QPCF: Higher-Order Languages and Quantum Circuits

QPCF: higher-order languages and quantum circuits

Luca Paolini · Mauro Piccolo · Margherita Zorzi

Received: 2017 / Accepted: 2018 / Published online: November 18, 2019

Abstract qPCF is a paradigmatic quantum programming language that extends PCF with quantum circuits and a quantum co-processor. Quantum circuits are treated as classical data that can be duplicated and manipulated in flexible ways by means of a dependent type system. The co-processor is essentially a standard QRAM device, albeit we avoid to store permanently quantum states in between two co-processor's calls. Despite its quantum features, qPCF retains the classic programming approach of PCF. We introduce qPCF syntax, typing rules, and its operational semantics. We prove fundamental syntactic properties of the system. Moreover, we provide some higher-order examples of circuit encoding.

Keywords PCF · Quantum Computing · Quantum Programming Languages

1 Introduction

Quantum computing is an intriguing trend in computer science research. The interest about quantum computing is due to R. Feynman. In [8], the first relationships between the quantum mechanical model and a formal computational model such as Turing machines is stated. The first concrete proposal for a quantum abstract computer is due to D. Deutsch, who introduced quantum Turing machines [16]. Later, a number of results about computational complexity [64, 65, 29, 32] showed quantum computing is not only a challenging theoretical subject, but also a promising paradigm for the concrete realization of powerful machines.

L. Paolini · M. Piccolo
Department of Computer Science, University of Torino (Italy)
E-mail: paolini@di.unito.it,piccolo@di.unito.it

M. Zorzi
Department of Computer Science, University of Verona (Italy)
E-mail: margherita.zorzi@univr.it

Nowadays, quantum computers are a long term industrial goal and a tangible reality in term of prototypes. Even if physicists and engineers have to face tricky problems in the realization of quantum devices, the advance of these innovative technologies is ceaseless. As a consequence, to fully understand how to program quantum devices is become an urgent need.

Typically, calculi for quantum computable functions present two different computational features. On the first hand, there is the unitary aspect of the calculus, that captures the essence of quantum computing as algebraic transformations. On the other hand, it should be possible to *control* the quantum steps, "embedding" the pure quantum evolution in a classical computation. Behind this second point, we have the usual idea of computation as a sequence of discrete steps on (the mathematical description of) an abstract machine. The relationship between these different aspects gives rise to different approaches to quantum functional calculi (as observed in [4]). If we divide the two features, i.e. we separate data from control, we adopt the so called *quantum data & classical control* (*qd&cc*) approach. This means that quantum computation is *hierarchically dependent* from the classical part: a classical program (ideally in execution on a classical machine) computes some "directives" and, these directives are sent to a hypothetical device which applies them to quantum data. Therefore quantum data are manipulated by the classical program or, in other words, classical computational steps control the unitary part of the calculus.

This idea is inspired to an architectural model called Quantum Random Access Machine (QRAM). The QRAM has been defined in [36] and can be viewed as a classically controlled machine enriched with a quantum device. On the grounds of the QRAM model, P. Selinger defined the first functional language based on the quantum data-classical control paradigm [62]. This work represents a milestone in the development of quantum functional calculi and inspired a number of different investigations. Many quantum programming languages implementing the *qd&cc* approach can be found in literature [62, 63, 55, 60]. Details about related works are in Section 7.

In this paper we propose some new contributions to the research on the *qd&cc* paradigm by formalizing the quantum language qPCF, based on a simplified version of the QRAM model restricted to total measurements (cf. Section 2.1). qPCF extends PCF, namely the prototype of typed functional language. Some qPCF features is listed below.

- *Absence of explicit linear typing constraints*: the management of linear resources is radically different from the mainstream in languages inspired to Linear Logic such as [62, 13, 10, 11, 37, 77, 60, 38]; so, we do not use linear/exponential modalities.
- *Use of dependent types* : we decouple the classical control from the quantum computation and we use dependent types to manage quantum circuits. Dependent types allow to write not only individual quantum circuits, but also parametric programs.

- *Emphasis on the Circuit Construction*: In accord with the recent trends in quantum programming theory [69, 55, 59], qPCF focuses on the quantum circuit description aspects. This idea already considered in [36] aims to ease the programming of quantum algorithms.
- *No permanent quantum states.* Differently from other proposals (e.g. [62, 13, 77]), qPCF does not need types for quantum states (and its linear management). This is possible, since the interaction with the quantum co-processor is neatly decoupled by means of the operator dmeas. It offloads a quantum circuit to a co-processor for the evaluation which is immediately followed by a *(von Neumann) Total Measurement* [46]. This means that partial measures are forbidden. Luckily, the deferred measurement principle [46] says us that this restriction does not represent a theoretical limitation (cf. Section 8).

qPCF is an higher-order programming language that retains the standard classic programming approach. Potentially, this can ease the transition to computers endowed with quantum co-processors.

A preliminary version of qPCF has been proposed in [54]. With respect to the first version, we extended our proposal following several directions.

- We slightly reformulate syntax and heavily improve the type system, that is radically more refined.
- We carefully provide details of proofs (just stated in the first version). Proofs are non trivial, since one has to consider infinite ground types and to unravel the mutual relationships that holds between syntactic classes.
- We extend the semantic in order to take into account the probabilities of quantum measurements.
- We add many examples.
- We provide a deep discussion about related works by focusing on classical control quantum languages.
- We define a restricted version of the general QRAM (cf. Section 2) that represents the architecture required by qPCF.

Summing up, we propose a new, stand-alone quantum programming language that aims to combine classical programming style, parametric circuit programming, higher-order and quantum features in a unified setting.

Synopsis

In Section 2 we describe the idealized quantum programming environment (a restricted version of the QRAM machine) behind qPCF. Section 3 and Section 4 introduce the syntax and typing rules of qPCF respectively. The operational semantic of qPCF and main properties of the system are in Section 5. In Section 6 we discuss the implementation of some quantum algorithms. A detailed overview about the state of art of classical control quantum languages is in Section 7. In Section 8 we propose some conclusive considerations with special care for the language expressivity.

2 Background: the QRAM model

We assume some familiarity with notions as *quantum bits* (or *qubits* the quantum equivalent of classical data), quantum states [46, 39, 72, 73] (systems of n quantum bits), quantum circuit and quantum circuit families [47]. In Section 2.1 we introduce a simplified version of the QRAM (the architectural model behind the quantum data and classical control approach) called rQRAM. The rQRAM represents the idealized hardware to execute qPCF.

2.1 Introducing an idealized co-processor

The physical realization of basic components necessary for universal quantum computation has gathered much attention in recent years, and many realistic technologies have emerged (see [41] for a survey): trapped ions, quantum dots, polar molecules, and superconductors among others. It is commonly accepted that a physical quantum computer has to fulfill the following five criteria which were proposed by Di Vincenzo at IBM in [20]: (i) a scalable physical system of qubits; (ii) the ability to initialize the state of the qubits; (iii) long relevant decoherence times (qubits lose their quantum properties exponentially quickly); (iv) a universal set of quantum gates; and (v) a measurement capability. These criteria correspond quite directly to various engineering hurdles that implementations have to face. In particular, in the last years many efforts have been supplied to increase the decoherence times and to overcome adjacency/neighboring constraints on qubits (see [9, 20, 41]).

It is clear that future quantum hardware may differ in many details, so we have to look for some abstract model of quantum computations. qPCF is designed to be executed on the QRAM programming environment (see [36, 61, 42]) which is commonly accepted as a reasonable model of computation for describing quantum computing devices. However, qPCF rests on a restricted QRAM model that relaxes two crucial issues: (i) bounded dechoerence times are sufficient because the decoherence-care is needed only during the evaluation of a single circuit and, (ii) free-rewiring hardware ability is not required, since we strictly rest on the basic gates provided by the co-processor.

We quote the QRAM introduction provided in [36].

> It is increasingly clear that practical quantum computing will take place on a classical machine with access to quantum registers. The classical machine performs off-line classical computations and controls the evolution of the quantum registers by initializing them to certain states, operating on them with elementary unitary operations and measuring them when needed.

We also quote some concise remarks done by Selinger in [61].

> Typically, the quantum device will implement a fixed, finite set of unitary transformations that operate on one or two quantum bits at a time. The classical controller communicates with the quantum device by sending a sequence of instructions, specifying which fundamental operations are to be performed. The only output from the quantum device consists of the results of measurements, which are sent back to the classical controller.

It is worthwhile to note that the QRAM model is sometimes considered too restrictive to support proposed quantum programming languages; therefore, it is enhanced to include more possible interactions between the classical and the quantum devices. (see [68][p.3])

qPCF is designed to operate in a restricted QRAM programming environment named rQRAM, that corresponds, quite well, to an idealized co-processor for a classical computer. The idea is that our classical computers compute circuits (i.e. a sequences of gates) that are classical data. A circuit can be offloaded to the quantum device in order to be applied to a quantum register suitably initialized. The co-processor has to be able: (i) to initialize a register to a given classic value; (ii) to apply a given sequence of gates on the state stored in the register; (iii) to perform a final quantum measurement of the whole register. It is worthwhile to remark some peculiarities of our rQRAM. It neatly isolates the application of non-unitary transformations from unitary ones; the set of available quantum gates (hopefully, a universal set of quantum gates) determines the possible permutations of qubits; and, more interestingly, the register is not assumed to be able to store permanently qubits (with their unbounded decoherence time issues). Indeed, the rQRAM co-processor has to support a bounded decoherence time (the maximal of times needed to the application of gates in the device) and it is even possible to imagine how to include several classical controllers to share the access to the same single quantum device. However, qPCF can be easily executed on standard QRAM models. We are convinced that its realization can be easier that the standard QRAM models and interesting for implementation. In Section 8.1 we explain why our model does not cause any theoretical limitation, albeit some algorithms cannot be directly executed on it.

3 qPCF

qPCF is quantum programming language based on the lambda-calculus, like many other quantum calculi (see, for instance, [62,13,77]). In accord with the recent trend in the development of quantum programming languages [28,54, 55,59,60], qPCF focuses on the abilities to generate and manipulate quantum circuits.

No permanent quantum state can be stored in qPCF, thus linear types are avoided: this distinguishes qPCF from other typed quantum programming languages. The linearity for quantum control is completely confined to atomic datatypes by using a simplified form of dependent types as that suggested in [56]: a dependent type picks up a family of types that bring in the type auxiliary information (just the arity of a circuit, in our case).

3.1 Syntax Overview.

qPCF extends PCF [57,23] to manage some additional atomic data structures: *indexes* (normalizing number expressions) and *circuits*. Index expressions are

essentially built by means of variables, numerals and some *total* operations on expressions: $\odot \in \{+, *\}$ (viz. sum, product). Circuits expressions are obtained by means of suitable operators combining gates; their evaluation is expected to produce (when terminating) strings on gates.

The row syntax of qPCF follows:

$$\begin{aligned} \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q} ::= {} & \mathtt{x} \mid \lambda \mathtt{x}.\mathtt{M} \mid \mathtt{MN} \mid \underline{\mathtt{n}} \mid \mathtt{pred} \mid \mathtt{succ} \mid \mathtt{if} \mid \mathtt{Y}_\sigma \mid \mathtt{set} \mid \mathtt{get} \\ & \mid \mathtt{U} \mid \fatsemi \mid \| \mid \mathtt{iter} \mid \mathtt{reverse} \mid \odot \mathtt{EE}' \mid \mathtt{size} \mid \mathtt{dMeas} \end{aligned}$$

where E ranges over index expressions, namely terms typed as indexes.

In the first row, we include PCF extended with some syntactic sugar in order to facilitate the bit-wise access to numerals: `get` allows us to extract (to read) the i-th digit of the binary representation of a numeral, i.e. its i-th bit; `set` allows to modify the i-th bit of a numeral. They are added to simplify the initialization and decomposition of states.

qPCF is parameterized by a set of quantum gates that correspond to the unitary operators made available by the quantum co-processor. We assume U to range on available gates and, in accord with [55], we can assume that a universal subset of unitary gates (see [33, 35, 46]) is available. Let $\mathcal{U}$ be the set of available computable unitary operators, such for each gate U there is a unique $\mathbf{U} \in \mathcal{U}$. If $k \in \mathbb{N}$ then we denote $\mathcal{U}(k)$ the gates in $\mathcal{U}$ having arity $k+1$, so $\mathcal{U} = \bigcup_0^\omega \mathcal{U}(k)$. More explicitly, the gates of arity 1 are in $\mathcal{U}(0)$ and so on.

The syntax of (evaluated) *circuits* is generated by $\fatsemi, \|$ and gate-names. The symbol $\fatsemi$ sequentializes two circuits of the same arity, while $\|$ denotes the parallel composition of circuits. We build circuit expressions by means of `iter` and `reverse`. We use `iter` to produce the parallel composition of a first circuit with a given number of a second one. We use `reverse` to transform a circuit into another one of the same arity.

Indexes are operated by *total* operations. W.l.o.g. we assume $\odot \in \{+, *\}$ (viz. sum, product). Types of circuits include index expressions. We add `size` to qPCF to emphasize the gain that dependent type can concretely provide, although this makes the proofs of the language properties more complex. `size` is an operator that applied to a circuit-expression returns the arity of the corresponding circuit, viz. an index information.

Last, but not least, we use `dMeas` to evaluate circuits suitably initialized: `dMeas` returns a numeral being the binary representation of a quantum measure executed on all quantum wires (of the circuit) as a whole.

qPCF is indeed conceived to manage circuits that can be freely duplicated and erased, while quantum states are hidden by means of `dMeas`. In some sense, `dMeas` offloads a quantum circuit to a co-processor, it waits the end of circuit execution and it returns the final measure of all wires.

3.2 Dependent Types.

Dependent types are widely used in proof-theoretical research. Typically, in presence of strongly normalizing languages. Unfortunately, the type-checking

of dependent types requires to decide the equality of terms (that can be included in types) and the strong normalization is not realistic for programming languages [52]. Therefore, the management of terms in types is the crucial issue that has to be faced in a programming language. This point is discussed in page 75, Section 2.8 of [7] where different programming approaches are compared. Following [74, 76], qPCF forbids the inclusion of arbitrary terms in types. A suitable subclass of terms (numeric expressions always normalizing) is identified to this purpose: these terms are called *indexes*.

