this fact to establish that $\rightrightarrows_{\mathsf{s}}$ is a normalizing strategy for $\Rightarrow$.

6.2 Confluence of ⇒

We modularize the proof of confluence by using a classical technique, Hindley-Rosen lemma, stating that if $\Rightarrow_1$ and $\Rightarrow_2$ are binary relations on the same set $\mathcal{R}$, then their union $\Rightarrow_1 \cup \Rightarrow_2$ is confluent if both $\Rightarrow_1$ and $\Rightarrow_2$ are confluent, and $\Rightarrow_1$ and $\Rightarrow_2$ commute.

▶ **Theorem 6.4.** *The reduction* $\Rightarrow$ *satisfies confluence.*

Proof. The proof that $\Rightarrow_\beta \cup \Rightarrow_q$ is confluent, is easily obtained from Lemma 6.2, by using Hindley-Rosen Lemma. We already have most of the elements: $\Rightarrow_\beta$ is confluent: because $\to_\beta$ is; $\Rightarrow_q$ is confluent: because it is diamond (Proposition 6.3); $\Rightarrow_q$ and $\Rightarrow_\beta$ commute: by Lemma 6.2, we already know that $\Rightarrow_q$ and $\Rrightarrow_{\mathsf{s}}{}_\beta$ commute, hence we only need to verify that $\Rightarrow_q$ and $\rightrightarrows_{\neg\mathsf{s}}{}_\beta$ commute, which is easily done. ◀

6.3 Surface Standardization

We show that any sequence $\Rightarrow^*$ can be factorized as $\Rrightarrow_{\mathsf{s}}^* \cdot \rightrightarrows_{\neg\mathsf{s}}^*$ (Theorem 6.7). Standardization is proved via the modular technique proposed in [1], which in our notation can be stated as follows:

▶ **Lemma 6.5** (Modular Factorization [1]). $\Rightarrow^* \subseteq \Rrightarrow_{\mathsf{s}}^* \cdot \rightrightarrows_{\neg\mathsf{s}}^*$ *if the following conditions hold:*

1. $\Rightarrow_\beta^* \subseteq \Rrightarrow_{\mathsf{s}}{}_\beta^* \cdot \rightrightarrows_{\neg\mathsf{s}}{}_\beta^*$*, and*
2. $\rightrightarrows_{\neg\mathsf{s}}{}_\beta \cdot \Rrightarrow_{\mathsf{s}}{}_q \subseteq \Rrightarrow_{\mathsf{s}}{}_q \cdot \Rightarrow_\beta$*.*

Condition 1. in Lemma 6.5 is immediate consequence of Simpson's surface standardization for the $\Lambda^!$ calculus [43] stating that $\to_\beta^* \subseteq \to_{\mathsf{s}}{}_\beta^* \cdot \to_{\neg\mathsf{s}}{}_\beta^*$. Condition 2. in Lemma 6.5 is obtained from the following *pointed* version:

▶ **Lemma 6.6.** $[\![\mathtt{Q}; M]\!] \rightrightarrows_{\neg \mathsf{s}}^{\beta} \{\!\{ [\![\mathtt{Q}; P]\!] \}\!\}$ *and* $[\![\mathtt{Q}; P]\!] \to_q \mathtt{n}$ *implies* $[\![\mathtt{Q}; M]\!] \to_q \cdot \Rightarrow_\beta \mathtt{n}$.

Proof. By induction on the context $\mathbf{S}$ such that $P = \mathbf{S}(\!|R|\!)$ and $[\![\mathtt{Q}; \mathbf{S}(\!|R|\!)]\!] \to_q \{\!\{ p_i[\![\mathtt{Q}_i; \mathbf{S}(\!|R_i|\!)]\!] \}\!\} = \mathtt{n}$. We exploit in an essential way the fact that M and P have the same shape. ◀

By Lemmas 6.5 and 6.6, we obtain the main result of this section:

▶ **Theorem 6.7** (Surface Standardization). $\mathtt{m} \Rightarrow^* \mathtt{n}$ *implies* $\mathtt{m} \underset{\mathsf{s}}{\Rightarrow}^* \cdot \underset{\neg \mathsf{s}}{\Rightarrow}^* \mathtt{n}$

▶ Remark 6.8 (Strict vs non-strict). Please observe that standardization is stated in terms of the *non-strict* lifting ($\underset{\mathsf{s}}{\Rightarrow}$) of $\underset{\mathsf{s}}{\to}$, as $\underset{\mathsf{s}}{\rightrightarrows}$ could reduce more than what is desired. Dually, normalization holds in terms of the *strict* lifting $\underset{\mathsf{s}}{\rightrightarrows}$, for the reasons already discussed in Example 4.6.

A Reading of Surface Standardization. A program $\mathbf{p}$ in snf will no longer modify the qubits state. Intuitively, $\mathbf{p}$ has already produced the maximal amount of quantum data that it could possibly do. We can read Surface Standardization as follows. Assume $\{\!\{ \mathbf{p} \}\!\} \Rightarrow^* \mathtt{n}$ where all terms in $\mathtt{n}$ are in snf (we use metavariables S_i, S_i' to indicate terms in snf). Standardization guarantees that surface steps suffice to reach a multidistribution $\mathtt{n}'$ whose programs have the exact *same information content* as $\mathtt{n}$:

$$\{\!\{ \mathbf{p} \}\!\} \Rightarrow^* \mathtt{n} = \{\!\{ p_i[\![\mathtt{Q}_i; S_i]\!] \}\!\}_{i \in I} \quad \text{implies} \quad \{\!\{ \mathbf{p} \}\!\} \underset{\mathsf{s}}{\Rightarrow}^* \mathtt{n}' = \{\!\{ p_i[\![\mathtt{Q}_i; S_i']\!] \}\!\}_{i \in I}.$$

This because Theorem 6.7 implies $\{\!\{ \mathbf{p} \}\!\} \underset{\mathsf{s}}{\Rightarrow}^* \mathtt{n}' \underset{\neg \mathsf{s}}{\Rightarrow}^* \mathtt{n}$, and from $\mathtt{n}' \underset{\neg \mathsf{s}}{\Rightarrow}^* \mathtt{n}$ we deduce that each element $p_i[\![\mathtt{Q}_i; S_i]\!]$ in $\mathtt{n}$ must come form an element $p_i[\![\mathtt{Q}_i; S_i']\!]$ in $\mathtt{n}'$ where S_i' is in snf and where the qubits state $\mathtt{Q}_i$ (and the associated probability p_i) are *exactly the same*.

7 Probabilistic Termination and Asymptotic Normalization

What does it mean for a program to reach surface normal form (snf)? Since measurement makes the reduction probabilistic, we need to give a quantitative answer.

Probabilistic Termination. The probability that the system described by the multidistribution $\mathtt{m} = \{\!\{ p_i[\![\mathtt{Q}_i; M_i]\!] \mid i \in I \}\!\}$ is in surface normal form is expressed by a scalar $p = \mathbb{P}(\mathtt{m}) \in [0, 1]$ which is defined as follows:

$$\mathbb{P}(\mathtt{m}) = \sum_{i \in I} q_i \qquad q_i = \begin{cases} p_i & \text{if } M_i \text{ } \mathsf{snf} \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{p} = [\![\mathtt{Q}; M]\!]$ and $\mathtt{m}_0 = \{\!\{ \mathbf{p} \}\!\}$. Let $\mathtt{m}_0 \Rightarrow \mathtt{m}_1 \Rightarrow \mathtt{m}_2 \Rightarrow \cdots$ a reduction sequence. $\mathbb{P}(\mathtt{m}_k)$ expresses the probability that after k steps $\mathbf{p}$ is in snf. The *probability that* $\mathbf{p}$ *reaches* snf along the (possibly infinite) reduction sequence $\langle \mathtt{m}_n \rangle_{n \in \mathbb{N}}$ is easily defined as a limit: $\sup_n \{\mathbb{P}(\mathtt{m}_n)\}$. We also say that the sequence $\langle \mathtt{m}_n \rangle_{n \in \mathbb{N}}$ *converges* with probability $\sup_n \{\mathbb{P}(\mathtt{m}_n)\}$.