Our approach to dependent types is closely inspired to that mentioned in [56, §30.5] to manage vector's types: the decoration carries with it, some dimensional (i.e. numeric) information. We avoid general dependent types systems (see [7] for a survey) because their great expressiveness is exceeding our needs. We prefer to maintain the qPCF type system as simple as possible in order to show the feasibility of the approach and its concrete benefits.

3.3 Types Overview.

Traditional types of PCF (i.e. integers and arrows) are extended to include 2 new types: a type for *indexes* (strong normalizing numeric expressions) and a type for *circuits* (carrying around indexes). Types of qPCF are formalized as follows:

$$\sigma, \tau ::= \mathsf{Nat} \mid \mathsf{Idx} \mid \mathsf{circ}(\mathtt{E}) \mid \Pi \mathtt{x}^{\sigma}.\tau$$

where $\mathtt{E}$ is an index expression (viz. a normalizing numeric expression). As common in presence of dependent types, we replace arrows by quantified types that include more information than arrows: they make explicit, in the type, the variable-name which is bounded. The variable-name can be α-renamed whenever the standard capture-free proviso is satisfied. We write $\sigma \to \tau$ as an abbreviation for $\Pi \mathtt{x}^{\sigma}.\tau$ whenever, either the variable-name $\mathtt{x}$ does not occur in τ or we are not concerned with $\mathtt{x}$.

3.4 Indexes.

The type Idx picks up a subset of numeric normalizing expressions. In particular, Idx does not allow to type non-normalizing expressions as $\mathtt{Y}(\lambda \mathtt{x}.\mathtt{x})$. The main use of Idx is to type all proper-terms used in dependent types. Indexes are built on numerals and total operations on them. We limit our operations to addition and multiplication, but we assume that this set is conveniently tuned in a concrete case, e.g. by adding the (positive subtraction) $\dot{-}$, or the modulus %, or a selection $\mathtt{if}^{\mathtt{x}}$ and so on. See Remark 1 for more details.

4 Typing system

Finite sets of pairs "variable:type" are called *bases* whenever variable-names are disjoint: we use B to range over them. We denote $\mathrm{dom}(B)$ the finite set of variable names included in B and we denote $\mathrm{ran}(B)$ the finite set of types that B associates to variables. As usual, we use the notation $B \cup \{\mathtt{x} : \sigma\}$ to extend the base B with the pair $\mathtt{x} : \sigma$ under the proviso that, either we are adding a fresh variable (i.e. $\mathtt{x} \notin \mathrm{dom}(B)$) or that $\mathtt{x} : \sigma$ is already in B. We assume analogous conditions writing $B \cup B'$, for any B'.

qPCF includes a typing axiom of the shape $x : \sigma \vdash x : \sigma$ for each type σ. In presence of dependent types, σ can contain terms and, consequently, it is a valid type only when such terms are well-typed. Luckily we have a unique type that includes dependencies, namely the type of circuit; moreover, it includes a term that must be typed as index. We solve this issue by using the notation $\vDash B$ in order to express the proviso that all types in B are well typed.

Definition 1 We write $\vDash B$ in order to meaning that, for all $\mathtt{x} : \sigma$ in B, if $\mathsf{circ}(\mathtt{E})$ occurs in σ then $B\backslash_{\{\mathtt{x}:\sigma\}} \vdash \mathtt{E} : \mathsf{Idx}$ is a valid typing (cf. Definition 2). □

Example 1 For instance, we write $\vDash \{\mathtt{x} : \mathsf{Idx}, \mathtt{z} : \mathsf{circ}(\mathtt{x} \oplus \mathtt{E}')\}$ in order to mean that $\mathtt{x} : \mathsf{Idx} \vdash \mathtt{z} \oplus \mathtt{E}' : \mathsf{Idx}$ is a valid typing and that $\{\mathtt{x} : \mathsf{Idx}\} \vdash \mathtt{E} : \mathsf{Idx}$ holds for all types $\mathsf{circ}(\mathtt{E})$ eventually occurring in $\mathtt{E}'$. □

Usually, dependent types are formalized via the introduction of super-types (named kinds) and super-typing rules (a.k.a. kinding rules) that identify well-given types, see [7] for instance. To limit the number of kinding rules, dependent type systems introduce formal tricks that allow to re-use the (ground) typing rules in the kinding system. We taken advantage of our limited use of dependent types to circumvent the introduction of kinds and kinding rules aiming at: (i) to avoid the explosion of the number of rules and complex overlapping of rules; (ii) to make the extension of PCF as clear as possible; and, (iii) to avoid further complexity (viz. mutual induction) in proofs. For the sake of completeness, we remark that it is possible to reformulate the use of $\vDash$ by adding a unique kind $\square$ to identify well-given types, and to add kinding rules checking that terms in types are well-typed.

Definition 2 The rules of the typing system are given in Table 1. A typing is *valid* whenever it is the conclusion of a finite type derivation built on the given rules. Types are considered up to the congruence $\simeq$, which is the smallest equivalence including: (i) the α-conversion of bound variables; (ii) β-inter-convertibility of β-redexes (occurring in terms included in types); (iii) associativity, commutativity and distributivity of sum and product together with the properties of neutral elements (viz. 0, 1). □

We consider types up to inter-convertibility of included terms. For the sake of simplicity, we avoid to formalize the interconvertibility via additional rules. Moreover, we remark that we introduce the untyped row syntax (cf. Section

$$\frac{\vDash B \cup \{\mathtt{x} : \sigma\}}{B \cup \{\mathtt{x} : \sigma\} \vdash \mathtt{x} : \sigma}\ (P_0) \qquad \frac{B \cup \{\mathtt{x} : \sigma\} \vdash \mathtt{N} : \tau}{B \vdash \lambda \mathtt{x}^{\sigma}.\mathtt{N} : \Pi \mathtt{x}^{\sigma}.\tau}\ (P_1) \qquad \frac{B \vdash \mathtt{P} : \Pi \mathtt{x}^{\sigma}.\tau \quad B \vdash \mathtt{Q} : \sigma}{B \vdash \mathtt{PQ} : \tau[\mathtt{Q}/\mathtt{x}]}\ (P_2)$$

$$\frac{\vDash B}{B \vdash \mathtt{succ} : \mathsf{Nat} \to \mathsf{Nat}}\ (P_3) \qquad \frac{\vDash B}{B \vdash \mathtt{pred} : \mathsf{Nat} \to \mathsf{Nat}}\ (P_4)$$

$$\frac{\vDash B}{B \vdash \mathtt{if} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}}\ (P_5) \qquad \frac{B \vdash \mathtt{E} : \mathsf{Idx}}{B \vdash \mathtt{if} : \mathsf{Nat} \to \mathsf{circ}(\mathtt{E}) \to \mathsf{circ}(\mathtt{E}) \to \mathsf{circ}(\mathtt{E})}\ (P_5')$$

$$\frac{\sigma = \tau_1 \to \ldots \to \tau_n \to \gamma \quad (\gamma \in \{\mathsf{Nat}, \mathsf{circ}(\mathtt{E})\} \text{ and } n \geq 0) \qquad \vDash B \cup \{\mathtt{x} : \sigma\} \text{ where } \mathtt{x} \text{ is fresh}}{B \vdash \mathtt{Y}_{\sigma} : (\sigma \to \sigma) \to \sigma}\ (P_6)$$

$$\frac{\vDash B}{B \vdash \mathtt{get} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}}\ (B_1) \qquad \frac{\vDash B}{B \vdash \mathtt{set} : \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}}\ (B_2)$$

$$\frac{B \vdash \mathtt{M} : \mathsf{Idx}}{B \vdash \mathtt{M} : \mathsf{Nat}}\ (I_0) \qquad \frac{\vDash B}{B \vdash \underline{\mathtt{n}} : \mathsf{Idx}}\ (I_1) \qquad \frac{B \vdash \mathtt{E}_0 : \mathsf{Idx} \quad B \vdash \mathtt{E}_1 : \mathsf{Idx}}{B \vdash \odot\, \mathtt{E}_0\, \mathtt{E}_1 : \mathsf{Idx}}\ (I_2) \qquad \frac{B \vdash \mathtt{M} : \mathsf{circ}(\mathtt{E})}{B \vdash \mathtt{size}\ \mathtt{M} : \mathsf{Idx}}\ (I_3)$$

$$\frac{\mathtt{U} \in \mathcal{U}(k) \qquad \vDash B}{B \vdash \mathtt{U} : \mathsf{circ}(\underline{\mathtt{k}})}\ (C_1) \qquad \frac{B \vdash \mathtt{E} : \mathsf{Idx}}{B \vdash \mathtt{;} : \mathsf{circ}(\mathtt{E}) \to \mathsf{circ}(\mathtt{E}) \to \mathsf{circ}(\mathtt{E})}\ (C_2)$$

$$\frac{B \vdash \mathtt{E}_0 : \mathsf{Idx} \qquad B \vdash \mathtt{E}_1 : \mathsf{Idx}}{B \vdash \| : \mathsf{circ}(\mathtt{E}_0) \to \mathsf{circ}(\mathtt{E}_1) \to \mathsf{circ}(\mathtt{E}_0 + \mathtt{E}_1 + 1)}\ (C_3) \qquad \frac{B \vdash \mathtt{E} : \mathsf{Idx}}{B \vdash \mathtt{reverse} : \mathsf{circ}(\mathtt{E}) \to \mathsf{circ}(\mathtt{E})}\ (C_4)$$

$$\frac{B \vdash \mathtt{E}_0 : \mathsf{Idx} \qquad B \vdash \mathtt{E}_1 : \mathsf{Idx}}{B \vdash \mathtt{iter} : \Pi \mathtt{x}^{\mathsf{Idx}}.\, \mathsf{circ}(\mathtt{E}_0) \to \mathsf{circ}(\mathtt{E}_1) \to \mathsf{circ}(\mathtt{E}_0 + ((1 + \mathtt{E}_1) * \mathtt{x}))}\ (C_5)$$

$$\frac{B \vdash \mathtt{E} : \mathsf{Idx}}{B \vdash \mathtt{dMeas} : \mathsf{Nat} \to \mathsf{circ}(\mathtt{E}) \to \mathsf{Nat}}\ (M)$$

Table 1 Typing Rules.

3) only in order to help the reading. Nevertheless, we are interested only to explicitly typed terms (à la Church) where all terms are considered together with their whole typing information.

4.1 Type rules overview

Rules (P_0), (P_1), (P_2), (P_3), (P_4), (P_5), (P_5'), (P_6) are directly inherited from PCF. We add $\vDash$ constraint in the premises of rules where bases are introduced in order to ensure that all terms included in (considered) types are well typed. Rules $(P_5), (P_5'), (P_6)$ restrict types allowed for conditional and recursive terms in order to ensure that the evaluation of terms typed with Idx are strongly normalizing. Note that the premise of (P_5') implies that $\vDash B$. Rules $(P_1), (P_2)$ involve the type binder Π that generalizes standard arrow-type and take care of possible free variables in types. As expected in dependent type systems, (P_2) substitutes the argument also in types. In rules $(P_3), (P_4), (P_5), (P_5'), (P_6)$ we use arrows as an abbreviation for Π-types, since the corresponding variable-names is never typed with Idx. Note that all numerals are typed Nat by means of rules I_0 and I_1.

Remark 1 It is worth to notice that the choice of the operators that we admit in index expressions has a strong impact on the decidability of the type-checking

of the language because they can occur in types. First, a set of strong normalizing expression ensures that the evaluation of closed terms can be decided and no run-time error can arise. On the other hand, when we build a program we have to manage open terms and open expressions also in types. More explicitly, let us assume $B \vdash \mathtt{P} : \Pi \mathtt{x}^{\mathsf{circ}(\mathtt{E}_P)}.\tau$ and $B \vdash \mathtt{Q} : \mathsf{circ}(\mathtt{E}_Q)$: then, we can apply the rule (P_2) only whenever $\mathtt{E}_P \simeq \mathtt{E}_Q$. If the language of index expressions is not endowed with a decidable equality then we can run into unwanted programming awkwardness. The decidability of the identity between index expressions follows from [17]. Appealing extensions to elementary function or primitive recursive functions are possible, but their impact on the programming practice should be carefully considered facing decidability issues [58]. □

The rule (B_1) types `get`, that returns 0 or 1 when applied to two integers. More precisely it returns the bit (in the binary representation) of the first integer in the position pointed by the second integer. The rule (B_2) types `set` that takes in input two integers: the second one selects a bit in the binary representation of the first one and it returns as output the numeral obtained by setting to 1 the selected bit in the binary representation of the first numeral. Their typing agree with these behaviors.

The rule (I_0) allows to use an index expression as a term typed Nat. Rules $(I_1), (I_2)$ type our basic index expressions as expected. The rule (I_3) brings back in term the arity information included in the type of a circuit.

Rules $(C_1), (C_2), (C_3), (C_4), (C_5)$ type circuit expressions. We recall that the index 0 has to be intended denoting the arity 1. (C_1) makes available the basic gates. (C_2) types the sequential composition of circuits having the same arity. (C_3) types the parallel composition of two circuits. (C_4) types an operator that (possibly) transforms a circuit in its adjoint, so the arity is preserved. (C_5) types the parallel composition of some circuits, namely a base circuit `M` and some copies of a circuit `N`.

Finally, (M) types an operator taking in input a state (the binary representation of a numeral) and a circuit, that gives back another state.

Example 2 An interesting example of term that provides evidence of the circularity arising from dependent types follows.

$$\dfrac{\dfrac{}{\mathtt{x} : \Pi \mathtt{z}^{\mathsf{Idx}}.\, \mathsf{circ}(\mathtt{z}) \vdash \mathtt{x} : \Pi \mathtt{z}^{\mathsf{Idx}}.\, \mathsf{circ}(\mathtt{z})}\,(P_0) \qquad \dfrac{\begin{array}{c}\vdots\\ \vdash \mathtt{M} : \mathsf{circ}(\mathtt{E})\end{array}}{\vdash \mathtt{size}\,(\mathtt{M}) : \mathsf{Idx}}\,(I_3)}{\mathtt{x} : \Pi \mathtt{z}^{\mathsf{Idx}}.\, \mathsf{circ}(\mathtt{z}) \vdash \mathtt{x}\,\mathtt{size}(\mathtt{M}) : \mathsf{circ}(\mathtt{size}(\mathtt{M}))}\,(P_2)$$

`M` can be any closed term of qPCF typed as circuit and `E` can be any closed term of qPCF typed Idx.