▶ **Example 7.1** (Recursive coin, cont.). Consider again Example 4.3. After 4 steps, the program terminates with probability $\frac{1}{2}$. After 4 more steps, it terminates with probability $\frac{1}{2} + \frac{1}{4}$, and so on. At the limit, the reduction sequence *converges* with probability $\sum_{k:1}^{\infty} \frac{1}{2^k} = 1$.

Table 2 Limit of (possibly infinite) reduction sequences.

Convergence (Def.n7.2)	
$\mathtt{m} \Downarrow p$	there is a $\Rightarrow$-sequence from $\mathtt{m}$ which converges with probability p
$\mathtt{m} \Downarrow_s p$	there is a $\underset{s}{\Rightarrow}$-sequence from $\mathtt{m}$ which converges with probability p
$\mathtt{m} \downdownarrows_s p$	there is a $\underset{s}{\rightrightarrows}$-sequence from $\mathtt{m}$ which converges with probability p

7.1 Accounting for Several Possible Reduction Sequences

Since $\Rightarrow$ is not a deterministic reduction, given a multidistribution $\mathtt{m}$, there are *several possible reduction sequences* from $\mathtt{m}$, and therefore several outcomes (limits) are possible. Following [23], we adopt the following terminology:

▶ **Definition 7.2** (Limits). *Given* $\mathtt{m}$*, we write*

- $\mathtt{m} \Downarrow p$*, if there exists a* $\Rightarrow$*-sequence* $\langle \mathtt{m}_n \rangle_{n\in\mathbb{N}}$ *from* $\mathtt{m}$ *whose limit is* p*.*
- $\mathtt{Lim}(\mathtt{m}, \Rightarrow) \coloneqq \{p \mid \mathtt{m} \Downarrow p\}$ *is the set of limits of* $\mathtt{m}$*.*
- $[\![\mathtt{m}]\!]$ *denotes the greatest element of* $\mathtt{Lim}(\mathtt{m}, \Rightarrow)$*, if any exists.*

Intuitively, $[\![\mathbf{p}]\!]$ is the *best result* that any $\Rightarrow$-sequence from $\mathbf{p}$ can *effectively* produce. If the set $\mathtt{Lim}(\mathbf{p}, \Rightarrow)$ has a sup α but not a greatest element (think of the open interval $[0, 1)$), it means that in fact, no reduction can produce α as a limit. Notice also that, when reduction is deterministic, from any $\mathbf{p}$ there is only one maximal reduction sequence, and so it is always the case that $[\![\mathbf{p}]\!] = \sup_n\{\mathbb{P}(\mathbf{p}_n)\}$. Below we exploit the interplay between different rewriting relations, and their limit; it is useful to summarize our notations in Table 2.

7.2 Asymptotic Normalization

Given a quantum program $\mathbf{p}$, does $[\![\mathbf{p}]\!]$ exists? If this is the case, can we define a *normalizing strategy* which is guarantee to converge to $[\![\mathbf{p}]\!]$? The answer is positive. The main result of this section is that such a normalizing strategy does exist, and it is $\underset{s}{\rightrightarrows}$. More precisely, we show that *any* $\underset{s}{\rightrightarrows}$-reduction sequence from $\mathbf{p}$ converges to the same limit, which is exactly $[\![\mathbf{p}]\!]$. We establish the following results, for any arbitrary $\mathtt{m} \in \mathtt{MD}(\mathcal{P})$. Theorem 7.3 is a direct – and the most important – consequence of the diamond property of $\underset{s}{\rightrightarrows}$. The proof uses both point 1. and point 2. of Lemma 6.2. For Theorem 7.4, the proof relies on an abstract technique from [24].

▶ **Theorem 7.3** (Random Descent). *All* $\underset{s}{\rightrightarrows}$*-sequences from* $\mathtt{m}$ *converge to the same limit.*

▶ **Theorem 7.4** (Asymptotic completeness). $\mathtt{m} \Downarrow p$ *implies* $\mathtt{m} \downdownarrows_s q$*, with* $p \leq q$*.*

Theorem 7.4 states that, for each $\mathtt{m}$, if $\Rightarrow$ reduction *may* converge to snf with probability p, then $\underset{s}{\rightrightarrows}$ reduction *must* converge to snf with probability (at least) p. Theorem 7.3 states that, for each $\mathtt{m}$, the limit q of strict surface reductions ($\underset{s}{\rightrightarrows}$) from $\mathtt{m}$ is *unique*.