Since can be `M` can be any closed term, t his example shows that types can (possibly) include sub-terms being either non-terminating or open variables, not typed Idx. However, such terms are always argument of `size` that looks only for the index term in the "more external" type: but `size` throws away such information (cf. Definition 2 for more details). □

4.2 Some typing properties.

Many standard properties can be easily adapted to typing system in Table 1. If $B \vdash \mathtt{M} : \tau$ then $\mathrm{FV}(\mathtt{M}), \mathrm{FV}(\tau) \subseteq dom(B)$. Bases of typing can be weakened, viz. if $B \vdash \mathtt{M} : \tau$ and $dom(B) \cup dom(B') = \emptyset$ then $B \cup B' \vdash \mathtt{M} : \tau$. Moreover, it is easy to check that $B \vdash \mathtt{M} : \tau$ implies that $\vDash B$. Straightforward adaptation of standard Generation Lemmas hold too. However our interest is more focused on dynamic properties of the typing system than on its logical properties.

Lemma 1 (Substitution lemma)
If $B \cup \{\mathtt{x} : \sigma\} \vdash \mathtt{M} : \tau$ and $B' \vdash \mathtt{N} : \sigma$ then $B[\mathtt{N}/\mathtt{x}] \cup B' \vdash \mathtt{M}[\mathtt{N}/\mathtt{x}] : \tau[\mathtt{N}/\mathtt{x}]$.

Proof By induction on the derivation $B \cup \{\mathtt{x} : \sigma\} \vdash \mathtt{M} : \tau$. In accordance with the notation introduced at the beginning of Section 4, $\mathrm{dom}(B) \cup dom(B')$ can be not empty. □

From Lemma 1 the subject reduction follows easily.

Lemma 2 *Let $\mathcal{D}$ be a derivation concluding $B \vdash \mathtt{M}[N/\boldsymbol{x}] : \sigma$.*

1. *If $\boldsymbol{x} \in \mathrm{FV}(\mathtt{M})$ then, $\mathcal{D}$ includes a subderivation $\mathcal{D}^N$ concluding $B^N \vdash N : \tau$, for some $B \subseteq B^N$ and τ; and, moreover, $B \vdash \lambda\boldsymbol{x}.\mathtt{M} : \Pi\boldsymbol{x}^\tau.\sigma$.*
2. *If $\boldsymbol{x} \notin \mathrm{FV}(\mathtt{M})$ and $\boldsymbol{x} \notin dom(B)$ then $B \vdash \lambda\boldsymbol{x}.\mathtt{M} : \Pi\boldsymbol{x}^\tau.\sigma$ for any τ.*
3. *If $\mathcal{D}^N$ is a derivation concluding $B \vdash N : \tau$, for some τ then $B \vdash (\lambda\boldsymbol{x}.\mathtt{M})N : \sigma$.*

Proof 1. First, by induction on $\mathcal{D}$ we prove that in $\mathcal{D}$ there is a subderivation $B^{\mathtt{N}} \vdash \mathtt{N} : \tau$. Second, by induction on $\mathcal{D}$ we prove that we can transform $\mathcal{D}$ in a derivation $\mathcal{D}^*$ concluding $B \cup \{\mathtt{x} : \tau\} \vdash \mathtt{M} : \sigma$. We conclude by using the rule (P_1).
2. Since $\mathtt{M}[\mathtt{N}/\mathtt{x}] = \mathtt{M}$, it is easy to prove that $B \cup \{\mathtt{x} : \tau\} \vdash \mathtt{M} : \sigma$ by induction on $\mathcal{D}$. We conclude by using the rule (P_1).
3. By the previous cases of this Lemma and by using the rule (P_2) (note that $\mathtt{x}$ does not occur in σ by hypothesis). □

Let $C[.]$ denote a context for qPCF.

Lemma 3 (Typed subject expansion) *Let $\mathcal{D}^N$ be a derivation concluding $B \vdash N : \tau$, for some τ. If $B \vdash C[\mathtt{M}[N/\boldsymbol{x}]] : \sigma$ then $B \vdash C[(\lambda\boldsymbol{x}.\mathtt{M})N] : \sigma$.*

Proof W.l.o.g. we can assume that $\mathtt{x}$ is fresh, so the proof follows by induction on $B \vdash C[\mathtt{M}[\mathtt{N}/\mathtt{x}]] : \sigma$ and by using the Lemma 2. □

It is worth to remark some peculiarity of this typing system.

Lemma 4 *1. If $B \vdash \mathtt{M} : \mathsf{Idx}$ then $B \vdash \mathtt{M} : \mathsf{Nat}$.*
2. Let $B \vdash \mathtt{M} : \sigma$ be a typing derivation. If $\mathsf{circ}(E)$ occurs in it, then $B \vdash E : \mathsf{Idx}$.
3. If $B \vdash \mathtt{M} : \sigma$ then $\vDash B \cup \{\mathtt{x} : \sigma\}$ where $\mathtt{x}$ is fresh.

Proof 1. By rule (I_0). *2,3.* The proof can proceed by induction on the derivation $B \vdash \mathtt{M} : \sigma$ by using the Substitution Lemma. □

5 Semantics of qPCF

As for PCF, the evaluation of qPCF focuses on programs, viz. closed terms of ground types. Nevertheless, PCF has just one (or two) ground types, while qPCF has denumerable closed ground types, namely Nat, Idx and circ($\underline{\mathtt{n}}$).

As for PCF, the evaluation of a term typed Nat can either diverge or give back a numeral. The evaluation of a term typed Idx should always converge, and give back a numeral. The evaluation of a term typed circ(E) can either diverge or give back an evaluated circuit. In the following, we use C to denote circuits (resulting from the evaluation of circuit expressions), i.e. strings built by gate-names, parallel composition and serial composition. Moreover, numerals and circuit strings are sometimes denoted V.

Definition 3 We formalize the evaluation of qPCF by means of statements of the shape $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ obtained as conclusion of a (finite) derivation $\mathcal{D}$ built with the rules in Table 2, such that: (i) M is a closed ground term; and, (ii) $0 < \alpha \leq 1$ is the probability that $\mathcal{D}$ is the evaluation. For the sake of simplicity, we write $\mathtt{M} \Downarrow \mathtt{V}$ to mean that there is a derivation (not necessarily unique) concluding $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ for some $0 < \alpha \leq 1$ and, we write $\mathtt{M} \Uparrow$ to mean that no $\alpha > 0$ and V exist, such that $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$. □

We remark that the semantics of Table 2 rests on an external semantics via the premise of the rule (m), that executes the circuit in accord with the laws of quantum mechanics (cf. Definition 4).

Table 2 includes the standard call-by-name semantics of PCF, namely the first two lines of rules are well-known.

The rules (sz), (op) evaluate some index expressions. In particular, (sz) uses the typing information of M to recover its arity information. Since types are preserved during the evaluation (cf. Section 5.2), we can be sure that the information we extract from types is an index. Moreover, it is strongly normalizing by Theorem 1.

Example 3 We consider an example that, in some sense, allows us to complement the Example 2. It is easy to see that $\vdash \mathtt{Y}(\lambda \mathtt{x}^{\mathsf{circ}(8)}.\mathtt{x}) : \mathsf{circ}(8)$ and $\vdash \mathtt{size}\,(\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{circ}(8)}.\mathtt{x})) : \mathsf{Idx}$. Although $\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{circ}(8)}.\mathtt{x}) \Uparrow$, it is interesting to note that $\mathtt{size}\,(\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{circ}(8)}.\mathtt{x})) \Downarrow^{1} \underline{8}$. □

Let $\lceil \underline{\mathtt{m}} \rceil^{\underline{\mathtt{n}}}$ be notation for $(\overbrace{(\underline{\mathtt{m}} \,/\, \underline{2}) \ldots / \,\underline{2}}^{\underline{\mathtt{n}}})\%\underline{2}$ where / is the integer division (neglecting the remainder of the division) and % is the modulo (giving back the remainder of the division). Thus, $\lceil \underline{\mathtt{m}} \rceil^{\underline{0}}$ is the rightmost bit of the binary representation of $\underline{\mathtt{m}}$. The rules (gt) and (st) get/set a bit of the first argument (the one selected by the second argument). For example, the numeral $\mathtt{set}\,\underline{0}\,\underline{\mathtt{n}+1}$ is the decimal representation of the binary state $1\underbrace{0\ldots0}_{n}$ and $\mathtt{get}\,\underline{3}\,\underline{0}$ yields the bit 1. Clearly, set, get are syntactic sugar to manage input states.

$$\frac{}{\underline{\mathtt{n}} \Downarrow^{1} \underline{\mathtt{n}}}\ (n) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{n}}}{\mathtt{succ}(\mathtt{M}) \Downarrow^{\alpha} \underline{\mathtt{n}+1}}\ (s) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{n}+1}}{\mathtt{pred}(\mathtt{M}) \Downarrow^{\alpha} \underline{\mathtt{n}}}\ (p) \qquad \frac{\mathtt{M}[\mathtt{N}/\mathtt{x}]\mathtt{P}_1 \cdots \mathtt{P}_m \Downarrow^{\alpha} \mathtt{V}}{(\lambda \mathtt{x}.\mathtt{M})\mathtt{N}\mathtt{P}_1 \cdots \mathtt{P}_m \Downarrow^{\alpha} \mathtt{V}}\ (\beta)$$

$$\frac{\mathtt{M} \Downarrow^{\alpha} \underline{0} \quad \mathtt{L} \Downarrow^{\alpha'} \mathtt{V}}{\mathtt{if}\ \mathtt{M}\ \mathtt{L}\ \mathtt{R} \Downarrow^{\alpha\cdot\alpha'} \mathtt{V}}\ (\mathrm{if}_l) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{n}+1} \quad \mathtt{R} \Downarrow^{\alpha'} \mathtt{V}}{\mathtt{if}\ \mathtt{M}\ \mathtt{L}\ \mathtt{R} \Downarrow^{\alpha\cdot\alpha'} \mathtt{V}}\ (\mathrm{if}_r) \qquad \frac{\mathtt{M}(\mathtt{YM})\mathtt{P}_1 \cdots \mathtt{P}_i \Downarrow^{\alpha} \mathtt{V}}{\mathtt{YM}\mathtt{P}_1 \cdots \mathtt{P}_i \Downarrow^{\alpha} \mathtt{V}}\ (Y)$$

$$\frac{\vdash \mathtt{M} : \mathsf{circ}(\mathtt{E}) \qquad \mathtt{E} \Downarrow^{\alpha} \underline{\mathtt{n}}}{\mathtt{size}\,\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{n}}}\ (\mathrm{sz}) \qquad \frac{\mathtt{E}_0 \Downarrow^{\alpha} \underline{\mathtt{m}} \quad \mathtt{E}_1 \Downarrow^{\alpha'} \underline{\mathtt{n}}}{\odot\,\mathtt{E}_0\,\mathtt{E}_1 \Downarrow^{\alpha\cdot\alpha'} \underline{\mathtt{m}\odot\mathtt{n}}}\ (\mathrm{op}) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{m}} \qquad \mathtt{N} \Downarrow^{\alpha'} \underline{\mathtt{n}}}{\mathtt{get}\,\mathtt{M}\,\mathtt{N} \Downarrow^{\alpha\cdot\alpha'} \lceil \underline{\mathtt{m}} \rceil^{\underline{\mathtt{n}}}}\ (\mathrm{gt})$$

$$\frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{m}} \quad \mathtt{N} \Downarrow^{\alpha'} \underline{\mathtt{n}} \quad \text{and } \underline{\mathtt{m}}' \text{ is such that } \lceil \underline{\mathtt{m}}' \rceil^{\underline{\mathtt{n}}} = 1 \text{ and } \forall \underline{\mathtt{k}} \neq \underline{\mathtt{n}} \lceil \underline{\mathtt{m}}' \rceil^{\underline{\mathtt{k}}} = \lceil \underline{\mathtt{m}} \rceil^{\underline{\mathtt{k}}}}{\mathtt{set}\,\mathtt{M}\,\mathtt{N} \Downarrow^{\alpha\cdot\alpha'} \underline{\mathtt{m}}'}\ (\mathrm{st})$$

$$\frac{}{\mathtt{U} \Downarrow^{\alpha} \mathtt{U}}\ (\mathrm{u}) \qquad \frac{\mathtt{M}_0 \Downarrow^{\alpha} \mathtt{C}_0 \quad \mathtt{M}_1 \Downarrow^{\alpha'} \mathtt{C}_1}{\mathtt{M}_0 \fatsemi \mathtt{M}_1 \Downarrow^{\alpha\cdot\alpha'} \mathtt{C}_0 \fatsemi \mathtt{C}_1}\ (\mathrm{u}') \qquad \frac{\mathtt{M}_0 \Downarrow^{\alpha} \mathtt{C}_0 \qquad \mathtt{M}_1 \Downarrow^{\alpha'} \mathtt{C}_1}{\mathtt{M}_0 \parallel \mathtt{M}_1 \Downarrow^{\alpha\cdot\alpha'} \mathtt{C}_1 \parallel \mathtt{C}_0}\ (\mathrm{u}'')$$

$$\frac{\mathtt{M} \Downarrow^{\alpha} \mathtt{U} \quad (\ddagger \mathtt{U}) = \mathtt{U}'}{\mathtt{reverse}\,\mathtt{M} \Downarrow^{\alpha} \mathtt{U}'}\ (\mathrm{r}_0) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \mathtt{C}_0 \fatsemi \mathtt{C}_1 \quad \mathtt{reverse}\,\mathtt{C}_0 \Downarrow^{\alpha'} \mathtt{C}'_0 \quad \mathtt{reverse}\,\mathtt{C}_1 \Downarrow^{\alpha''} \mathtt{C}'_1}{\mathtt{reverse}\,\mathtt{M} \Downarrow^{\alpha\cdot\alpha'\cdot\alpha''} \mathtt{C}'_1 \fatsemi \mathtt{C}'_0}\ (\mathrm{r}_1)$$

$$\frac{\mathtt{E} \Downarrow^{\alpha} \underline{\mathtt{n}} \qquad \mathtt{M}_0 \Downarrow^{\alpha'} \mathtt{C}_0 \qquad \mathtt{M}_1 \Downarrow^{\alpha''} \mathtt{C}_1}{\mathtt{iter}\,\mathtt{E}\,\mathtt{M}_0\,\mathtt{M}_1 \Downarrow^{\alpha\cdot\alpha'\cdot\alpha''} \underbrace{\mathtt{C}_1 \parallel \cdots \parallel \mathtt{C}_1}_{\underline{\mathtt{n}}} \parallel \mathtt{C}_0}\ (\mathrm{it}) \qquad \frac{\mathtt{M} \Downarrow^{\alpha} \mathtt{C}_0 \parallel \mathtt{C}_1 \quad \mathtt{reverse}\,\mathtt{C}_0 \Downarrow^{\alpha'} \mathtt{C}'_0 \quad \mathtt{reverse}\,\mathtt{C}_1 \Downarrow^{\alpha''} \mathtt{C}'_1}{\mathtt{reverse}\,\mathtt{M} \Downarrow^{\alpha\cdot\alpha'\cdot\alpha''} \mathtt{C}'_0 \parallel \mathtt{C}'_1}\ (\mathrm{r}_2)$$

$$\frac{\mathtt{M} \Downarrow^{\alpha} \underline{\mathtt{m}} \quad \mathtt{N} \Downarrow^{\alpha'} \mathtt{C} \quad \mathtt{N} : \mathsf{circ}(\underline{\mathtt{k}}) \qquad (n, \alpha'') \in \mathrm{circuitEval}^{k}(\underline{\mathtt{m}}\rfloor_{\underline{\mathtt{k}}}, \mathtt{C})}{\mathtt{dMeas}(\mathtt{M}, \mathtt{N}) \Downarrow^{\alpha\cdot\alpha'\cdot\alpha''} \underline{\mathtt{n}}}\ (\mathrm{m})$$

Table 2 Operational Semantics.