Summing-up, the limit q of $\underset{s}{\rightrightarrows}$ reduction is the best convergence result that any sequence from $\mathtt{m}$ can produce. Since $\underset{s}{\rightrightarrows} \subseteq \Rightarrow$, then q is also the greatest element in $\mathtt{Lim}(\mathtt{m}, \Rightarrow)$, *i.e.* $[\![\mathtt{m}]\!] = q$. We hence have proved the following, where item (2.) is the *asymptotic analogue* of the normalization results in Table 1.

▶ **Theorem 7.5** (Asymptotic normalization). *For each* $\mathbf{p} \in \mathcal{P}$*, (1.) the limit* $\mathtt{Lim}(\mathbf{p}, \Rightarrow)$ *has a greatest element* $[\![\mathbf{p}]\!]$*, and (2.)* $\mathbf{p} \downdownarrows_s [\![\mathbf{p}]\!]$*.*

8 Related Work and Discussion

In this paper, we propose a foundational notion of (untyped) quantum λ-calculus with a general reduction, encompassing the full strength of β-reduction while staying compatible with quantum constraints. We then introduce an evaluation strategy, and derive standardization and confluence results. We finally discuss normalization of programs at the limit.

Related Works. For quantum λ-calculi *without measurement*, hence without probabilistic behavior, *confluence* [13, 5] (and even a special form of standardization [13]) have been studied since early work. When dealing with measurement, the analysis is far more challenging. To our knowledge, only confluence has been studied, in pioneering work by Dal Lago, Masini and Zorzi [14]. Remarkably, in order to deal with probabilistic and asymptotic behavior, well before the advances in probabilistic rewriting of which we profit, the authors introduce a very elaborated technique. Notice that in [14] reduction is non-deterministic, but restricted to *surface reduction*. In our paper, their result roughly corresponds to the diamond property of $\overset{*}{\underset{s}{\Rightarrow}}$, together with Theorem 7.3.

No "standard" *standardization* results (like the classical ones we recall in Table 1) exist in the literature for the quantum setting. Notice that the form of standardization in [13] is a reordering of the (surface) *measurement-free* reduction steps, so to perform first beta steps, then quantum steps, in agreement with the idea that a quantum computer consists of a classical device "setting up" a quantum circuit, which is then fed with an input. A similar refinement is also possible for the corresponding fragment of our calculus (namely measurement-free $\underset{s}{\rightarrow}$), but clearly does not scale: think of $(\lambda x.x)\,\mathtt{meas}(U_H\,\mathtt{new}, M, N)$, where the argument of a function is guarded by a measurement.

Our term language is close to [14]. How such a calculus relate with a Call-by-Value λ-calculus such as [42]? A first level of answer is that our setting is an *untyped* λ-calculus; linear abstraction, together with well forming rules, allows for the management of quantum data. In [42], the same role is fulfilled by the (Linear Logic based) *typing system*.

Despite these differences, we do expect that our results can be transferred. As already mentioned, the redex $(\lambda^! x.M)!N$ reflects a Call-by-Push-Value mechanism, which in *untyped form* has been extensively studied in the literature with the name of *Bang calculus* [20, 31, 10], as a uniform framework to encode both Call-by-Name (CbN) and Call-by-Value (CbV). Semantical but also syntactical properties, including *confluence* [20, 31] and *standardization* [24, 3] are analyzed in the Bang setting, and then transferred via *reverse simulation* to both CbV and CbN. More precisely, a CbV (resp. CbN) translation maps forth-and-back weak (resp. head) reduction into surface reduction. Surface normal forms are the CbV image of values (and the CbN image of head normal forms). Since the *Bang calculus* is exactly the fragment of Simpson's calculus [43] without linear abstraction, one may reasonably expect that our calculus can play a similar role in the quantum setting. It seems however that a back-and forth translation of CbV (or CbN) will need to encompass types.