The rules $(\mathrm{u}), (\mathrm{u}'), (\mathrm{u}''), (\mathrm{r}_0), (\mathrm{r}_1), (\mathrm{r}_2), (\mathrm{it})$ evaluate circuit expressions in circuits, viz. strings on $\fatsemi, \parallel$ and the gate-names $\mathtt{U}$. Note that (u') and (u'') evaluate sequential and parallel composition of circuit expressions.

Example 4 (Sequential composition of quantum circuits) Let $\mathtt{C} : \mathsf{circ}(\underline{\mathtt{k}})$ be a given circuit. We can use $\lambda \mathtt{x}^{\mathsf{circ}(\underline{\mathtt{k}})}.\mathtt{x} \fatsemi \mathtt{x} : \mathsf{circ}(\underline{\mathtt{k}}) \to \mathsf{circ}(\underline{\mathtt{k}})$ in order to concatenate two copies of $\mathtt{C}$. Let $\underline{\mathtt{k}}$ be an arbitrary numeral and let $\mathsf{M}_{\mathsf{seq}}$ be

$$\lambda \mathtt{u}^{\mathsf{circ}(\underline{\mathtt{k}})}.\lambda \mathtt{x}^{\mathsf{Nat}}.Y\mathsf{W}\mathtt{ux} : \mathsf{circ}(\underline{\mathtt{k}}) \to \mathsf{Nat} \to \mathsf{circ}(\underline{\mathtt{k}})$$

where $\mathsf{W} = \lambda \mathtt{w}^{\sigma}.\lambda \mathtt{u}^{\mathsf{circ}(\underline{\mathtt{k}})}.\lambda \mathtt{y}^{\mathsf{Nat}}.\,\mathtt{if}\ \mathtt{y}\ (\mathtt{u})\ (\fatsemi\,(\mathtt{u})\,(\mathtt{w}\,\mathtt{u}\,(\mathtt{pred}\,\mathtt{y})))$ has type $\sigma \to \sigma$, with $\sigma = \mathsf{circ}(\underline{\mathtt{k}}) \to \mathsf{Nat} \to \mathsf{circ}(\underline{\mathtt{k}})$. We can use $\mathsf{M}_{\mathsf{seq}}$ applied to $\mathtt{C}$ and $\underline{\mathtt{n}}$ to concatenate $n+1$ copies of $\mathtt{C}$. It is straightforward to parameterize $\mathsf{M}_{\mathsf{seq}}$ in order to transform it in a template for a circuit-builder that can be used for any arity. It suffices to replace $\underline{\mathtt{k}}$ with the variable $\mathtt{k}^{\mathsf{Idx}}$ and to abstract it; so that the resulting term $\mathsf{M}^{\mathsf{A}}_{\mathsf{seq}} = \lambda \mathtt{k}^{\mathsf{Idx}}.\lambda \mathtt{u}^{\mathsf{circ}(\underline{\mathtt{k}})}.\lambda \mathtt{x}^{\mathsf{Nat}}.Y\mathsf{W}\mathtt{ux}$ has type $\Pi \mathtt{k}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathtt{k}) \to \mathsf{Nat} \to \mathsf{circ}(\mathtt{k})$. □

The rule (it) provides a mechanism to compose circuits in parallel. It is driven by an argument of type Idx in order to ensure that iteration is strong normalizing and, consequently, that the arity of the generated circuit is always a numeral.

Example 5 (Parallel composition of quantum circuit) Let $\mathsf{M}_{\mathsf{par}}$ be

$$\lambda \mathtt{x}^{\mathsf{Idx}}.\lambda \mathtt{u}^{\mathsf{circ}(\underline{\mathtt{k}})}\lambda \mathtt{w}^{\mathsf{circ}(\underline{\mathtt{h}})}.\, \mathtt{iter}\ \mathtt{x}\,\mathtt{u}\,\mathtt{w} : \Pi \mathtt{x}^{\mathsf{Idx}}.\, \mathsf{circ}(\underline{\mathtt{k}}) \to \mathsf{circ}(\underline{\mathtt{h}}) \to \mathsf{circ}(\underline{\mathtt{k}} + (\mathtt{x} * (\underline{\mathtt{h}} + 1)))\,.$$

$\mathsf{M}_{\mathsf{par}}$ when applied to an $\underline{\mathtt{n}}$ and two unitary gates $\mathtt{U}_1 : \mathsf{circ}(\underline{\mathtt{k}})$ and $\mathtt{U}_2 : \mathsf{circ}(\underline{\mathtt{h}})$ generates a simple circuit built upon a copy of gate $\mathtt{U}_1$ in parallel with n copies of gate $\mathtt{U}_2$. It is straightforward to parameterize the above example. It suffices to replace numerals $\underline{\mathtt{k}}$ and $\underline{\mathtt{h}}$ in the above example by variables and to abstract to obtain a single parametric term typed $\Pi \mathtt{k}^{\mathsf{Idx}}.\Pi \mathtt{h}^{\mathsf{Idx}}.\Pi \mathtt{x}^{\mathsf{Idx}}.\, \mathsf{circ}(\mathtt{k}) \to \mathsf{circ}(\mathtt{h}) \to \mathsf{circ}(\mathtt{k} + (\mathtt{x} * (\mathtt{h} + 1)))$. □

More recent quantum programming languages [28,54,55,60] include the possibility to manipulate quantum circuits and, in particular, of reversing circuits. Likewise, our operator `reverse` is expected to produce the adjoint circuit of its input. Its definition rests on the choice of total endo-function (mapping each gate of arity k in a gate of arity k) that we denote with the symbol ‡, and that returns the adjoint of each gate. The circuit reversibility is implemented by rewiring gates in reverse order and, then, by replacing each gate by its adjoint. Rules (r0), (r1) and (r2) implement this policy.

The evaluation of circuits is characterized by Lemma 5.

Quantum Co-processor. The rule (m) evaluates the `dMeas` arguments and it uses the results of these evaluations to feed an external evaluating device: a quantum co-processor [69]. It is considered as a black-box that receiving a suitable evaluated circuit for the evaluation and its initialization, gives back a total measurement executed on the final state. In contrast with the other quantum programming languages based on the QRAM model, the co-processor of qPCF is not expected to record states between calls to it. Therefore, the decoherence issues are limited to its internal operations (cf. Section 2.1).

The circuit evaluation is described in the standard way by means of the Hilbert's spaces and von Neumann Measurements [31,45,33].

Remark 2 Following the standard axiomatization of quantum mechanics, usually proposed in terms of some *postulates* (see [77] for a simple formulation), we include in the language an explicit operator that represents the so called *von Neumann Total Measurement*, a special kind of *projective* measurements (see e.g. [31,45,33]). Informally, given a quantum state, total projective measure destroys superposition and returns a classical state, i.e. a sequence of classical bits $|b_1, b_2, \ldots, b_n\rangle$. The restriction to total measurement is not too restrictive because of the Principle of Deferred Measurement [45, p.186]:

> Measurements can always be moved from an intermediate stage of a quantum circuit to the end of the circuit; if the measurement results are used at any stage of the circuit then the classically controlled operations can be replaced by conditional quantum operations.

□

In accord with Section 3.1, we recall that $\mathtt{C} : \mathsf{circ}(n)$ aims to represent a quantum circuit operating on $n + 1$ qubits.

Definition 4 Let Circ^n be the set of (evaluated) circuits typed $\mathsf{circ}(\underline{\mathtt{n}})$ and let $N = 2^{n+1}$. Let $\mathcal{H}^N$ be a Hilbert space of finite dimension N, let $\{|\varphi_i\rangle\}$ be a orthonormal basis on $\mathcal{H}^N$ and let $\mathcal{H}^N \to \mathcal{H}^N$ be the set of unitary operators on $\mathcal{H}^N$.

1. $\mathrm{Hilb}^n : \mathrm{Circ}^n \to (\mathcal{H}^N \to \mathcal{H}^N)$ is a mapping from evaluated circuits into their corresponding algebraic operators defined as follows: $\mathrm{Hilb}^n(\mathtt{U}) ::= \mathbf{U}$ whenever $\mathtt{U}$ is typed $\mathsf{circ}(\underline{\mathtt{n}})$, so that $\mathbf{U} : \mathcal{H}^N \to \mathcal{H}^N$; $\mathrm{Hilb}^n(\mathtt{C}_0 \parallel \mathtt{C}_1) ::= \mathrm{Hilb}^{n_0}(\mathtt{C}_0) \otimes \mathrm{Hilb}^{n_1}(\mathtt{C}_1)$ whenever $\mathtt{C}_i$ is typed $\mathsf{circ}(\underline{\mathtt{n}}_i)$ and $n = n_0 + n_1$; $\mathrm{Hilb}^n(\mathtt{C}_0 \fatsemi \mathtt{C}_1) ::= \mathrm{Hilb}^n(\mathtt{C}_0) \circ \mathrm{Hilb}^n(\mathtt{C}_1)$.
2. A von Neumann measurement (see page 49 of [33], for instance) with respect to the basis of $\mathcal{H}^N$ and a given state
$$\psi = \sum_i \alpha_i \, |\varphi_i\rangle$$
outputs the i with probability $|\alpha_i|^2 \in (0, 1]$.
3. $\mathrm{circuitEval}^n : \mathsf{Nat} \times \mathrm{Circ}^n \to 2^{\mathbb{N}\times(0,1]}$ is a mapping from a pair (an initial state and a circuit) to a powerset of pairs (a vector of the basis and its probability) defined as follows:
$$\mathrm{circuitEval}^n(x, \mathtt{C}) = \left\{ (i, |\alpha_i|^2) \,\middle|\, \mathrm{Hilb}^n(\mathtt{C})(|\varphi_x\rangle) = \sum_i \alpha_i \, |\varphi_i\rangle \right\} .$$

□

The rule (m) describes a call to an external quantum co-processor that has to be able to evaluate quantum circuits. The co-processor is not assumed to store any quantum state between calls.

Lemma 5 *1. If $\vdash \mathtt{M} : \mathsf{circ}(\underline{\mathtt{n}})$ and $\mathtt{M} \Downarrow \mathtt{N}$ then $\vdash \mathtt{N} : \mathsf{circ}(\underline{\mathtt{n}})$ where $\mathtt{N}$ is a circuit.*
2. If $\mathtt{C}$ is a circuit such that $\vdash \mathtt{C} : \mathsf{circ}(\underline{\mathtt{n}})$ then $\mathrm{Hilb}^n(\mathtt{C})$ is well-defined.

Proof We recall that circuits are built on $\mathtt{U} \mid \fatsemi \mid \parallel$.

1. $\mathtt{M} \Downarrow \mathtt{N}$ means that there exists a derivation $\mathcal{D}$ concluding $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ where $\alpha > 0$. The proof is by induction on the last rule applied in $\mathcal{D}$. Rules (n), (s), (p),(sz), (op), (gt), (st) and (m) are not possible because of the typing hypothesis. Rules (β), (if_l), (if_r) and (Y) follow immediately by the induction hypothesis. The rule (u) is immediate. The rule (r_0) is straightforward, because $\ddagger$ (cf. rule (r_0)) is a total operator that respects arities. The rules (u'), (u''), (r_1), (r_2), (it) follow immediately by induction.
2. The proof follows by induction on the typing rules. By the previous point of this lemma, it is sufficient to check the rules $(C_1), (C_2), (C_3)$. □

Let $\underline{\mathtt{m}}\restriction_{\underline{\mathtt{k}}}$ be the restriction of the binary representation of $\underline{\mathtt{m}}$ to the first $\underline{\mathtt{k}}$ bits.

Example 6 (Creating and evaluating the EPR Circuit) Let $\mathtt{CNOT} : \mathsf{circ}(\underline{1})$ be the cnot gate and let $\mathtt{H} : \mathsf{circ}(\underline{0})$ and $\mathtt{I} : \mathsf{circ}(\underline{0})$ be the (unary) Hadamard and Identity gates, respectively. The following qPCF-term represents a simple circuit that generates the well-known EPR state [45]:

$$\mathtt{EPR} = (\mathtt{I} \parallel \mathtt{H}) \fatsemi \mathrm{CNOT} : \mathsf{circ}(\underline{1}) \quad .$$

Given a sequence $(b_1, \ldots, b_k)$ of bits, we write $\mathbf{n}(b_1, \ldots, b_k)$ to denote the corresponding numeral.
The evaluation of $\mathtt{dMeas}(\underline{0}, \mathtt{EPR})$ asks to the quantum co-processor the execution of the circuit $\mathtt{EPR}$ on the initial state $\mathbf{n}(00)$. This execution returns a fair superposition of $|00\rangle$ and $|11\rangle$, i.e. the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$. Since

$$\mathrm{circuitEval}(0, \mathtt{EPR}) = \{(\mathbf{n}(00), \frac{1}{2}), (\mathbf{n}(11), \frac{1}{2})\} \quad ,$$

it follows that both $\mathtt{dMeas}(\underline{0}, \mathtt{EPR}) \Downarrow^{\frac{1}{2}} \underline{0}$ and $\mathtt{dMeas}(\underline{0}, \mathtt{EPR}) \Downarrow^{\frac{1}{2}} \underline{3}$ (note that $\underline{0} = \mathbf{n}(0,0)$ and $\underline{3} = \mathbf{n}(1,1)$). □

It is well-known that quantum measures break the deterministic evolution of a quantum system. As a consequence, in presence of a measurement operator in a quantum language (equipped with an universal basis of quantum gates), or generally in a language equipped with a choice constructor, one necessarily loses confluence. For details on this argument see [11,19,5,6].

5.1 On the Probabilistic evaluation

The probabilistic evaluation of formal quantum programming languages is usually defined by means of a small-step operational semantics (e.g. [62,63, 48]) that formalizes the desired reduction strategy. Each reduction rule is labeled with the probability that the reduction fires. This essentially means that the reduction strategy is associated to a discrete time Markov chain, whose states are terms and stationary states are evaluated terms. Moreover, various probabilistic and non-deterministic extensions of PCF have been proposed in literature, see [14,21,22,25,26]. In order to stress the strict correspondence of qPCF with PCF, we define the evaluation in terms of a big-step operational semantics that hides the single-step details (unessential for our purposes): this is done by following a quite standard approach (see [34,12,53] for instances).

In all the above operational evaluations there might be many evaluations paths from $\mathtt{M}$ to $\mathtt{V}$, so that the probability has to be determined has the sum of the probabilities of such paths.

Definition 5 Let $\mathbf{D}(\mathtt{M}, \mathtt{V})$ denote the set of derivations proving $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ for any α; let $\mathbf{D}(\mathtt{M})$ denote the set of derivations proving $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ for any α and any $\mathtt{V}$; and, let $\mathsf{prb}(\mathcal{D})$ denotes the probability α whenever $\mathcal{D}$ concludes $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$. □

It is easy to see that $\sum_{\mathcal{D}_i \in \mathbf{D}(\mathtt{M},\mathtt{V})} \mathsf{prb}(\mathcal{D}_i) \leq 1$ and $\sum_{\mathcal{D}_i \in \mathbf{D}(\mathtt{M})} \mathsf{prb}(\mathcal{D}_i) \leq 1$ is the probability that the evaluation of $\mathtt{M}$ stops, while $1 - \sum_{\mathcal{D}_i \in \mathbf{D}(\mathtt{M})} \mathsf{prb}(\mathcal{D}_i)$ is the probability that the evaluation of $\mathtt{M}$ diverges.

Example 7 – Let Ω^{Nat} be $\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{Nat}}.\mathtt{x})$. It is clear that Ω^{Nat} is a closed term typed Nat such that $\Omega^{\mathsf{Nat}} \Uparrow$. Therefore, $\mathbf{D}(\Omega^{\mathsf{Nat}}, \underline{\mathtt{n}}) = \emptyset$ and $\mathbf{D}(\Omega^{\mathsf{Nat}}) = \emptyset$.

– Let $\mathtt{EPR}$ be the term defined in the Example 6.
Let $\mathtt{M}_8$ be $\mathtt{if}\,(\mathtt{dMeas}(\underline{0}, \mathtt{EPR}))\,\underline{8}\,(\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{Nat}}.\mathtt{x}))$ then $\mathbf{D}(\mathtt{M}_8)$ contains only the derivation concluding $\mathtt{M}_8 \Downarrow^{\frac{1}{2}} \underline{8}$.

– Let $\mathtt{M}_8^{\infty}$ be $\mathtt{Y}(\lambda \mathtt{x}^{\mathsf{Nat}}.\,\mathtt{if}\,(\mathtt{dMeas}(\underline{0}, \mathtt{EPR}))\,\underline{8}\,\mathtt{x})$ then $\mathbf{D}(\mathtt{M}_8^{\infty})$ contains denumerable derivations. More precisely, $\mathbf{D}(\mathtt{M}_8^{\infty})$ contains a derivation concluding $\mathtt{M}_8^{\infty} \Downarrow^{(\frac{1}{2})^k} \underline{8}$ for each $k \geq 1$. Thus, the limit of the sum of these probabilities is 1. □

We can define the observational equivalence of qPCF by following [21].

Definition 6 Let $\mathtt{M}$ and $\mathtt{M}$ be closed terms of the same type. They are observationally equivalent whenever, for each context $C[.]$ it holds that: (i) if $B \vdash C[\mathtt{M}] : \mathsf{Nat}$ then $B \vdash C[\mathtt{N}] : \mathsf{Nat}$; (ii) if $B \vdash C[\mathtt{N}] : \mathsf{Nat}$ then $B \vdash C[\mathtt{M}] : \mathsf{Nat}$; and, (iii) $\sum_{\mathcal{D}_i \in \mathbf{D}(C[\mathtt{M}],\underline{0})} \mathsf{prb}(\mathcal{D}_i) = \sum_{\mathcal{D}_i \in \mathbf{D}(C[\mathtt{N}],\underline{0})} \mathsf{prb}(\mathcal{D}_i)$. □

As noted in [21], the last constraint can be replaced by, for each numeral $\underline{\mathtt{k}}$, $\sum_{\mathcal{D}_i \in \mathbf{D}(C[\mathtt{M}],\underline{\mathtt{k}})} \mathsf{prb}(\mathcal{D}_i) = \sum_{\mathcal{D}_i \in \mathbf{D}(C[\mathtt{N}],\underline{\mathtt{k}})} \mathsf{prb}(\mathcal{D}_i)$. The operational equivalence is defined on closed terms of type Nat because the operational differences in the other types can be traced back to Nat (while the reverse can be easily proved be false). Anyway, in this paper we do not plan to further study the observational equivalence of qPCF, therefore in the following, we sometimes use $\mathtt{M} \Downarrow \mathtt{V}$ (i.e. $\mathtt{M} \Downarrow^{\alpha} \mathtt{V}$ for some $0 < \alpha \leq 1$).

5.2 Evaluation properties

To prove properties about the evaluation mechanisms of qPCF, we have to consider infinite ground types (viz. Nat, Idx, $\mathsf{circ}(\underline{\mathtt{n}})$) and we have to unravel the mutual relationships that hold between syntactic classes (cf. Example 2). A first goal is to prove that Idx picks up a class of terms which is expected to be endowed with an always terminating evaluation, i.e. always normalizing.

Simply minded arguments do not work for showing the strong normalization property of the typed lambda-calculi: reduction increases the size of terms, which precludes an induction on their size, and preserves their types, which seems to preclude an induction on types. We follow the well-known Tait's idea for proofs of strong normalizations based on a suitable predicate, see [52] for instance. More precisely, we prove our property by adapting the computability predicate given in [23, 57, 49] for PCF.

Definition 7 The predicate $Comp(B, \mathtt{M}, \sigma)$ holds whenever $B \vdash \mathtt{M} : \sigma$ and one of the following cases is satisfied:

1. $B = \emptyset$, $\sigma = \mathsf{Nat}$;
2. $B = \emptyset$, $\sigma = \mathsf{Idx}$ and $\mathtt{M} \Downarrow^1 \underline{\mathtt{n}}$, for some $\underline{\mathtt{n}}$;
3. $B = \emptyset$, $\sigma = \mathsf{circ}(\mathtt{E})$ and $Comp(\emptyset, \mathtt{E}, \mathsf{Idx})$;
4. $B = \emptyset$, $\sigma = \Pi \mathtt{x}^{\mu}.\tau$ and $Comp(\emptyset, \mathtt{MN}, \tau[\mathtt{N}/\mathtt{x}])$, for all $Comp(\emptyset, \mathtt{N}, \mu)$;
5. $B = \{\mathtt{x} : \nu\} \cup B'$ implies $Comp(B'[\mathtt{N}/\mathtt{x}], \mathtt{M}[\mathtt{N}/\mathtt{x}], \sigma[\mathtt{N}/\mathtt{x}])$ for all $Comp(\emptyset, \mathtt{N}, \nu)$.

□

Let us assume $B \vdash \mathtt{M} : \sigma$ holds. Focus on ground types: (i) *Comp* always holds for Nat; (ii) *Comp* holds for Idx when the term evaluation is terminating; (iii)*Comp* holds for $\mathsf{circ}(\mathtt{E})$ independently from the typed term, whenever $\mathtt{E}$ is well typed and its evaluation is terminating. The remaining cases ensure that *Comp* hold: (i) for all well-typed closing substitution of $\mathtt{M}$ and, (ii) for all well-typed application.

Notation 1 *Let $\mathtt{x}_1, ..., \mathtt{x}_n$ be variables, let $\mathtt{N}_1, ..., \mathtt{N}_n$ be terms and let $1 \leq j \leq k \leq n$. We write $\mathtt{Q}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_j^k$ as a shortening for $((\mathtt{Q}[\mathtt{N}_j/\mathtt{x}_j]) \cdots [\mathtt{N}_k/\mathtt{x}_k])$ when $\mathtt{Q}$ is a term. We write $\sigma\overrightarrow{[\mathtt{N}/\mathtt{x}]}_j^k$ as a shortening for $((\sigma[\mathtt{N}_j/\mathtt{x}_j]) \cdots [\mathtt{N}_k/\mathtt{x}_k])$ when σ is a type. As expected $k < j$ means no substitution.* □

Lemma 6

1. *Let $\kappa = \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\mathsf{Nat}$; $Comp(B, \mathtt{M}, \kappa)$ iff $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Nat}$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^{j-1})$ where $j \leq n$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n\overrightarrow{[\mathtt{P}/\mathtt{z}]}_1^{i-1})$ where $i \leq m$.*
2. *Let $\kappa = \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\mathsf{Idx}$; $Comp(B, \mathtt{M}, \kappa)$ iff $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $Comp(\emptyset, \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m, \mathsf{Idx})$, for all $\mathtt{N}_j$ s.t. $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^{j-1})$ where $j \leq n$, for all $\mathtt{P}_i$ s.t. $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n\overrightarrow{[\mathtt{P}/\mathtt{z}]}_1^{i-1})$ where $i \leq m$.*
3. *Let $\kappa = \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\mathsf{circ}(\mathtt{E})$. $Comp(B, \mathtt{M}, \kappa)$ iff $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{circ}(\mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n\overrightarrow{[\mathtt{P}/\mathtt{z}]}_1^m)$ and $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n\overrightarrow{[\mathtt{P}/\mathtt{z}]}_1^m, \mathsf{Idx})$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^{j-1})$ where $j \leq n$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n\overrightarrow{[\mathtt{P}/\mathtt{z}]}_1^{i-1})$ where $i \leq m$.*

Proof 1. First, we prove by induction on n that $Comp(B, \mathtt{M}, \kappa)$ if and only if $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $Comp(\emptyset, \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n, \kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n)$ for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^{j-1})$ where $j \leq n$. Then, we conclude by induction on m.

2. The proof is similar to the previous one. It is worth to emphasize that, in the statement, we write $Comp(\emptyset, \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m, \mathsf{Idx})$ as a shortening for $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Idx}$ and $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]}_1^n \mathtt{P}_1 \ldots \mathtt{P}_m \Downarrow^1 \underline{\mathtt{n}}$, for some $\underline{\mathtt{n}}$.
3. Similar to that of 1. □

Lemma 6 provides an alternative charactrerization of *Comp*, because it is easy to check that each type has the shape $\Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\gamma$ for a unique $m \in \mathbb{N}$ and $\gamma \in \{\mathsf{Nat}, \mathsf{Idx}, \mathsf{circ}(\mathtt{E})\}$.

Theorem 1 *If $B \vdash \mathtt{M} : \kappa$ then $Comp(B, \mathtt{M}, \kappa)$.*

Proof The proof is by induction on the derivation $\mathcal{D}$ proving $B \vdash \mathtt{M} : \kappa$.