A last line of works worth mentioning is the series of works based on Lineal [5, 4, 18]. However, these works differ from our approach in the sense that the λ-terms themselves are subject to superposition: the distinction between classical and quantum data in an untyped setting is unclear.

References

1 Beniamino Accattoli, Claudia Faggian, and Giulio Guerrieri. Factorize factorization. In *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 6:1–6:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CSL.2021.6.

2 Zena M. Ariola and Stefan Blom. Skew confluence and the lambda calculus with letrec. *Annals of Pure and Applied Logic*, 117(1):95–168, 2002. doi:10.1016/S0168-0072(01)00104-X.

3 Victor Arrial, Giulio Guerrieri, and Delia Kesner. The benefits of diligence. *International Joint Conference on Automated Reasoning, IJCAR 2024,*, 2024.

4 Pablo Arrighi, Alejandro Díaz-Caro, and Benoît Valiron. The vectorial lambda-calculus. *Information and Computation*, 254:105–139, 2017. doi:10.1016/j.ic.2017.04.001.

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

6 Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. *Sci. Comput. Program.*, 185, 2020. doi:10.1016/J.SCICO.2019.102338.

7 Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1984.

8 Benjamin Bichsel, Maximilian Baader, Timon Gehr, and Martin T. Vechev. Silq: a high-level quantum language with safe uncomputation and intuitive semantics. In Alastair F. Donaldson and Emina Torlak, editors, *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI'20*, pages 286–300. ACM, 2020. doi:10.1145/3385412.3386007.

9 Olivier Bournez and Florent Garnier. Proving positive almost sure termination under strategies. In *Rewriting Techniques and Applications, RTA*, pages 357–371, 2006. doi:10.1007/11805618_27.

10 Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. The bang calculus revisited. *Inf. Comput.*, 293:105047, 2023. doi:10.1016/J.IC.2023.105047.

11 Christophe Chareton, Sébastien Bardin, Franccois Bobot, Valentin Perrelle, and Benoît Valiron. An automated deductive verification framework for circuit-building quantum programs. In Nobuko Yoshida, editor, *Proceedings of the 30th European Symposium on Programming Languages and Systems, ESOP 2021*, volume 12648 of *Lecture Notes in Computer Science*, pages 148–177. Springer, 2021. doi:10.1007/978-3-030-72019-3_6.

12 Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. The geometry of parallelism: classical, probabilistic, and quantum effects. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 833–845. ACM, 2017. doi:10.1145/3009837.

13 Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. On a measurement-free quantum lambda calculus with classical control. *Math. Struct. Comput. Sci.*, 19(2):297–335, 2009. doi:10.1017/S096012950800741X.

14 Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Confluence results for a quantum lambda calculus with measurements. *Electr. Notes Theor. Comput. Sci.*, 270(2):251–261, 2011. doi:10.1016/j.entcs.2011.01.035.

15 Ugo Dal Lago and Margherita Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO Theor. Informatics Appl.*, 46(3):413–450, 2012. doi:10.1051/ita/2012012.

16 Ugo de'Liguoro and Adolfo Piperno. Non deterministic extensions of untyped lambda-calculus. *Inf. Comput.*, 122(2):149–177, 1995. doi:10.1006/INCO.1995.1145.

17 Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Probabilistic lambda-calculus and quantitative program analysis. *J. Log. Comput.*, 15(2):159–179, 2005. doi:10.1093/LOGCOM/EXI008.

18 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'19*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785834.

19 Alejandro Díaz-Caro and Guido Martinez. Confluence in probabilistic rewriting. *Electr. Notes Theor. Comput. Sci.*, 338:115–131, 2018.

20 Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, pages 174–187. ACM, 2016. doi:10.1145/2967973.2968608.

21 Thomas Ehrhard, Michele Pagani, and Christine Tasson. The computational meaning of probabilistic coherence spaces. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 87–96. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.29.

22 Claudia Faggian. Probabilistic rewriting: Normalization, termination, and unique normal forms. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 19:1–19:25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.FSCD.2019.19.