- Rule (P_0). Let $\mathtt{M} = \mathtt{x}_k$ for some $k \leq n$ and $\kappa = \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\gamma$ where $\gamma \in \{\mathsf{Nat}, \mathsf{Idx}, \mathsf{circ}(\mathtt{E})\}$. If $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ then we aim to prove that $Comp(\emptyset, \mathtt{x}_k\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, \kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$ for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$. Since $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ implies $\vdash \mathtt{N}_j : \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}}$, we are sure that $\mathtt{N}_j$ and $\nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}}$ do not contain free variables. Therefore $Comp(\emptyset, \mathtt{x}_k\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, \kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}) = Comp(\emptyset, \mathtt{N}_k, \kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$. Since (P_0) is the last rule used in $\mathcal{D}$, then its final typing has shape $B'' \cup \{\mathtt{x}_k : \nu_k\} \vdash \mathtt{x}_k : \kappa$ where $\nu_k = \kappa$. Then, $Comp(\emptyset, \mathtt{N}_k, \kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}) = Comp(\emptyset, \mathtt{N}_k, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ which is assumed to hold.
- Rule (P_1). Let $\mathtt{M} = \lambda \mathtt{x}^\sigma.\mathtt{Q}$ and $\kappa = \Pi \mathtt{x}^\sigma.\Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\gamma$ where γ is a ground type. By induction, $Comp(B \cup \{\mathtt{x} : \sigma\}, \mathtt{Q}, \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\gamma)$ holds.
  - Let $\gamma = \mathsf{Nat}$; by applying the Lemma 6 to the induction hypothesis we know that $B \cup \{\mathtt{x} : \sigma\} = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Nat}$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}})$ where $i \leq m$. Let $k \leq n$ be such that $\mathtt{x} : \sigma$ is $\mathtt{x}_k : \nu :_k$. So $\vdash (\lambda \mathtt{x}^\sigma.\mathtt{Q})\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{k-1}}\overrightarrow{[\mathtt{N}/\mathtt{x}]_{k+1}^n}\mathtt{N}_k\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Nat}$ follows by Lemma 3, since $\mathtt{N}_k$ is closed.
  - Let $\gamma = \mathsf{Idx}$; by applying the Lemma 6 to the induction hypothesis we know that if we assume $B \cup \{\mathtt{x} : \sigma\} = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ then $Comp(\emptyset, \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\mathtt{P}_1 \ldots \mathtt{P}_m, \mathsf{Idx})$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}})$ where $i \leq m$. More explicitly, $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Idx}$ and $\mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\mathtt{P}_1 \ldots \mathtt{P}_m \Downarrow^1 \underline{\mathtt{n}}$, for some $\underline{\mathtt{n}}$. Let $k \leq n$ be such that $\mathtt{x} : \sigma$ is $\mathtt{x}_k : \nu :_k$. Since $\mathtt{N}_k$ is closed, we conclude by Lemma 3 and the evaluation rule β.
  - Let $\gamma = \mathsf{circ}(\mathtt{E})$; by applying the Lemma 6 to the induction hypothesis we know that $B \cup \{\mathtt{x} : \sigma\} = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash \mathtt{M}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{circ}(\mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m})$ and $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m}, \mathsf{Idx})$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, for all $\mathtt{P}_i$ s.t. $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}})$ where $i \leq m$. Let $k \leq n$ be such that $\mathtt{x} : \sigma$ is $\mathtt{x}_k : \nu :_k$. Since $Comp(\emptyset, \mathtt{E}[\mathtt{P}/\mathtt{x}, \vec{\mathtt{N}}/\vec{\mathtt{x}}, \mathtt{P}_1/\mathtt{z}_1, ..., \mathtt{P}_m/\mathtt{z}_m], \mathsf{Idx})$ already holds, it remains to prove that $\vdash (\lambda \mathtt{x}.\mathtt{Q}))\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{k-1}}\overrightarrow{[\mathtt{N}/\mathtt{x}]_{k+1}^n}\mathtt{N}_k\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{circ}(\mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m})$. This latter follows by Lemma 3 because $\mathtt{N}_k$ is closed. Thus we conclude by Lemma 6.
- Rule (P_2). Let $\mathtt{M} = \mathtt{PQ}$ and $\kappa = \Pi \mathtt{z}_1^{\tau_1}....\Pi \mathtt{z}_m^{\tau_m}.\gamma$ where γ is a ground type. Let σ be the type such that $Comp(B, \mathtt{P}, \Pi \mathtt{y}^\sigma.\kappa)$ and $Comp(B, \mathtt{Q}, \sigma)$ hold by induction. Thus, if $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ then, we have $Comp(\emptyset, \mathtt{P}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, (\Pi \mathtt{y}^\sigma.\kappa)\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$ for all $\mathtt{N}_j$ s.t. $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$. But $Comp(\emptyset, \mathtt{Q}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, \sigma\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$ holds too. Thus, by Definition 7, we

have $Comp(\emptyset, (\mathtt{P}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})\mathtt{Q}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, (\kappa\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})[\mathtt{Q}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}/\mathtt{y}])$ that, in it is turn, implies $Comp(\emptyset, (\mathtt{PQ})\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}, (\kappa[\mathtt{Q}/\mathtt{y}])\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$.

- Rule (P_3). By Lemma 6, we have to prove that $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash (\mathtt{pred}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})\mathtt{P} : \mathsf{Nat}$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, and $\mathtt{P}$ such that $Comp(\emptyset, \mathtt{P}, \mathsf{Nat})$. Namely, $\vdash \mathtt{pred}\,\mathtt{P} : \mathsf{Nat}$ has to hold whenever $\vdash \mathtt{P} : \mathsf{Nat}$ holds. This is true by rules (P_2) and (P_3).
- Rules $(P_4), (P_5), (B_1), (B_2)$ are similar to the previous case.
- Rule (P_5'). Let $\mathtt{M} = \mathtt{if}$ and $\kappa = \Pi\mathtt{z}_1^{\mathsf{Nat}}.\Pi\mathtt{z}_2^{\mathsf{circ}(E)}.\Pi\mathtt{z}_3^{\mathsf{circ}(E)}.\,\mathsf{circ}(E)$. By Lemma 6, we have to prove that $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash (\mathtt{if}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})\mathtt{P}_1\mathtt{P}_2\mathtt{P}_3 : \mathsf{circ}(\mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^3})$ and $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^3}, \mathsf{Idx})$ for $Comp(\emptyset, \mathtt{P}_1, \mathsf{Nat})$, $Comp(\emptyset, \mathtt{P}_2, \mathsf{circ}(E\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{P}_1/\mathtt{z}_1]))$, $Comp(\emptyset, \mathtt{P}_3, \mathsf{circ}(E\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^3}))$ and for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$. The typing follows by rule (P_2) and (P_5'), since the comp-hypothesis implies that $\vdash \mathtt{P}_1 : \mathsf{Nat}$, $\vdash \mathtt{P}_2 : \mathsf{circ}(E\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{P}_1/\mathtt{z}_1])$ and $\vdash \mathtt{P}_3 : \mathsf{circ}(E\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{P}_1/\mathtt{z}_1][\mathtt{P}_2/\mathtt{z}_2])$. Moreover, by induction on $B \vdash \mathtt{E} : \mathsf{Idx}$ we have $Comp(B, \mathtt{E}, \mathsf{Idx})$, therefore $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{P}_1/\mathtt{z}_1][\mathtt{P}_2/\mathtt{z}_2][\mathtt{P}_3/\mathtt{z}_3], \mathsf{Idx})$ can be immediately concluded.
- Rule (P_6). Let $\mathtt{M} = \mathtt{Y}$ and $\kappa = \Pi\mathtt{y}^{(\sigma\to\sigma)}.\sigma$ such that $\sigma = \Pi\mathtt{z}_1^{\tau_1}....\Pi\mathtt{z}_m^{\tau_m}.\gamma$ where $\gamma \in \{\mathsf{Nat}, \mathsf{circ}(\mathtt{E})\}$.
  - The proof of $\gamma = \mathsf{Nat}$ is similar to the proof of the rule (P_3). By Lemma 6, we have to prove that $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies $\vdash (\mathtt{Y}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})\mathtt{Q}\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{Nat}$, for all $\mathtt{N}_j$ such that $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, for $\mathtt{Q}$ such that $Comp(\emptyset, \mathtt{Q}, (\sigma \to \sigma)\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}})$ where $i \leq m$. Namely, we have to prove that $\vdash \mathtt{YQP}_1 \ldots \mathtt{P}_m : \mathsf{Nat}$ whenever $\vdash \mathtt{Q} : (\sigma \to \sigma)\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}$ and $\vdash \mathtt{P}_i : \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}}$. This is true by rules (P_2) and (P_6).
  - The proof of $\gamma = \mathsf{circ}(\mathtt{E})$ is similar to the proof of the rule (P_5').By Lemma 6, we have to prove that $B = \{\mathtt{x}_1 : \nu_1, ..., \mathtt{x}_n : \nu_n\}$ implies

$$\vdash (\mathtt{Y}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})\mathtt{Q}\mathtt{P}_1 \ldots \mathtt{P}_m : \mathsf{circ}(\mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m}) \qquad \text{and}$$

    $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m}, \mathsf{Idx})$ for all $\mathtt{N}_j$ s.t. $Comp(\emptyset, \mathtt{N}_j, \nu_j\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^{j-1}})$ where $j \leq n$, for $\mathtt{Q}$ such that $Comp(\emptyset, \mathtt{Q}, (\sigma \to \sigma)\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n})$, for all $\mathtt{P}_i$ such that $Comp(\emptyset, \mathtt{P}_i, \tau_i\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^{i-1}})$ where $i \leq m$. The typing requirement is similar to that of the previous case. Moreover, it is easy to check that $\vDash B \cup \{\mathtt{x} : \sigma\}$ requires that $B \vdash \mathtt{E} : \mathsf{Idx}$ is in the premises of (P_6). Thus $Comp(B, \mathtt{E}, \mathsf{Idx})$ follows by hypothesis, and $Comp(\emptyset, \mathtt{E}\overrightarrow{[\mathtt{N}/\mathtt{x}]_1^n}[\mathtt{Q}/\mathtt{y}]\overrightarrow{[\mathtt{P}/\mathtt{z}]_1^m}, \mathsf{Idx})$ can be concluded.
- Rule (I_0). The proof follows by induction, it suffices to apply (I_0) to obtain the typing in the induction hypothesis.
- Rule (I_1). Immediate by the evaluation rule (n).
- Rule (I_2). The proof follows by induction, because we assume that $\odot$ is a (generic) total operator.
- Rule (I_3). The proof follows by induction and by the evaluation rule (sz).

- Rule (C_1). Immediately $B \vdash \mathtt{U} : \mathsf{circ}(\underline{\mathtt{k}})$, $B \vdash \underline{\mathtt{k}} : \mathsf{Idx}$ and $\underline{\mathtt{k}} \Downarrow^1 \underline{\mathtt{k}}$ hold, thus $Comp(B, \mathtt{U}, \mathsf{circ}(\underline{\mathtt{k}}))$.
- Rules $(C_2), (C_3), (C_4), (C_5)$. The proofs are similar to that of the rule (P_5').
- Rule (M). The proof is similar to that of the rule (P_5). □

Comp has been defined in order to obtain the next corollary that states the strong normalization of closed term typed Nat and the that its evaluation provides a unique result.

Corollary 1 (Idx-normalization) *Let* $\vdash \mathtt{E} : \mathit{Idx}$.

1. $\mathtt{E} \Downarrow^1 \underline{\mathtt{n}}$, *for some* $\underline{\mathtt{n}}$.
2. *If* $\mathsf{circ}(\mathtt{E})$ *occurs in a valid type derivation then* $\mathtt{E} \Downarrow^1 \underline{\mathtt{n}}$, *for some* $\underline{\mathtt{n}}$.

Proof 1. $\vdash \mathtt{E} : \mathsf{Idx}$ implies $Comp(\emptyset, \mathtt{E}, \mathsf{Idx})$ by Theorem 1; so $\mathtt{E} \Downarrow^1 \underline{\mathtt{n}}$ by Definition 7.

2. Let $\mathcal{D}$ be a valid type derivation. By induction on $\mathcal{D}$ we can prove that if $\mathsf{circ}(\mathtt{E}')$ occurs in the conclusion of the derivation then $B \vdash \mathtt{E}' : \mathsf{Idx}$ is required for some B. Thus, since we assumed $\vdash \mathtt{E} : \mathsf{Idx}$, we conclude by the previous point. □

We can now focus on standard programming properties (see [56]). A first property of a paradigmatic programming language as qPCF is preservation, i.e. if a well-typed term takes a step of evaluation then the resulting term is also well typed. A second property expected for a programming language is progress: well-typed terms evaluation does not get stuck. Roughly, a term P gets stuck whenever the evaluation of P ends in a normal form, which is not a value. It is easy to see that our evaluation cannot stuck.

Corollary 2 $\vdash \mathtt{M} : \mathit{Idx}$ *and* $\mathtt{M} \Downarrow^1 \mathtt{N}$ *then* $\vdash \mathtt{N} : \mathit{Idx}$.
$\vdash \mathtt{M} : \mathit{Idx}$ *and* $\mathtt{M} \Downarrow^1 \mathtt{N}$ *then* N *is a numeral* $\underline{\mathtt{n}}$.

Proof The proof follows by Theorem 1. □

For remaining ground types we prove our properties together.

Theorem 2 *1.* $\vdash \mathtt{M} : \mathit{Nat}$ *and* $\mathtt{M} \Downarrow \mathtt{N}$ *then* $\vdash \mathtt{N} : \mathit{Nat}$ *and* N *is a numeral.*
2. If $\vdash M : \mathsf{circ}(\mathtt{E})$ *and* $\mathtt{M} \Downarrow \mathtt{N}$ *then* $\vdash N : \mathsf{circ}(\mathtt{E})$ *where* N *is a circuit; moreover,* $\mathtt{E} \Downarrow \underline{\mathtt{m}}$ *for some* $\underline{\mathtt{m}}$.

Proof We recall that $\mathtt{M} \Downarrow \mathtt{N}$ means that there exists a derivation $\mathcal{D}$ concluding $\mathtt{M} \Downarrow^\alpha \mathtt{V}$ where $\alpha > 0$. The statements are proved by induction on the derivation proving $\mathtt{M} \Downarrow \mathtt{N}$ by considering the typing hypothesis.

1. The proof is done by induction on the derivation proving $\mathtt{M} \Downarrow \mathtt{N}$. (n) is trivial. If the last applied evaluation rule is one between (s), (p), (β), (if_l), (if_r), (Y), (gt), (st) then the proof follows by induction. We remark that the last rules can be (sz) or (op) because its result can be typed Nat via the typing rule (x_3): luckily, in both cases the proof is still immediate by Theorem 1. The cases (u), (u'), (u''), (r_0), (r_1), (r_2), (it) are not possible, because the typing hypothesis in the statement excludes them. The case (m) easily follows by induction and the definition of circuitEval.
2. $\mathtt{E} \Downarrow \underline{\mathtt{m}}$ follows by Corollary 2. The proof is similar to that of Lemma 5. □

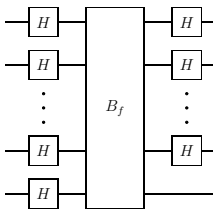
6 Examples

In this section we propose some examples of quantum circuit families implementing interesting algorithms. As previously done, given a sequence $(b_1, \ldots, b_k)$ of bits, we write $\mathbf{n}(b_1, \ldots, b_k)$ to denote the numeral that represents it.

Example 8 (Deutsch-Jozsa) We aim to program the Deutsch's algorithm [46] in qPCF. This can be done by a term that represents the entire (infinite) quantum circuit family.

The "basic case" of Deutsch's problem can be formulated as follows. Given a block box B_f implementing some function $f : \{0,1\} \to \{0,1\}$, determine whether f is constant or balanced. The classical computation to determine whether f is constant or balanced is very simple: one computes $f(0)$ and $f(1)$, and then check if $f(0) = f(1)$. This requires two different calls to B_f. Deutsch showed how to achieve this result with a single call to B_f: Deutsch's algorithm exploits *quantum parallelism* phenomenon that allows to evaluate a function $f(x)$ for different values x at the same time.

The problem can be generalized considering a function $f : \{0,1\}^n \to \{0,1\}$ which acts on many input bits. This yields the n-bit generalization of Deutsch's algorithm, known as the Deutsch-Josza algorithm. The following picture represents the circuit, up to the last, measurement phase.



When fed with a classical input state of the form $|0\ldots01\rangle$, the output measurement of the first $n-1$ bits reveals if the function f is constant or not. If all $n-1$ measurement results are 0, we can conclude that the function was constant. Otherwise, if at least one of the measurement outcomes is 1, we conclude that the function was balanced. See [45] for more details.

Let $\mathtt{H} : \mathsf{circ}(\underline{0})$ and $\mathtt{I} : \mathsf{circ}(\underline{0})$ be the (unary) Hadamard and Identity gates respectively. Suppose $\mathsf{M}^{B_f} : \mathsf{circ}(\underline{\mathsf{n}})$ is given for some n such that $\mathsf{M}^{B_f} \Downarrow \mathsf{U}_f$ where $\mathsf{U}_f : \mathsf{circ}(\underline{\mathsf{n}})$ is the qPCF-circuit that represents the the black-box function f having arity $n+1$.

Observe that $\lambda \mathsf{x}^{\mathsf{Idx}}.\,\mathtt{iter}\ \mathtt{xHH} : \Pi \mathsf{x}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathsf{x})$ generates $x+1$ parallel copies of Hadamard gates $\mathtt{H}$, and $\lambda \mathsf{x}^{\mathsf{Idx}}.\,\mathtt{iter}\ \mathtt{xIH} : \Pi \mathsf{x}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathsf{x})$ concatenates in parallel x copies of Hadamard gates $\mathtt{H}$ and one copy of the identity gate $\mathtt{I}$. Thus the parametric measurement-free Deutsch-Jozsa circuit can be defined as

$$\mathsf{DJ}^{-} = \lambda \mathsf{x}^{\mathsf{Idx}}.\lambda \mathsf{y}^{\mathsf{circ}(\mathsf{x})}.((\mathtt{iter}\ \mathsf{x}\,\mathtt{H}\,\mathtt{H})\ ⨾\ \mathsf{y})\ ⨾ (\mathtt{iter}\ \mathsf{x}\,\mathtt{I}\,\mathtt{H}) : \sigma$$

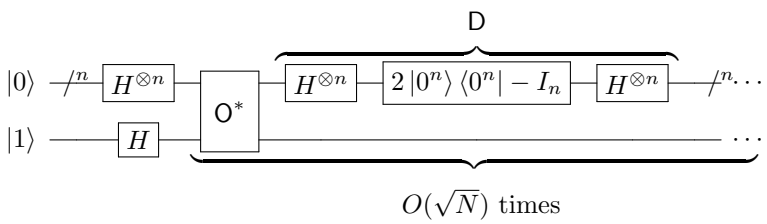
where $\sigma = \Pi \mathsf{x}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathsf{x}) \to \mathsf{circ}(\mathsf{x})$.

The last phase is performed by the operator `dMeas`, suitably fed with the representation of the classical input state, i.e. $\mathbf{n}(\underbrace{0\ldots0}_{n}1)$. The evaluation yields $\texttt{dMeas}(\mathbf{n}(\underbrace{0\ldots0}_{n}1), \mathsf{DJ}^{-}\underline{\mathtt{n}}\mathsf{M}^{B_f}) \Downarrow^1 \underline{\mathtt{m}}$ where $\underline{\mathtt{m}}$ is the result (with probability 1).

It is straightforward to make parametric the above term. It suffices to replace $\underline{\mathtt{n}}$ with the variable $\mathtt{n}^{\mathsf{Idx}}$, to replace the black-box with a variable $\mathsf{b}^{\mathsf{circ(n)}}$ so that, the resulting term is typed $\Pi\mathtt{n}^{\mathsf{Idx}}.\Pi\mathsf{b}^{\mathsf{circ(n)}}.\mathsf{Nat}$, or more simply $\Pi\mathtt{n}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathtt{n}) \to \mathsf{Nat}$. □

Example 9 (Grover's searching algorithm) We provide the qPCF encoding of the circuit that implements Grover's searching algorithm [45]. More precisely, Grover's algorithm solves the problem of a search in a given a set $X = \{x_1, x_2, \ldots, x_N\}$ of $N = 2^{n+1}$ elements. Given a boolean function $f : X \to \{0, 1\}$, the target is to find an element x^* in X such that $f(x^*) = 1$. With a classical circuits, one cannot do better than performing a linear number of queries to find the target element. Grover's quantum solves algorithm the search in $O(\sqrt{N})$. The main idea of Grover's searching algorithm is to make a fair superposition of input elements and, then, to iterate $O(\sqrt{N})$ time a subroutine applying an "oracle gate" O^* that encodes the function f and a suitable diffusion operator D. After each application of the subroutine, it is possible to show that the probability to measure the target element x^* increases: the amplitudes α^* goes up by more than $\frac{1}{\sqrt{N}}$. This meas that, after $O(\sqrt{N})$ repetitions, α^* is very close to 1, and thus a final, total measurement, yields x^* with a negligible error.

Some slightly different circuit implementations have been proposed in literature [33, 43], we follow [45]:



Let O^* be the circuit that represents the function mapping $|x, q\rangle$ to $|x, q \oplus f(x)\rangle$. Look at the figure above: the lower wire (initialized to $|1\rangle$) feeds O^* with the state $(|0\rangle - |1\rangle)/\sqrt{2}$ while the other ones (initialized to $\underbrace{|0\ldots0\rangle}_{n}$) feed O^* with the fair superposition of all possible inputs. Therefore, O^* maps $|x\rangle \left|\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right\rangle$ on $(-1)^{f(x)}|x\rangle \left|\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right\rangle$; if we neglect the (bottom) ancillary wire then $|x\rangle$ is mapped on $(-1)^{f(x)}|x\rangle$ producing a key flipping behavior.

Let $\mathsf{M}^{O^*} : \mathsf{circ}(\underline{\mathtt{n}})$ be a qPCF term such that $\mathsf{M}^{O^*} \Downarrow^1 \mathsf{O}^*$ (we recall that $\mathsf{circ}(\underline{\mathtt{n}})$ is the type for $n+1$ wires.) Let $\mathsf{M}^D : \mathsf{circ}(\mathtt{x})$ be such that $(\lambda \mathtt{x}^{\mathsf{Idx}}.\mathsf{M}^D)(\mathtt{pred}\,\underline{\mathtt{n}}) \Downarrow \mathsf{D}$ where D is the circuit that represents the diffusion operator that magnifies amplitudes: D maps the superposition $\sum_{i=0}^{N-1} \alpha_i V_i$ in $\sum_{i=0}^{N-1}(2m-\alpha_i)V_i$ where $m = (\frac{1}{N})\sum_{i=0}^{N-1}\alpha_i$ is the average of all amplitudes (see [66]). Some interesting decomposition of D in terms of smaller quantum circuits can be found in [18, 24].

Let $\sqrt{\ }$: $\mathsf{Nat} \to \mathsf{Nat}$ be a qPCF-term calculating (integer approximation of) the square root and let $\mathsf{M}^{\times}_{\mathsf{seq}} : \mathsf{circ}(\mathtt{x}) \to \mathsf{Nat} \to \mathsf{circ}(\mathtt{x})$ be $\mathsf{M}^{\mathsf{A}}_{\mathsf{seq}}\,\mathtt{x}$, i.e. the application of the term defined in Example 4 to $\mathtt{x}$. The term that implements the core of Grover's algorithm is:

$$\mathsf{M}_{\mathsf{G}} = \lambda \mathtt{x}^{\mathsf{Idx}}.\mathtt{z}^{\mathsf{circ}(\mathtt{x})}.(\mathtt{iter}\ \mathtt{x}\,\mathtt{H}\,\mathtt{H}) \fatsemi (\mathsf{M}^{\times}_{\mathsf{seq}}(\mathtt{z} \fatsemi (\mathsf{M}^D \parallel \mathtt{I}))\,\sqrt{\mathtt{x}}) : \sigma$$

where $\sigma = \Pi \mathtt{x}^{\mathsf{Idx}}.\,\mathsf{circ}(\mathtt{x}) \to \mathsf{circ}(\mathtt{x})$. A measurement of M_{G} applied to a suitable arity $\underline{\mathtt{n}}$ and a suitable oracle operator O^* allows to execute Grover's algorithm.

For example, we can consider a search in a space of $2^3 = 8$ states looking for $|011\rangle$. Grover's circuit take as input the (classical) state $|0001\rangle$.

It is easy to verify that $\mathsf{G}_3 = \mathtt{dMeas}(\mathbf{n}(0001), \mathsf{M}_{\mathsf{G}}(\underline{3})(\mathsf{M}^{\mathsf{O}^*}))$ solves the search problem with a bounded error. In particular, $\mathbf{D}(\mathsf{G}_3, \mathbf{n}(0110))$ is a set including just one derivation, namely the derivation concluding $\mathsf{G}_3 \Downarrow^{\alpha} \mathbf{n}(0110)$ such that $\alpha = 0,945$. Thus, the evaluation of G_3 gives the right results with a 94,5% of probability. See [67] for the details. □

7 Related Work

In this section, we sketch the state of the art of quantum programming languages by focusing on calculi related to qPCF.

After a first formal attempt by Maymin [40], Selinger rigorously defined a first-order quantum functional language [61]. Subsequently the author, in a joint work with Valiron [62], defined a quantum λ-calculus, (that we dub λ_{sv} in what follows), with classical control and explicitly based on the QRAM architecture [36]. λ_{sv} rests on unitary transformations on quantum states and an explicit measurement operator which allows the program to observe the value of one quantum bit. The separation between data and control is explicit. The type system of λ_{sv} avoids run time errors, enjoys good properties such as subject reduction, progress, and safety and is based on the affine intuitionistic Linear Logic. This permits a fine control over the linearity of the system, by distinguishing between duplicable and non-duplicable resources. λ_{sv} can be seen as the departing point of several investigations. On the foundational side, see for instance [13, 10]. On the semantic side, we just cite [63, 30, 48].

qPCF follows the slogan quantum data&classical control ("*qd&cc*") proposed for λ_{sv} in [62], albeit its typing system does not explicitly include linear/exponential types.

Looking for implementation-oriented proposal, the most interesting reality is Quipper, an embedded, scalable functional quantum programming language. Quipper is essentially a circuit description language: circuits can be created, manipulated, evaluated in a mixture of procedural and declarative programming styles. The most important quantum algorithms can be easily encoded thanks to a number of programming tools, macros, and extensive libraries of quantum functions.

Quipper is based on the lambda calculus with classical control proposed in [62], and this relationship has been partially explained in [60] by means of the calculus Proto-Quipper. Reduction rules are defined between configurations as in [62, 77] and, since the calculus is measurement free, they are totally deterministic (likewise to [13, 10, 77]). Proto-Quipper type system is based on intuitionistic linear logic (with both additive and multiplicative modalities) plus a type for circuits. A more recent version of Proto-Quipper, called Proto-QuipperM, has been defined by Selinger and Rios in [59] together with an interesting categorical semantics.

qPCF follows the trend started by Quipper about the management of quantum circuits as prominent classical data. However, it differs from Proto-Quipper formalization, at least, for the absence of quantum states and linear/exponential types, but also for the presence of dependent types.

Another recent quantum language is QWire introduced in [55]. Also QWire can be seen as a language for circuit manipulation. It rests on the QRAM model and aims at separating classical and quantum part of the computation. QWire is a very simple and manageable linear language for the definition of quantum circuit that, through a sophisticate interface, can be treated as a "quantum plugin" for an host classical language. This is reflected by the type system, inspired to Benton's LNL Logic that partitions the exponential data into a purely linear fragment and a purely non-linear fragment connected via a categorical adjunction. Circuits are treated as classical data in the host language through a clever interface based on a box-unbox mechanism. Moreover, the authors show that QWire is able to deal with a depend-type based host language via an interesting example. Albeit they have been developed independently, QWire and qPCF have a many aspects in common, and both move the focus from states to circuits. Thus, QWire deserves a direct comparison with qPCF.
qPCF bans linear typing, while QWire use it to provide a very helpful support for a quantum circuit language description that borrows the best features from the "hardware circuit description" languages (Verilog, VHDL, ...). A future extension of qPCF should include a circuit language inspired to QWire.
Second, QWire is a plug-in quantum extension of a classical language, and its dependent types are made available from the host language, while qPCF is a stand-alone language that includes a limited form of dependent types. Finally, QWire allows partial measurements, thus it supports quantum states that mutually interact with the classical environment in the frame of the general QRAM model; in qPCF we focused on restricted co-processors, as widely

explained in Section 2.1. Summarizing, QWire and qPCF shares many aspects, but QWire provides more programming flexibility in the implementation of quantum algorithms than qPCF, while qPCF is a standalone language with cheaper hardware requirements than QWire.

For the sake of completeness, we remark that other approches to quantum programming languages exist in literature. Interesting proposal are, among the others: the functional quantum language (based on strict Linear Logic) QML [2,1,27]; the linear algebraic λ-calculi [4,70,71,3]; the *measurement calculus* [44,15] developed as an efficient rewriting system for measurement based quantum computation; and, last, the quantum-control quantum-data paradigm described in [75].

8 Conclusions

8.1 On the expressive power of qPCF

qPCF is a language able to describe parametric quantum circuit families, in accord with [59], where the difference between the notions of *parameter* (informally, the information available at the compile time, as e.g. the input dimension) and *state* (the information available at run time, e.g. the effective value of quantum data) is highlighted. Indeed, Examples 8 and 9 show how to use dependent types to represent circuit families. The study of the exact expressive power of qPCF, in terms of a formal notion of uniform quantum families is left to future work. Since the limitations we assumed on expressions of type Idx, it is clear that many representations are unsuitable.