23 Claudia Faggian. Probabilistic rewriting and asymptotic behaviour: on termination and unique normal forms. *Log. Methods Comput. Sci.*, 18(2), 2022. doi:10.46298/LMCS-18(2:5)2022.

24 Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021. doi:10.1007/978-3-030-71995-1_11.

25 Claudia Faggian, Gaetan Lopez, and Benoît Valiron. A rewriting theory for quantum λ-calculus. *CoRR*, abs/2411.14856, 2024. arXiv:2411.14856.

26 Claudia Faggian and Simona Ronchi Della Rocca. Lambda calculus and probabilistic computation. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785699.

27 Francesco Gavazzo and Claudia Faggian. A relational theory of monadic rewriting systems, part I. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470633.

28 Simon J. Gay. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006. doi:10.1017/S0960129506005378.

29 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

30 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In Hans-Juergen Boehm and Cormac Flanagan, editors, *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'13*, pages 333–342. ACM, 2013. doi:10.1145/2491956.2462177.

31 Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 15–30, 2019. doi:10.4204/EPTCS.292.2.

32 Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From innermost to full almost-sure termination of probabilistic term rewriting. In Naoki Kobayashi and James Worrell, editors, *Foundations of Software Science and Computation Structures - 27th International Conference, FoSSaCS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14575 of *Lecture Notes in Computer Science*, pages 206–228. Springer, 2024. doi:10.1007/978-3-031-57231-9_10.

33 Maja H. Kirkeby and Henning Christiansen. Confluence and convergence in probabilistically terminating reduction systems. In *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017*, pages 164–179, 2017. doi:10.1007/978-3-319-94460-9_10.

34 Emanuel H. Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, Los Alamos, New Mexico, US., 1996.

35 Dongho Lee, Valentin Perrelle, Benoît Valiron, and Zhaowei Xu. Concrete categorical model of a quantum circuit description language with measurement. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *Proceedings of the 41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021*, volume 213 of *LIPIcs*, pages 51:1–51:20, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.51.

36 M.H.A. Newman. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2), 1942.

37 Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2002.

38 Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to higher-order quantum computing. In Suresh Jagannathan and Peter Sewell, editors, *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'14)*, pages 647–658. ACM, 2014. doi:10.1145/2535838.2535879.

39 Luca Paolini, Mauro Piccolo, and Margherita Zorzi. qPCF: higher-order languages and quantum circuits. *Journal of Automated Reasoning*, 63(4):941–966, 2019. doi:10.1007/s10817-019-09518-y.

40 Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*, pages 846–858. ACM, 2017. doi:10.1145/3009837.3009894.

41 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

42 Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16:527–552, 2006. doi:10.1017/S0960129506005238.

43 Alex K. Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In *Term Rewriting and Applications, 16th International Conference (RTA 2005)*, volume 3467 of *Lecture Notes in Computer Science*, pages 219–234, 2005. doi:10.1007/978-3-540-32033-3_17.

A Convention for Garbage Collection

In the definition of programs, we use the convention that the size of the memory is exactly the number of registers manipulated in the term. The memory will grow when new qubits are allocated, and shrink when qubits are read (see Figure 1): the reduction perform garbage collection on the fly.

If this makes it easy to identify identical programs, it makes the proofs a bit cumbersome. We therefore rely for them on an equivalent representation, where a program can have spurious qubits, as long as they are not *entangled* with the rest of the memory – i.e. when measuring them would not change the state of the registers manipulated by the term. So for instance, in this model $[|0\rangle \otimes |\psi\rangle; r_1]$ is the same as $[\phi; r_0]$.

B Technical properties

In all proofs we freely use the following closure property, which is immediate by definition of context and surface context.

▶ **Fact B.1** (Closure).

$$\frac{[\mathtt{Q}; M] \to_c \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}}{[\mathtt{Q}; \boldsymbol{S}(\!|M|\!)] \to_q \{\!\{ p_i[\mathtt{Q}_i; \boldsymbol{S}(\!|M_i|\!)] \}\!\}}\ 1. \qquad \frac{[\mathtt{Q}; M] \to_\beta \{\!\{ [\mathtt{Q}; M'] \}\!\}}{[\mathtt{Q}; \boldsymbol{C}(\!|M|\!)] \to_\beta \{\!\{ [\mathtt{Q}; \boldsymbol{C}(\!|M'|\!)] \}\!\}}\ 2.$$

Surface closure (point 1.) also holds with $\to_\beta$ in place of $\to_q$.