A related interesting point is the expressiveness of qPCF w.r.t. quantum algorithms. Since we restrict measurements to total, deferred ones, in qPCF one cannot directly represent algorithms that exploit partial measurement during the computation (see, for example, Shor's original formulation of the factorization algorithm [64]).

Of course, it is possible to encode their "deferred versions" exploiting the deferred measurement principle. See [46] (Section 4.4) for a careful account about the rewriting of a circuit that allows intermediate partial measurement into an equivalent deferred form.

Finally, we remark that by endowin qPCF with an *universal basis* of quantum gates ensures the full expressivity w.r.t. quantum transformations, up to a definition of gate approximation (see [47] for details). We also argue that particular choices of gates in qPCF could return interesting instances of the language. For instance, all reversible circuits are a subset of quantum ones. See [50,51] for a recent characterization of the reversible computing. In particular, qPCF appears to be a simple setting where reversible and classical computation coexists and, potentially, can cooperate.

8.2 Conclusive Statement

We study qPCF, an extension of PCF for quantum circuit generation and evaluation. qPCF pursues seriously the *qd&cc* paradigm in a restricted QRAM environment where only total measurements are allowed. First, this makes quantum programming easy: we can program circuit descriptions by using only classical data. Second, this approach is cheaper than the usual on hardware requirements.

In this work, we explain qPCF syntax, typing rules and a possible formulation of the evaluation semantics. We prove some basic properties of the language. We provide some encoding examples of parametric circuit families that exploit the expressive power of qPCF.

The careful analysis of the exact expressive power of qPCF w.r.t. formal notion of circuit families is an open question we are currently addressing (following [13], where a two-way correspondence between a formal calculus and the finitely generated quantum circuit families [47] is proved). Finally, even if the use of total measurement does not represent a theoretical limitation, a partial measurement operator can represent a useful programming tool. Recently, we proposed an extension of qPCF in order to integrate the possibility to perform partial measurements [53].

References

1. T. Altenkirch and J. Grattage. A functional quantum programming language. In *LICS05*, 2005.
2. T. Altenkirch, J. Grattage, J. K. Vizzotto, and A. Sabry. An algebra of pure quantum programming. In *QPL05*, 2005. ENTCS.
3. P. Arrighi and A. Diaz-Caro. A System F accounting for scalars. *Logical Methods in Computer Science*, Volume 8, Issue 1, Feb. 2012.
4. P. Arrighi and G. Dowek. Linear-algebraic lambda-calculus: higher-order, encodings, and confluence. In A. Voronkov, editor, *RTA*, volume 5117 of *Lecture Notes in Computer Science*, pages 17–31. Springer, 2008.
5. F. Aschieri and M. Zorzi. Non-determinism, non-termination and the strong normalization of system T. In *Typed Lambda Calculi and Applications, 11th International Conference, TLCA 2013, Eindhoven, The Netherlands, June 26-28, 2013. Proceedings*, volume 7941 of *Lecture Notes in Computer Science*, pages 31–47, 2013.
6. F. Aschieri and M. Zorzi. On natural deduction in classical first-order logic: Curry-howard correspondence, strong normalization and herbrand's theorem. *Theoretical Computer Science*, 625:125–146, 2016.
7. D. Aspinall and M. Hofmann. Dependent types. In B. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 2, pages 45–86. MIT Press, 2005.
8. P. Benioff. The computer as a physical system: a microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *J. Statist. Phys.*, 22(5):563–591, 1980.
9. S. Bettelli, T. Calarco, and L. Serafini. Toward an architecture for quantum programming. *The European Physical Journal D - Atomic, Molecular, Optical and Plasma Physics*, 25(2):181–200, Aug 2003.
10. U. Dal Lago, A. Masini, and M. Zorzi. Quantum implicit computational complexity. *Theor. Comput. Sci.*, 411(2):377–409, 2010.
11. U. Dal Lago, A. Masini, and M. Zorzi. Confluence results for a quantum lambda calculus with measurements. *Electr. Notes Theor. Comput. Sci.*, 270(2):251–261, 2011.

12. U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. *RAIRO - Theor. Inf. and Applic.*, 46(3):413–450, 2012.
13. U. d. Dal Lago, A. Masini, and M. Zorzi. On a measurement-free quantum lambda calculus with classical control. *Mathematical. Structures in Comp. Sci.*, 19(2):297–335, 2009.
14. V. Danos and T. Ehrhard. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Information and Computation*, 209(6):966 – 991, 2011.
15. V. Danos, E. Kashefi, and P. Panangaden. The measurement calculus. *J. ACM*, 54(2), Apr. 2007.
16. D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Ser. A*, A400:97–117, 1985.
17. R. Di Cosmo and T. Dufour. The equational theory of $\langle \mathbb{N}, 0, 1, +, \times, \uparrow \rangle$ is decidable, but not finitely axiomatisable. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 240–256, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
18. Z. Diao, M. S. Zubairy, and G. Chen. A quantum circuit design for grover's algorithm. *Zeitschrift für Naturforschung A*, 57(8):701–708, 2002.
19. A. Díaz-Caro, P. Arrighi, M. Gadella, and J. Grattage. Measurements and confluence in quantum lambda calculi with explicit qubits. *Electr. Notes Theor. Comput. Sci.*, 270(1):59–74, 2011.
20. D. P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48, 2000.
21. T. Ehrhard, M. Pagani, and C. Tasson. Full abstraction for probabilistic pcf. *J. ACM*, 65(4):23:1–23:44, Apr. 2018.
22. M. Escardó. Semi-decidability of may, must and probabilistic testing in a higher-type setting. *Electronic Notes in Theoretical Computer Science*, 249:219 – 242, 2009. Proceedings of the 25th Conference on Mathematical Foundations of Programming Semantics (MFPS 2009).
23. M. Gaboardi, L. Paolini, and M. Piccolo. On the reification of semantic linearity. *Mathematical Structures in Computer Science*, 26(5):829–867, 2016.
24. A. Glos and P. Sadowski. Constructive quantum scaling of unitary matrices. *Quantum Information Processing*, 15(12):5145–5154, Dec 2016.
25. J. Goubault-Larrecq. Full abstraction for non-deterministic and probabilistic extensions of pcf i: The angelic cases. *Journal of Logical and Algebraic Methods in Programming*, 84(1):155 – 184, 2015. Special Issue: The 23rd Nordic Workshop on Programming Theory (NWPT 2011) Special Issue: Domains X, International workshop on Domain Theory and applications, Swansea, 5-7 September, 2011.
26. J. Goubault-Larrecq and D. Varacca. Continuous random variables. In *2011 IEEE 26th Annual Symposium on Logic in Computer Science*, pages 97–106, June 2011.
27. J. Grattage. An overview of QML with a concrete implementation in haskell. *Electr. Notes Theor. Comput. Sci.*, 270(1):165–174, 2011.
28. A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 333–342, New York, NY, USA, 2013. ACM.
29. L. K. Grover. Quantum search on structured problems. In *Quantum computing and quantum communications (Palm Springs, CA, 1998)*, volume 1509 of *Lecture Notes in Comput. Sci.*, pages 126–139. Springer, Berlin, 1999.
30. I. Hasuo and N. Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS'11*, pages 237–246, 2011.
31. C. J. Isham. *Lectures on quantum theory.* Imperial College Press, London, 1995. Mathematical and structural foundations.
32. R. Jain, Z. Ji, S. Upadhyay, and J. Watrous. QIP = PSPACE. *J. ACM*, 58(6):30, 2011.
33. P. Kaye, R. Laflamme, and M. Mosca. *An introduction to quantum computing.* Oxford University Press, Oxford, 2007.
34. S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Algorithmic probabilistic game semantics. *Form. Methods Syst. Des.*, 43(2):285–312, Oct. 2013.
35. A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi. *Classical and quantum computation.* AMS, 2002.

36. E. Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Laboratory, 1996. Technical Report.
37. U. D. Lago and M. Zorzi. Wave-style token machines and quantum lambda calculi. In *Proceedings Third International Workshop on Linearity, LINEARITY 2014, Vienna, Austria, 13th July, 2014, Electronic Proceedings in Theoretical Computer Science 176*, pages 64–78, 2014.
38. M. Mahmoud and A. P. Felty. Formalization of Metatheory of the Quipper Programming Language in a Linear Logic. University of Ottawa, Canada, 2018.
39. A. Masini, L. Viganò, and M. Zorzi. Modal Deduction Systems for Quantum State Transformations. *Multiple-Valued Logic and Soft Computing*, 17(5-6):475–519, 2011.
40. P. Maymin. The lambda-q calculus can efficiently simulate quantum computers. Technical Report arXiv:quant-ph/9702057, arXiv, 1997.
41. T. S. Metodi, A. I. Faruque, and F. T. Chong. *Quantum Computing for Computer Architects, Second Edition.* Morgan & Claypool Publishers, 2nd edition, 2011.
42. J. A. Miszczak. *High-level Structures for Quantum Computing.* Morgan and Claypool Publishers, 1st edition, 2014.
43. M. Nakahara and T. Ohmi. *Quantum Computing - From Linear Algebra to Physical Realizations.* CRC Press, 2008.
44. M. Nielsen. Universal quantum computation using only projective measurement, quantum memory, and preparation of the 0 state. *Physical Letters*, 308(2-3):96–100, 2003.
45. M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information.* Cambridge University Press, Cambridge, 2000.
46. M. A. Nielsen and I. L. Chuang. *Quantum computation and quantum information, 10h Anniversary Edition.* Cambridge University Press, Cambridge, 2010.
47. H. Nishimura and M. Ozawa. Perfect computational equivalence between quantum turing machines and finitely generated uniform quantum circuit families. *Quantum Information Processing*, 8(1):13–24, 2009.
48. M. Pagani, P. Selinger, and B. Valiron. Applying quantitative semantics to higher-order quantum computing. In *Proceedings of POPL '14*, pages 647–658. ACM, 2014.
49. L. Paolini. A stable programming language. *Information and Computation*, 204(3):339 – 375, 2006.
50. L. Paolini, M. Piccolo, and L. Roversi. A class of reversible primitive recursive functions. *Electronic Notes in Theoretical Computer Science*, 322(18605):227–242, 2016.
51. L. Paolini, M. Piccolo, and L. Roversi. On a class of reversible primitive recursive functions and its turing-complete extensions. *New Generation Computing*, 36(3):233–256, 2018.
52. L. Paolini, E. Pimentel, and S. Ronchi Della Rocca. An operational characterization of strong normalization. *LNCS*, 3921:367–381, 2006.
53. L. Paolini, L. Roversi, and M. Zorzi. Quantum programming made easy. In V. d. P. Thomas Ehrhard, Maribel Fernández and L. T. de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018), Oxford, UK*, volume 290 of *Electronic Proceedings in Theoretical Computer Science*, pages 58–72, 2019.
54. L. Paolini and M. Zorzi. qPCF: a language for quantum circuit computations. In T. Gopal, G. Jäger, and S. Steila, editors, *Theory and Applications of Models of Computation - 14th Annual Conference, TAMC 2017, Bern, Switzerland, April 20-22, 2017, Proceedings*, volume 10185 of *Lecture Notes in Computer Science*, pages 455–469. Springer, 2017.
55. J. Paykin, R. Rand, and S. Zdancewic. Qwire: A core language for quantum circuits. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 846–858, New York, NY, USA, 2017. ACM.
56. B. C. Pierce. *Types and Programming Languages.* The MIT Press, 2002.
57. G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.
58. D. Richardson and J. P. Fitch. The identity problem for elementary functions and constants. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC '94*, pages 285–290, 1994.

59. F. Rios and P. Selinger. A categorical model for a quantum circuit description language (extended abstract). In B. Coecke and A. Kissinger, editors, Proceedings 14th International Conference on *Quantum Physics and Logic,* Nijmegen, The Netherlands, 3-7 July 2017, volume 266 of *Electronic Proceedings in Theoretical Computer Science*, pages 164–178. Open Publishing Association, 2018.
60. N. J. Ross. *Algebraic and Logical Methods in Quantum Computation.* PhD thesis, Department of Mathematics and Statistics, Dalhousie University, 2015. Available from arXiv:1510.02198.
61. P. Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, Aug. 2004.
62. P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16:527–552, 2006.
63. P. Selinger and B. Valiron. *Semantic Techniques in Quantum Computation*, chapter Quantum lambda calculus, pages pp. 135–172. Cambridge University Press, 2009.
64. P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994)*, pages 124–134. IEEE Comput. Soc. Press, Los Alamitos, CA, 1994.
65. P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.*, 41(2):303–332 (electronic), 1999.
66. P. W. Shor. Introduction to quantum algorithms. In *Proceedings of Symposia in Applied Mathematics*, volume 58, pages 143–160, 2002.
67. E. Strubell. An introduction to quantum algorithms, lecture notes, university of massachusetts. 2011.
68. B. Valiron. Quantum computation: From a programmer's perspective. *New Generation Computing*, 31(1):1–26, Jan 2013.
69. B. Valiron, N. J. Ross, P. Selinger, D. S. Alexander, and J. M. Smith. Programming the quantum future. *Commun. ACM*, 58(8):52–61, 2015.
70. L. Vaux. Lambda-calculus in an algebraic setting, 2006.
71. L. Vaux. The algebraic lambda-calculus, 2009.
72. L. Viganò, M. Volpe, and M. Zorzi. Quantum state transformations and branching distributed temporal logic. In *Proceedings of the 21st International Workshop on Logic, Language, Information, and Computation (WoLLIC)*, volume 8652 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2014.
73. L. Viganò, M. Volpe, and M. Zorzi. A branching distributed temporal logic for reasoning about entanglement-free quantum state transformations. *Information and Computation*, 255:311–333, 2017.
74. H. Xi and F. Pfenning. Dependent types in practical programming. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 214–227, New York, NY, USA, 1999. ACM.
75. M. Ying. *Foundations of Quantum Programming.* Morgan Kaufmann, 2016.
76. C. Zenger. Indexed types. *Theoretical Computer Science*, 187(1):147 – 165, 1997.
77. M. Zorzi. On quantum lambda calculi: a foundational perspective. *Mathematical Structures in Computer Science*, 26(7):1107–1195, 2016.