We will also use the following lemma (analog to substitutivity in [7], p.54) The proof is straightforward.

▶ **Lemma B.2** (Substitutivity). *Assume $[\mathtt{Q}; P] \in \mathcal{P}$ and $[\mathtt{Q}; P] \to \{\!\{ p_i[\mathtt{Q}_i; P_i] \}\!\}$. Then for each term N : $[\mathtt{Q}; P\{N/x\}] \to \{\!\{ p_i[\mathtt{Q}_i; P_i\{N/x\}] \}\!\}$.*

The converse also holds, and it is simply Fact B.1, that can be reformulated as follows.

▶ **Fact B.3.** *Assume $[\mathtt{Q}; N] \in \mathcal{P}$, $[\mathtt{Q}; N] \to_q \{\!\{ p_i[\mathtt{Q}_i; N_i] \}\!\}$ and P a term such that x is* linear *in P. Then $[\mathtt{Q}; P\{N/x\}] \to_q \{\!\{ p_i[\mathtt{Q}_i; P\{N_i/x\}] \}\!\}$*

Surface Reduction. has a prominent role. We spell-out the definition.

Surface Reduction Step $\underset{s}{\to}$

$$\underset{s}{\to} := \underset{s}{\to_\beta} \cup \to_q$$

Surface Beta Step $\underset{s}{\to_\beta}$	**(Surface) q-Step $\to_q$**
$\dfrac{[\mathtt{Q}; M] \mapsto_\beta \{\!\{ [\mathtt{Q}; M'] \}\!\}}{[\mathtt{Q}; \mathbf{S}(\!|M|\!)] \underset{s}{\to_\beta} \{\!\{ [\mathtt{Q}; \mathbf{S}(\!|M'|\!)] \}\!\}}$	$\dfrac{[\mathtt{Q}; M] \mapsto_q \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}}{[\mathtt{Q}; \mathbf{S}(\!|M|\!)] \to_q \{\!\{ p_i[\mathtt{Q}_i; \mathbf{S}(\!|M_i|\!)] \}\!\}}$

Figure 5 Surface Reduction Steps.

C Surface Reduction has the Diamond Property

We obtain the diamond property (Proposition 6.3) from the pointed diamond, result using the following technique (from [26]) , which allows us to *work pointwise*.

▶ **Lemma** (pointwise Criterion (FaggianRonchi19)). *Let $\to_a, \to_b \subseteq \mathcal{P} \times \mathtt{MD}(\mathcal{P})$ and $\Rightarrow_a, \Rightarrow_b$ their lifting. To prove that $\Rightarrow_a, \Rightarrow_b$ diamond-commute,* i.e.

$$\textit{If } \mathbf{p} \Rightarrow_b \mathtt{m}_1 \textit{ and } \mathbf{p} \Rightarrow_a \mathtt{m}_2, \textit{ then } \exists \mathtt{r} \textit{ s.t. } \mathtt{n} \Rightarrow_a \mathtt{r} \textit{ and } \mathtt{s} \Rightarrow_b \mathtt{r}.$$

it suffices to prove the property (#) below (stated in terms of a single program $\mathbf{p}$)

$$\textit{(\#) If } \mathbf{p} \to_b \mathtt{m}_1 \textit{ and } \mathbf{p} \to_a \mathtt{m}_2, \textit{ then } \exists \mathtt{r} \textit{ s.t. } \mathtt{n} \Rightarrow_a \mathtt{r} \textit{ and } \mathtt{s} \Rightarrow_b \mathtt{r}.$$

The same result holds with $\rightrightarrows$ in place of $\Rightarrow$.

The criterion together with Lemma 6.2 (Point 1.) yields

▶ **Prop** (6.3). *Surface reduction $\underset{s}{\Rightarrow}$ has the diamond property. The same holds for $\underset{s}{\rightrightarrows}$.*

D Finitary Standardization

Shape Preservation. We recall a basic but key property of contextual closure. If a step $\to_\gamma$ is obtained by closure under *non-empty context* of a rule $\mapsto_\gamma$, then it preserves the shape of the term. We say that T and T' have *the same shape* if both terms belong to the same production (*i.e.*, both terms are an application, an abstraction, a variable, a register, a term of shape $!P$, $\mathtt{new}$, etc).

▶ **Fact D.1** (Shape preservation). *Assume* $[\mathtt{Q}; M] \to \{\!\{ p_i[\mathtt{Q}_i; M_i] \}\!\}$, $M = \boldsymbol{C}\langle\!| R |\!\rangle, M_i = \boldsymbol{C}\langle\!| R_i |\!\rangle$ *and that the context* $\boldsymbol{C}$ *is* non-empty. *Then (for each* i*),* M *and* M_i *have the same shape.*

An easy-to-verify consequence is the following, stating that *non-surface steps* ($\underset{\neg \mathsf{s}}{\Rightarrow}$)

- do not change the quantum memory
- do not change the shape of the terms

Notice that the qubit state is *unchanged* by $\underset{\neg \mathsf{s}}{\Rightarrow}$ steps, since it can only be a $\underset{\neg \mathsf{s}}{\Rightarrow}_\beta$ step

▶ **Lemma D.2** (Redexes and normal forms preservation). *Assume* $[\mathtt{Q}; M] \underset{\neg \mathsf{s}}{\Rightarrow}_\beta \{\!\{ [\mathtt{Q}; M'] \}\!\}$.

1. M *is a redex iff* M' *is a redex. In this case, either both are* β*-redexes, or both meas-redexes.*
2. M *is* s*-normal if and only if* M' *is* s*-normal.*

Proof of Lemma 6.6.

▶ **Lemma** (Lemma 6.6). $[\mathtt{Q}; M] \underset{\neg \mathsf{s}}{\Rightarrow}_\beta \{\!\{ [\mathtt{Q}; P] \}\!\}$ *and* $[\mathtt{Q}; P] \to_q \mathtt{n}$ *implies* $[\mathtt{Q}; M] \to_q \cdot \Rightarrow_\beta \mathtt{n}$.

Proof. By induction on the context $\mathbf{S}$ such that $P = \mathbf{S}\langle\!| R |\!\rangle$ and $[\mathtt{Q}; \mathbf{S}\langle\!| R |\!\rangle] \to_q \{\!\{ p_i[\mathtt{Q}_i; \mathbf{S}\langle\!| R_i |\!\rangle] \}\!\} = \mathtt{n}$. We exploit in an essential way the fact that M and P have the same shape. ◀

E Asymptotic normalization

Proof Sketch. The proof of Theorem 7.4 relies on an abstract result from the literature [24], which here we reformulate in our setting:

▶ **Lemma E.1** (Asymptotic completeness criterion [24]). *Assume*

i. s-factorisation*: if* $\mathtt{m} \Rightarrow^* \mathtt{n}$ *then* $\mathtt{m} \underset{\mathsf{s}}{\Rightarrow}^* \cdot \underset{\neg \mathsf{s}}{\Rightarrow}^* \mathtt{n}$*;*

ii. $\neg$s-neutrality *:* $\mathtt{m} \underset{\neg \mathsf{s}}{\Rightarrow} \mathtt{m}'$ *implies* $\mathbb{P}(\mathtt{m}) = \mathbb{P}(\mathtt{m}')$.

Then: $\mathtt{m} \Downarrow p$ *implies* $\mathtt{m} \Downarrow_s p$.

Proof of Theorem 7.4. We establishing the two items below, and then compose them.

1. $\mathtt{m} \Downarrow p$ implies $\mathtt{m} \Downarrow_s p$
2. $\mathtt{m} \Downarrow_s p$ implies $\mathtt{m} \downdownarrows_s p'$, with $p \leq p'$

Item (1.) holds because $\underset{\mathsf{s}}{\Rightarrow}$ satisfies both conditions in Lemma E.1: point (i.) holds by Theorem 6.7, point (ii.) by Lemma D.2. Item (2.) is immediate. ◀